НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет програмних систем та прикладної математики
Кафедра системного програмування і спеціалізованих комп'ютерних систем

До захисту допущено:

Завідувач кафедри

_____ Віталій РОМАНКЕВИЧ

«___»_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою

«Системне програмування та спеціалізовані комп'ютерні системи»

спеціальності 123 «Комп'ютерна інженерія»

на тему: «Комп'ютерна система прогнозування енергоспоживання в інфраструктурі розумного дому»

Виконав:
студент IV курсу, групи КВ-23
Перетятько Богдан Вікторович _____

Керівник:
д.т.н., проф.
Терейковський Ігор Анатолійович _____

Консультант з нормконтролю:
доц. каф. СП і СКС, к.т.н
Клятченко Ярослав Михайлович _____

Рецензент:

_____

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

Київ – 2026 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет програмних систем та прикладної математики
Кафедра системного програмування і спеціалізованих комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Системне програмування та спеціалізовані комп'ютерні системи»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Віталій РОМАНКЕВИЧ
«___»_____20__ р.

ЗАВДАННЯ
на дипломний проєкт студенту
Перетятьку Богдану Вікторовичу

1. Тема проєкту «Комп'ютерна система прогнозування енергоспоживання в інфраструктурі розумного дому», керівник проєкту Терейковський І.А., Доктор технічних наук, Професор.

2. Термін подання студентом проєкту _____

3. Вихідні дані проєкту: див. Технічне завдання.

4. Зміст пояснювальної записки

- Вступ;
- Аналіз предметної області та існуючих рішень;
- Проєктування інформаційної інфраструктури та підготовка даних;
- Розробка та навчання моделей машинного навчання;
- Оцінка ефективності системи та аналіз результатів.

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій, тощо)

- Структурна схема інформаційної системи прогнозування;

- Блок-схема алгоритму прогнозування енергоспоживання;
- Порівняльні графіки результатів прогнозування та екзамену моделі;

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Клятченко Я.М., к.т.н., доцент каф. СП і СКС		

7. Дата видачі завдання: 30 жовтня 2025р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Терміни виконання етапів проєкту	Примітка
1.	Вивчення літератури за тематикою проєкту	13.11.2025	
2.	Розроблення та узгодження технічного завдання	17.11.2025	
3.	Аналіз існуючих рішень	03.12.2025	
4.	Підготовка матеріалів першого розділу дипломного проєкту	13.01.2026	
5.	Підготовка матеріалів другого розділу дипломного проєкту	20.01.2026	
6.	Підготовка графічної частини дипломного проєкту	09.02.2026	
7.	Оформлення документації дипломного проєкту	15.05.2026	
8.	Попередній огляд матеріалів диплому на кафедрі	20.05.2026	

Студент _____ Богдан ПЕРЕТЯТЬКО

(підпис)

Керівник проєкту _____ Ігор ТЕРЕЙКОВСЬКИЙ

(підпис)

АНОТАЦІЯ

Бакалаврський дипломний проєкт включає пояснювальну записку (77 стор., 7 рис., список використаної літератури з 12 найменувань, 4 додатки).

Об’єкт розробки – програмна система прогнозування енергоспоживання в інфраструктурі Розумного дому на основі аналізу часових рядів з використанням методів машинного навчання.

Розроблена система дозволяє: інтегруватися з платформою Home Assistant для отримання телеметрії з розумних розеток; накопичувати та зберігати історичні дані у базі даних часових рядів InfluxDB; моделювати різні профілі навантаження електроприладів; здійснювати високоточне прогнозування енергоспоживання на 24 години вперед. В процесі розробки були використані алгоритми машинного та глибокого навчання, зокрема адитивна статистична модель Prophet, ансамбль дерев рішень XGBoost та рекурентна нейронна мережа довгої короткострокової пам'яті (LSTM). Програмну частину реалізовано мовою програмування Python з використанням бібліотек TensorFlow/Keras. Для оцінки точності прогнозування застосовано метрики абсолютної (MAE) та середньоквадратичної (RMSE) похибок.

В ході розробки:

- проведено аналіз існуючих підходів до прогнозування часових рядів та систем збору телеметрії Розумного дому;

- сформульовані вимоги до інформаційної інфраструктури та конвеєра обробки даних (Data Pipeline);

- розроблена архітектура підсистеми збору та зберігання даних;

- розроблено програмне забезпечення для навчання, тестування (Train-Test Split) та порівняння моделей машинного навчання;

- проведено експериментальне тестування системи на історичних даних та виконано порівняльний аналіз ефективності обраних алгоритмів.

Ключові слова: ПРОГНОЗУВАННЯ ЕНЕРГОСПОЖИВАННЯ, МАШИННЕ НАВЧАННЯ, ЧАСОВІ РЯДИ, РОЗУМНИЙ ДІМ, HOME ASSISTANT, INFLUXDB, PYTHON, LSTM, XGBOOST, PROPHET.

ABSTRACT

The bachelor's qualification project includes an explanatory note (77 pages, 7 figures, a list of 12 references, and 4 appendices).

The object of development is a software system for energy consumption forecasting in a Smart Home infrastructure based on time series analysis using machine learning methods.

The developed system provides the following capabilities: integration with Home Assistant to collect telemetry data from smart plugs; accumulation and storage of historical data in the InfluxDB time-series database; modeling of various electrical appliance load profiles; and highly accurate forecasting of energy consumption 24 hours ahead. During the development process, machine learning and deep learning algorithms were used, including the additive statistical model Prophet , the decision-tree ensemble XGBoost , and the Long Short-Term Memory (LSTM) recurrent neural network. The software component was implemented in the Python programming language using the TensorFlow and Keras libraries. To evaluate forecasting accuracy, the Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) metrics were applied.

