НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ
В.о. Завідувача кафедри
____ Михайло НОВОТАРСЬКИЙ
(підпис)

“__” _____2025 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Комп’ютерні системи та мережі”

спеціальності 123 “Комп’ютерна інженерія”

на тему: Система аналізу та прогнозування погодних умов з використанням методів глибинного навчання

Виконав : студент __4__ курсу, групи _ІО-14
(шифр групи)

_____Чижов Олексій Юрійович_____
(прізвище, ім’я, по батькові) (підпис)

Керівник _____ас. Гордієнко Нікіта Юрійович_____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) ас. Гончаренко Олександр Олексійович_____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент __ас. Соколовський Владислав Володимирович_____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2025 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Комп’ютерні системи та мережі”

спеціальності 123 “Комп'ютерна інженерія”

ЗАТВЕРДЖУЮ
В.о. Завідувача кафедри
____ Михайло НОВОТАРСЬКИЙ
(підпис)

“__” _____2025 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Чижова Олексія Юрійовича

1. Тема проєкту *Система аналізу та прогнозування погодних умов з використанням мтеодів глибинного навчання*
керівник проєкту Гордієнко Нікіта Юрійович, асистент,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 23 травня 2025 року №1705-с

2. Термін здачі студентом закінченого проєкту *6 червня 2025 р.*

3. Вихідні дані до проєкту *технічна документація, теоретичні дані.*

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

Розділ 1. Аналіз предметної області.

Розділ 2. Огляд алгоритмів та технологій для розробки системи.

Розділ 3. Деталі розробки системи.

Розділ 4. Дослідження та аналіз розробленої системи.

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) структурна схема системи, функціональна схема (діаграма класів), алгоритм дій програмного забезпечення.

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Гончаренко О. О.	02.06.2025	05.06.2025

7. Дата видачі завдання «30» серпня 2024 р.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*29.01.2025-03.02.2025*	
2.	*Вивчення та аналіз завдання*	*03.02.2025-15.04.2025*	
3.	*Розробка архітектури та загальної структури системи*	*15.04.2025.-26.04.2025*	
4.	*Розробка структур окремих підсистем*	*26.04.2025-05.05.2025*	
5.	*Програмна реалізація системи*	*05.05.2025-16.05.2025*	
6.	*Оформлення пояснювальної записки*	*15.04.2025-19.05.2025*	
7.	*Захист програмного продукту*	*25.05.2025*	
8.	*Передзахист*	*02.06.2025*	
9.	*Захист*	*20.06.2025*	

Студент-дипломник _____ Олексій ЧИЖОВ
(підпис)

Керівник проєкту _____ Нікіта ГОРДІЄНКО
(підпис)

АНОТАЦІЯ

Дана дипломна робота присвячена створенню системи аналізу та прогнозування погодних умов. Проєкт включає в себе дослідження предметної області, а саме існуючих систем, що мають схоже застосування. Під час виконання також були розглянуті та проаналізовані різні підходи до виконання поставленої задачі, що включають в себе різні методи машиного навчання, серед яких моделі глибокого навчання, модель градієнтного бустингу; для порівняння також використано статистичну модель часових рядів. Взаємодія користувача зі системою відбувається за допомогою інтерфейсу програмного застосунку. У якості датасету використано історичні дані для великих міст України, що отримані за допомогою сервісу Visual Crossing. Розроблена програма надає користувачу можливість переглядати різні погодні дані для обраних міст. Система була розроблена на мові програмування Python.

Ключові слова: прогнозування погоди, машинне навчання, глибоке навчання, часові ряди, Python.

ANNOTATION

This thesis is dedicated to the development of a system for the analysis and forecasting of weather conditions. The project includes a study of the subject area, particularly existing systems with similar applications. During the implementation, various approaches to solving the stated problem were considered and analyzed, including different machine learning methods such as deep learning models and a gradient boosting model. For comparison, a statistical time series model was also used. User interaction with the system is carried out through a software application interface. Historical weather data for major cities of Ukraine, obtained via the Visual Crossing service, was used as the dataset. The developed program allows users to view various weather parameters for selected cities. The system was developed using the Python programming language.

Keywords: weather forecasting, machine learning, deep learning, time series, Python.

справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпля	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.467200.002 ТЗ***	Система аналізу та	3		
			Прогнозування погодних			
			умов з використанням			
			методів машинного			
			навчання			
			Технічне завдання			
	А4	***ІАЛЦ.467200.003 ПЗ***	Пояснювальна записка	61		
	А4	***ІАЛЦ.467200.004 Д1***	Діаграма компонентів	1		
	А4	***ІАЛЦ.467200.005 Д2***	Діаграма класів	1		
	А4	***ІАЛЦ.467200.006 Д3***	Алгоритм дій користувача	1		
	А4	***ІАЛЦ.467200.007 Д4***	Текст програмного коду	22		

					ІАЛЦ.467200.001 ОА
Зм	***Лист***	***№ докум.***	***Підп***	***Дата***	
Розроб		Чижов О.Ю.			*Система аналізу та прогнозування погодних умов з використанням методів глибинного навчання* Опис альбому
Перев		Волокита А.М.			

Літ.	Аркуш	Аркушів
	1	1

НТУУ "КПІ" ФІОТ ІО-14

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Система аналізу та прогнозування погодних умов з використанням методів глибинного навчання*»

Київ – 2025

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на розробку системи аналізу та прогнозування погодних умов з використанням методів глибинного навчання, а також на подальшу підтримку та вдосконалення розробленої системи.

Областю застосування цієї системи є метеорологічне прогнозування, аналіз кліматичних даних, підтримка сільськогосподарських і транспортних систем, планування заходів у сфері екології, управління ризиками, пов’язаними з погодними умовами, а також надання точних прогнозів для потреб населення та бізнесу. Система може бути використана для обробки історичних і поточних погодних даних, створення короткострокових і середньострокових прогнозів.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки даної системи є завдання для виконання роботи кваліфікаційно-освітнього рівня «бакалавр комп’ютерної інженерії», який був затверджений факультетом “Інформатики та обчислювальної техніки” кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою та призначенням даної роботи є розробка системи аналізу та прогнозування погодних умов з використанням методів глибинного навчання, що дозволить ефективно обробляти історичні та поточні метеорологічні дані, створювати точні прогнози погоди та надавати користувачам зручний інтерфейс для перегляду погодних умов і прогнозів.

4 ДЖЕРЕЛА РОЗРОБКИ

Для створення цього дипломного проекту використано офіційні документи, наукові статті, публікації в мережі Інтернет, а також спеціалізовану науково-технічну літературу.

5 ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

Розроблена система аналізу та прогнозування погодних умов має відповідати таким вимогам:

- Зручний та інтуїтивно зрозумілий інтерфейс: Графічний інтерфейс на базі Tkinter із підтримкою інтерактивної карти (tkintermapview) для простого доступу користувачів до прогнозу погоди.
- Обробка та відображення даних: Можливість автоматичного завантаження історичних погодних даних (наприклад, із Visual Crossing чи даних користувача) та отримання поточних даних через Timeline Weather API з відображенням прогнозів для обраних міст.
- Автоматизоване прогнозування: Система самостійно обробляє дані та надає користувачу точний прогноз погоди без необхідності налаштування параметрів моделей.
- Документація: Надання чіткої, вичерпної та зрозумілої документації, що описує функціонал системи, інструкції з використання та особливості роботи програмного продукту.

5.2. Вимоги до програмного забезпечення

- ОС Windows, Mac або Linux.
- Python версії 3.12 або вище.

5.3. Вимоги до апаратної частини

- ЦП не менше ніж AMD Ryzen 5 3600.
- ROM не менше ніж 64 ГБ.
- RAM не менше ніж 8 ГБ.

6 ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	29.01.2025-03.02.2025
Вивчення та аналіз завдання	03.02.2025-15.04.2025
Розробка архітектури та загальної структури системи	15.04.2025.-26.04.2025
Розробка структур окремих частин системи	26.04.2025-05.05.2025
Програмна реалізація системи	05.05.2025-10.05.2025
Виправлення помилок	10.05.2025-16.05.2025
Оформлення пояснювальної записки	15.04.2025-19.05.2025

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Система аналізу та прогнозування погодних умов з використанням методів глибинного навчання*»

Київ – 2025

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

API — Application Programming Interface

LSTM — Long Short-Term Memory

TCN — Temporal Convolutional Network

GUI — Graphical User Interface

XGBoost — Extreme Gradient Boosting

GPU — Graphics Processing Unit

ВСТУП

Сучасна метеорологія та кліматологія значною мірою спираються на передові технології для аналізу даних і прогнозування погодних умов. Такі завдання є складними через нелінійність метеорологічних процесів, високу розмірність даних, їх мультимодальність, а також значну обчислювальну складність, спричинену необхідністю врахування численних факторів, таких як температура, вологість чи атмосферні явища. Традиційні підходи до прогнозування часто виявляються недостатньо точними, особливо для середньо- та довгострокових прогнозів, через складну природу погодних явищ.

У цьому контексті методи глибинного навчання набувають дедалі більшої популярності завдяки своїй здатності обробляти великі обсяги даних і виявляти складні закономірності. Такі методи дозволяють створювати точніші прогнози, використовуючи як історичні дані, так і поточні метеорологічні показники, отримані з різних джерел.

Ця дипломна робота присвячена розробці системи аналізу та прогнозування погодних умов із застосуванням методів глибинного навчання. Метою роботи є створення зручного програмного продукту з графічним інтерфейсом, який забезпечує користувачам доступ до точних прогнозів погоди для обраних міст України. У роботі розглядаються основні підходи до моделювання погодних даних, включаючи статистичні методи, нейронні мережі та гібридні моделі, а також аналізуються сучасні тенденції в метеорологічному прогнозуванні на основі україномовних і англомовних джерел.

Запропонована система має практичне значення для метеорологічних служб, сільського господарства, транспортних компаній та інших сфер, де точні прогнози погоди є критично важливими. Робота також включає класифікацію застосованих підходів і обґрунтування вибору технологій для реалізації проєкту, з акцентом на їх ефективність і доступність для користувачів.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Актуальність сервісів прогнозування погодних умов

Важко переоцінити значення прогнозування погоди для людей у сучасному світі. Точні та своєчасні прогнози погоди відіграють важливу роль у сільському господарстві, транспортній логістиці, енергетиці, туризмі, а також у плануванні заходів із запобігання надзвичайним ситуаціям, спричиненим погодними умовами, такими як урагани, повені чи посухи. Крім того, у зв'язку з глобальними кліматичними змінами актуальність розробки ефективних сервісів прогнозування погоди лише зростає, оскільки метеорологічні явища стають менш передбачуваними, а їх наслідки – більш масштабними.

Сучасні сервіси прогнозування погоди використовуються не лише професійними метеорологами, але й широким колом користувачів, включаючи фермерів, які планують посівні роботи, авіакомпанії, що оптимізують маршрути польотів, та звичайних громадян, які приймають повсякденні рішення, наприклад, щодо вибору одягу чи планування відпочинку.

Розвиток інформаційних технологій, зокрема методів глибинного навчання, відкриває нові можливості для підвищення якості прогнозів. Такі методи дозволяють ефективно обробляти великі обсяги метеорологічних даних, враховувати нелінійні залежності та створювати моделі, здатні прогнозувати погодні умови з урахуванням складних взаємозв'язків між різними параметрами.

З сучасними можливостями створення великої кількості мобільних дешевих станцій, наявність системи, що могла б аналізувати незалежні дані, є дуже корисною. Доступ до актуальної інформації щодо змін погодних умов, особливо у місцях, де відсутні стаціонарні метеостанції, став би в нагоді працівникам аграрної та інших сфер.

1.2.Загальний огляд існуючих сервісів для аналізу та прогнозування погодних умов

Сервіси прогнозування погоди відіграють ключову роль у наданні точної метеорологічної інформації, яка застосовується в повсякденному житті та професійній діяльності. Нижче представлено короткий огляд сервісів, що вирізняються своєю функціональністю та популярністю.

- AccuWeather - глобальний лідер у прогнозуванні погоди, який використовує передові технології для створення високоточних прогнозів. Надає локалізовані дані, погодинні прогнози та унікальні функції, такі як оцінка суб'єктивного відчуття температури.
- Met Office - офіційна метеорологічна служба Великобританії, що забезпечує надійні прогнози для багатьох країн. Відрізняється простим інтерфейсом і спеціалізованими послугами для різних галузей.
- Sinoptik.ua - український сервіс із зручним інтерфейсом. Орієнтований на місцевих користувачів, забезпечує погодинні та довгострокові прогнози з акцентом на простоту використання.

Ці сервіси демонструють ефективне поєднання сучасних технологій і зручних інтерфейсів, що забезпечує їхню популярність. Подальший детальний аналіз кожного сервісу розкриє їхні унікальні особливості та переваги.

1.3. AccuWeather

За заявами розробників, «AccuWeather, визнана та задокументована як найточніше джерело прогнозів погоди та попереджень у світі, врятувала десятки тисяч життів, запобігла сотням тисяч травм і допомогла зберегти майно на десятки мільярдів доларів. Сьогодні AccuWeather є найвідомішим і найпопулярнішим джерелом прогнозів погоди та попереджень у світі, відомим мільярдам людей, і доведено, що є найточнішим. У цифровому просторі AccuWeather – це провідне місце для прогнозів погоди у світі та один із найпопулярніших сайтів загалом» [1]. Сервіс вирізняється високою точністю

прогнозів, інноваційними технологіями та широкою доступністю завдяки офіційному вебсайту, мобільним додаткам та телебаченню [2]. AccuWeather – це професійний метеорологічний сервіс, призначений для забезпечення користувачів у всьому світі, зокрема в Україні, точними та деталізованими прогнозами погоди. Основною метою сервісу є надання актуальних метеорологічних даних, які сприяють ефективному плануванню особистих і професійних завдань, включаючи організацію подорожей, логістичних операцій та інших видів діяльності. Сервіс обробляє значні обсяги даних, отриманих із супутників, метеостанцій і радарів, та презентує їх у структурованому й зрозумілому форматі для кінцевих користувачів. На рисунку 1.1 зображено головну сторінку сайту.

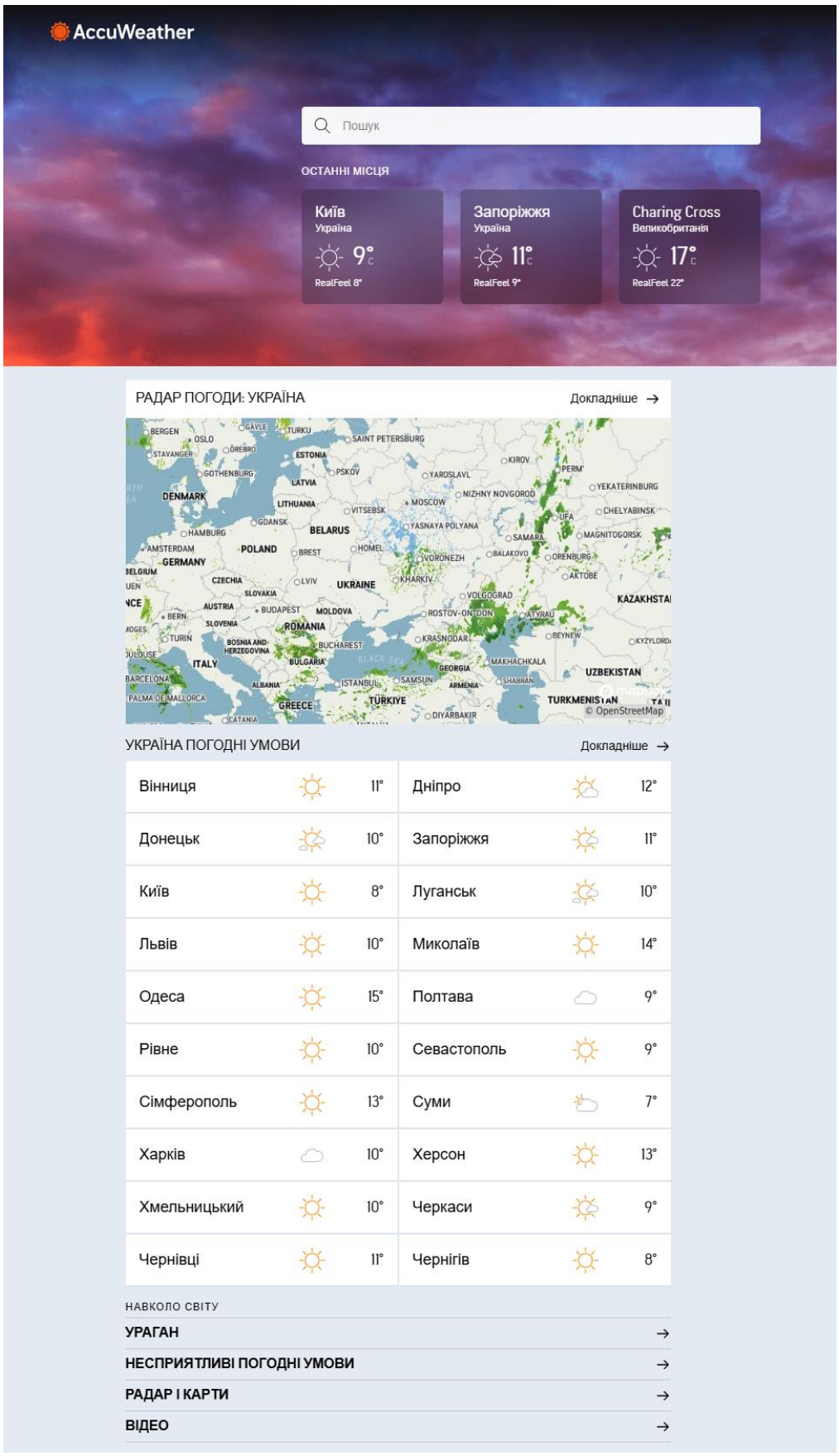


Рисунок 1.1 – Головна сторінка сайту accuweather.com

На вебсайті AccuWeather користувачі можуть переглядати погодні прогнози для різних локацій, включаючи міста, регіони чи навіть конкретні адреси. Доступні погодинні, щоденні та довгострокові прогнози (до 90 днів), які включають такі параметри, як температура, опади, вологість, швидкість і напрям вітру, ультрафіолетовий індекс та якість повітря. Унікальною особливістю є функція MinuteCast, яка надає похвилинний прогноз опадів на найближчі чотири години, що корисно для планування короткострокових заходів (рис. 1.2).

Рисунок 1.2 – функція MinuteCast

Сервіс також пропонує інтерактивні карти, які дозволяють відстежувати рух хмар, опади, температуру та інші метеорологічні явища в реальному часі (рис. 1.3). Користувачі можуть переглядати супутникові зображення (RealVue™ та Enhanced RealVue™), карти радарів для відстеження опадів, штормів і навіть тропічних циклонів. Окрім цього, AccuWeather надає попередження про екстремальні погодні умови, такі як урагани, повені чи снігопади, що сприяє підвищенню безпеки [3].

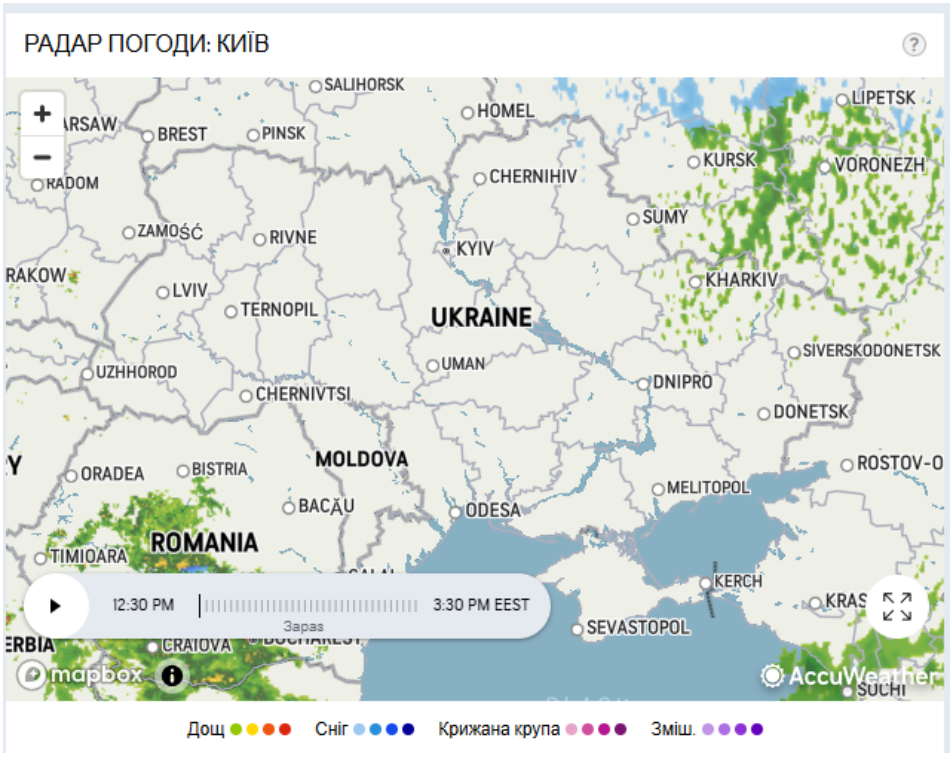


Рисунок 1.3 – карта радару погоди

Для специфічних потреб, наприклад, для людей із алергіями, сервіс пропонує розділ із даними про рівень пилку та якість повітря. Користувачі можуть налаштувати інтерфейс, зберігати кілька локацій і отримувати персоналізовані сповіщення про зміни погоди. AccuWeather також підтримує аналітику для бізнесу, надаючи спеціалізовані прогнози для авіації, сільського господарства, логістики та інших галузей.