During the development process, the following tasks were completed:

- an analysis of existing approaches to time series forecasting and Smart Home telemetry collection systems was conducted;

- requirements for the information infrastructure and the data processing pipeline were formulated;

- the architecture of the data collection and storage subsystem was developed;

- software for training, testing (Train-Test Split), and comparing machine learning models was implemented;

- experimental testing of the system on historical data was carried out, and a comparative analysis of the effectiveness of the selected algorithms was performed.
Keywords: ENERGY CONSUMPTION FORECASTING, MACHINE LEARNING, TIME SERIES, SMART HOME, HOME ASSISTANT, INFLUXDB, PYTHON, LSTM, XGBOOST, PROPHET.

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045430.002 ТЗ	Комп'ютерна система	4	А4	
			прогнозування			
			енергоспоживання в			
			інфраструктурі			
			розумного дому			
			Технічне завдання			
	А4	ІАЛЦ.045430.003 ТП	Комп'ютерна система	1	А4	
			прогнозування			
			енергоспоживання в			
			інфраструктурі			
			розумного дому			
			Відомість технічного			
			проекту			
	А4	ІАЛЦ.045430.004 ПЗ	Комп'ютерна система	77	А4	
			прогнозування			
			енергоспоживання в			
			інфраструктурі			
			розумного дому			
			Пояснювальна записка			

Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.045430.001 ОА		
Розробив		Перетятько Б.В.			Комп'ютерна система прогнозування енергоспоживання в інфраструктурі розумного дому **Опис альбому**	Літ.	Аркуш 1 / Ар 2
Перевірив		Терейковський І.А.					
Консульт.						КПІ ім. Ігоря Сікорського, ФПСПМ, КВ-23	
Н. контроль		Клятченко Я.М.					
Зав. каф.		Романкевич В.О.					

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045430.005 Д1	Система прогнозування	1	А4	
			енергоспоживання.			
			Структурна схема			
	А4	ІАЛЦ.045430.006 Д2	Генерація синтетичної	1	А4	
			історії			
			Схема алгоритму			
	А4	ІАЛЦ.045430.007 Д3	Авторегресивний	1	А4	
			Схема алгоритму			
	А4	ІАЛЦ.045430.008 Д4	Класи модуля	1	А4	
			прогнозування.			
			Структурна схема			
		Диск CD-ROM	Текст пояснювальної			
			записки.			
			Графічний матеріал			

ЗМІСТ

					ІАЛЦ.045430.002 ТЗ		
Зм.	Арк.	№ докум.	Підп.	Дата			
Розроб.		Перетятько Б.В.			Комп'ютерна система прогнозування енергоспоживання в інфраструктурі розумного дому **Технічне завдання**	Літ.	Аркуш 1 / Аркушів 4
		Терейковський І.А.					
Н. контр.		Клятченко Я.М.				КПІ ім. Ігоря Сікорського, ФПСПМ, КВ-23	
Затв.		Романкевич В.О.					

1. НАЙМЕНУВАННЯ І ОБЛАСТЬ ЗАСТОСУВАННЯ

Найменування проєкту - «Система прогнозування енергоспоживання в інфраструктурі розумного дому».

Область застосування: інформаційні технології, системи розумного дому (Smart Home), Інтернет речей (IoT), предиктивна аналітика, системи підтримки прийняття рішень.

2. ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання першого (бакалаврського) рівня вищої освіти, затверджене кафедрою системного програмування і спеціалізованих комп'ютерних систем Національного технічного університету України "Київський Політехнічний Інститут імені Ігоря Сікорського".

3. ЦІЛЬ І ПРИЗНАЧЕННЯ РОБОТИ

Метою дипломного проєкту є розробка програмної системи для безперервного збору телеметрії, збереження часових рядів та високоточного прогнозування енергоспоживання побутових приладів на 24 години вперед із застосуванням статистичних, ансамблевих та нейромережевих алгоритмів машинного навчання.

4. ДЖЕРЕЛА РОБОТИ

Джерелами інформації для розроблення є технічна література, офіційна документація до платформ автоматизації та баз даних, наукові публікації з машинного навчання та аналізу часових рядів у періодичних виданнях, а також Інтернет-ресурси.

5. ТЕХНІЧНІ ВИМОГИ

5.1 Вимоги до модулю програмного забезпечення, що розробляється

- підтримка імітаційної генерації історичних даних енергоспоживання;
- можливість інтеграції та передачі потокових даних до бази даних часових рядів InfluxDB;
- підтримка попередньої обробки часових рядів та конструювання ознак;
- можливість навчання, налаштування та валідації моделей машинного навчання;
- автоматичний розрахунок метрик похибки (MAE, RMSE) та візуалізація результатів прогнозування на графіках.

5.2 Рекомендовані вимоги до апаратного забезпечення

- процесор: Intel Core i5 / AMD Ryzen 5 або подібний;
- оперативна пам'ять: від 8 Гб (рекомендовано 16 Гб для ефективного навчання нейронних мереж);
- простір на диску: від 10 Гб;
- серверний вузол (для локального розгортання інфраструктури IoT): Raspberry Pi 4 або локальний персональний комп'ютер.

5.3 Вимоги до мінімального програмного забезпечення

- операційна система: macOS, Linux або Windows;
- середовище виконання: інтерпретатор Python версії 3.9 або вище;
- розгорнута база даних InfluxDB;
- розгорнута платформа автоматизації Home Assistant;
- встановлений набір спеціалізованих бібліотек Python.

6. ЕТАПИ РОЗРОБКИ

№ з/п	Назва етапів виконання дипломного проєкту	Терміни виконання етапів проєкту	Примітка
1.	Вивчення літератури за тематикою проєкту	13.11.2025	
2.	Розроблення та узгодження технічного завдання	17.11.2025	
3.	Аналіз існуючих рішень	03.12.2025	
4.	Підготовка матеріалів першого розділу дипломного проєкту	13.01.2026	
5.	Підготовка матеріалів другого розділу дипломного проєкту	20.01.2026	
6.	Підготовка графічної частини дипломного проєкту	09.02.2026	
7.	Оформлення документації дипломного проєкту	15.05.2026	
8.	Попередній огляд матеріалів диплому на кафедрі	20.05.2026	

ВІДОМІСТЬ ТЕХНІЧНОГО ПРОЄКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4	ІАЛЦ.045430.000	Завдання на дипломний проєкт	2	
2	А4	ІАЛЦ.045430.001 ОА	Опис альбому	2	
3	А4	ІАЛЦ.045430.002 ТЗ	Технічне завдання	4	
4	А4	ІАЛЦ.045430.003 ТП	Відомість технічного проєкту	1	
5	А4	ІАЛЦ.045430.004 ПЗ	Пояснювальна записка	77	
6	А4	ІАЛЦ.045430.005 Д1	Система прогнозування енергоспоживання. Схема структурна	1	
7	А4	ІАЛЦ.045430.006 Д2	Генерація синтетичної історії електроспоживання. Схема алгоритму	1	
8	А4	ІАЛЦ.045430.007 Д3	Авторегресивний прогноз. Схема алгоритму	1	
9	А4	ІАЛЦ.045430.008 Д4	Класи модуля прогнозування. Схема структурна	1	

				ІАЛЦ.045430.003 ТП		
	ПІБ	Підп.	Дата			
Розробн.	Перетятько Б.В.			Комп'ютерна система прогнозування енергоспоживання в інфраструктурі розумного дому **Відомість технічного проєкту**	Лист	Листів
Керівн.	Терейковський І.А.				1	1
Консульт.					КПІ ім. Ігоря Сікорського Каф. СП і СКС Гр. КВ-23	
Н/контр.	Клятченко Я.М.					
Зав.каф.	Романкевич В.О.					

Пояснювальна записка до дипломного проєкту

на тему: Система прогнозування енергоспоживання в інфраструктурі розумного дому

Київ – 2026

ЗМІСТ

ДОДАТКИ

Додаток 1. Копії графічних матеріалів

- ІАЛЦ.045430.005 Д1. Система прогнозування енергоспоживання. Структурна схема
- ІАЛЦ. 045430.006 Д2. Генерація синтетичної історії електроспоживання. Схема алгоритму
- ІАЛЦ. 045430.007 Д3. Авторегресивний прогноз. Схема алгоритму
- ІАЛЦ. 045430.008 Д4. Класи модуля прогнозування. Структурна схема

Додаток 2. Фрагменти програмного коду

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

СКБД – Система Керування Базами Даних

API – Application Programming Interface

ARIMA – AutoRegressive Integrated Moving Average

CLI – Command-Line Interface

CPU – Central Processing Unit

CSV – Comma-Separated Values

GPU – Graphics Processing Unit

GRU – Gated Recurrent Unit

GUI – Graphical User Interface

HA – Home Assistant

HTTP – HyperText Transfer Protocol

IoT – Internet of Things

LSTM – Long Short-Term Memory

MAE – Mean Absolute Error

MAP – Maximum A Posteriori

MQTT – Message Queuing Telemetry Transport

MSE – Mean Squared Error

NILM – Non-Intrusive Load Monitoring

RMSE – Root Mean Squared Error

RNN – Recurrent Neural Network

SARIMA – Seasonal AutoRegressive Integrated Moving Average

TCN – Temporal Convolutional Network

TSM – Time-Structured Merge Tree

UTC – Coordinated Universal Time

XGBoost – Extreme Gradient Boosting

ML – Machine Learning

ВСТУП

Сучасний етап розвитку житлово-комунального сектору та побутової автоматизації характеризується активним впровадженням концепції «Розумного дому» та інтелектуальних енергетичних мереж. Одним із найбільш критичних викликів для сучасної інженерії є підвищення енергоефективності, оптимізація споживання ресурсів та зниження пікових навантажень на локальні й магістральні електромережі. Побутові споживачі є однією з основних груп, що формують нерівномірність навантаження в енергосистемі, що зумовлено поведінковими патернами користувачів та одночасним увімкненням потужних приладів.

Традиційні підходи до управління енергоспоживанням у системах автоматизації (наприклад, на базі поширеної open-source платформи Home Assistant) переважно обмежуються реактивним моніторингом поточних показників телеметрії та відображенням історичних даних за допомогою спеціалізованих баз даних часових рядів. Проте для переходу до предиктивного, проактивного управління - автоматичного зміщення графіків роботи приладів у часові зони з мінімальними тарифами або низьким загальним навантаженням - необхідна наявність точного прогнозу енергоспоживання на короткострокову перспективу.

Вирішення цієї задачі ускладнюється нелінійною природою коливань навантаження, що поєднує в собі постійне базове споживання, циклічні процеси та виражені інтервальні піки. Застосування класичних лінійних статистичних методів є недостатнім для моделювання таких комплексних процесів. Це зумовлює необхідність дослідження та впровадження сучасних методів машинного навчання, зокрема адитивних статистичних моделей, ансамблів дерев рішень із гнучким конструюванням ознак та глибоких рекурентних нейронних мереж . Таким чином, розробка системи предиктивного аналізу роботи розумних IoT-пристроїв на основі ML-алгоритмів є актуальним науково-практичним завданням.