У таблиці, що наведена нижче, описані переваги та недоліки сервісу AccuWeather.

Таблиця 1.1. – Плюси та мінуси AccuWeather

Категорія	**Опис**
Плюси	
Висока точність прогнозів	Передові алгоритми та численні джерела даних Функція MinuteCast для точного передбачення початку/кінця опадів

Кінець таблиці 1.1.

Категорія	**Опис**
Широкий функціонал	Погодинні, щоденні й довгострокові прогнози з деталями (температура, вологість, УФ-індекс тощо) Інтерактивні карти та супутникові зображення
Персоналізація	Збереження кількох локацій, налаштування сповіщень, дані про алергени та якість повітря, адаптивний інтерфейс для різних пристроїв
Попередження про негоду	Своєчасні сповіщення про небезпечні погодні явища Інформація про шторми, урагани, снігопади
Доступність	Безкоштовна версія з широким функціоналом Доступ через вебсайт, мобільні додатки та інші платформи
Мінуси	
Платний контент	Деякі функції доступні лише за підпискою (наприклад, Premium+ за $4/місяць) Обмеження для користувачів без платної версії
Неточності в регіонах	Можливі похибки в локальних прогнозах у регіонах з малою кількістю метеостанцій Складність прогнозування екстремальних умов
Реклама у безкоштовній версії	Значна кількість реклами Деякі оголошення є нав’язливими
Конфіденційність даних	Збір персональних даних, включаючи геолокацію Дані можуть передаватися третім сторонам (наприклад, Amazon, Microsoft Azure) [4]

Таким чином, AccuWeather є універсальним інструментом для прогнозування погоди, який пропонує широкий набір функцій для різних категорій користувачів. Незважаючи на деякі обмеження, такі як платний контент і реклама, сервіс залишається одним із найпопулярніших завдяки своїй точності, зручності та доступності.

1.4. Met Office

Наступним розглянутим сервісом є MetOffice. За інформацією розробників, «Met Office є національною метеорологічною службою Великобританії, яка надає критично важливі погодні послуги та провідну у світі кліматичну науку, допомагаючи кожному залишатися в безпеці та процвітати» [5]. Сервіс відомий своєю точністю, науковими інноваціями та широким спектром метеорологічних послуг, доступних через офіційний вебсайт, мобільні додатки, телебачення, радіо та спеціалізовані канали для бізнесу й уряду.

Основна мета сервісу полягає в наданні точних метеорологічних даних і попереджень, які підтримують безпеку, планування та прийняття рішень у різних сферах, від повсякденного життя до авіації, сільського господарства й управління надзвичайними ситуаціями. Met Office обробляє дані з глобальної мережі спостережень, використовуючи суперкомп’ютери для створення прогнозів, які є основою для багатьох погодних сервісів. На рисунку 1.4 зображено головну сторінку сайту.

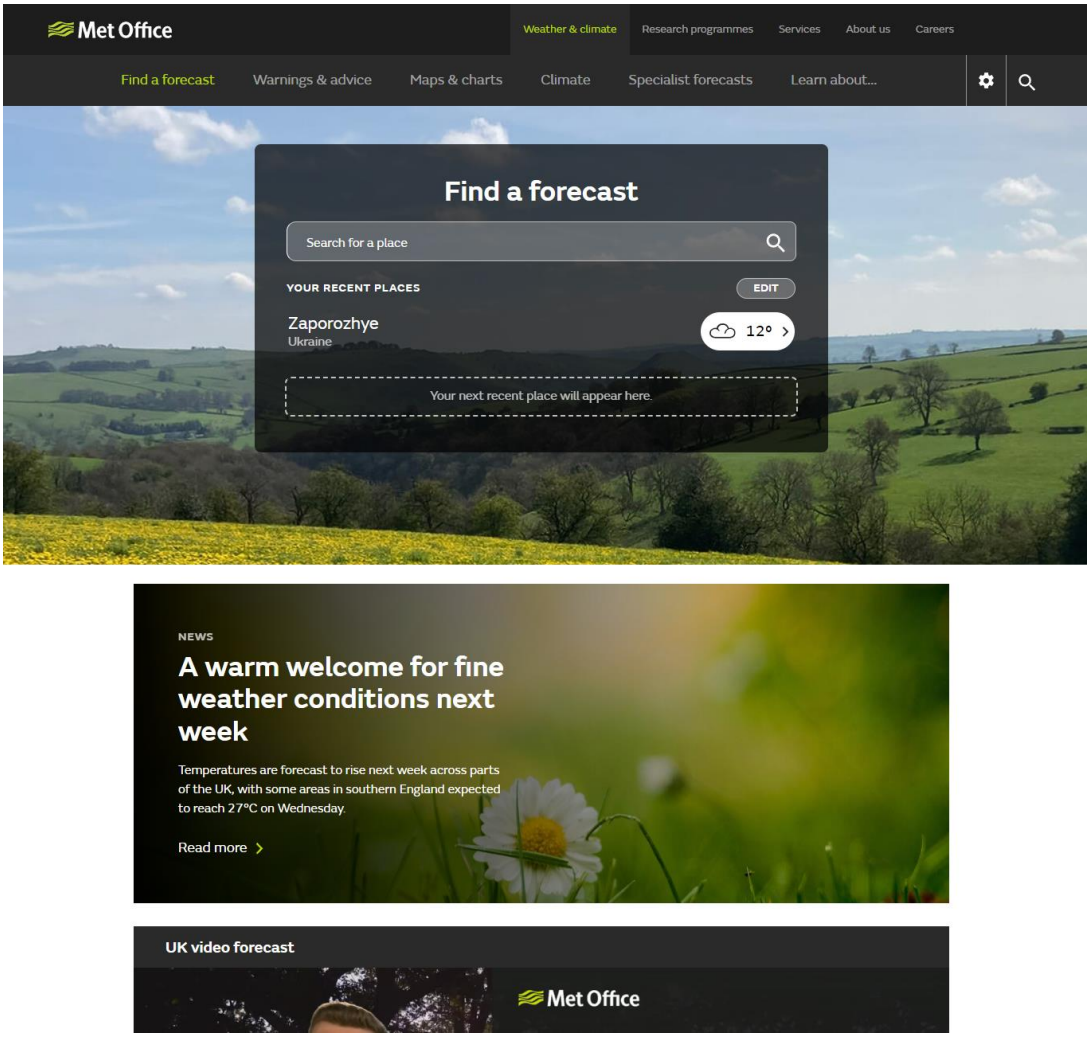


Рисунок 1.4 – Головна сторінка сайту metoffice.gov.uk

На вебсайті Met Office користувачі можуть отримувати прогнози погоди для будь-якої локації, включаючи міста та регіони. Сервіс пропонує погодинні, щоденні та тижневі прогнози (до 7 днів), а також довгострокові прогнози для Великобританії (до 28 днів). Прогнози включають такі параметри, як температура, ймовірність опадів, швидкість і напрям вітру, вологість, видимість, ультрафіолетовий індекс і тиск. Однією з ключових особливостей для користувачів у Великобританії є National Severe Weather Warnings, які попереджають про небезпечні погодні явища, такі як шторми, повені чи сильні снігопади, із детальними картами зон ризику [6] (рис. 1.5). Для інших країн, включаючи Україну, доступні загальні прогнози та карти без локалізованих попереджень.

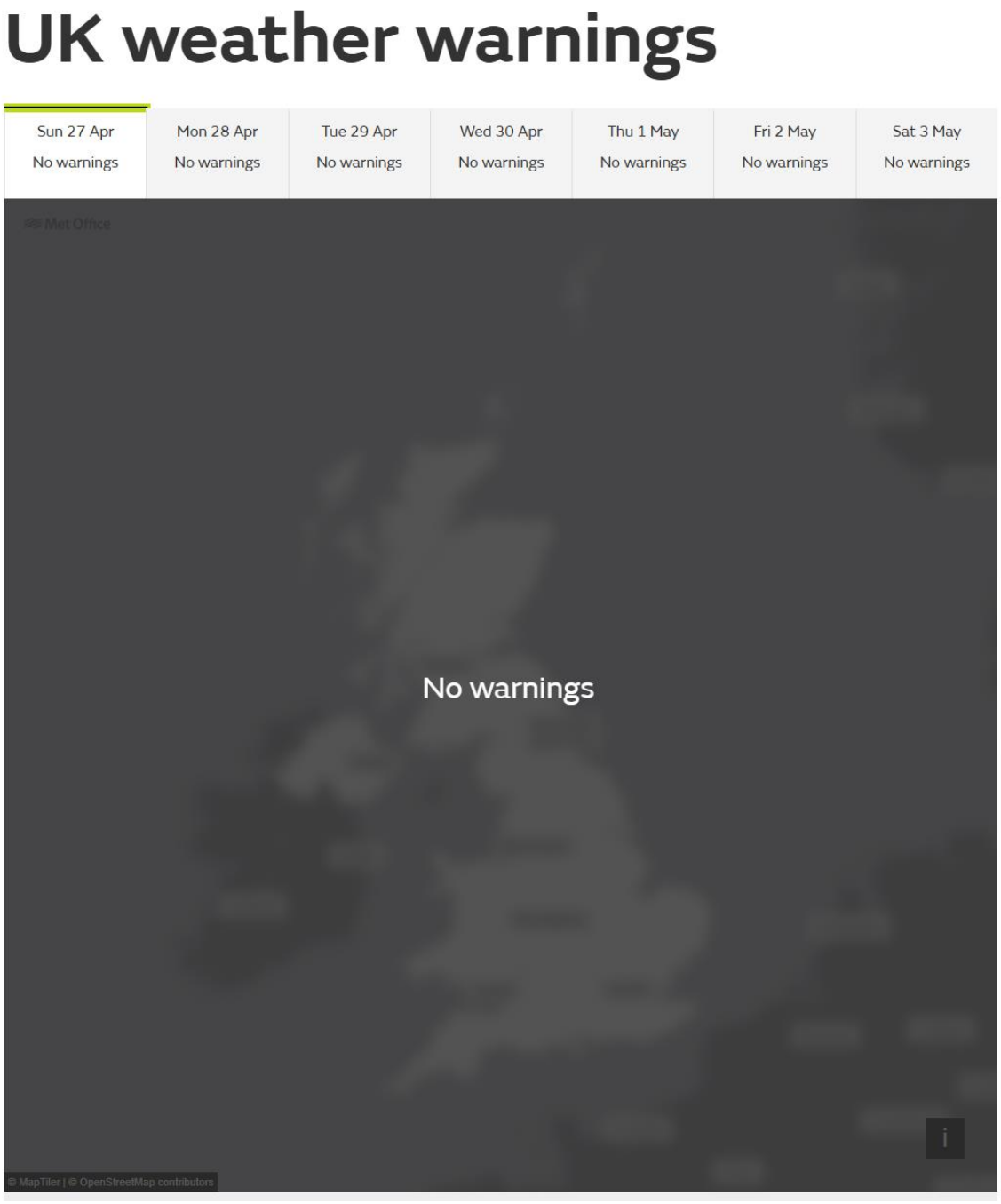


Рисунок 1.5 – Функція попередження про небезпечні погодні явища

Met Office також надає інтерактивні карти, які дозволяють відстежувати опади, хмарність і тиск у реальному часі, а також переглядати дані за останні 24 години. Для користувачів доступні спеціалізовані розділи, наприклад, прогнози якості повітря, рівня пилку (з березня по вересень). Сервіс підтримує персоналізацію: користувачі можуть зберігати кілька локацій і отримувати push-сповіщення про зміни погоди чи попередження через мобільний додаток.

Для професійного використання Met Office пропонує спеціалізовані послуги, такі як прогнози для авіації [7], морські прогнози [8] і консультації для бізнесу. Сервіс активно застосовує штучний інтелект і машинне навчання для вдосконалення моделей прогнозування, що забезпечує високу точність.

У таблиці 1.2. наведено переваги та недоліки сервісу MetOffice.

Таблиця 1.2. – Плюси та мінуси MetOffice

Категорія	**Опис**
Плюси	
Висока точність прогнозів	Завдяки використанню великих обчислювальних потужностей досягається висока точність прогнозів [5] [9]
Широкий спектр даних	Прогнози на годину, день і тиждень, різні параметри: якість повітря, пилок, космічна погода, інтерактивні карти
Попередження про негоду	National Severe Weather Warnings: сповіщення про шторми, повені, снігопади, карти зон ризику
Професійні послуги	Прогнози для авіації, судноплавства, агросектору та уряду, консультації щодо кліматичних ризиків, участь у програмах IPCC, WAFS [10]
Зручність і доступність	Інтуїтивний інтерфейс вебсайту та додатку, нагороди WMO за якість [10], безкоштовний доступ до основних функцій

Кінець таблиці 1.2.

Категорія	**Опис**
Мінуси	
Обмежена локалізація	Менш точні прогнози поза Великобританією (наприклад, для України), обмежений доступ до деяких функцій, як прогнози пилку
Платні професійні послуги	Доступ до спеціалізованих даних лише за підпискою, API потребує комерційної ліцензії
Реклама у безкоштовній версії	Наявність реклами у додатку- Можливість вимкнення за £2.99 [10]. Деякі користувачі вважають рекламу нав’язливою
Конфіденційність даних	- Збір геолокації та даних використання додатку- Дані можуть передаватися партнерам для аналітики, що викликає занепокоєння [11]

1.5. Sinoptik.ua

Останній розглянутий сервіс це Sinoptik.ua. Sinoptik.ua – це популярний український метеорологічний сервіс, призначений для надання точних і зручних прогнозів погоди, насамперед для користувачів в Україні. Він охоплює 29 815 населених пунктів України, а також понад 104 000 міст світу з прогнозом до 10 днів. Відомий своєю локалізацією, простим інтерфейсом і доступністю через вебсайт та інтеграцію з іншими платформами, Sinoptik.ua забезпечує населення актуальними даними про погодні умови для планування повсякденних справ, сільськогосподарських робіт, подорожей чи інших видів діяльності. Дані включають температуру, ймовірність опадів, вологість,

швидкість і напрям вітру, атмосферний тиск. На рисунку 1.6 зображено головну сторінку сайту.

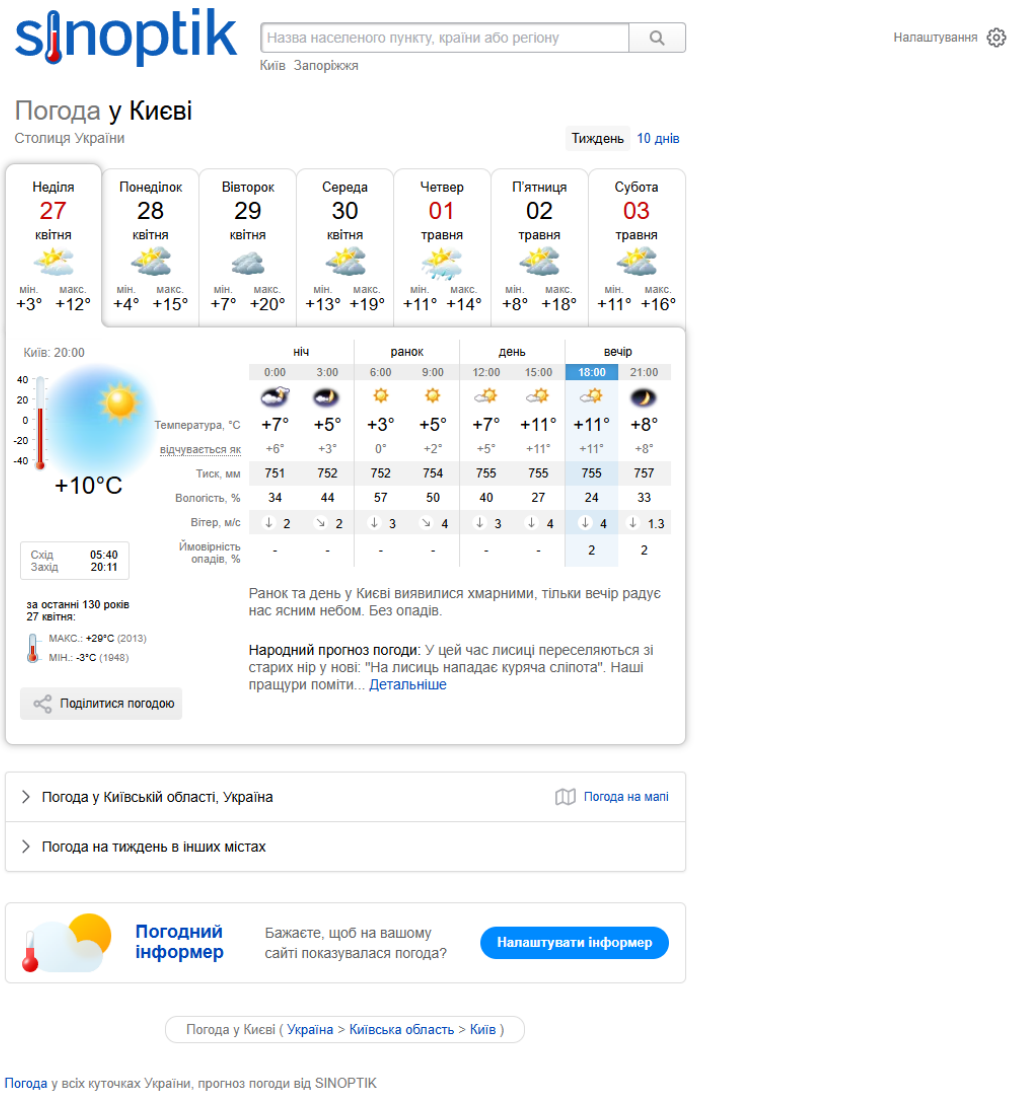


Рисунок 1.6 – Головна сторінка сайту sinoptik.ua

Sinoptik.ua надає доступ до інтерактивної карти, які дозволяють відстежувати температуру та стан неба (хмарність або ясність) у реальному часі по всій Україні. Сервіс також надає інформацію про схід і захід Сонця, тривалість дня.

Для зручності користувачів Sinoptik.ua підтримує персоналізацію: можна обрати кілька локацій для швидкого доступу до прогнозів, а також отримати сповіщення про зміни погоди через інтеграцію з браузером або партнерськими платформами.

У таблиці нижче підсумовано переваги та недоліки сервісу Sinoptik.ua.

Таблиця 1.3. – Плюси та мінуси Sinoptik.ua

Категорія	**Опис**
Плюси	
Локалізація для України	Охоплює всі 29 815 населених пунктів України, включаючи малі села, інтерфейс повністю україномовний
Широкий спектр даних	Погодинні, щоденні та 10-денні прогнози, параметри: температура, тиск, вологість, інтерактивні карти для відстеження погоди в реальному часі
Зручність і доступність	Простий і зрозумілий інтерфейс, що підходить для будь-якого віку, безкоштовний доступ до всіх основних функцій через вебсайт
Мінуси	
Обмежена точність	Менша точність прогнозів у віддалених регіонах через обмежену кількість метеостанцій, низька надійність довгострокових прогнозів (понад 7 днів)
Реклама на сайті	Значна кількість реклами
Відсутність мобільного додатку	Відсутність офіційного мобільного додатку, доступ лише через вебверсію або партнерів
Конфіденційність даних	Збір геолокації та поведінкових даних, можливість передачі рекламним партнерам [12]

ВИСНОВКИ ДО РОЗДІЛУ

У першому розділі проєкту було проведено аналіз предметної області прогнозування погоди, підкреслено актуальність створення точної незалежної системи на основі розглянутих сучасних аналогів. Встановлено, що точні прогнози погоди мають критичне значення для багатьох сфер, таких як сільське господарство, транспорт, енергетика, туризм і управління надзвичайними ситуаціями. Зростання попиту на персоналізовані та локалізовані прогнози, а також розвиток інформаційних технологій, зокрема методів глибинного навчання, відкривають нові можливості для підвищення якості прогнозів і створення зручних систем для користувачів.