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1.1 Аналіз науково-практичних рішень у галузі прогнозування енергоспоживання розумного дому

Прогнозування електроспоживання на рівні окремого домогосподарства - відносно молода, але активно розвивана галузь прикладного машинного навчання. Вона сформувалася на перетині кількох технологічних напрямів: масового поширення інтелектуальних лічильників та IoT-датчиків, розвитку платформ домашньої автоматизації, а також зростання доступності обчислювальних потужностей, достатніх для роботи сучасних алгоритмів навчання на побутових пристроях. Тривалий час задача прогнозування електричного навантаження розглядалася лише у контексті великих енергосистем та промислових мереж; натомість упродовж останнього десятиліття дослідницький інтерес поступово зміщується до мікрорівневого аналізу - окремих будівель, квартир і навіть окремих побутових приладів.

Основними практичними завданнями, для яких застосовують прогнозування побутового енергоспоживання, є: оптимізація тарифних планів з диференційованим погодинним обліком (Time-of-Use); управління автономними джерелами живлення на базі сонячних панелей та акумуляторних систем; виявлення аномальних режимів роботи побутових приладів; формування рекомендацій користувачам щодо ефективного використання електроенергії. Для всіх цих сценаріїв критичним є саме короткостроковий прогноз - на горизонт від декількох годин до однієї доби. Саме у цьому часовому проміжку приймаються основні рішення з управління навантаженням.

Серед готових комерційних рішень, представлених на ринку, можна виділити кілька характерних категорій. Першу складають вбудовані аналітичні модулі систем розумного обліку від великих електропостачальних компаній: вони, як правило, надають користувачу базовий статистичний аналіз минулого споживання та найпростіший лінійний прогноз без використання сучасних алгоритмів машинного навчання. Друга категорія - спеціалізовані хмарні сервіси

на кшталт Sense, Smappee та Bidgely, які поєднують власне обладнання моніторингу з пропрієтарними хмарними алгоритмами розпізнавання роботи окремих приладів та прогнозування споживання. Їх загальним недоліком з точки зору приватного користувача є закритість алгоритмів, необхідність передачі телеметрії на сторонні сервери та підписковий характер обслуговування. Третя категорія - рішення на базі відкритих платформ домашньої автоматизації, передусім Home Assistant, які, попри велику популярність самої платформи, історично не мали повноцінних вбудованих засобів прогностичної аналітики і зазвичай обмежувалися візуалізацією минулого споживання.

В академічних дослідженнях, представлених у профільних виданнях останніх років, простежуються три магістральні напрями розробки методів короткострокового прогнозування побутового енергоспоживання. Перший напрям - застосування класичних статистичних моделей сімейства ARIMA та їх сезонних модифікацій SARIMA, які забезпечують високу інтерпретованість результатів, проте потребують ретельного попереднього аналізу стаціонарності ряду та погано адаптуються до нелінійних залежностей. Другий - використання ансамблевих методів машинного навчання, зокрема Random Forest, Gradient Boosting та XGBoost, що демонструють високу точність при відносно невеликому обсязі навчальної вибірки та простоті налаштування гіперпараметрів. Третій напрям - застосування глибоких нейронних мереж, у першу чергу рекурентних архітектур LSTM та GRU, а також згорткових мереж типу Temporal Convolutional Network (TCN), які теоретично здатні моделювати складні нелінійні часові залежності, проте вимагають значно більших обсягів даних та обчислювальних ресурсів.

Окрему практичну нішу займає бібліотека Prophet, розроблена дослідницькою групою Meta, яка хоч і не є новаторською з точки зору алгоритмічної основи, проте суттєво знизила поріг входу у задачі прогнозування часових рядів для широкого кола розробників завдяки автоматичному виявленню сезонностей та відсутності необхідності у конструюванні ознак. Її поява стимулювала проведення низки порівняльних досліджень з більш

складними методами на типових прикладних задачах, де Prophet часто виступає у ролі базової лінії (baseline) для оцінки ефективності складніших алгоритмів.

Огляд галузі виявляє декілька прогалин, які роблять актуальною розробку нових прогностичних рішень для побутового сектору. По-перше, комерційні продукти переважно є закритими та прив'язаними до фірмового обладнання виробника. Це обмежує гнучкість розширення системи й її інтеграцію з тією інфраструктурою, що вже є у користувача. По-друге, готові пакети для Home Assistant зосереджені переважно на візуалізації минулого споживання та налаштуванні автоматизацій; порівняльний аналіз кількох прогностичних методів на одному наборі даних зустрічається у них вкрай рідко. По-третє, академічні публікації, як правило, фокусуються на одному обраному алгоритмі та використовують узагальнені публічні набори даних, через що складно оцінити реальну застосовність методу до конкретного домогосподарства з його унікальним профілем споживання.

Окреслені прогалини обґрунтовують доцільність розробки відкритої програмної системи, що інтегрується з популярною платформою домашньої автоматизації Home Assistant, працює локально без передачі даних у хмару та дає змогу прямо порівняти три принципово різні алгоритми машинного навчання - статистичну модель Prophet, ансамблевий метод XGBoost та рекурентну нейронну мережу LSTM - на одній і тій самій вибірці реальної телеметрії. Саме такий підхід покладено в основу дипломного проєкту; технічні аспекти його реалізації розглянуто у наступних розділах роботи.

1.2 Характеристики платформи автоматизації Home Assistant

Home Assistant є відкритою платформою домашньої автоматизації з вихідним кодом, що поширюється за ліцензією Apache 2.0. Сьогодні платформа займає лідируючі позиції серед некомерційних рішень у сегменті розумного дому. Проєкт ініціював у 2013 році розробник Пол Шравен (Paulus Schoutsen) як невелику Python-бібліотеку для управління кількома пристроями у власному помешканні. За роки розвитку вона перетворилася на повноцінну екосистему з

тисячами підтримуваних інтеграцій та активною спільнотою користувачів і контриб'юторів. Архітектурно Home Assistant позиціонується як локальне рішення з пріоритетом приватності даних: уся обробка телеметрії, виконання сценаріїв автоматизації та збереження історії відбувається безпосередньо на пристрої користувача, без обов'язкової передачі інформації у хмарні сервіси сторонніх виробників.

Ядро платформи реалізоване мовою Python з використанням асинхронної моделі виконання на базі бібліотеки asyncio. Це забезпечує високу продуктивність при одночасній взаємодії з великою кількістю фізичних пристроїв та програмних сервісів. Концептуально архітектура Home Assistant побудована навколо чотирьох базових абстракцій: сутностей (entities), які представляють окремі джерела стану (датчики, лампи, перемикачі, медіапрогравачі тощо); пристроїв (devices), що групують пов'язані сутності одного фізичного об'єкта; інтеграцій (integrations), які реалізують протоколи зв'язку з конкретними виробниками обладнання; та автоматизацій (automations), тобто декларативно описаних правил, що пов'язують події у системі з відповідними діями. Така багаторівнева модель дозволяє абстрагуватися від технічних деталей конкретних протоколів та працювати з різнорідним обладнанням за уніфікованим інтерфейсом.

Однією з ключових переваг платформи є винятково широка підтримка протоколів та виробників обладнання: на момент написання роботи кількість офіційних інтеграцій перевищує 2700, що покриває практично всі поширені стандарти у сфері IoT - Zigbee, Z-Wave, Matter, Thread, Bluetooth Low Energy, Modbus, KNX, MQTT[8], а також пропрієтарні API провідних виробників розумних пристроїв. Окремо слід відзначити підтримку протоколу MQTT, який де-факто став стандартним механізмом обміну телеметрією у системах IoT та використовується значною частиною бюджетних розумних пристроїв на базі мікроконтролерів ESP8266 та ESP32. У контексті розробленої системи саме MQTT забезпечує отримання погодинних показників активної потужності від розумної розетки з вбудованим вимірювачем.

Уся історія станів сутностей у Home Assistant зберігається у внутрішній реляційній базі даних, що за замовчуванням реалізована на SQLite, однак може бути замінена на PostgreSQL або MySQL для високонавантажених інсталяцій. Цей механізм є зручним для базової візуалізації минулих значень, проте має суттєві обмеження при роботі зі значним обсягом телеметрії високої роздільної здатності: реляційні бази даних не оптимізовані для зберігання та агрегації часових рядів, що призводить до зростання обсягу файлу бази даних та погіршення продуктивності запитів за тривалий період. Для розв'язання цієї проблеми у Home Assistant передбачено офіційну інтеграцію з InfluxDB - спеціалізованою СКБД часових рядів, до якої телеметрія дублюється у режимі реального часу та з якої надалі може зчитуватися зовнішніми аналітичними інструментами.

Розширюваність платформи реалізується через систему користувацьких компонентів, що дозволяє розробникам створювати власні інтеграції на Python без необхідності модифікації основного коду. Окремо існує неофіційний репозиторій HACS (Home Assistant Community Store), через який спільнота поширює тисячі сторонніх інтеграцій, тем оформлення та користувацьких карток для веб-інтерфейсу. Така відкритість архітектури принципово відрізняє Home Assistant від замкнених комерційних рішень: будь-який користувач має технічну можливість реалізувати власну логіку обробки даних, інтегрувати платформу зі сторонніми сервісами або підключити обладнання, для якого офіційна підтримка відсутня.

Засоби візуалізації Home Assistant побудовано на базі веб-інтерфейсу Lovelace. Цей інтерфейс дозволяє гнучко налаштовувати дашборди з готового набору карток для відображення поточних значень сутностей, історичних графіків, переключення сценаріїв та керування пристроями. Окрім візуалізації, платформа надає механізм автоматизацій - він дозволяє створювати правила виду «якщо настала певна подія - виконати визначену дію». Такі правила охоплюють широкий спектр практичних сценаріїв: вмикання освітлення на заході сонця, повідомлення користувачу про аномальний стан датчика,

виконання послідовностей дій за розкладом тощо. Однак вбудовані засоби аналітики обмежуються переважно ретроспективним аналізом - відображенням минулих значень - і не містять повноцінних механізмів прогностичного моделювання. Це залишає простір для розробки додаткових компонентів машинного навчання.

Можливість інтеграції з InfluxDB разом із відкритістю архітектури платформи стали ключовими аргументами на користь вибору Home Assistant як базового програмного середовища для розробленої системи. Платформа виконує роль джерела телеметрії: приймає погодинні показники від розумної розетки та постійно дублює їх у спеціалізовану базу часових рядів. З цієї бази дані надалі завантажує аналітичне ядро системи для навчання моделей та формування прогнозів. Такий поділ відповідальностей - Home Assistant як шлюз збору даних, а власна підсистема як аналітичне ядро - дозволяє повністю використати сильні сторони платформи без необхідності реалізовувати власні драйвери обладнання. Водночас зберігається повний контроль над логікою прогностичного моделювання.

1.3 Аналіз методів прогнозування часових рядів енергоспоживання

Прогнозування часових рядів електроспоживання є класичною задачею прикладної статистики та машинного навчання, для якої за останні десятиліття було розроблено та апробовано широкий спектр методів - від простих ковзних середніх до глибоких нейронних мереж з мільйонами параметрів. Вибір конкретного методу для практичної реалізації визначається кількома взаємопов'язаними чинниками: обсягом доступної історичної вибірки, характером сезонних патернів у даних, горизонтом прогнозу, вимогами до інтерпретованості результатів та доступними обчислювальними ресурсами. У контексті побутового енергоспоживання, де часові ряди, як правило, мають виражену добову та тижневу сезонність, помірний обсяг доступних даних та потребують короткострокового прогнозу на 24 години, найбільш доцільним є

застосування методів, які поєднують здатність до моделювання періодичних патернів з обчислювальною ефективністю.