Детальний огляд трьох популярних сервісів – AccuWeather, Met Office і Sinoptik.ua – показав їхні сильні та слабкі сторони, а також ключові функціональні особливості. AccuWeather вирізняється високою точністю, інноваційними функціями, такими як MinuteCast, і широким набором даних, включаючи інтерактивні карти та аналітику для бізнесу, але має обмеження у вигляді платного контенту та реклами. Met Office, як національна метеорологічна служба Великобританії, пропонує надійні прогнози, підкріплені суперкомп'ютерними обчисленнями, і спеціалізовані послуги для професійних користувачів, однак його функціонал менш деталізований для регіонів поза Великобританією, таких як Україна. Sinoptik.ua адаптований до потреб українських користувачів, охоплює всі населені пункти країни та має простий інтерфейс, але обмежений відсутністю мобільного додатку та значною кількістю реклами.

Аналіз цих сервісів виявив спільні тенденції у розвитку метеорологічних платформ: використання великих обсягів даних із супутників і метеостанцій, інтеграція інтерактивних карт, персоналізація для користувачів і акцент на сповіщеннях про екстремальні погодні умови. Водночас можна відмітити

відсутність підтримки користувацьких даних, що може бути виправлено у даній роботі.

РОЗДІЛ 2
ОГЛЯД ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ СИСТЕМИ

У 2 розділі будуть детально розглянуті основні бібліотеки та алгоритми, використані для створення системи.

2.1. Огляд доступних технологій

Для створення системи прогнозування погоди важливо обрати ефективні інструменти для обробки даних, побудови моделей машинного навчання та реалізації інтерфейсу для користувачів. У цьому підрозділі наведено короткий огляд популярних бібліотек та фреймворків, які можуть бути застосовані на кожному з етапів розробки – від попередньої обробки даних до побудови інтерфейсу користувача.

2.1.1. Бібліотеки для роботи з даними

Для обробки табличних даних у простих проєктах найчастіше використовується бібліотека Pandas, яка забезпечує зручний і гнучкий API для фільтрації, агрегації, об’єднання даних та роботи з датами [13]. Це ідеальний інструмент для аналізу середніх за обсягом наборів даних, але її продуктивність знижується при масштабуванні.

Для великих обсягів даних можливим вибором є Dask – бібліотека, яка розширює можливості Pandas у паралельному та розподіленому середовищі. Вона дозволяє обробляти дані, що не поміщаються в оперативну пам’ять, але потребує додаткових зусиль для налаштування та оптимізації [14].

2.1.2. Фреймворки глибокого навчання

Серед найпопулярніших платформ для глибокого навчання вирізняються TensorFlow та його високорівнева надбудова Keras [15]. Ці інструменти пропонують інфраструктуру для розробки нейронних мереж, включаючи

моделі для аналізу послідовностей, такі як LSTM і TCN. Їхні переваги – масштабованість, оптимізація для продуктивності та активна спільнота, хоча складність налагодження може бути перешкодою для новачків.

Альтернативою є PyTorch, який має більш гнучкий підхід завдяки динамічному графу обчислень. Він особливо популярний у наукових дослідженнях завдяки зручності відлагодження та інтерактивній роботі, але є складнішим для освоєння [16].

2.1.3. Моделі машинного навчання

Для високоточної обробки табличних даних часто застосовують XGBoost – алгоритм градієнтного бустингу дерев рішень, який добре контролює перенавчання, ефективний і масштабований [17]. Його недоліком є високе споживання ресурсів при роботі з великими датасетами.

Для більш традиційних задач можна використовувати Scikit-learn – універсальну бібліотеку, яка містить широкий набір алгоритмів класифікації, регресії та кластеризації. Вона відзначається простотою інтерфейсу, але менш ефективна на великих даних [18].

2.1.4. Інструменти для прогнозування часових рядів

Для обробки часових рядів з вираженою сезонністю популярною є бібліотека Prophet, розроблена в Facebook [19]. Вона дозволяє легко моделювати тренди та циклічність за часовими мітками, але має обмежену гнучкість для складніших задач.

NeuralProphet поєднує ідеї Prophet із силою нейронних мереж. Цей фреймворк дозволяє враховувати додаткові ознаки та більш складні залежності, але потребує тоншого налаштування [20].

2.1.5. Інтерфейс користувача

Для створення простих графічних інтерфейсів вбудованим засобом Python є Tkinter. Він дозволяє швидко створювати вікна, кнопки, списки, однак має обмежені можливості стилізації [21]. Для розширення функціоналу можна використати tkintermapview, що дозволяє інтегрувати мапи з підтримкою координат, маркерів і масштабування [22].

Для створення більш складного і сучасного GUI застосовують PySide або PyQt – фреймворки на базі Qt. Вони підтримують адаптивні макети, теми, взаємодію з системними компонентами, але вимагають більше ресурсів і знань [23].

2.1.6. Геовізуалізація

Для візуалізації географічних даних використовується бібліотека Cartopy, яка інтегрується з matplotlib і дозволяє створювати карти з різними шарами: межі країн, координати, погодні дані. Цей інструмент гнучкий і потужний, але вимагає певного досвіду в налаштуванні [24].

Далі представлена таблиця з коротким порівнянням наведених технологій.

Таблиця 2.1. – Порівняння розглянутих технологій

Технологія	**Призначення**	**Переваги**	**Недоліки**
Pandas	Табличні дані	Простота, гнучкий API	Обмежена масштабованість
Dask	Великі дані	Паралельна обробка, масштабування	Складніше налаштування
TensorFlow / Keras	Нейромережі	Велика екосистема, оптимізація продуктивності	Вища складність у відлагодженні
PyTorch	Нейромережі	Інтерактивність, гнучкість	Менш зручний для початківців

Кінець таблиці 2.1.

Технологія	**Призначення**	**Переваги**	**Недоліки**
XGBoost	Табличні прогнози	Висока точність, контроль над перенавчанням	Часове споживання на великому обсязі
Scikit-learn	Класичні ML-моделі	Простий API, велика документація	Менш ефективний для великих задач
Prophet	Сезонні часові ряди	Швидкість, простота використання	Обмежена складність моделей
NeuralProphet	Розширене прогнозування	Можливість додаткових ознак, нейронний підхід	Вища складність налаштування
Tkinter	Базовий GUI	Стандартна бібліотека, простота	Обмежений дизайн
PySide / PyQt	Сучасний GUI	Потужний інструментарій, кросплатформеність	Вища складність
tkintermapview	Інтерактивна мапа	Легке використання з Tkinter	Обмежена кастомізація
Cartopy	Географічні карти	Потужна візуалізація, інтеграція з matplotlib	Складніша конфігурація

2.2.Pandas

Pandas – це відкрита бібліотека Python, що надає засоби для обробки та аналізу структурованих даних. Вона широко використовується в задачах обробки табличної інформації, зокрема в науці про дані та машинному навчанні [25]. Основними структурами даних є Series та DataFrame.

•Series – це одномірний індексований масив, який може містити значення будь-якого типу: числові, рядкові, дати тощо.

•DataFrame – двовимірна таблиця з іменованими стовпцями, в якій кожен стовпець може мати власний тип даних. Така структура подібна до таблиці в SQL або аркуша Excel [13].

Pandas підтримує імпорт і експорт з різних форматів, зокрема: CSV, TSV, Excel (XLSX, XLS), JSON, HTML (таблиці з веб-сторінок), SQL-бази (через SQLAlchemy), Apache Parquet, Feather, HDF5, Msgpack, а також pickle. Є можливість читати великі файли частинами з використанням параметра chunksize, що дозволяє контролювати обсяг пам'яті під час обробки [25]. Для роботи з часовими рядами реалізовано функції ресемплінгу, зсуву (shift), обчислень у вікнах (rolling, expanding), а також обробки періодів і інтервалів. Передбачено зміну форматів та приведення часових індексів до різних часових зон.

У бібліотеці реалізовано інструменти для фільтрації даних, сортування (sort_values, sort_index), обробки пропущених значень (dropna, fillna), заміни значень (replace), об'єднання таблиць (merge, join, concat), зміни форми таблиці (pivot, melt) і групування (groupby) з можливістю подальших агрегацій (agg) та трансформацій (transform) [13].

Pandas використовується для виконання різноманітних завдань, пов'язаних із підготовкою, очищенням та аналізом даних. Однією з основних задач є обробка неструктурованих або частково структурованих джерел, де дані можуть містити дублікати, пропущені значення або потребувати об'єднання з іншими таблицями. У таких випадках бібліотека надає зручні засоби для очищення: видалення повторів, заповнення пропусків з використанням методів інтерполяції, обробку груп даних, зібраних із різних джерел.

Для економії оперативної пам'яті можна змінювати типи даних вручну, наприклад, перетворювати змінні у категоріальний формат. Ще однією важливою задачею є формування нових ознак на основі наявної інформації. Наприклад, з часових міток можна виділити день тижня, місяць або квартал. На основі числових даних – обчислити ковзне середнє, кумулятивну суму або інші

агреговані характеристики. Якщо таблиця містить географічні координати, їх можна використовувати для кластеризації або розрахунку відстаней.

Завдяки сумісності з іншими інструментами, Pandas легко інтегрується в машинне навчання. Дані з DataFrame можуть бути перетворені у формат масивів NumPy, які використовуються у scikit-learn, XGBoost або LightGBM [15, 17, 18]. Крім того, вона дозволяє підготовлювати послідовні батчі для навчання нейронних мереж у TensorFlow та PyTorch за допомогою генераторів або спеціалізованих класів на кшталт tf.data.Dataset [16].

Серед особливостей використання варто згадати індексування та методи об’єднання операцій у ланцюжки, що дозволяє скорочувати кількість проміжного коду. Підтримується багаторівнева система індексів, яка корисна у роботі з панельними або ієрархічно організованими даними. Разом з тим, бібліотека має певні обмеження. Усі дані в об'єкті DataFrame зберігаються в оперативній пам’яті, тому при роботі з дуже великими наборами виникає ризик перевантаження системи. Також більшість операцій виконується в одному потоці, і без використання зовнішніх бібліотек типу Dask або Modin швидкодія обмежена [14]. Ще одна особливість – при використанні ланцюжків методів складно відстежити помилки, тому часто доводиться розбивати обчислення на окремі етапи.

Попри ці обмеження, у задачах середньої складності Pandas залишається оптимальним вибором завдяки своїй гнучкості, зручному API та потужним можливостям для попередньої обробки даних. Це особливо актуально для побудови систем прогнозування погодних умов, де обсяги даних не перевищують кількох мільйонів записів, а обробка виконується локально [19].

У процесі розробки такої системи були розглянуті різні бібліотеки для роботи з табличними даними – зокрема, Dask, який підтримує паралельну обробку великих масивів у розподіленому середовищі [14]. Втім, у конкретному випадку перевагу було надано Pandas, оскільки він повністю покриває

функціональні потреби проєкту без необхідності додаткової інфраструктури для масштабування.

Бібліотека дає змогу легко поєднувати інформацію з різних джерел – CSV-файлів, JSON API або баз даних SQL – та формувати з неї єдину таблицю для подальшої обробки. Завдяки вбудованим засобам роботи з часовими мітками можна виконувати ресемплінг із довільною періодичністю (наприклад, за годинами, днями чи тижнями), вирівнювати часові ряди, а також заповнювати пропущені значення методами інтерполяції. Крім того, Pandas дозволяє здійснювати підготовку даних безпосередньо під час формування батчів для навчання LSTM-моделей, що усуває потребу в окремих скриптах і додаткових кроках попереднього збереження. Сукупність цих переваг робить Pandas ефективним і практичним інструментом для вирішення прикладних задач у сфері погодного прогнозування [19].

2.3. LSTM

Першою використаною моделлю є LSTM. LSTM (Long Short-Term Memory) – це різновид рекурентних нейронних мереж, розроблений для подолання проблеми «затухаючого градієнта», яка виникає під час обробки довгих послідовностей у звичайних RNN [26]. Завдяки своїй архітектурі LSTM здатна зберігати інформацію протягом тривалого часу, що особливо важливо у задачах, де кожне значення залежить від контексту попередніх, – наприклад, у прогнозуванні погоди [19].

Ключовою особливістю LSTM є наявність так званих воріт (gates), які керують потоком інформації всередині мережі. Вони вирішують, яку частину інформації зберегти, яку – оновити, а яку – відкинути [26]. Це забезпечує гнучкість у роботі з послідовностями, де залежності можуть охоплювати десятки або сотні кроків назад.

У контексті прогнозування погоди на основі історичних даних LSTM працює наступним чином: моделі надається послідовність попередніх значень,

наприклад, за останні 30 днів, а вона на основі вивчених закономірностей формує прогноз на наступний період (наприклад, 7 днів) [27]. Внутрішня пам’ять дозволяє їй враховувати не лише найближчі до поточного моменту значення, а й віддаленіші події, які могли вплинути на загальну динаміку.

Ця архітектура довела свою ефективність у різних задачах обробки часових рядів, зокрема там, де потрібен аналіз довгострокових залежностей – від прогнозу погоди до фінансових ринків [19].

LSTM-моделі мають низку переваг, що робить їх особливо привабливими для задач, пов’язаних із часовими рядами. Завдяки своїй архітектурі, вони ефективно працюють із даними, де присутні довготривалі залежності між значеннями. Наприклад, у прогнозуванні погоди нинішній стан атмосфери часто залежить від тенденцій, що спостерігались упродовж кількох попередніх днів. Саме здатність зберігати історичний контекст і враховувати його при генерації наступного прогнозу робить LSTM надзвичайно корисним інструментом для таких завдань.

Окрім цього, моделі LSTM демонструють високу гнучкість: їх можна адаптувати як для одномірних задач, так і для багатовимірного аналізу – коли потрібно враховувати не лише температуру, а й інші характеристики, як-от вологість, тиск, швидкість вітру, географічні координати тощо [15]. Мережа здатна ефективно працювати з такими багатофакторними наборами даних у межах одного навчального процесу. Ще однією перевагою є легка інтеграція з іншими інструментами машинного навчання: наприклад, для передобробки часто використовується масштабування за допомогою MinMaxScaler, а остаточний прогноз формується через щільні (Dense) шари, які підключаються до виходу LSTM [16].

Водночас використання LSTM пов’язане з певними обмеженнями. Моделі цього типу є досить ресурсоємними – як з погляду пам’яті, так і обчислювальної потужності [28]. Через послідовний характер обробки даних вони погано піддаються паралелізації, а тренування на великих наборах займає

значний час. Крім того, хоч LSTM і знижує ризик виникнення проблеми «затухаючого градієнта», вона все ж може проявлятися в глибоких мережах або при обробці надто довгих послідовностей [26].

У випадку побудови моделі прогнозування погодних умов LSTM виявилася найбільш доцільним вибором. Природа метеорологічних даних передбачає наявність як часових залежностей, так і багатовимірності вхідних параметрів. LSTM дає змогу зберігати інформацію про попередні значення у своїй пам’яті, ефективно працювати з кількома ознаками одночасно (наприклад, температура, вологість, широта, довгота, висота) й інтегруватися з допоміжними техніками, що забезпечують якісну підготовку та обробку даних. Такий підхід дозволяє моделі адекватно описувати складну динаміку атмосферних процесів і формувати точні прогнози навіть у довгостроковій перспективі [19].

2.4. TCN

Другою розглянутою моделлю є TCN. Temporal Convolutional Network — це тип нейронної мережі, який адаптовано для роботи з часовими рядами шляхом використання згорткових операцій. У традиційних задачах аналізу послідовностей найчастіше застосовувалися рекурентні архітектури, однак TCN пропонує альтернативний підхід, у якому обробка виконується на основі згорток, що проходять по часовій осі [29]. Основною особливістю є використання каузальних згорток, які не дозволяють інформації з майбутнього впливати на поточний прогноз. Завдяки цьому модель відповідає реальним умовам задачі прогнозування, де поточний стан залежить лише від минулих значень [15].

Ще однією важливою характеристикою TCN є здатність працювати з тривалими часовими залежностями. Це досягається шляхом застосування згорток з розширеним кроком (dilated convolutions), які дозволяють мережі охоплювати значну історичну глибину без збільшення кількості параметрів

[29]. Таким чином, модель здатна одночасно враховувати як найближчі до поточного моменту значення, так і віддаленіші події. У задачах прогнозування погоди це відкриває можливість враховувати складні погодні закономірності, що розгортаються в часі [19].

Обробка в TCN виконується паралельно, що дозволяє скоротити час навчання порівняно з послідовними моделями [30]. Це може бути особливо корисним у випадках, коли обробляється великий обсяг історичних даних. Архітектура моделі передбачає використання кількох згорткових шарів з однаковою кількістю вихідних каналів, а також залишкових зв'язків, які спрощують навчання глибоких мереж і стабілізують процес оптимізації [28].

У контексті прогнозування погодних умов TCN отримує на вхід послідовність метеорологічних показників за попередні дні, таких як температура, вологість, кількість опадів, швидкість вітру та інші. Обробка цих даних згортками дозволяє моделі визначити шаблони, які часто повторюються, та зробити прогноз на основі таких закономірностей. Здатність TCN ефективно працювати з багатоканальними послідовностями робить її придатною для аналізу комплексних погодних умов [19].

Попри відсутність внутрішньої пам'яті, як у LSTM, TCN за рахунок глибокої структури та розширених згорток здатна моделювати залежності на великій часовій глибині [29]. Це дозволяє формувати прогнози, які враховують не лише короткотермінові зміни, але й більш тривалі тенденції. Такий підхід виявляється особливо корисним у регіонах із вираженою сезонністю або циклічністю атмосферних явищ.

TCN також добре масштабується й може бути адаптована під різні варіанти задач: від прогнозу на наступний день до довгострокових передбачень на тиждень і більше [19]. Це забезпечується гнучкою архітектурою, яка дозволяє змінювати глибину моделі та параметри згорток залежно від особливостей вхідних даних. У результаті TCN може ефективно моделювати

метеорологічні ряди та доповнювати або навіть замінювати рекурентні підходи в сучасних системах прогнозування.

2.5. XGBoost

Однією з використаних моделей машинного навчання є XGBoost. XGBoost (Extreme Gradient Boosting) – це алгоритм машинного навчання, заснований на принципі градієнтного бустингу. Він комбінує велику кількість слабких моделей, зазвичай дерев рішень, у сукупну модель, яка дозволяє поступово зменшувати похибки попередніх прогнозів. Такий підхід застосовується в задачах, де потрібно враховувати складні залежності між вхідними ознаками [17].

Алгоритм працює поетапно: кожне нове дерево створюється з урахуванням помилок попередніх. На кожному етапі в модель додається нове дерево, яке враховує цю відмінність та оновлює прогноз [17]. Механізм XGBoost також включає регуляризацію – метод, що дозволяє контролювати складність моделі. Це допомагає зменшити ризик перенавчання та зробити модель більш стабільною при роботі з новими даними. Регуляризація враховує кількість листків у дереві та ваги, які надаються цим листкам [31].

Під час навчання модель також використовує спеціальні параметри, які впливають на швидкість та якість навчання. Наприклад, коефіцієнт навчання визначає, наскільки сильно нові дерева впливають на загальний прогноз. Важливою частиною навчального процесу є також вибір структури дерева, включаючи глибину та кількість вузлів [15].

XGBoost застосовується для задач класифікації та регресії, а також здатна працювати з даними, які містять пропущені значення. Завдяки цьому модель може бути використана у задачах прогнозування погоди, де історичні метеодані часто мають пропуски або існують складні залежності між змінними [19]. У таких задачах модель враховує ознаки, такі як температура, вологість, опади,

швидкість вітру тощо, та дозволяє будувати короткострокові прогнози на основі попередніх спостережень.

Під час побудови моделі необхідне налаштування гіперпараметрів, яке суттєво впливає на результат. При великих обсягах даних час навчання може збільшуватись, тому часто застосовуються методи оптимізації та паралельного обчислення [31]. Модель не є повністю інтерпретованою, оскільки об'єднання великої кількості дерев ускладнює розуміння логіки прийняття рішень, хоча окремі дерева мають просту структуру [28].

Таким чином, XGBoost є одним з підходів, який може бути використаний у задачах прогнозування погоди поряд з іншими алгоритмами. Його основними характеристиками є покрокове навчання з урахуванням помилок, підтримка регуляризації та здатність працювати з різними типами вхідних даних.

2.6. Prophet

Останньою використаною моделлю є Prophet. Prophet — це модель прогнозування часових рядів, розроблена дослідниками з Facebook, яка в даній роботі використовувалася для порівняння з іншими підходами, включаючи глибокі нейронні мережі [20]. Її створено спеціально для роботи з даними, що містять сезонні коливання, тренди та нерівномірно заповнені часові ряди. Prophet орієнтований на простоту застосування, високу інтерпретованість і здатність відображати складні структури часових рядів без потреби в глибокому налаштуванні параметрів [32].

У своїй основі Prophet представляє часовий ряд як суму кількох ключових компонентів: тренду, сезонності, ефекту від подій (наприклад, свят) та випадкового шуму. Тренд відображає довготривалі зміни значень з часом, сезонність відповідає за регулярні повторювані цикли, а складова подій враховує вплив зовнішніх факторів на поведінку ряду. В даній задачі ця остання складова не використовується, оскільки погодні процеси не залежать від свят

чи подібних подій [20]. Випадковий шум відображає ті коливання, які не можуть бути пояснені іншими компонентами моделі.

Особливу увагу в Prophet приділено моделюванню тренду. Модель підтримує два основні підходи до його формування — лінійний та логістичний. Лінійний підхід передбачає постійний або змінний темп росту значення в часі, тоді як логістичний використовується, коли очікується насичення або обмеження росту на певному рівні. Prophet дозволяє автоматично визначати так звані переломні точки — моменти, у яких швидкість зміни тренду змінюється [32]. Ці точки можуть також задаватися вручну, якщо є відповідна інформація.

Для опису сезонності модель використовує підхід на основі розкладу в ряд Фур'є. Це дозволяє гнучко моделювати циклічні коливання з будь-яким періодом, зокрема річну чи тижневу сезонність [19]. Кількість гармонік, що використовуються в розкладі, визначає точність апроксимації сезонних змін. Такий підхід дає змогу моделі адаптуватися до складної сезонної структури без надмірного ускладнення.

Випадкова компонента в моделі представляє собою шум, який охоплює ті зміни у часовому ряді, які не можуть бути пояснені трендом, сезонністю чи зовнішніми подіями. Це дозволяє Prophet залишатися стійким до неочікуваних або неструктурованих відхилень у даних [32].

Серед основних переваг Prophet варто відзначити його простоту у використанні — модель не потребує спеціальних знань у сфері машинного навчання, і навіть базове налаштування дозволяє отримати точні прогнози [20]. Крім того, її компоненти можна легко інтерпретувати й аналізувати окремо, що є особливо корисним у прикладних задачах. Prophet також добре справляється з пропущеними значеннями та нерівномірними часовими рядами, що робить його зручним для реальних даних, які часто мають прогалини [15].

Однак модель має і свої обмеження. Вона менш ефективна у випадках, коли залежності в даних є складними та нелінійними, порівняно з більш гнучкими методами, такими як XGBoost або глибокі нейронні мережі [17].

Prophet в основному працює з часовою ознакою як єдиною вхідною змінною і має обмежену підтримку додаткових регресорів, що може знизити точність у задачах з великою кількістю взаємозалежних ознак. Крім того, модель менш придатна для роботи з високочастотними даними, наприклад, із хвилинними або секундними інтервалами, оскільки її архітектура не оптимізована для таких сценаріїв [32].

У контексті прогнозування погоди, Prophet може бути корисним у випадках, коли дані демонструють чітко виражену сезонність, як-от температурні коливання протягом року [19]. У задачах з вираженими трендами та повторюваними шаблонами навіть базове налаштування Prophet може дати прийнятний результат. Проте через обмежену здатність моделювати складні взаємозв'язки між багатьма змінними, такими як температура, тиск, географічні координати тощо, Prophet доцільно використовувати переважно як базову або порівняльну модель у поєднанні з більш складними підходами.

2.7. Використані фреймворки

У цьому підрозділі наведено огляд фреймворків, що були використані для реалізації моделей прогнозування. Кожен із них забезпечує інструменти для ефективної побудови, навчання та тестування відповідного типу моделі. Далі буде розглянуто особливості обраних фреймворків, їхні переваги та недоліки в контексті задачі прогнозування погодних умов.

2.7.1. TensorFlow (Keras)

Для реалізації моделей глибокого навчання типу LSTM та TCN у даній роботі було обрано фреймворк TensorFlow з високорівневою надбудовою Keras [15]. TensorFlow є однією з поширених бібліотек для задач машинного навчання, яка підтримується компанією Google, активно оновлюється та має широку спільноту користувачів [33]. Вибір саме цього фреймворку серед

альтернатив, таких як PyTorch чи MXNet, зумовлений його придатністю до розв’язання поставленого завдання.

Однією з ключових характеристик TensorFlow є його гнучкість і масштабованість. Бібліотека дозволяє створювати як прості послідовні моделі, так і складні архітектури з власною логікою навчання. TensorFlow підтримує апаратне прискорення — навчання моделей може виконуватися з використанням графічних процесорів (GPU) або тензорних процесорів (TPU), що скорочує час тренування при роботі з великими обсягами даних [33].

TensorFlow дозволяє розгортати моделі на різних платформах — від серверів до мобільних пристроїв і вебсередовищ. Така універсальність полегшує інтеграцію моделей у практичні застосування. Крім того, наявність інструментів для моніторингу, як-от TensorBoard, а також велика кількість навчальних матеріалів і прикладів, спрощує процес розробки [15].

Серед недоліків TensorFlow варто зазначити складність при реалізації нетипових моделей на базовому API, що може ускладнити розробку для користувачів без досвіду. Також моделі глибокого навчання є ресурсоємними — ефективне тренування без GPU може бути надто повільним [28]. Крім того, процес підбору гіперпараметрів та оптимізації потребує додаткового часу й обчислювальних ресурсів.

Незважаючи на зазначені обмеження, підхід на базі TensorFlow/Keras є придатним для задачі прогнозування метеорологічних параметрів, забезпечуючи необхідний баланс між функціональністю, продуктивністю та можливістю інтеграції з іншими інструментами.

2.7.2. XGBoost

XGBoost є фреймворком для реалізації градієнтного бустингу дерев рішень, що широко застосовується в задачах регресії, класифікації та прогнозування часових рядів [17]. Його розроблено з урахуванням високої

продуктивності, ефективної обробки великих обсягів табличних даних та зручної інтеграції у практичні робочі процеси. У межах цього проєкту XGBoost було використано для реалізації однойменної моделі.

Однією з ключових технічних переваг XGBoost є підтримка паралельного навчання дерев, що суттєво зменшує час обробки при роботі з великими наборами погодних даних. Завдяки внутрішній оптимізації процесу побудови дерев та обчислень градієнтів фреймворк демонструє стабільну продуктивність навіть на слабших обчислювальних ресурсах [31].

Крім цього, XGBoost підтримує автоматичну обробку пропущених значень у даних. Під час навчання модель самостійно визначає оптимальні гілки дерева для прикладів, де певні ознаки відсутні. Це особливо важливо в умовах неповних погодних датасетів, де кількість і якість даних можуть змінюватися в залежності від регіону або періоду спостережень [17].

Для контролю складності моделей у XGBoost передбачено регуляризацію, зокрема підтримка як L1-, так і L2-підходів. Це дозволяє уникати перенавчання, особливо у випадках, коли модель використовується для узагальнення на нові часові інтервали [31]. Крім того, простий інтерфейс програмування та наявність широкої документації роблять інструмент зручним для інтеграції у більші розробницькі середовища або прототипи [34].

У порівнянні з альтернативними підходами, такими як реалізація бустингу дерев на базі загальних фреймворків для глибокого навчання, XGBoost пропонує більш оптимізовану реалізацію "з коробки". Ручне створення ансамблів у таких середовищах, як PyTorch або TensorFlow, зазвичай вимагає значно більше зусиль та не гарантує порівняної ефективності [15].

Ще однією популярною альтернативою є LightGBM, яка в окремих випадках забезпечує вищу швидкість навчання. Однак у задачах, де дані містять пропуски або мають обмежений обсяг, XGBoost демонструє кращу стабільність без необхідності попередньої обробки [34]. У свою чергу, CatBoost добре

працює з категоріальними ознаками, проте у даному проєкті такі ознаки не використовуються, що зменшує релевантність цієї бібліотеки.

Разом із тим, XGBoost має низку особливостей, які потребують врахування. Для досягнення найкращих результатів часто необхідно проводити ретельний підбір гіперпараметрів, що може вимагати значних обчислювальних ресурсів. Крім того, хоча окремі дерева в ансамблі можна аналізувати, загальна структура моделі лишається складною для інтерпретації [28]. Це обмежує можливості глибокого пояснення результатів, особливо в задачах, де важливо розуміти вплив кожної ознаки на підсумковий прогноз.

Ще одне обмеження полягає в складності моделювання довготривалих залежностей у часових рядах. XGBoost ефективно виявляє короткострокові шаблони та загальні тренди, але може бути менш придатним для аналізу довгих послідовностей, які часто мають значення у погодних процесах [19]. Проте, попри це, фреймворк залишається практичним інструментом для швидкого створення точної моделі.

2.7.3. Prophet

Prophet — це фреймворк для прогнозування часових рядів, розроблений дослідниками з компанії Facebook. У цьому проєкті він використовується як зручний інструмент побудови моделей, що дає змогу швидко сформувати прогноз із мінімальним налаштуванням [20]. Основною метою його застосування стало порівняння отриманих результатів з тими, що забезпечують більш складні підходи, зокрема моделі глибокого навчання або ансамблеві методи.

Фреймворк реалізує компонентну структуру часових рядів, де загальна поведінка пояснюється сумою кількох частин: тренд, сезонність, зовнішні події та залишкова складова. Такий підхід дозволяє окремо моделювати кожен з основних чинників, що впливають на зміну значень у часі [32]. У межах цього

проєкту увага зосереджувалася на тренді та сезонності, оскільки погодні змінні не залежать від подій на кшталт свят, а випадковий шум є неминучим супутником усіх реальних даних.

Формування тренду в Prophet ґрунтується на двох варіантах: лінійному та логістичному. Лінійний варіант передбачає постійний або змінний темп росту, тоді як логістичний застосовується у випадках насичення або граничних значень. Завдяки механізму автоматичного виявлення точок зламу, фреймворк здатен враховувати зміни в динаміці ряду без додаткового втручання з боку користувача [32]. При потребі такі точки можна задати вручну.

Сезонна складова моделюється через розклад у ряд Фур'є, що дає змогу відобразити періодичні коливання різної частоти — наприклад, річні або тижневі цикли [19]. Гнучкість цього підходу дозволяє адаптувати модель до різноманітних форм сезонності без потреби у жорстко заданих шаблонах. Кількість гармонічних компонентів визначає точність відображення сезонних змін і може регулюватися в залежності від складності вхідних даних.

Фреймворк також враховує залишкову компоненту, яка охоплює всі відхилення, що не пояснюються іншими елементами моделі. Це дозволяє зберігати стійкість до нерегулярних чи непередбачуваних змін у даних, що є характерною особливістю багатьох реальних часових рядів [32]. При цьому Prophet автоматично обробляє пропущені значення та не вимагає рівномірного інтервалу між спостереженнями, що підвищує його зручність у прикладному використанні.

Попри зручність та інтерпретованість, фреймворк має певні обмеження. Його підхід не передбачає складного моделювання взаємозалежностей між численними змінними, тому в задачах із великою кількістю характеристик або високою нелінійністю даних він може поступатися іншим підходам [17]. Prophet здебільшого орієнтований на обробку одновимірних часових рядів і має обмежену підтримку сторонніх регресорів, що також варто враховувати при застосуванні до багатовимірних задач.

У рамках прогнозування метеорологічних показників Prophet може бути корисним, коли дані мають виражену сезонну структуру або стабільні тренди [19]. Його застосування дозволяє швидко оцінити загальну динаміку та отримати базовий прогноз, що, в свою чергу, слугує орієнтиром при оцінці ефективності більш гнучких або складних моделей.

2.7.4. Tkinter та TkinterMapView

Для створення графічного інтерфейсу користувача в цьому проєкті було використано стандартну бібліотеку Tkinter, що входить до складу Python [21]. Її застосування дало змогу швидко реалізувати функціональні елементи керування, необхідні для взаємодії користувача з системою. Зокрема, було реалізовано інтерфейс для вибору необхідних параметрів погоди, запуску моделей прогнозування та виводу результатів.

Доповненням до основної бібліотеки став модуль TkinterMapView, який дав змогу інтегрувати в додаток інтерактивні мапи на основі OpenStreetMap [22]. Завдяки цьому стало можливим відображення географічної інформації без необхідності звертатися до сторонніх рішень. Мапи забезпечують зручну візуалізацію координат, масштабування та навігацію, що є важливою частиною інтерфейсу при роботі з геоприв’язаними даними, такими як погодні показники в окремих містах.

Використання Tkinter виявилося доцільним у межах цього проєкту, оскільки його функціоналу було достатньо для реалізації поставлених завдань. Бібліотека повністю інтегрується з Python, що дозволило безпосередньо поєднати логіку машинного навчання з графічною частиною програми [21]. Хоча засоби кастомізації вигляду інтерфейсу є дещо обмеженими порівняно з сучасними альтернативами, такими як PyQt або web-технології, реалізованого дизайну виявилося цілком достатньо для демонстрації основних можливостей застосунку.

Загалом, обрані інструменти виявилися придатними для побудови інтерфейсу середньої складності, що поєднує обробку погодних даних, географічну візуалізацію та доступ до функцій моделювання у зручній формі.

У наведеній нижче таблиці наданий короткий підсумок щодо використаних фреймворків.

Таблиця 2.2. – Переваги та недоліки використаних інструментів

Інструмент	**Призначення**	**Основні переваги**	**Основні недоліки**
TensorFlow (Keras)	Глибоке навчання (LSTM, TCN)	Гнучкість і масштабованість Підтримка GPU/TPU Розгортання на різних платформах Інструменти моніторингу (TensorBoard)	Складність реалізації нетипових моделей Високі обчислювальні вимоги Трудомісткий підбір гіперпараметрів
XGBoost	Градієнтний бустинг дерев рішень для регресії, класифікації та прогнозування	Висока продуктивність Паралельне навчання дерев Обробка пропущених значеньРегуляризація (L1, L2)- Простота інтеграції Хороша стабільність на обмежених/неповних даних	Неінтерпретованість ансамблю дерев Потреба в налаштуванні гіперпараметрів Складність моделювання довготривалих залежностей
Prophet	Прогнозування часових рядів (лінійні/сезонні тренди)	Простота використання Автоматичне врахування тренду/сезонності Обробка пропусків і нерівномірного інтервалу Інтерпретованість компонент (тренд, сезонність, шум)	Орієнтований на одновимірні ряди Обмежена робота з багатовимірними змінними Менш ефективний у задачах з високою нелінійністю або великою кількістю змінних

Кінець таблиці 2.2.

Інструмент	Інструмент	Інструмент	Інструмент
Tkinter	Графічний інтерфейс користувача	Стандартна бібліотека Python Простота реалізації елементів керування Не потребує сторонніх залежностей	Обмежені можливості дизайну Не підтримує сучасні GUI-компоненти
TkinterMapView	Відображення інтерактивних мап (OpenStreetMap) у GUI	Інтеграція мап без сторонніх сервісів Підтримка масштабування, навігації Зручна візуалізація геоданих	Залежність від зовнішніх джерел карт (наприклад, інтернет-з'єднання) Обмежена кастомізація мап

2.8.Датасет

У цій роботі для навчання та тестування моделей прогнозування використано історичний погодний датасет, отриманий за допомогою API Visual Crossing [35]. Visual Crossing є надійним комерційним сервісом, який надає погодні дані з високою точністю та простим у використанні API, що дозволяє автоматично завантажувати великі обсяги даних у зручному форматі, наприклад, CSV або JSON [35].

Дані охоплюють період з 1 січня 2014 року по поточну дату, що забезпечує достатню глибину історії для виявлення сезонних закономірностей, довготривалих кліматичних трендів і короткострокових аномалій [19]. Це дозволяє навчити моделі як на глобальних, так і на локальних особливостях погодних змін.

Вибірка включає інформацію для восьми великих міст України, а саме: Київ, Харків, Одеса, Дніпро, Львів, Запоріжжя, Кривий Ріг та Донецьк. Така географічна різноманітність дозволяє врахувати відмінності в кліматичних

умовах між різними регіонами: від прибережних до континентальних, від рівнинних до більш підвищених територій [24].

Датасет має добову частоту та містить такі основні параметри:

- Дата – часова мітка спостереження;
- Температура повітря (°C) – основна цільова змінна для прогнозування;
- Вологість (%) – фактор, що впливає на теплове відчуття та динаміку атмосферних явищ;
- Кількість опадів (мм) – важлива змінна для виявлення фронтальних зон і циклонів;
- Швидкість вітру (м/с) – впливає на терморегуляцію та формування погодних систем;
- географічні координати – широта, довгота та висота над рівнем моря, які враховуються як фічі при побудові моделей.

Зібрані дані мають добову частоту, що узгоджується з поставленим завданням прогнозу температури на кілька днів уперед. Перед використанням усі дані проходили базову обробку: перевірку на пропущені значення, агрегацію за містами та підготовку до подачі в моделі прогнозування [13]. Для цього використовувалися інструменти бібліотеки Pandas, які забезпечують ефективну обробку табличних даних, включаючи заповнення пропусків і приведення даних до потрібного формату [25].

Таким чином, побудований датасет поєднує як метеорологічні, так і географічні характеристики, що дозволяє застосовувати моделі різного типу – від статистичних до глибоких нейронних мереж – для задач прогнозування погодних умов на декілька днів уперед.

ВИСНОВКИ ДО РОЗДІЛУ

У цьому розділі було розглянуто набір інструментів і моделей, що використовувалися для побудови системи прогнозування погоди. Першочергово увагу приділено бібліотеці pandas, яка є основним засобом для обробки структурованих даних у форматі таблиць. Саме за її допомогою здійснювалась попередня обробка погодних даних, агрегація та підготовка до моделювання.

Серед моделей, що застосовувались у дослідженні, кожна має власний підхід до роботи з часовими рядами. LSTM – це тип нейронної мережі, яка дозволяє аналізувати послідовності даних, враховуючи залежності між подіями у минулому. Це дає змогу будувати прогнози на основі температурних змін за попередні дні.

TCN аналізує послідовності, застосовуючи спеціальні згорткові операції. Вона ефективно виявляє закономірності у часових рядах, навіть якщо залежності розтягнуті в часі, і при цьому забезпечує хорошу швидкість обчислень.

XGBoost використовує принцип поетапного навчання набору простих моделей для досягнення високої точності. Цей підхід дозволяє ефективно працювати з табличними погодними даними, де кожна ознака (наприклад, висота чи координати міста) впливає на підсумкове рішення.

Prophet – це інструмент, створений спеціально для задач прогнозування з яскраво вираженою сезонністю. Його простота у використанні робить його зручним для швидкого отримання базових результатів без складного налаштування.

Дані для моделювання були отримані з сервісу Visual Crossing за допомогою API. Датасет охоплює період із 2014 року до поточної дати й містить добову інформацію про температуру, вологість, опади, а також географічні характеристики – широту, довготу та висоту над рівнем моря. Дані зібрані для

восьми найбільших міст України, що дозволяє врахувати кліматичну різноманітність регіонів.

Для реалізації кожної з моделей було використано відповідні фреймворки: TensorFlow/Keras – для побудови нейронних мереж LSTM та TCN, XGBoost – для однойменного алгоритму бустингу, Prophet – як окремий фреймворк для сезонного прогнозування.

Крім того, для створення графічного інтерфейсу було використано Tkinter у поєднанні з TkinterMapView. Завдяки цьому реалізовано можливість вибору міста, запуску моделі, виводу прогнозованих значень та інтерактивного перегляду результатів на карті.

Використання цих інструментів дозволило охопити як класичні, так і сучасні підходи до обробки часових рядів та реалізувати повноцінний застосунок для прогнозування погоди з графічним інтерфейсом.

РОЗДІЛ 3
ДЕТАЛІ РОЗРОБКИ СИСТЕМИ

У цьому розділі буде детально розглянуто процес розробки системи.

Для розробки системи було обрано об’єктно-орієнтований підхід, що забезпечує модульність, масштабованість і зручність підтримки коду. Програма структурована на окремі модулі, кожен із яких інкапсулює специфічний функціонал. Основні модулі включають:

- Модуль обробки даних, відповідальний за завантаження, попередню обробку, очищення та підготовку метеорологічних даних для прогнозування.
- Модулі моделей, що реалізують навчання та застосування чотирьох прогнозних моделей (LSTM, TCN, XGBoost, Prophet), кожна з яких інкапсулює унікальну логіку прогнозування.
- Модуль інтерфейсу, який забезпечує реалізацію GUI для взаємодії з даними, моделями та візуалізацією прогнозів.

Такий поділ дозволяє чітко розмежувати функції, спрощує тестування та подальший розвиток системи.

3.1. Створення датасету та його оновлення

Як вже було згадано раніше, для створення датасету використовувався сервіс Visual Crossing, що надає доступ для історичних записів про різні погодні характеристики, такі як температура, вологість, швидкість вітру, кількість опадів та інші.

Процес збору необхідного для навчання набору даних було автоматизовано за допомогою скрипту з використанням Timeline Weather API, що надається самим сервісом. Для базового датасету було обрано щоденні погодні показники у восьми великих містах України, починаючи з 2014 року.

Крім метеоданих також включено інформацію про географічне розташування міста, а саме його довготу, широту, та висоту над рівнем моря, що дозволяє покращити точність моделей.