Усі сучасні методи прогнозування часових рядів можна умовно поділити на три великі класи відповідно до їх алгоритмічної природи: класичні статистичні моделі, методи машинного навчання на базі ансамблів дерев рішень та методи глибокого навчання на основі нейронних мереж. Кожен з цих класів має свої характерні переваги та обмеження, які доцільно розглянути окремо для обґрунтованого вибору підходів, що будуть реалізовані у межах розробленої системи.

Класичні статистичні моделі представлені передусім сімейством ARIMA (AutoRegressive Integrated Moving Average) та його сезонними модифікаціями SARIMA та SARIMAX, а також моделями експоненціального згладжування Холта-Вінтерса. Узагальнена форма моделі ARIMA(p, d, q) для попередньо інтегрованого ряду описується співвідношенням:

$$y_t = c + \sum_{i=1}^{p} \varphi_i y_{t-i} + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j} + \varepsilon_t$$

де y_t - значення ряду в момент часу t, φ_i - коефіцієнти авторегресії порядку p, θ_j - коефіцієнти ковзного середнього порядку q, ε_t - стохастична похибка. Теоретичною основою цих методів є припущення про стаціонарність ряду - стабільність статистичних характеристик у часі - та лінійний характер залежностей між послідовними значеннями. Ці методи мають низку важливих переваг: висока інтерпретованість результатів, наявність формальних статистичних тестів для перевірки якості моделі, помірні обчислювальні вимоги та усталена теоретична база. Проте їх практичне застосування потребує ретельного попереднього аналізу ряду - перевірки стаціонарності, ідентифікації порядків авторегресії та інтегрування, діагностики залишків. Крім того, лінійна природа цих моделей принципово обмежує їх здатність до апроксимації складних нелінійних залежностей, що є суттєвим недоліком при роботі з реальною

побутовою телеметрією, де поведінка користувачів та режими роботи приладів зумовлюють виражено нелінійні патерни.

Окрему практичну нішу у класі статистичних методів займає бібліотека Prophet, розроблена дослідницькою групою Meta. Її алгоритмічну основу складає узагальнена адитивна модель, що розкладає часовий ряд на компоненти тренду, сезонностей різних періодів та аномальних подій:

$$y(t) = g(t) + s(t) + h(t) + \varepsilon_t$$

де g(t) - функція тренду, s(t) - періодична сезонна складова, h(t) - вплив нерегулярних подій (свят, аномалій), ε_t - стохастична похибка. На відміну від ARIMA, Prophet не вимагає попереднього приведення ряду до стаціонарного вигляду та автоматично виявляє сезонні закономірності за допомогою рядів Фур'є. Такий підхід суттєво знижує поріг входу у задачу прогнозування для прикладних розробників та робить Prophet природним вибором у ролі базової лінії (baseline) для порівняння з більш складними алгоритмами[1].

Другий клас - методи машинного навчання на основі ансамблів дерев рішень - представлений алгоритмами Random Forest та сімейством градієнтного бустингу (Gradient Boosting Machines, GBM), серед яких найбільш популярними реалізаціями є XGBoost, LightGBM та CatBoost. Концептуально ці методи поєднують прогнози великої кількості простих базових моделей (неглибоких дерев рішень) у єдиний ансамбль, при цьому процедура побудови дерев організована таким чином, щоб кожне нове дерево виправляло помилки попередніх. Прогноз ансамблю формується як зважена сума прогнозів окремих базових моделей:

$$\widehat{y_i} = \sum_{k=1}^{K} f_k(x_i), \quad f_k \in \mathcal{F}$$

де K - кількість дерев в ансамблі, f_k - окреме регресійне дерево, що відображає вектор ознак x_i у скалярне прогнозне значення, F - простір можливих регресійних дерев заданої структури. Для застосування цих методів до

задач прогнозування часових рядів необхідне попереднє конструювання матриці ознак - перетворення послідовності у табличну форму з використанням лагових значень та закодованих часових змінних. Ансамблеві методи демонструють високу точність при відносно невеликому обсязі навчальної вибірки, нечутливі до масштабу ознак, стійкі до мультиколінеарності та надають вбудовані механізми оцінки важливості окремих предикторів. Серед недоліків - необхідність авторегресивного формування багатокрокових прогнозів та обмежена здатність до моделювання дуже довгих часових залежностей.

Третій клас - методи глибокого навчання - об'єднує сімейство нейромережевих архітектур, спеціально пристосованих до роботи з послідовностями. Найбільш широко застосовуваними у задачах прогнозування часових рядів є рекурентні мережі типу LSTM (Long Short-Term Memory) та GRU (Gated Recurrent Unit), які завдяки внутрішнім механізмам керованої пам'яті здатні моделювати довгострокові залежності у даних. Окрему групу складають одновимірні згорткові мережі та архітектури Temporal Convolutional Network (TCN), а останніми роками активно досліджуються трансформерні моделі (Temporal Fusion Transformer, Informer) для задач прогнозування. Теоретичною перевагою глибоких моделей є здатність до автоматичного виявлення складних нелінійних патернів без ручного конструювання ознак, проте на практиці ця перевага реалізується лише за умови достатньо великого обсягу навчальних даних - типово від десятків тисяч відліків та більше. У випадках обмеженої вибірки нейромережеві моделі схильні до перенавчання та можуть поступатися за точністю значно простішим алгоритмам.