Для оновлення датасету у системі створено окремий модуль, що перевіряє наявність актуальних даних у відповідному файлі та за потреби робить запит для оновлення до поточного дня. Також передбачено додавання нових даних за умови дотримання формату файлів.

На рисунку 3.1 наведено приклад одного запису у форматі JSON.

Рисунок 3.1 Приклад зберігання даних

3.2. Створення моделей

У системі використано чотири моделі: дві моделі глибокого навчання — LSTM і TCN, а також ансамблевий бустинг XGBoost і модель прогнозування часових рядів Prophet.

LSTM-модель розроблена з використанням TensorFlow Keras, де основна архітектура включає LSTM-шар (64 нейрони) для обробки часових послідовностей тривалістю 30 днів і щільний шар для статичних ознак (широта, довгота, висота), об’єднаних для прогнозування чотирьох параметрів (температура, вологість, опади, швидкість вітру) на 7 днів уперед. Дані нормалізуються за допомогою MinMaxScaler, а модель навчається з функцією

втрат MSE та оптимізатором Adam протягом 7 епох. Після навчання модель зберігається в файл для подальшого використання в інтерфейсі застосунку.

TCN-модель, реалізована з використанням бібліотеки tcn на базі TensorFlow Keras, використовує згорткову архітектуру з 64 фільтрами та дилатаціями [1, 2, 4, 8] для ефективного моделювання довгострокових залежностей у часових послідовностях тривалістю 30 днів, замість рекуррентного підходу LSTM. Статичні ознаки (широта, довгота, висота) обробляються через щільний шар із 16 нейронами та активацією ReLU, що забезпечує компактне представлення географічних даних. Вихідний шар формує прогноз для тих же чотирьох метеорологічних параметрів на 7 днів уперед, зберігаючи сумісність із задачею LSTM-моделі, але з акцентом на паралельну обробку даних та потенційно швидше навчання завдяки згортковій структурі.

Модель XGBoost, побудована на основі градієнтного бустингу з використанням бібліотеки xgboost, застосовує ансамбль із 500 дерев рішень глибиною до 5, з швидкістю навчання 0.1, субвибіркою 80% даних і 80% ознак на дерево, для прогнозування виключно температури на 7 днів уперед. На відміну від рекуррентної LSTM чи згорткової TCN, які обробляють 30-денні послідовності для всіх метеорологічних параметрів, XGBoost використовує 14-денні лаги температури, доповнені середнім за 7 днів і днем року, а також статичними ознаками (широта, довгота, висота). Обмеження прогнозуванням лише температури зумовлене високою помилкою для інших параметрів (вологість, опади, швидкість вітру), що спрощує модель і уникає ускладнення системи, оскільки LSTM, TCN і Prophet уже забезпечують повний набір прогнозів. Ітеративне прогнозування, де кожне нове значення температури додається до вхідних даних, відрізняє XGBoost від одноетапного підходу нейронних мереж.

Модель Prophet, реалізована через бібліотеку prophet, використовує статистичний підхід із річною сезонністю для прогнозування всіх чотирьох

метеорологічних параметрів (температура, вологість, опади, швидкість вітру) на 7 днів уперед. На відміну від рекуррентної LSTM, згорткової TCN чи ансамблевої XGBoost, Prophet розкладає часові ряди на тренд і сезонні компоненти, навчаючи окремі моделі для кожного параметра та міста, що полегшує врахування нелінійних змін і періодичних коливань. Географічні ознаки (широта, довгота, висота) додаються як регресори, забезпечуючи контекст без потреби в лагах чи послідовностях, як у інших моделях. Prophet доповнює систему, усуваючи прогалину XGBoost, обмеженого температурою, і пропонує гнучке моделювання сезонних ефектів.

3.3. Реалізація інтерфейсу

Інтерфейс користувача системи розроблено з використанням бібліотек tkinter і tkintermapview, що забезпечують реалізацію інтуїтивно зрозумілого графічного середовища для взаємодії з даними, моделями та результатами прогнозування. Основним компонентом є клас ForecastApp, який включає весь функціонал інтерфейсу, координуючи роботу модулів обробки даних і моделей у межах об’єктно-орієнтованого підходу. Інтерфейс дозволяє користувачу завантажувати та оновлювати метеорологічні дані, за потреби заново навчати моделі LSTM і TCN, генерувати прогнози за допомогою чотирьох моделей (LSTM, TCN, XGBoost, Prophet), візуалізувати результати на графіках і відображати їх на інтерактивній мапі.

Основне вікно програми (рис. 3.2) поділено на два блоки: панель керування та область відображення мапи. Панель керування містить набір кнопок для виконання ключових операцій, таких як оновлення та завантаження даних через виклик модуля data_module, додавання нових даних у форматі Parquet із перевіркою відповідності структури, навчання моделей LSTM і TCN, створення прогнозів для всіх чотирьох моделей із збереженням результатів у CSV-файли, завантаження збережених прогнозів для повторного використання

та побудова графіків для порівняння прогнозів за температурою, вологістю, опадами та швидкістю вітру в кожному місті.

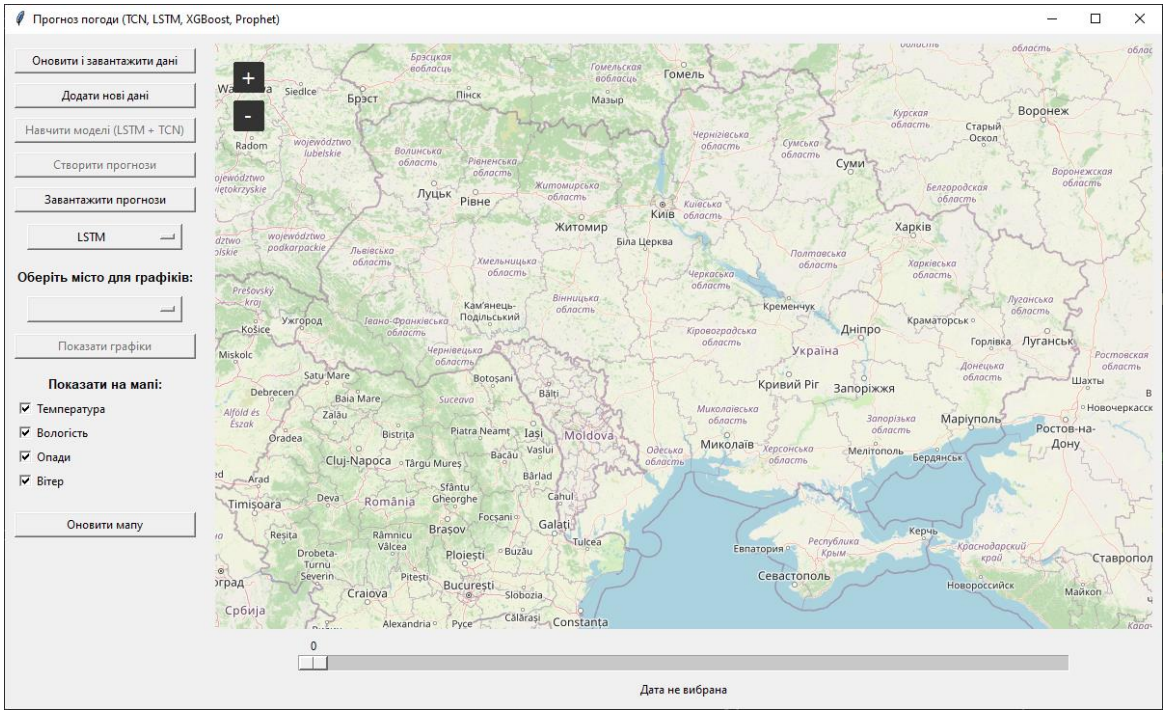


Рисунок 3.2 Інтерфейс системи

Вибір моделі для відображення прогнозів здійснюється через випадаючий список, а чекбокси дозволяють користувачу налаштувати відображення метеорологічних параметрів на мапі. Окрема кнопка «Оновити мапу» забезпечує усунення візуальних артефактів шляхом повторного створення мапи з відновленням попереднього стану, що підвищує стабільність візуалізації.

Область мапи, реалізована через TkinterMapView, відображає географічні маркери для міст України з прогнозними значеннями, які оновлюються за допомогою повзунка для вибору дати. Повзунок синхронізується з датами прогнозів, дозволяючи динамічно переглядати передбачення для кожного дня. Для моделі XGBoost, яка прогнозує лише температуру, інші параметри (вологість, опади, швидкість вітру) не відображаються, що враховується при побудові мапи та графіків, забезпечуючи коректне представлення даних.

Такий дизайн інтерфейсу забезпечує гнучкість і зручність, дозволяючи користувачу легко керувати даними, моделями та їх результатами. Модульна

структура системи, реалізована через ООП, спрощує інтеграцію нових функцій і подальший розвиток програми.

ВИСНОВКИ ДО РОЗДІЛУ

У третьому розділі проєкту були розглянуті особливості реалізації системи прогнозування погоди.

Система прогнозування метеопараметрів побудована на об’єктно-орієнтованому підході, що забезпечує модульність і легкість розширення. Поділ на модулі обробки даних, моделей і графічного інтерфейсу спрощує розробку та тестування.

Модуль обробки даних автоматизує збір і оновлення метеоінформації з Visual Crossing, включаючи географічні ознаки для підвищення точності. Чотири моделі — LSTM, TCN, XGBoost і Prophet — пропонують різноманітні методи прогнозування: LSTM і TCN обробляють усі параметри, XGBoost обмежується температурою, а Prophet гнучко моделює сезонність.

Інтерфейс створено на основі tkinter і tkintermapview. Він забезпечує зручне керування даними, моделями та візуалізацію прогнозів на мапі й графіках. Модульна структура та ООП-підхід створюють надійну основу для подальшого вдосконалення системи.

РОЗДІЛ 4
ОГЛЯД СИСТЕМИ АНАЛІЗУ ТА ПРОГНОЗУВАННЯ ПОГОДНИХ УМОВ

У четвертому розділі описано функціонал системи, розробленої в межах проєкту.

При запуску програми користувач бачить інтерфейс із панеллю керування та вікном мапи (рис. 4.1). Панель керування містить набір кнопок, кожна з яких відповідає за певну функцію (рис. 4.2).

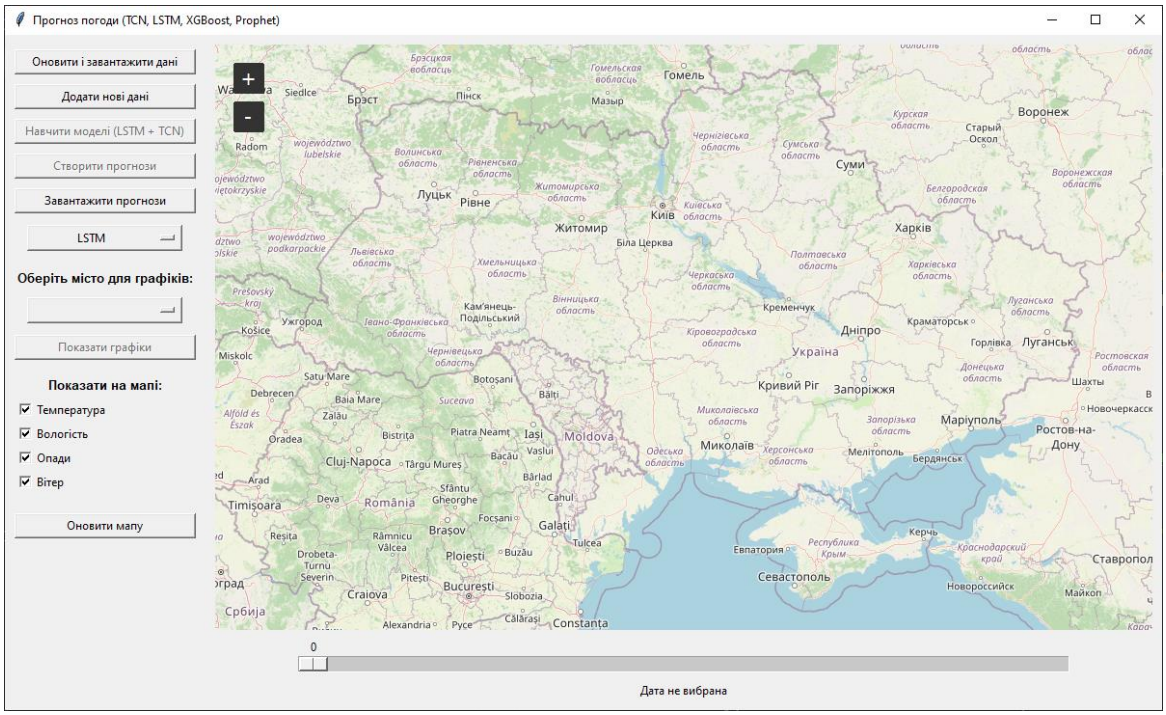


Рисунок 4.1 Вигляд інтерфейсу при запуску програми

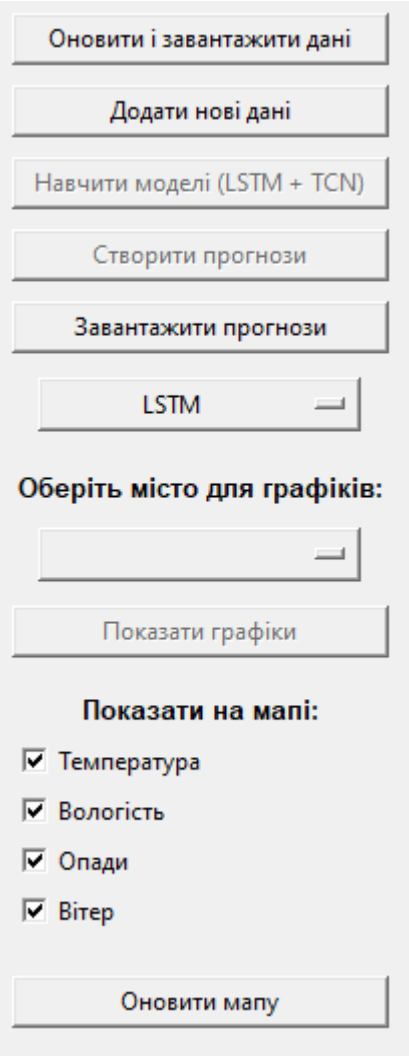


Рисунок 4.2 Кнопки меню

Кнопка «Оновити і завантажити дані» активує методи модуля data_module, які оновлюють датасет до поточної дати через Timeline Weather API і завантажують його для подальшої роботи. Кнопка «Додати нові дані» дає змогу розширити датафрейм системи інформацією з іншого Parquet-файлу, якщо його структура відповідає наявному формату.

Після успішного завантаження даних стають доступними кнопки «Навчити моделі» та «Створити прогнози». Система підтримує збереження навчених моделей, що дозволяє використовувати їх повторно без додаткового навчання. Процес прогнозування займає менше хвилини, а якщо прогнози на поточну дату вже створено, їх можна швидко завантажити через відповідну кнопку.

Створений або завантажений прогноз автоматично відображається на мапі (рис. 4.3). За допомогою випадаючого меню можна обрати модель (LSTM, TCN, XGBoost або Prophet), результати якої будуть показані. Чекбокси під меню дозволяють вибрати метеорологічні параметри для відображення, а повзунок під мапою — дату прогнозу.

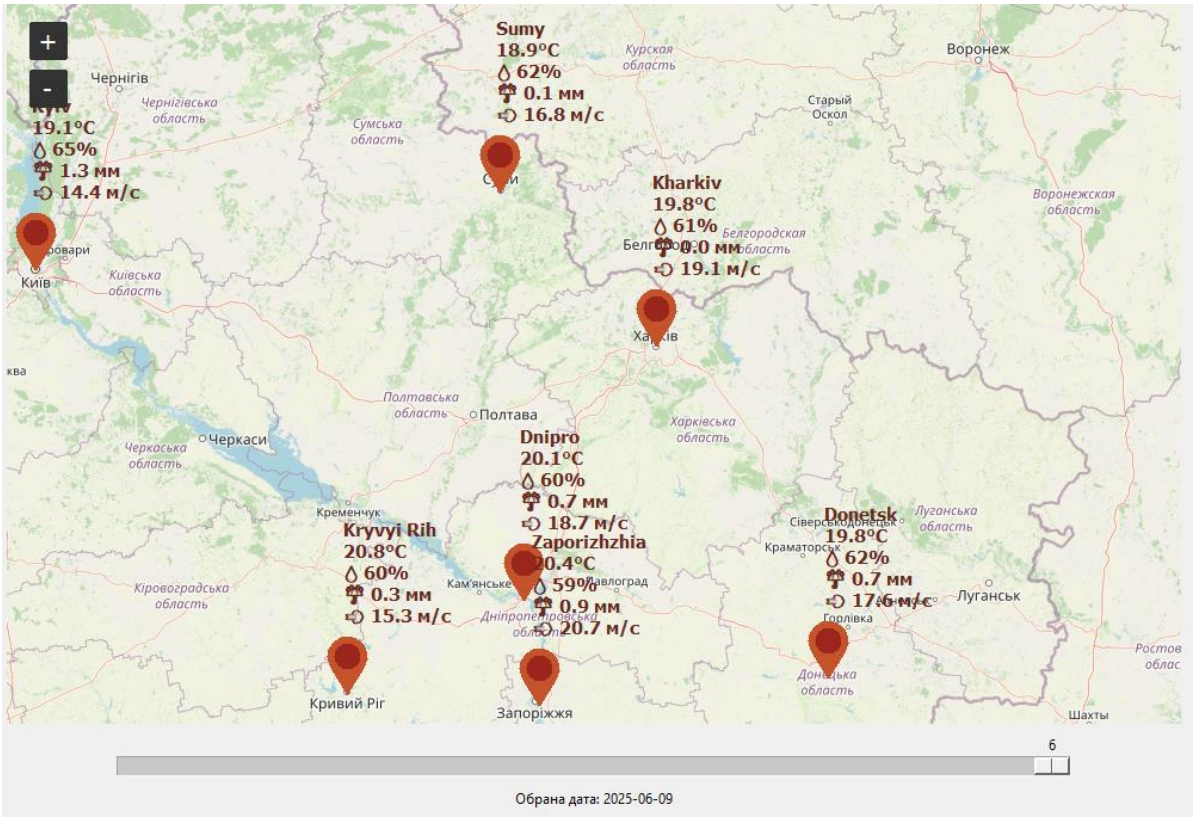


Рисунок 4.3 Мапа з відображеними прогнозами

Кнопка «Показати графіки» відкриває послідовність графіків, що порівнюють прогнози всіх моделей для обраного міста за різними параметрами (рис. 4.4).

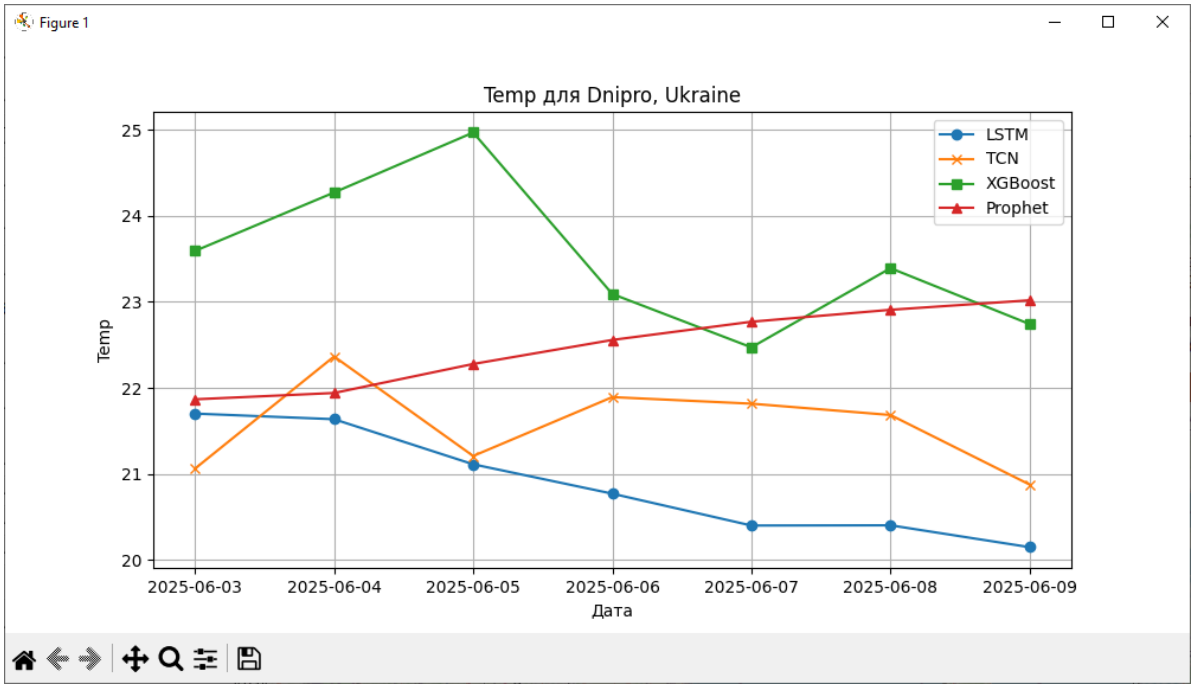


Рисунок 4.4. Приклад відображення прогнозу у вигляді графіку

Останньою є кнопка «Оновити мапу». Вона створює віджет мапи заново зберігаючи попередній стан. Це зроблено для уникнення візуальних багів які можуть виникнути, наприклад, при частому оновленні мапи за допомогою повзунка.

ВИСНОВКИ ДО РОЗДІЛУ

Розроблена система прогнозування метеорологічних параметрів вирізняється зручним інтерфейсом, який спрощує взаємодію користувача з даними та прогнозами. Панель керування забезпечує швидке оновлення погодної інформації через API, додавання нових даних у форматі Parquet і гнучке керування процесом створення чи завантаження прогнозів. Можливість повторного використання збережених моделей і швидке генерування прогнозів підвищують ефективність роботи.

Інтерактивна мапа дозволяє візуалізувати прогнози для обраної моделі, метеопараметрів і дати, що обирається повзунком, забезпечуючи наочний аналіз даних. Порівняння результатів усіх моделей у вигляді графіків для конкретного міста сприяє оцінці їхньої точності, зокрема з урахуванням обмеження XGBoost до прогнозування температури.

Функціонал оновлення мапи усуває можливі візуальні баги, гарантуючи стабільне відображення. Система поєднує простоту використання з широкими можливостями аналізу, створюючи ефективний інструмент для прогнозування погоди з перспективами подальшого розвитку.

ВИСНОВКИ

Дипломний проєкт присвячено розробці системи для аналізу та прогнозування погодних умов. У процесі роботи досліджено наявні рішення, проаналізовано методи й підходи до передбачення такої складної системи, як погода.

У першому розділі розглянуто предметну область, зокрема популярні сервіси прогнозування погоди. Вивчено їхній функціонал і методи, що застосовуються в сучасних рішеннях. На основі цього аналізу визначено перспективи створення власної системи та напрями її вдосконалення.

Другий розділ присвячено технічним засобам, придатним для реалізації завдання. Описано бібліотеки для обробки даних, фреймворки глибокого навчання, моделі машинного навчання та інструменти для створення графічного інтерфейсу. Обґрунтовано вибір конкретних технологій і методів їх застосування.

У третьому розділі детально викладено процес програмної реалізації системи. Описано створення й особливості датасету, налаштування моделей прогнозування та взаємодію інтерфейсу з модулями, що забезпечують серверну частину програми.

Четвертий розділ висвітлює функціонал розробленого застосунку. Система з графічним інтерфейсом дозволяє користувачам зручно працювати з даними, генерувати прогнози, переглядати їх на інтерактивній мапі та аналізувати передбачені параметри через графіки.

Розроблена система спрощує прогнозування погоди, надаючи зручний інструмент для отримання точних передбачень навіть у регіонах із обмеженою кількістю метеостанцій. Вона поєднує функціональність, доступність і потенціал для подальшого розвитку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. AccuWeather [Електронний ресурс] – Режим доступу до ресурсу: https://www.linkedin.com/company/accuweather/
2. AccuWeather on TV [Електронний ресурс] – Режим доступу до ресурсу: https://corporate.accuweather.com/resources/accuweather-on-tv/
3. Severe Weather – USA [Електронний ресурс] – Режим доступу до ресурсу: https://www.accuweather.com/en/us/severe-weather
4. AccuWeather Privacy Policy [Електронний ресурс] – Режим доступу до ресурсу: https://www.accuweather.com/en/privacy
5. Met Office – About Us [Електронний ресурс] – Режим доступу до ресурсу: https://www.metoffice.gov.uk/about-us/
6. UK Weather Warnings and Advice [Електронний ресурс] – Режим доступу до ресурсу: https://www.metoffice.gov.uk/weather/warnings-and-advice/uk-warnings
7. WAFC – World Area Forecast Centre [Електронний ресурс] – Режим доступу до ресурсу: https://www.metoffice.gov.uk/services/transport/aviation/regulated/international-aviation/wafc/index#:~:text=What%20is%20a%20WAFC%3F
8. Shipping Forecast – Met Office [Електронний ресурс] – Режим доступу до ресурсу: https://weather.metoffice.gov.uk/specialist-forecasts/coast-and-sea/shipping-forecast
9. Weather Forecasts – Our World in Data [Електронний ресурс] – Режим доступу до ресурсу: https://ourworldindata.org/weather-forecasts
10. Met Office Weather Forecast App [Електронний ресурс] – Режим доступу до ресурсу: https://apps.apple.com/us/app/met-office-weather-forecast/id1068146838
11. Privacy Policy – Met Office [Електронний ресурс] – Режим доступу до ресурсу: https://www.metoffice.gov.uk/about-us/legal/privacy

12. Політика конфіденційності – Sinoptik.ua [Електронний ресурс] – Режим доступу до ресурсу: https://sinoptik.ua/privacypolicy
13. McKinney W. Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython. – O'Reilly Media, 2017.
14. Rocklin M. Dask: Parallel Computation with Blocked Algorithms and Task Scheduling // Proc. of the 14th Python in Science Conference, 2015.
15. Géron A. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. – O'Reilly Media, 2019.
16. Paszke A., Gross S., Massa F., et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library // Advances in Neural Information Processing Systems (NeurIPS), 2019.
17. Chen T., Guestrin C. XGBoost: A Scalable Tree Boosting System // Proc. of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016.
18. Pedregosa F., Varoquaux G., Gramfort A., et al. Scikit-learn: Machine Learning in Python // Journal of Machine Learning Research. 2011. Vol. 12. P. 2825–2830.
19. Taylor S. J., Letham B. Forecasting at Scale // The American Statistician. 2018. Vol. 72(1). P. 37–45.
20. Triebe O., Laptev N., Rajagopal R. NeuralProphet: Explainable Forecasting at Scale // arXiv preprint arXiv:2005.06676, 2021.
21. Grayson J. Python and Tkinter Programming. – Manning Publications, 2012.
22. tkintermapview [Електронний ресурс] – Режим доступу: https://github.com/TomSchimansky/TkinterMapView.
23. Summerfield M. Rapid GUI Programming with Python and Qt. – Prentice Hall, 2015.
24. Met Office. Cartopy: A library providing cartographic tools for Python [Електронний ресурс] – Режим доступу: https://scitools.org.uk/cartopy/.

25. Pandas Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://pandas.pydata.org/docs/.
26. Long Short-Term Memory (LSTM) [Електронний ресурс] – Режим доступу до ресурсу: https://www.mitpressjournals.org/doi/abs/10.1162/neco.1997.9.8.1735.
27. Time-series Forecasting with Deep Learning [Електронний ресурс] – Режим доступу до ресурсу: https://royalsocietypublishing.org/doi/10.1098/rsta.2020.0209.
28. Deep Learning [Електронний ресурс] – Режим доступу до ресурсу: https://mitpress.mit.edu/9780262035613/deep-learning/.
29. Temporal Convolutional Networks for Sequence Modeling [Електронний ресурс] – Режим доступу до ресурсу: https://arxiv.org/abs/1803.01271.
30. Temporal Convolutional Networks for Action Segmentation [Електронний ресурс] – Режим доступу до ресурсу: https://openaccess.thecvf.com/content_cvpr_2017/html/Lea_Temporal_Convolutional_Networks_CVPR_2017_paper.html.
31. Greedy Function Approximation: A Gradient Boosting Machine [Електронний ресурс] – Режим доступу до ресурсу: https://projecteuclid.org/journals/annals-of-statistics/volume-29/issue-5/Greedy-function-approximation-A-gradient-boosting-machine/10.1214/aos/1013203451.full.
32. Prophet Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://facebook.github.io/prophet/.
33. TensorFlow Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.tensorflow.org/.
34. XGBoost Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://xgboost.readthedocs.io/.
35. Visual Crossing Weather API Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.visualcrossing.com/weather-api.

ДОДАТОК А

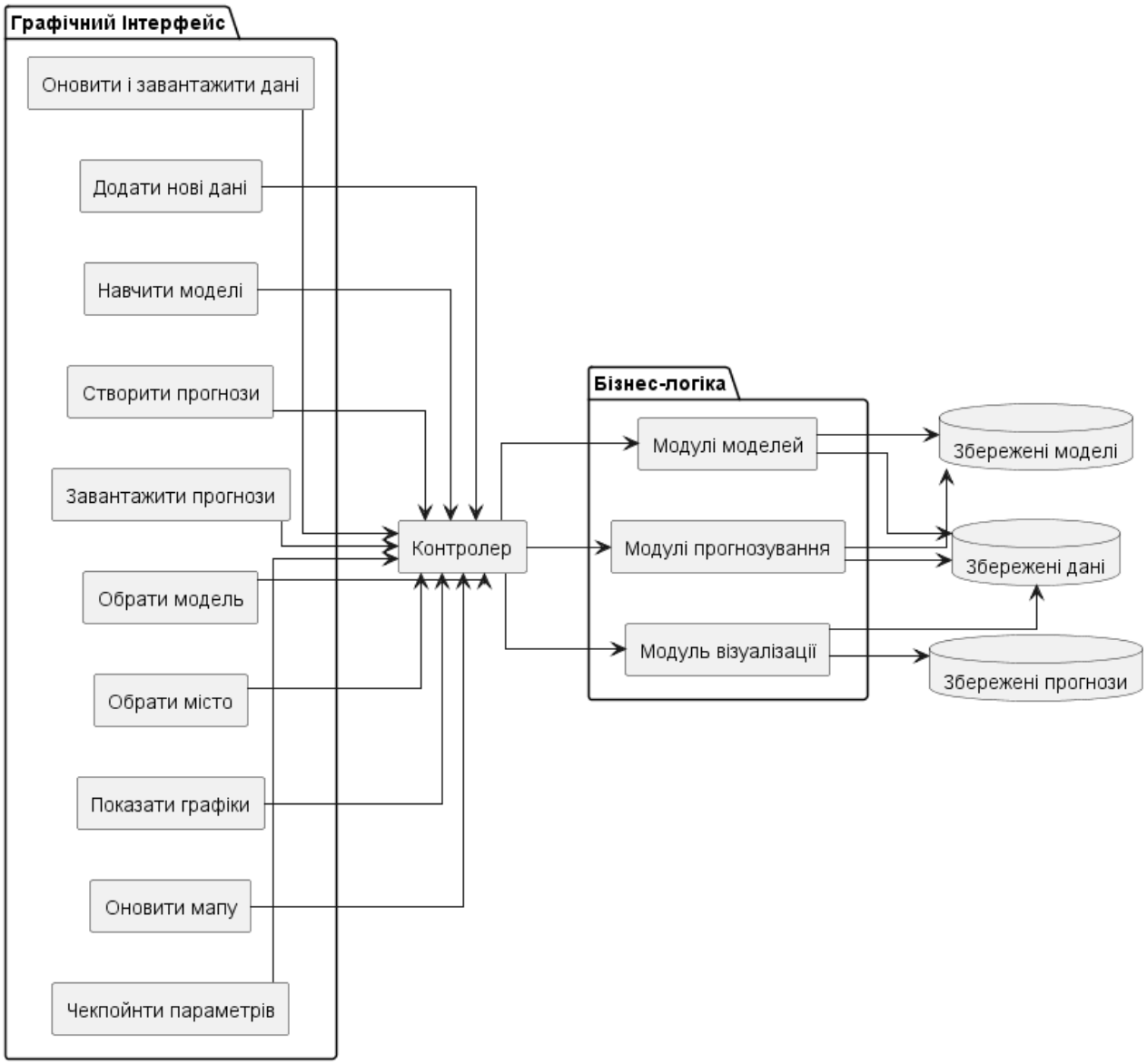
Система аналізу та прогнозування погодних умов з використанням методів глибинного навчання

Діаграма компонентів

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ 2025 р



					ІАЛЦ.467200.004 Д1		
		№ докум.	Підпис	Дата			
Розробив		Чижов О. Ю.			**Система аналізу та прогнозування погодних умов з використанням методів глибинного навчання** Діаграма компонентів	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Волокита А. М.					
						НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІО-14	
Н. Контр.		Гончаренко О.О.					
Затвердив							

ДОДАТОК Б

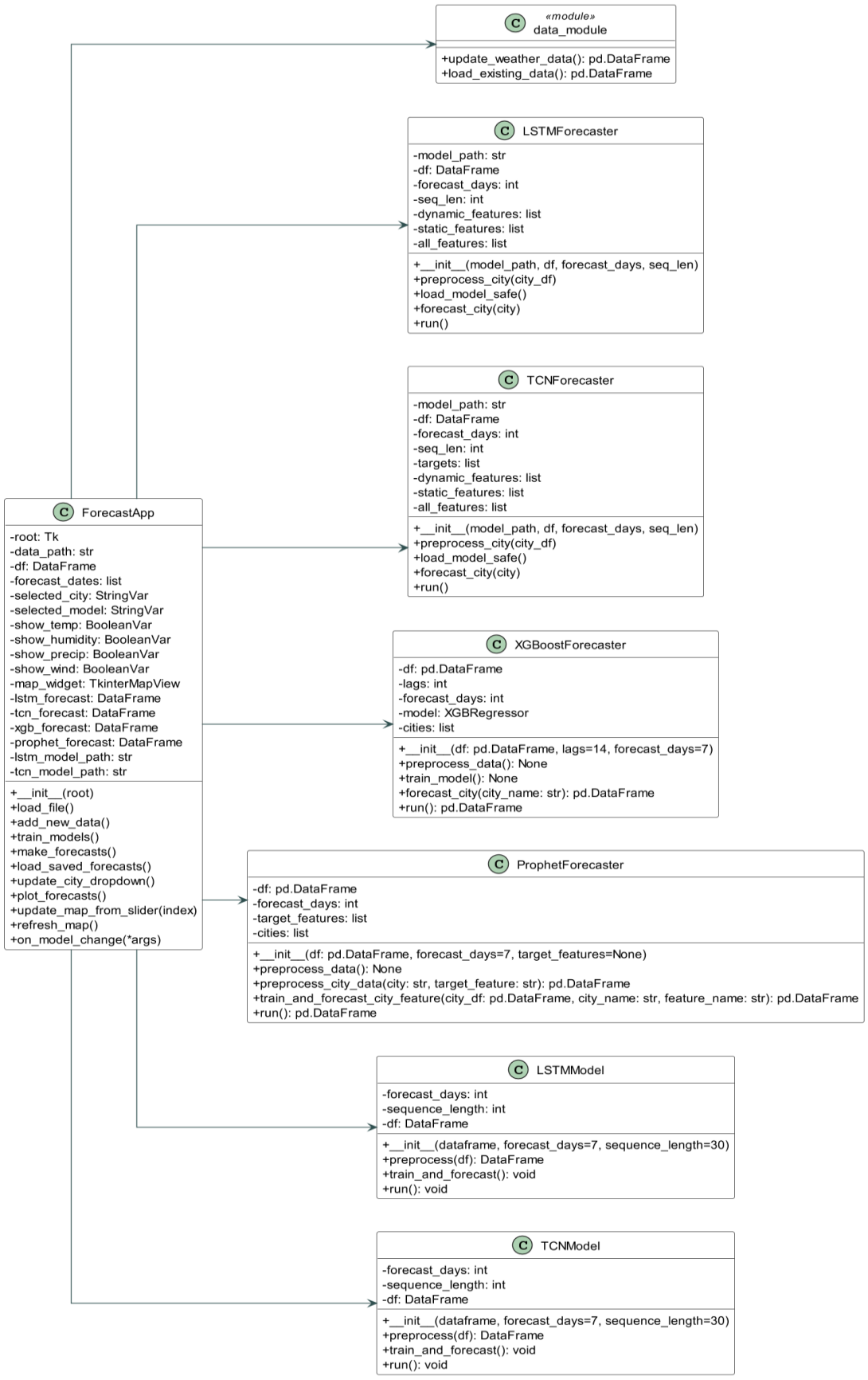
Система аналізу та прогнозування погодних умов з використанням методів глибинного навчання

Діаграма класів

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ 2025 р



					ІАЛЦ.467200.005 Д2		
		№ докум.	Підпис	Дата			
Розробив		Чижов О. Ю.			**Система аналізу та прогнозування погодних умов з використанням методів глибинного навчання** Діаграма класів	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Волокита А. М.					
Н. Контр.		Гончаренко О.О.				НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІО-14	
Затвердив							

ДОДАТОК В

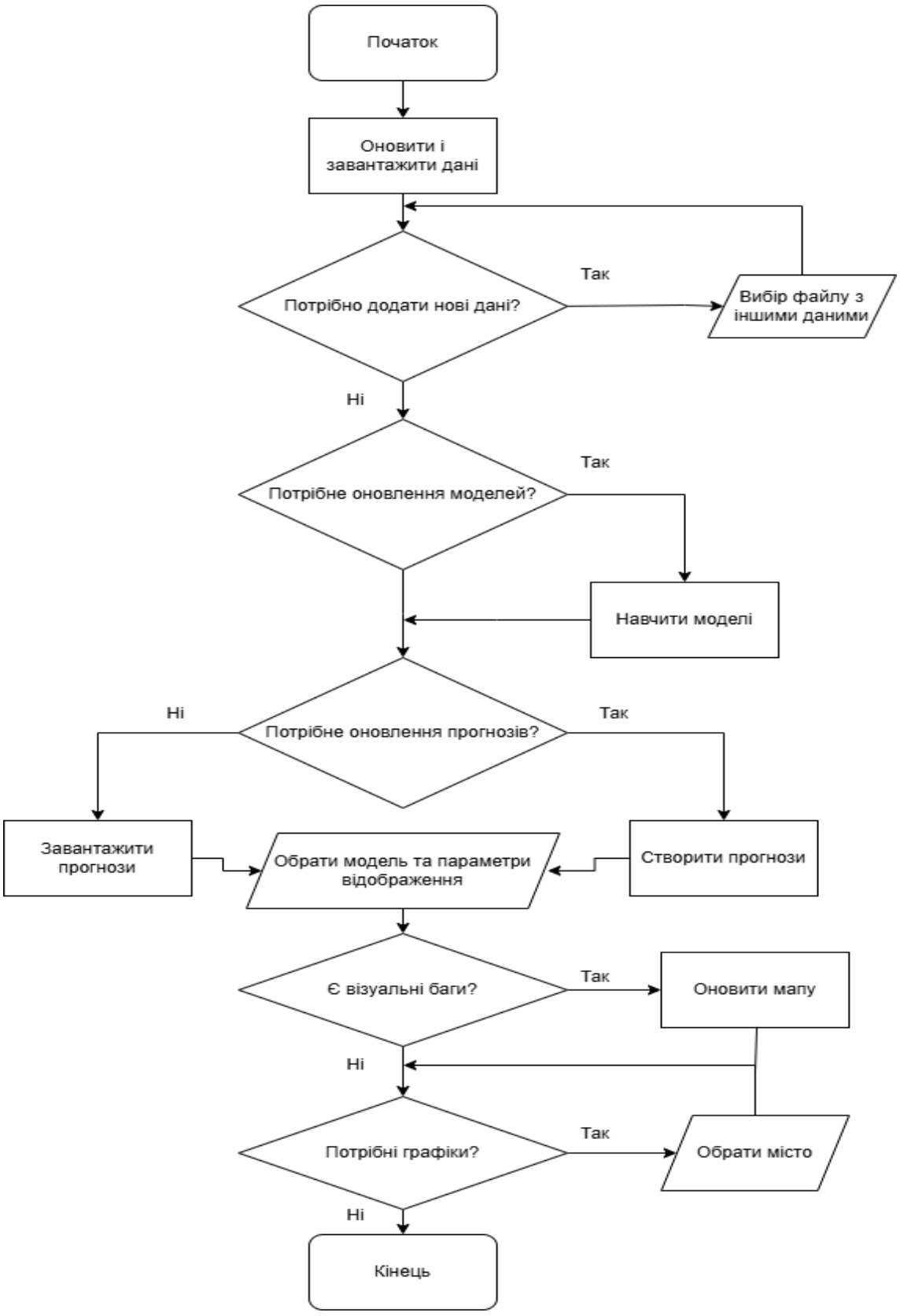
Система аналізу та прогнозування погодних умов з використанням методів глибинного навчання

Алгоритм дій користувача

ІАЛЦ.467200.006 Д3

Аркушів 1

Київ 2025 р



					ІАЛЦ.467200.006 Д3	
		№ докум.	Підпис	Дата		
Розробив		Чижов О. Ю.			**Система аналізу та прогнозування погодних умов з використанням методів глибинного навчання**	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Волокита А. М.				
Н. Контр.		Гончаренко О.О.			Алгоритм дій користувача	НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІО-14
Затвердив						