Окремо слід зазначити сучасну тенденцію до використання гібридних підходів, у яких комбінуються методи різних класів - наприклад, статистична модель використовується для моделювання тренду та сезонностей, а нейронна мережа - для прогнозування залишків. Такі підходи у деяких випадках демонструють вищу точність, проте їх реалізація суттєво ускладнює архітектуру системи та підвищує вимоги до кваліфікації розробника й обчислювальних ресурсів, що не завжди є виправданим у прикладних задачах.

Порівняння перерахованих класів методів дає змогу сформулювати критерії доцільності їх застосування у задачі короткострокового прогнозування[2] побутового енергоспоживання. Для адекватної оцінки ефективності алгоритмів треба одночасно дослідити їх роботу на одній і тій самій вибірці. Це дозволяє виявити переваги та обмеження кожного підходу у конкретних умовах задачі. З цих міркувань для реалізації у складі розробленої системи обрано трьох представників, кожен з яких репрезентує окремий клас методів. Перший - модель Prophet, що належить до адитивних статистичних моделей з простим інтерфейсом та автоматичним виявленням сезонностей. Другий - алгоритм XGBoost[3], високопродуктивний представник ансамблевих методів з можливістю гнучкого конструювання ознак. Третій - рекурентна нейронна мережа LSTM[4], представник сучасних методів глибокого навчання, теоретично оптимальних для моделювання часових залежностей.

Такий вибір забезпечує систематичне порівняння трьох різних алгоритмічних парадигм на ідентичних вхідних даних. Це дає змогу зробити обґрунтовані висновки щодо їх практичної ефективності у задачі прогнозування побутового енергоспоживання на горизонт 24 годин. Детальні теоретичні засади кожного з обраних методів, включно з математичним апаратом та особливостями реалізації, розглядаються у підрозділі 2.1 наступного розділу роботи.

1.4 Формулювання задач дипломної роботи

Виконаний у попередніх підрозділах аналіз існуючих рішень, характеристик платформи Home Assistant[5] та методів прогнозування часових рядів дозволяє чітко сформулювати мету, об'єкт, предмет та конкретні задачі цієї дипломної роботи.

Метою дипломної роботи є розробка програмної системи короткострокового прогнозування електроспоживання в інфраструктурі розумного дому, побудованої на базі відкритої платформи Home Assistant та бази даних часових рядів InfluxDB, з паралельною реалізацією та порівнянням трьох

принципово різних алгоритмів машинного навчання - Prophet, XGBoost та LSTM - на горизонт прогнозу 24 години.

Об'єктом дослідження є процес короткострокового прогнозування погодинного електроспоживання окремого побутового приладу, підключеного до інтелектуальної розетки з вбудованим вимірювачем активної потужності у складі інфраструктури розумного дому.

Предметом дослідження є методи, алгоритми та програмні засоби побудови прогностичних моделей для часових рядів електроспоживання, а також архітектурні рішення, що забезпечують їх інтеграцію з існуючими платформами домашньої автоматизації.

Для досягнення поставленої мети у роботі необхідно розв'язати такі основні задачі:

1. Виконати аналіз існуючих науково-практичних рішень у галузі прогнозування побутового енергоспоживання, дослідити архітектуру та можливості платформи Home Assistant, а також провести порівняльний огляд сучасних методів прогнозування часових рядів з обґрунтуванням вибору трьох представників різних алгоритмічних парадигм.

2. Розробити загальну архітектуру програмної системи з чітким розподілом функціональних обов'язків між підсистемами збору даних, їх попередньої обробки, машинного навчання, оцінки точності та візуалізації результатів.

3. Реалізувати підсистему генерації синтетичної річної історії погодинного електроспоживання, що математично відтворює реалістичні поведінкові патерни побутового домогосподарства - постійне фонове навантаження, циклічну роботу холодильного компресора та пікові навантаження побутового водонагрівача.

4. Реалізувати підсистему збереження та довгострокового зберігання телеметрії в базі даних часових рядів InfluxDB з організацією механізму щоденного дописування нових точок даних, що імітує реальну роботу IoT-датчика.

5. Розробити підсистему попередньої обробки даних з реалізацією конструювання часових ознак (циклічне кодування години та дня тижня), формування лагових змінних та нормалізації числових значень для подальшого використання трьома прогностичними моделями.

6. Реалізувати три прогностичні моделі - Prophet, XGBoost та LSTM - у єдиній архітектурі з абстрактним базовим класом, що забезпечує уніфікований програмний інтерфейс навчання, формування прогнозу та оцінки точності.

7. Реалізувати підсистему оцінки точності прогнозування за процедурою Train-Test Split з обчисленням метрик MAE та RMSE та автоматичним вибором найточнішої моделі для подальшого використання у режимі автоматичного прогнозу.

8. Розробити користувацькі інтерфейси системи - командний інтерфейс (CLI) на базі бібліотеки argparse для запуску у режимах окремих моделей, екзаменаційного тестування та автоматичного прогнозу, а також графічний інтерфейс (GUI) на базі бібліотеки tkinter з фоновим виконанням обчислень засобами багатопоточності.

9. Провести експериментальні дослідження розробленої системи на синтетичній річній вибірці погодинного електроспоживання, виконати порівняльний аналіз точності трьох прогностичних моделей за метриками MAE та RMSE та обґрунтувати висновок щодо найбільш ефективного підходу у досліджуваній задачі.

10. Скласти керівництво користувача з покроковим описом процедур розгортання середовища Home Assistant та InfluxDB, генерації первинної історичної вибірки, навчання моделей та отримання прогнозу через CLI та GUI інтерфейси.

Очікуваним практичним результатом виконання роботи є повнофункціональна модульна програмна система мовою Python, що інтегрується з платформою Home Assistant через базу даних InfluxDB, забезпечує паралельне навчання трьох прогностичних моделей та надає кінцевому

користувачу обґрунтований прогноз погодинного електроспоживання на наступні 24 години через зручні CLI та GUI інтерфейси.