ДОДАТОК Г

Система аналізу та прогнозування погодних умов з використанням методів глибинного навчання

Текст програмного коду

ІАЛЦ.467200.007 Д4

Аркушів 22

Київ 2025 р

```
import os
import tkinter as tk
from tkinter import filedialog, messagebox
import pandas as pd
import matplotlib.pyplot as plt
from lstm_module import LSTMForecaster
from tcn_module import TCNForecaster
from data_module import update_weather_data, load_existing_data
from xgboost_module import XGBoostForecaster
from prophet_module import ProphetForecaster
from tkintermapview import TkinterMapView
from lstmmodel import LSTMModel
from tcnmodel import TCNModel

class ForecastApp:
    def __init__(self, root):
        self.root = root
        root.title("Прогноз погоди (TCN, LSTM, XGBoost, Prophet)")
        root.geometry("1200x700")

        self.data_path = None
        self.df = None

        self.lstm_forecast = None
        self.tcn_forecast = None
        self.xgb_forecast = None
        self.prophet_forecast = None

        self.lstm_model_path = 'saved_models/global_multi_model.keras'
        self.tcn_model_path = 'saved_models/global_tcn_multi_model.keras'

        self.forecast_dates = []
        self.selected_city = tk.StringVar(value="")  # Змінна для обраного міста

        main_frame = tk.Frame(root)
        main_frame.pack(fill=tk.BOTH, expand=True)

        control_frame = tk.Frame(main_frame)
        control_frame.pack(side=tk.LEFT, padx=10, pady=10, fill=tk.Y)

        self.btn_load = tk.Button(control_frame, text="Оновити і завантажити дані",
command=self.load_file)
        self.btn_load.pack(pady=5, fill=tk.X)

        self.btn_add_new_data = tk.Button(control_frame, text="Додати нові дані",
command=self.add_new_data)
        self.btn_add_new_data.pack(pady=5, fill=tk.X)

        self.btn_train_models = tk.Button(control_frame, text="Навчити моделі (LSTM +
TCN)", command=self.train_models,
                                          state='disabled')
        self.btn_train_models.pack(pady=5, fill=tk.X)

        self.btn_forecast = tk.Button(control_frame, text="Створити прогнози",
command=self.make_forecasts, state='disabled')
```

```
        self.btn_forecast.pack(pady=5, fill=tk.X)

        self.btn_load_forecasts = tk.Button(control_frame, text="Завантажити прогнози",
                                            command=self.load_saved_forecasts)
        self.btn_load_forecasts.pack(pady=5, fill=tk.X)



        self.selected_model = tk.StringVar(value='LSTM')
        self.model_dropdown = tk.OptionMenu(control_frame, self.selected_model, 'LSTM',
'TCN', 'XGBoost', 'Prophet')
        self.model_dropdown.config(width=20)
        self.model_dropdown.pack(pady=5)

        tk.Label(control_frame, text="Оберіть місто для графіків:", font=("Arial", 10,
"bold")).pack(pady=(10, 2))
        self.city_dropdown = tk.OptionMenu(control_frame, self.selected_city,
"Завантажте прогнози")
        self.city_dropdown.config(width=20, state='disabled')
        self.city_dropdown.pack(pady=5)

        self.selected_model.trace_add("write", self.on_model_change)

        self.btn_plot = tk.Button(control_frame, text="Показати графіки",
command=self.plot_forecasts, state='disabled')
        self.btn_plot.pack(pady=5, fill=tk.X)

        tk.Label(control_frame, text="Показати на мапі:", font=("Arial", 10,
"bold")).pack(pady=(10, 2))

        self.show_temp = tk.BooleanVar(value=True)
        self.show_humidity = tk.BooleanVar(value=True)
        self.show_precip = tk.BooleanVar(value=True)
        self.show_wind = tk.BooleanVar(value=True)

        for text, var in [("Температура", self.show_temp),
                          ("Вологість", self.show_humidity),
                          ("Опади", self.show_precip),
                          ("Вітер", self.show_wind)]:
            cb = tk.Checkbutton(control_frame, text=text, variable=var,
                                command=lambda:
self.update_map_from_slider(self.date_slider.get()))
            cb.pack(anchor='w')

        self.btn_refresh_map = tk.Button(control_frame, text="Оновити мапу",
command=self.refresh_map)
        self.btn_refresh_map.pack(pady=(20, 5), fill=tk.X)

        self.map_frame = tk.Frame(main_frame)
        self.map_frame.pack(side=tk.LEFT, padx=10, pady=10, fill=tk.BOTH, expand=True)

        self.map_widget = TkinterMapView(self.map_frame, width=1000, height=550,
corner_radius=0)
        self.map_widget.set_position(48.5, 31.0)
        self.map_widget.set_zoom(6)
```

```
        self.map_widget.pack(fill=tk.BOTH, expand=True)

        self.date_slider = tk.Scale(self.map_frame, from_=0, to=0,
orient=tk.HORIZONTAL, length=800, command=self.update_map_from_slider)
        self.date_slider.pack(pady=5)

        self.slider_label = tk.Label(self.map_frame, text="Дата не вибрана")
        self.slider_label.pack()

    def refresh_map(self):
        current_slider_pos = self.date_slider.get()
        current_position = self.map_widget.get_position()
        current_label_text = self.slider_label.cget("text")
        slider_to = self.date_slider.cget("to")

        for widget in self.map_frame.winfo_children():
            widget.destroy()

        self.map_widget = TkinterMapView(self.map_frame, width=1000, height=550,
corner_radius=0)
        self.map_widget.set_position(current_position[0], current_position[1])
        self.map_widget.set_zoom(6)
        self.map_widget.pack(fill=tk.BOTH, expand=True)

        self.date_slider = tk.Scale(self.map_frame, from_=0, to=slider_to,
orient=tk.HORIZONTAL, length=800, command=self.update_map_from_slider)
        self.date_slider.set(current_slider_pos)
        self.date_slider.pack(pady=5)

        self.slider_label = tk.Label(self.map_frame, text=current_label_text)
        self.slider_label.pack()

        self.update_map_from_slider(current_slider_pos)

    def on_model_change(self, *args):
        self.update_map_from_slider(self.date_slider.get())

    def load_file(self):
        try:
            update_weather_data()
            self.df = load_existing_data()
            if not self.df.empty:
                self.data_path = "data.parquet"
                messagebox.showinfo("Успіх", f"Файл {os.path.basename(self.data_path)}
оновлено та завантажено!")
                self.btn_forecast.config(state='normal')
                self.btn_train_models.config(state='normal')
            else:
                messagebox.showwarning("Увага", "Дані не знайдено після оновлення.")
        except Exception as e:
            messagebox.showerror("Помилка", f"Помилка при завантаженні та
оновленні:\n{e}")

    def add_new_data(self):
        file_path = filedialog.askopenfilename(
```

```
            filetypes=[("Parquet файли", "*.parquet")],
            title="Виберіть файл з новими даними"
        )
        if not file_path:
            return

        if not hasattr(self, "data_path") or not os.path.exists(self.data_path):
            messagebox.showerror("Помилка",
                                 "Спочатку завантажте основний файл даних через
'Оновити і завантажити дані'.")
            return

        try:
            new_df = pd.read_parquet(file_path)
            existing_df = pd.read_parquet(self.data_path)

            if set(new_df.columns) != set(existing_df.columns):
                raise ValueError("Формат файлу не відповідає основному (різні
колонки).")

            combined_df = pd.concat([existing_df, new_df], ignore_index=True)
            combined_df.drop_duplicates(subset=["date", "city"], keep="last",
inplace=True)
            combined_df.sort_values(["city", "date"], inplace=True)

            self.df = combined_df

            messagebox.showinfo("Успіх", "Нові дані успішно додано до робочого набору!
Файл не змінено.")

        except Exception as e:
            messagebox.showerror("Помилка", f"Не вдалося додати дані:\n{str(e)}")

    def train_models(self):
        if self.df is None:
            messagebox.showwarning("Увага", "Завантажте дані перед навчанням моделей.")
            return

        try:
            messagebox.showinfo("Навчання", "Розпочато навчання моделей...")

            # Навчання LSTM
            lstm_trainer = LSTMModel(self.df)
            lstm_trainer.run()

            # Навчання TCN
            tcn_trainer = TCNModel(self.df)
            tcn_trainer.run()

            messagebox.showinfo("Успіх", "Моделі успішно навчені та збережені.")

        except Exception as e:
            messagebox.showerror("Помилка", f"Помилка під час навчання
моделей:\n{str(e)}")
```

```
    def update_city_dropdown(self):
        cities = set(self.lstm_forecast['city']) & set(self.tcn_forecast['city']) & \
                 set(self.xgb_forecast['city']) & set(self.prophet_forecast['city'])
        if cities:
            self.selected_city.set("")  # Не встановлюємо місто за замовчуванням
            menu = self.city_dropdown["menu"]
            menu.delete(0, "end")
            menu.add_command(label="Оберіть місто", command=lambda:
self.selected_city.set(""))
            for city in sorted(cities):
                menu.add_command(label=city, command=lambda c=city:
self.selected_city.set(c))
            self.city_dropdown.config(state='normal')
        else:
            self.selected_city.set("Немає міст")
            self.city_dropdown.config(state='disabled')

    def load_saved_forecasts(self):
        try:
            base_path = "saved_forecasts"
            lstm_path = os.path.join(base_path, "lstm_forecast.csv")
            tcn_path = os.path.join(base_path, "tcn_forecast.csv")
            xgb_path = os.path.join(base_path, "xgboost_forecast.csv")
            prophet_path = os.path.join(base_path, "prophet_forecast.csv")

            if not all(os.path.exists(p) for p in [lstm_path, tcn_path, xgb_path,
prophet_path]):
                messagebox.showwarning("Увага", "Не всі прогнози знайдено. Спочатку
створіть прогнози.")
                return

            self.lstm_forecast = pd.read_csv(lstm_path)
            self.tcn_forecast = pd.read_csv(tcn_path)
            self.xgb_forecast = pd.read_csv(xgb_path)
            self.prophet_forecast = pd.read_csv(prophet_path)

            dates = pd.to_datetime(self.lstm_forecast['date'].unique())
            dates = sorted(dates)
            self.forecast_dates = dates

            self.date_slider.config(to=len(dates) - 1)
            self.date_slider.set(len(dates) - 1)
            self.update_map_from_slider(len(dates) - 1)

            self.update_city_dropdown()

            self.btn_plot.config(state='normal')
            self.btn_forecast.config(state='normal')

            messagebox.showinfo("Успіх", "Прогнози успішно завантажено.")

        except Exception as e:
            if isinstance(e, TypeError) and "'NoneType' object is not subscriptable" in
str(e):
                messagebox.showwarning("Увага", "Спочатку потрібно завантажити дані.")
```

```
            else:
                messagebox.showerror("Помилка", f"Помилка при завантаженні
прогнозів:\n{e}")

    def make_forecasts(self):
        if self.df is None:
            messagebox.showwarning("Увага", "Завантажте файл з даними.")
            return

        os.makedirs('saved_forecasts', exist_ok=True)

        lstm = LSTMForecaster(self.lstm_model_path, self.df)
        self.lstm_forecast = lstm.run()
        if self.lstm_forecast is not None:
            self.lstm_forecast.to_csv('saved_forecasts/lstm_forecast.csv', index=False)

        tcn = TCNForecaster(self.tcn_model_path, self.df)
        self.tcn_forecast = tcn.run()
        if self.tcn_forecast is not None:
            self.tcn_forecast.to_csv('saved_forecasts/tcn_forecast.csv', index=False)

        xgb = XGBoostForecaster(self.df)
        self.xgb_forecast = xgb.run()
        if self.xgb_forecast is not None:
            self.xgb_forecast.to_csv('saved_forecasts/xgboost_forecast.csv',
index=False)

        prophet = ProphetForecaster(self.df)
        self.prophet_forecast = prophet.run()
        if self.prophet_forecast is not None:
            self.prophet_forecast.to_csv('saved_forecasts/prophet_forecast.csv',
index=False)

        dates = pd.to_datetime(self.lstm_forecast['date'].unique())
        dates = sorted(dates)
        self.forecast_dates = dates
        self.date_slider.config(to=len(dates) - 1)
        self.date_slider.set(len(dates) - 1)
        self.update_map_from_slider(len(dates) - 1)

        self.update_city_dropdown()

        messagebox.showinfo("Готово", "Прогнози створено та збережено.")
        self.btn_plot.config(state='normal')

    def plot_forecasts(self):
        if not all([self.lstm_forecast is not None, self.tcn_forecast is not None,
                    self.xgb_forecast is not None, self.prophet_forecast is not None]):
            messagebox.showwarning("Увага", "Створіть або завантажте всі прогнози
спочатку.")
            return

        cities = set(self.lstm_forecast['city']) & set(self.tcn_forecast['city']) & \
                 set(self.xgb_forecast['city']) & set(self.prophet_forecast['city'])
        if not cities:
```

```
            messagebox.showwarning("Увага", "Немає спільних міст у прогнозах.")
            return

        if not self.selected_city.get() or self.selected_city.get() == "Оберіть місто":
            messagebox.showwarning("Увага", "Оберіть місто зі списку.")
            return

        self.plot_selected_city()

    def plot_selected_city(self):
        city = self.selected_city.get()
        if not city or city == "Оберіть місто":
            return

        plt.close('all')

        for var in ['temp', 'humidity', 'precip', 'wind_speed']:
            plt.figure(figsize=(10, 5))
            plt.title(f"{var.capitalize()} для {city}")
            for forecast, label, marker in zip(
                    [self.lstm_forecast, self.tcn_forecast, self.xgb_forecast,
self.prophet_forecast],
                    ['LSTM', 'TCN', 'XGBoost', 'Prophet'],
                    ['o', 'x', 's', '^']
            ):
                city_df = forecast[forecast['city'] == city]
                if var in city_df.columns:
                    plt.plot(city_df['date'], city_df[var], label=label, marker=marker)

            plt.xlabel("Дата")
            plt.ylabel(var.capitalize())
            plt.legend()
            plt.grid(True)
            plt.show()

    def update_map_from_slider(self, slider_index):
        if isinstance(slider_index, str):
            slider_index = int(slider_index)

        if not self.forecast_dates:
            return

        selected_date = self.forecast_dates[slider_index]
        self.slider_label.config(text=f"Обрана дата: {selected_date.strftime('%Y-%m-
%d')}")

        forecast = {
            'LSTM': self.lstm_forecast,
            'TCN': self.tcn_forecast,
            'XGBoost': self.xgb_forecast,
            'Prophet': self.prophet_forecast
        }.get(self.selected_model.get())

        if forecast is None:
            return
```

```
        self.map_widget.delete_all_marker()

        day_forecast = forecast[forecast['date'] == selected_date.strftime('%Y-%m-%d')]

        merged = pd.merge(day_forecast, self.df[['city', 'latitude',
'longitude']].drop_duplicates(), on='city')

        for _, row in merged.iterrows():
            city_name = row['city'].replace(", Ukraine", "")

            temp_str = f"{row['temp']:.1f}°C" if self.show_temp.get() and 'temp' in row
and pd.notna(
                row['temp']) else ""
            humidity_str = f"💧 {row['humidity']:.0f}%" if self.show_humidity.get() and
'humidity' in row and pd.notna(
                row['humidity']) else ""
            precip_str = f"🌧 {row['precip']:.1f} мм" if self.show_precip.get() and
'precip' in row and pd.notna(
                row['precip']) else ""
            wind_str = f"🌬 {row['wind_speed']:.1f} м/с" if self.show_wind.get() and
'wind_speed' in row and pd.notna(
                row['wind_speed']) else ""

            lines = [city_name]
            for part in [temp_str, humidity_str, precip_str, wind_str]:
                if part:
                    lines.append(part)

            indent = "                        "
            self.map_widget.set_marker(
                row['latitude'], row['longitude'],
                text="\n".join([indent + line for line in lines])
            )


if __name__ == "__main__":
    root = tk.Tk()
    app = ForecastApp(root)
    root.mainloop()


import os
import numpy as np
import pandas as pd
from datetime import timedelta
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import load_model


class LSTMForecaster:
    def __init__(self, model_path, df, forecast_days=7, seq_len=30):
        self.model_path = model_path
        self.df = df
        self.forecast_days = forecast_days
```

```
        self.seq_len = seq_len
        self.dynamic_features = ['temp', 'humidity', 'precip', 'wind_speed',
'rolling_temp', 'dayofyear']
        self.static_features = ['latitude', 'longitude', 'elevation']
        self.all_features = self.dynamic_features + self.static_features

    def preprocess_city(self, city_df):
        city_df = city_df[['date', 'temp', 'humidity', 'precip', 'wind_speed',
                           'latitude', 'longitude', 'elevation']].copy()
        city_df['date'] = pd.to_datetime(city_df['date'])
        city_df = city_df.sort_values('date')
        city_df['rolling_temp'] = city_df['temp'].rolling(7).mean().bfill()
        city_df['dayofyear'] = city_df['date'].dt.dayofyear
        city_df.dropna(inplace=True)
        return city_df

    def load_model_safe(self):
        if not os.path.exists(self.model_path):
            print(f"Модель не знайдена: {self.model_path}")
            return None
        return load_model(self.model_path)

    def forecast_city(self, city):
        city_df = self.df[self.df['city'] == city].copy()
        city_df = self.preprocess_city(city_df)
        if len(city_df) < self.seq_len:
            print(f"Недостатньо даних для міста {city}")
            return None

        full_df = city_df[self.all_features].copy()
        scaler = MinMaxScaler()
        scaled = scaler.fit_transform(full_df)

        df_scaled = pd.DataFrame(scaled, columns=self.all_features)
        input_seq = df_scaled[self.dynamic_features].values[-self.seq_len:]
        static_vec = df_scaled[self.static_features].values[-1]

        input_seq = np.expand_dims(input_seq, axis=0)
        static_vec = np.expand_dims(static_vec, axis=0)

        model = self.load_model_safe()
        if model is None:
            return None

        pred_scaled = model.predict([input_seq, static_vec])[0]  # (forecast_days, 4)

        last_row = df_scaled.iloc[-1].copy()
        repeated_features = np.tile(last_row.values, (self.forecast_days, 1))
        repeated_features[:, 0:4] = pred_scaled

        denorm = scaler.inverse_transform(repeated_features)
        forecast_values = denorm[:, 0:4]  # temp, humidity, precip, wind_speed
        forecast_values[:, 2] = np.maximum(0, forecast_values[:, 2])

        forecast_dates = pd.date_range(start=city_df['date'].max() + timedelta(days=1),
```

```
periods=self.forecast_days)
        return pd.DataFrame({
            'date': forecast_dates,
            'temp': forecast_values[:, 0],
            'humidity': forecast_values[:, 1],
            'precip': forecast_values[:, 2],
            'wind_speed': forecast_values[:, 3],
            'city': city
        })

    def run(self):
        forecasts = []
        for city in self.df['city'].unique():
            f = self.forecast_city(city)
            if f is not None:
                forecasts.append(f)
        return pd.concat(forecasts) if forecasts else None


import os
import numpy as np
import pandas as pd
from datetime import timedelta
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import load_model


class TCNForecaster:
    def __init__(self, model_path, df, forecast_days=7, seq_len=30):
        self.model_path = model_path
        self.df = df
        self.forecast_days = forecast_days
        self.seq_len = seq_len

        # Має відповідати коду навчання
        self.targets = ['temp', 'humidity', 'precip', 'wind_speed']
        self.dynamic_features = self.targets + ['rolling_temp', 'dayofyear']
        self.static_features = ['latitude', 'longitude', 'elevation']
        self.all_features = self.dynamic_features + self.static_features

    def preprocess_city(self, city_df):
        city_df = city_df[['date', 'temp', 'humidity', 'precip', 'wind_speed',
                           'latitude', 'longitude', 'elevation']].copy()
        city_df['date'] = pd.to_datetime(city_df['date'])
        city_df = city_df.sort_values('date')
        city_df['rolling_temp'] = city_df['temp'].rolling(7).mean().bfill()
        city_df['dayofyear'] = city_df['date'].dt.dayofyear
        city_df.dropna(inplace=True)
        return city_df

    def load_model_safe(self):
        if not os.path.exists(self.model_path):
            print(f"Модель не знайдена: {self.model_path}")
            return None
        return load_model(self.model_path, compile=False)
```

```
    def forecast_city(self, city):
        city_df = self.df[self.df['city'] == city].copy()
        city_df = self.preprocess_city(city_df)
        if len(city_df) < self.seq_len:
            print(f"Недостатньо даних для міста {city}")
            return None

        full_df = city_df[self.all_features].copy()
        scaler = MinMaxScaler()
        scaled = scaler.fit_transform(full_df)
        df_scaled = pd.DataFrame(scaled, columns=self.all_features)

        # Підготовка входів
        input_seq = df_scaled[self.dynamic_features].values[-self.seq_len:]  # (30, 6)
        static_vec = df_scaled[self.static_features].values[-1]              # (3,)

        input_seq = np.expand_dims(input_seq, axis=0)  # (1, 30, 6)
        static_vec = np.expand_dims(static_vec, axis=0)  # (1, 3)

        model = self.load_model_safe()
        if model is None:
            return None

        pred_scaled = model.predict([input_seq, static_vec], verbose=0)[0]  # (7, 4)

        # Відновлення масштабу
        last_row = df_scaled.iloc[-1].copy()
        repeated_features = np.tile(last_row.values, (self.forecast_days, 1))
        repeated_features[:, 0:4] = pred_scaled  # вставити temp, humidity, precip,
wind_speed

        denorm = scaler.inverse_transform(repeated_features)
        forecast_values = denorm[:, 0:4]
        forecast_values[:, 2] = np.maximum(0, forecast_values[:, 2])  # опади не < 0

        forecast_dates = pd.date_range(start=city_df['date'].max() + timedelta(days=1),
periods=self.forecast_days)
        return pd.DataFrame({
            'date': forecast_dates,
            'temp': forecast_values[:, 0],
            'humidity': forecast_values[:, 1],
            'precip': forecast_values[:, 2],
            'wind_speed': forecast_values[:, 3],
            'city': city
        })

    def run(self):
        forecasts = []
        for city in self.df['city'].unique():
            f = self.forecast_city(city)
            if f is not None:
                forecasts.append(f)
        return pd.concat(forecasts) if forecasts else None
```

```
import pandas as pd
import numpy as np
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error
from datetime import timedelta


class XGBoostForecaster:
    def __init__(self, df: pd.DataFrame, lags=14, forecast_days=7):
        self.df = df.copy()
        self.lags = lags
        self.forecast_days = forecast_days
        self.model = None
        self.cities = []

    def preprocess_data(self):
        self.df['date'] = pd.to_datetime(self.df['date'])
        self.df = self.df[['city', 'date', 'temp', 'latitude', 'longitude',
'elevation']].dropna()
        self.df = self.df.sort_values(['city', 'date'])
        self.df['dayofyear'] = self.df['date'].dt.dayofyear
        self.df['rolling_mean'] = self.df.groupby('city')['temp'].transform(
            lambda x: x.rolling(window=7).mean().fillna(method='bfill'))

        for lag in range(1, self.lags + 1):
            self.df[f'lag_{lag}'] = self.df.groupby('city')['temp'].shift(lag)

        self.df.dropna(inplace=True)
        self.cities = self.df['city'].unique()

    def train_model(self):
        features = [f'lag_{i}' for i in range(1, self.lags + 1)] + [
            'dayofyear', 'rolling_mean', 'latitude', 'longitude', 'elevation'
        ]
        X = self.df[features]
        y = self.df['temp']

        self.model = XGBRegressor(
            n_estimators=500,
            max_depth=5,
            learning_rate=0.1,
            subsample=0.8,
            colsample_bytree=0.8
        )
        self.model.fit(X, y)

    def forecast_city(self, city_name):
        city_df = self.df[self.df['city'] == city_name].copy()
        if city_df.empty:
            raise ValueError(f"Місто '{city_name}' не знайдено в даних.")

        last_known = city_df.iloc[-self.lags:].copy()
        latitude = last_known['latitude'].iloc[-1]
        longitude = last_known['longitude'].iloc[-1]
        elevation = last_known['elevation'].iloc[-1]
```

```
        current_date = last_known['date'].iloc[-1]

        forecasts = []

        for _ in range(self.forecast_days):
            next_date = current_date + timedelta(days=1)
            dayofyear = next_date.timetuple().tm_yday
            rolling_mean = last_known['temp'][-7:].mean()
            lags = list(last_known['temp'][-self.lags:].values)

            features = lags + [dayofyear, rolling_mean, latitude, longitude, elevation]
            temp_pred = self.model.predict([features])[0]
            forecasts.append((next_date, temp_pred))

            new_row = pd.DataFrame({
                'temp': [temp_pred],
                'date': [next_date],
                'latitude': [latitude],
                'longitude': [longitude],
                'elevation': [elevation]
            })
            last_known = pd.concat([last_known, new_row], ignore_index=True)
            current_date = next_date

        forecast_df = pd.DataFrame(forecasts, columns=['date', 'temp'])
        forecast_df['city'] = city_name
        return forecast_df

    def run(self):
        self.preprocess_data()
        self.train_model()
        all_forecasts = []

        for city in self.cities:
            forecast_df = self.forecast_city(city)
            all_forecasts.append(forecast_df)

        return pd.concat(all_forecasts, ignore_index=True)


import pandas as pd
from prophet import Prophet
import warnings

warnings.filterwarnings("ignore")


class ProphetForecaster:
    def __init__(self, df: pd.DataFrame, forecast_days=7, target_features=None):
        self.df = df.copy()
        self.forecast_days = forecast_days
        self.target_features = target_features or ['temp', 'humidity', 'precip',
'wind_speed']
        self.cities = []
```

```
    def preprocess_data(self):
        required_columns = ['date', 'city', 'latitude', 'longitude', 'elevation'] +
self.target_features
        if not all(col in self.df.columns for col in required_columns):
            raise ValueError(f"DataFrame повинен містити стовпці: {required_columns}")

        self.df['date'] = pd.to_datetime(self.df['date'])
        self.df = self.df[self.df['date'] > '2014-01-01']
        self.df = self.df.dropna(subset=required_columns)
        self.cities = self.df['city'].unique()

    def preprocess_city_data(self, city, target_feature):
        city_df = self.df[self.df['city'] == city].copy()
        city_df = city_df[['date', target_feature, 'latitude', 'longitude',
'elevation']]
        city_df = city_df.rename(columns={'date': 'ds', target_feature: 'y'})
        city_df = city_df.sort_values('ds').reset_index(drop=True)

        if city_df['y'].isnull().any():
            city_df['y'] = city_df['y'].fillna(method='ffill')

        return city_df

    def train_and_forecast_city_feature(self, city_df, city_name, feature_name):
        model = Prophet(yearly_seasonality=True, daily_seasonality=False)
        model.add_regressor('latitude')
        model.add_regressor('longitude')
        model.add_regressor('elevation')

        model.fit(city_df)

        future = model.make_future_dataframe(periods=self.forecast_days)
        for col in ['latitude', 'longitude', 'elevation']:
            future[col] = city_df[col].iloc[0]

        forecast = model.predict(future)
        forecast = forecast[['ds', 'yhat']].tail(self.forecast_days)
        forecast = forecast.rename(columns={'ds': 'date', 'yhat': feature_name})
        forecast['city'] = city_name

        # Обмеження значень
        if feature_name == 'humidity':
            forecast[feature_name] = forecast[feature_name].clip(lower=0, upper=100)
        elif feature_name == 'precip':
            forecast[feature_name] = forecast[feature_name].clip(lower=0)

        return forecast

    def run(self):
        self.preprocess_data()
        all_forecasts = []

        for city in self.cities:
            city_forecast = None
```

```
            for feature in self.target_features:
                city_df = self.preprocess_city_data(city, feature)
                forecast_df = self.train_and_forecast_city_feature(city_df, city,
feature)

                if city_forecast is None:
                    city_forecast = forecast_df
                else:
                    city_forecast = city_forecast.merge(forecast_df[[feature]],
left_index=True, right_index=True)

            all_forecasts.append(city_forecast)

        return pd.concat(all_forecasts, ignore_index=True)


import requests
import pandas as pd
import os
import time
from datetime import datetime, timedelta

API_KEY = "EUXAXM298SS6SFWE9XZQ4CAEU"
OUTPUT_JSON = "data.json"
OUTPUT_PARQUET = "data.parquet"
BASE_URL =
"https://weather.visualcrossing.com/VisualCrossingWebServices/rest/services/timeline/"
ELEVATION_API_URL = "https://api.open-elevation.com/api/v1/lookup"
MAX_REQUESTS = 1000


def fetch_weather(date: str, city: str):
    url =
f"{BASE_URL}{city}/{date}?unitGroup=metric&include=days&key={API_KEY}&contentType=json"
    response = requests.get(url)
    if response.status_code == 200:
        return response.json()
    else:
        print(f"Помилка запиту для {city} на {date}: {response.status_code}")
        return None


def get_elevation(lat, lon):
    try:
        response = requests.get(f"{ELEVATION_API_URL}?locations={lat},{lon}")
        if response.status_code == 200:
            data = response.json()
            return data["results"][0]["elevation"]
        else:
            print(f"Помилка запиту висоти для {lat},{lon}: {response.status_code}")
    except Exception as e:
        print(f"Виняток при запиті висоти: {e}")
    return None
```

```
def load_existing_data():
    if os.path.exists(OUTPUT_PARQUET):
        return pd.read_parquet(OUTPUT_PARQUET)
    return pd.DataFrame()


def load_existing_json():
    if os.path.exists(OUTPUT_JSON):
        return pd.read_json(OUTPUT_JSON, orient="records")
    return pd.DataFrame()


def update_weather_data():
    all_data = []
    current_date = datetime.today()
    existing_data = load_existing_data()
    existing_json_data = load_existing_json()
    request_count = 0

    unique_cities = existing_data['city'].unique() if not existing_data.empty else
["Zaporizhzhia, Ukraine"]

    for city in unique_cities:
        date = current_date
        while date >= datetime(2025, 6, 1) and request_count < MAX_REQUESTS:
            date_str = date.strftime("%Y-%m-%d")

            if not existing_data.empty and not existing_data.query("date == @date_str
and city == @city").empty:
                print(f"Дані для {city} на {date_str} вже існують.")
            else:
                data = fetch_weather(date_str, city)
                if data:
                    day_info = data.get("days", [{}])[0]
                    all_data.append({
                        "date": date_str,
                        "city": city,
                        "latitude": data.get("latitude", None),
                        "longitude": data.get("longitude", None),
                        "temp": day_info.get("temp", None),
                        "humidity": day_info.get("humidity", None),
                        "precip": day_info.get("precip", None),
                        "wind_speed": day_info.get("windspeed", None),
                        "elevation": None
                    })
                    request_count += 1
                else:
                    break
            date -= timedelta(days=1)

    if not all_data:
        print("Нові дані відсутні.")
        return

    df_new = pd.DataFrame(all_data)
```

```
    if not existing_json_data.empty:
        df_new = pd.concat([existing_json_data, df_new], ignore_index=True)
        df_new = df_new.drop_duplicates(subset=['date', 'city'])

    df_new['date'] = pd.to_datetime(df_new['date'])
    df_new = df_new[df_new['date'] > '2014-01-01']
    df_new['date'] = df_new['date'].dt.strftime('%Y-%m-%d')

    elevation_map = {}
    for city in df_new['city'].unique():
        city_rows = df_new[df_new['city'] == city]
        coord_row = city_rows[['latitude', 'longitude']].dropna().head(1)

        if not coord_row.empty:
            lat = coord_row.iloc[0]['latitude']
            lon = coord_row.iloc[0]['longitude']
            if pd.notna(lat) and pd.notna(lon):
                elevation = get_elevation(lat, lon)
                if elevation is not None:
                    elevation_map[city] = elevation
                    print(f"🌍 {city}: elevation = {elevation} м")
                time.sleep(1)

    def fill_elevation(row):
        if pd.isna(row.get('elevation')) and row['city'] in elevation_map:
            row['elevation'] = elevation_map[row['city']]
        return row

    df_new = df_new.apply(fill_elevation, axis=1)

    df_new.to_json(OUTPUT_JSON, orient="records", indent=4)
    print(f"JSON збережено у {OUTPUT_JSON}")

    df_final = pd.concat([existing_data, df_new], ignore_index=True) if not
existing_data.empty else df_new
    df_final = df_final.drop_duplicates(subset=['date', 'city'])
    df_final['date'] = pd.to_datetime(df_final['date'])
    df_final = df_final[df_final['date'] > '2014-01-01']
    df_final.to_parquet(OUTPUT_PARQUET, index=False)
    print(f"Parquet збережено у {OUTPUT_PARQUET}")


if __name__ == "__main__":
    update_weather_data()


import pandas as pd
import numpy as np
from datetime import timedelta
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, LSTM, Dense, Reshape, Concatenate
import os
```

```
class LSTMModel:
    def __init__(self, dataframe, forecast_days=7, sequence_length=30):
        self.forecast_days = int(forecast_days)
        self.sequence_length = int(sequence_length)
        if isinstance(dataframe, pd.DataFrame):
            self.df = dataframe.copy()
        else:
            raise ValueError("Очікується pandas DataFrame.")

    def preprocess(self, df):
        df = df[['date', 'city', 'temp', 'humidity', 'precip', 'wind_speed',
                 'latitude', 'longitude', 'elevation']].copy()
        df['date'] = pd.to_datetime(df['date'])
        df = df.sort_values(['city', 'date'])
        df['rolling_temp'] = df.groupby('city')['temp'].transform(lambda x:
x.rolling(7).mean().bfill())
        df['dayofyear'] = df['date'].dt.dayofyear
        df.dropna(inplace=True)
        return df

    def train_and_forecast(self):
        os.makedirs("saved_models", exist_ok=True)
        df = self.preprocess(self.df)

        targets = ['temp', 'humidity', 'precip', 'wind_speed']
        dynamic_features = targets + ['rolling_temp', 'dayofyear']
        static_features = ['latitude', 'longitude', 'elevation']
        all_features = dynamic_features + static_features

        # Масштабування
        scaler = MinMaxScaler()
        df_features = df[all_features]
        scaled = scaler.fit_transform(df_features)
        df_scaled = pd.DataFrame(scaled, columns=all_features)
        df_scaled['city'] = df['city'].values
        df_scaled['date'] = df['date'].values

        X_dynamic, X_static, y = [], [], []

        for city in df_scaled['city'].unique():
            city_data = df_scaled[df_scaled['city'] == city]
            city_dyn = city_data[dynamic_features].values
            city_stat = city_data[static_features].values

            for i in range(self.sequence_length, len(city_dyn) - self.forecast_days):
                X_dynamic.append(city_dyn[i - self.sequence_length:i])
                X_static.append(city_stat[i])
                y.append(city_dyn[i:i + self.forecast_days, 0:4])  # тільки цільові

        X_dynamic = np.array(X_dynamic)
        X_static = np.array(X_static)
        y = np.array(y)

        input_dyn = Input(shape=(self.sequence_length, len(dynamic_features)))
```

```
        input_stat = Input(shape=(len(static_features),))

        x_dyn = LSTM(64)(input_dyn)
        x_stat = Dense(16, activation='relu')(input_stat)

        merged = Concatenate()([x_dyn, x_stat])
        dense_out = Dense(self.forecast_days * 4)(merged)
        output = Reshape((self.forecast_days, 4))(dense_out)

        model = Model(inputs=[input_dyn, input_stat], outputs=output)
        model.compile(loss='mse', optimizer='adam')
        model.fit([X_dynamic, X_static], y, epochs=7, verbose=1)

        model_path = f"saved_models/global_multi_model.keras"
        model.save(model_path)
        print(f"\nМодель збережено: {model_path}")

        result = []
        for city in df_scaled['city'].unique():
            city_scaled_data = df_scaled[df_scaled['city'] == city]
            city_orig_data = df[df['city'] == city]

            input_seq = city_scaled_data[dynamic_features].values[-
self.sequence_length:]
            static_vec = city_scaled_data[static_features].values[-1]

            input_seq = np.expand_dims(input_seq, axis=0)
            static_vec = np.expand_dims(static_vec, axis=0)

            pred_scaled = model.predict([input_seq, static_vec])[0]

            last_row_orig = city_orig_data[all_features].iloc[-1].copy()
            repeated = np.tile(last_row_orig.values, (self.forecast_days, 1))
            repeated[:, 0:4] = pred_scaled

            denorm = scaler.inverse_transform(repeated)
            forecast_values = denorm[:, 0:4]

            forecast_dates = pd.date_range(start=city_orig_data['date'].max() +
timedelta(days=1),
                                           periods=self.forecast_days)

            city_forecast = pd.DataFrame({
                'date': forecast_dates,
                'temp': forecast_values[:, 0],
                'humidity': forecast_values[:, 1],
                'precip': forecast_values[:, 2],
                'wind_speed': forecast_values[:, 3],
                'city': city
            })
            result.append(city_forecast)

        full_forecast = pd.concat(result)
        full_forecast.to_csv('global_lstm_multi_forecast_all_cities.csv', index=False)
        print("\nПрогноз збережено у global_lstm_multi_forecast_all_cities.csv")
```

```
    def run(self):
        self.train_and_forecast()


import pandas as pd
import numpy as np
from datetime import timedelta
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Reshape, Concatenate
from tcn import TCN
import os


class TCNModel:
    def __init__(self, dataframe, forecast_days=7, sequence_length=30):
        self.forecast_days = int(forecast_days)
        self.sequence_length = int(sequence_length)
        if isinstance(dataframe, pd.DataFrame):
            self.df = dataframe.copy()
        else:
            raise ValueError("Очікується pandas DataFrame.")

    def preprocess(self, df):
        df = df[['date', 'city', 'temp', 'humidity', 'precip', 'wind_speed',
                 'latitude', 'longitude', 'elevation']].copy()
        df['date'] = pd.to_datetime(df['date'])
        df = df.sort_values(['city', 'date'])
        df['rolling_temp'] = df.groupby('city')['temp'].transform(lambda x:
x.rolling(7).mean().bfill())
        df['dayofyear'] = df['date'].dt.dayofyear
        df.dropna(inplace=True)
        return df

    def train_and_forecast(self):
        os.makedirs("saved_models", exist_ok=True)
        df = self.preprocess(self.df)

        targets = ['temp', 'humidity', 'precip', 'wind_speed']
        dynamic_features = targets + ['rolling_temp', 'dayofyear']
        static_features = ['latitude', 'longitude', 'elevation']
        all_features = dynamic_features + static_features

        full_df = df[all_features].copy()
        scaler = MinMaxScaler()
        scaled = scaler.fit_transform(full_df)
        df_scaled = pd.DataFrame(scaled, columns=all_features)
        df_scaled['city'] = df['city'].values
        df_scaled['date'] = df['date'].values

        X_dynamic, X_static, y = [], [], []

        for city in df_scaled['city'].unique():
            city_data = df_scaled[df_scaled['city'] == city]
```

```
            city_dyn = city_data[dynamic_features].values
            city_static = city_data[static_features].values

            for i in range(self.sequence_length, len(city_dyn) - self.forecast_days):
                X_dynamic.append(city_dyn[i - self.sequence_length:i])
                X_static.append(city_static[i])
                y.append(city_dyn[i:i + self.forecast_days, 0:4])

        X_dynamic = np.array(X_dynamic)
        X_static = np.array(X_static)
        y = np.array(y)

        input_dyn = Input(shape=(self.sequence_length, len(dynamic_features)))
        input_stat = Input(shape=(len(static_features),))

        x_dyn = TCN(nb_filters=64, kernel_size=3, dilations=[1, 2, 4, 8])(input_dyn)
        x_stat = Dense(16, activation='relu')(input_stat)

        merged = Concatenate()([x_dyn, x_stat])
        dense_out = Dense(self.forecast_days * 4)(merged)
        output = Reshape((self.forecast_days, 4))(dense_out)

        model = Model(inputs=[input_dyn, input_stat], outputs=output)
        model.compile(loss='mse', optimizer='adam')
        model.fit([X_dynamic, X_static], y, epochs=7, verbose=1)

        model_path = f"saved_models/global_tcn_multi_model.keras"
        model.save(model_path)
        print(f"\nМодель TCN збережено в {model_path}")

        result = []
        for city in df_scaled['city'].unique():
            city_data = df_scaled[df_scaled['city'] == city]
            city_raw = df[df['city'] == city]

            input_seq = city_data[dynamic_features].values[-self.sequence_length:]
            static_vec = city_data[static_features].values[-1]

            input_seq = np.expand_dims(input_seq, axis=0)
            static_vec = np.expand_dims(static_vec, axis=0)

            pred_scaled = model.predict([input_seq, static_vec])[0]

            last_row = city_data[all_features].iloc[-1].copy()
            repeated_features = np.tile(last_row.values, (self.forecast_days, 1))
            repeated_features[:, 0:4] = pred_scaled

            denorm = scaler.inverse_transform(repeated_features)
            forecast_values = denorm[:, 0:4]

            forecast_dates = pd.date_range(start=city_raw['date'].max() +
timedelta(days=1), periods=self.forecast_days)
            city_forecast = pd.DataFrame({
                'date': forecast_dates,
                'temp': forecast_values[:, 0],
```

```
                'humidity': forecast_values[:, 1],
                'precip': forecast_values[:, 2],
                'wind_speed': forecast_values[:, 3],
                'city': city
            })
            result.append(city_forecast)

        full_forecast = pd.concat(result)
        full_forecast.to_csv('global_tcn_multi_forecast_all_cities.csv', index=False)
        print("\nПрогноз збережено у global_tcn_multi_forecast_all_cities.csv")

    def run(self):
        self.train_and_forecast()
```