2. РОЗРОБКА СИСТЕМИ ПРОГНОЗУВАННЯ ЕНЕРГОСПОЖИВАННЯ

Розробка програмної системи прогнозування електроспоживання є комплексною інженерною задачею, що поєднує елементи теорії часових рядів, об'єктно-орієнтованого проектування та практичної інтеграції з зовнішніми компонентами інфраструктури розумного дому. У межах цього розділу послідовно розкриваються теоретичні та архітектурні засади розробленої системи: від математичних моделей, покладених в основу прогностичного ядра, до програмної організації коду, який реалізує заплановану функціональність.

Виклад побудовано таким чином, щоб поступово рухатись від абстрактного до конкретного. Підрозділ 2.1 присвячено теоретичним основам обраних прогностичних моделей; у ньому наведено ключові математичні співвідношення, що описують адитивну статистичну модель Prophet, ансамблевий метод XGBoost та рекурентну нейронну мережу LSTM. Підрозділ 2.2 описує загальну архітектуру системи на рівні функціональних шарів, від джерела телеметрії до користувацьких інтерфейсів, та обґрунтовує обрані технологічні рішення. Підрозділ 2.3 деталізує програмну структуру системи на рівні модулів, класів та шаблонів проектування, що забезпечують її модульність, розширюваність та підтримуваність.

Результатом викладеного у цьому розділі матеріалу є цілісна архітектурна концепція розробленої системи, готова до подальшої детальної програмної реалізації, опис якої наведено у наступному розділі.

2.1 Теоретичні засади побудови системи прогнозування

Перш ніж переходити до архітектурних та реалізаційних аспектів розробленої системи, доцільно систематизувати математичні основи задачі прогнозування часових рядів та розглянути теоретичний апарат трьох прогностичних моделей, обраних у підрозділі 1.3. Такий виклад дозволяє в подальших підрозділах зосередитися виключно на інженерних аспектах реалізації, не повертаючись до математичних обґрунтувань.

Часовим рядом називається упорядкована послідовність числових спостережень y_1, y_2, …, y_T, отриманих через рівні проміжки часу. У задачі прогнозування електроспоживання кожне значення y_t відповідає сумарній активній потужності, виміряній у ваттах у момент часу t, а крок дискретизації становить одну годину. Класичним аналітичним інструментом дослідження таких послідовностей є їх адитивне розкладання на три структурні компоненти за виразом:

$$y_t = T_t + S_t + \varepsilon_t \tag{2.1}$$

де y_t - спостережуване значення; T_t - повільно змінний детермінований тренд, що відображає довгострокову динаміку процесу; S_t - періодична сезонна складова, обумовлена циклічними патернами поведінки; ε_t - стохастична залишкова складова, що описує випадкову флуктуацію навколо детермінованої частини. У контексті побутового енергоспоживання тренд як правило слабо виражений на короткострокових горизонтах, тоді як сезонна компонента є домінуючою: вона формується суперпозицією добового циклу (ранкові та вечірні піки споживання) та тижневого циклу (потенційна різниця між робочими днями та вихідними). Залишкова складова відображає випадковий вплив поведінкових чинників - нерегулярне користування приладами, стохастичні цикли роботи холодильного компресора тощо. Принциповою особливістю задачі прогнозування часових рядів, що відрізняє її від класичних задач регресії, є наявність часової залежності між послідовними спостереженнями, яка накладає специфічні вимоги на процедури навчання та валідації моделей: випадкове розділення вибірки на тренувальну та тестову частини є недопустимим, а замість нього застосовується виключно хронологічне розділення зі збереженням природного порядку відліків. Окремою методологічною особливістю є необхідність оцінки точності моделей за метриками, що враховують абсолютну величину похибок прогнозу, оскільки у прикладних задачах енергетики кінцевого користувача цікавлять не статистичні характеристики, а саме розмір потенційної помилки у фізично інтерпретованих одиницях (ватах). Конкретні

метрики, які використовуються для оцінки розробленої системи, розглядаються у підрозділі 3.3.

Першою з реалізованих у системі моделей є Prophet - узагальнена адитивна модель, розроблена дослідницькою групою Meta. Її алгоритмічну основу складає розкладання часового ряду на компоненти тренду, сезонностей та аномалій, формальний вигляд якого було наведено у формулі (1.2). Принциповою відмінністю Prophet від класичних авторегресивних моделей є те, що часова мітка t розглядається як незалежна змінна. Принциповою відмінністю Prophet від класичних авторегресивних моделей є те, що часова мітка t розглядається як незалежна змінна, а не як індекс послідовності, що дозволяє коректно обробляти ряди з пропусками та нерівномірним кроком дискретизації.

Функція тренду g(t) реалізується у вигляді набору лінійних ділянок, з'єднаних у точках зламу (changepoints), що автоматично визначаються алгоритмом за даними. Такий підхід дозволяє моделі адаптуватися до поступових змін у динаміці процесу без необхідності їх ручної ідентифікації. Сезонна компонента s(t) моделюється за допомогою скороченого ряду Фур'є з заданим періодом P:

$$s(t) = \sum_{n=1}^{N} \left(a_n \cos\left(\frac{2\pi n t}{P}\right) + b_n \sin\left(\frac{2\pi n t}{P}\right) \right) \qquad (2.2)$$

де N - порядок розкладання (за замовчуванням 10 для добової та 3 для тижневої сезонностей); P - період сезонності у тих же одиницях, що і t (24 для добового циклу, 168 для тижневого, 8760 для річного); a_n та b_n - невідомі коефіцієнти, що оцінюються разом з рештою параметрів моделі. Використання саме ряду Фур'є замість прямого моделювання конкретних годин дозволяє моделі гнучко апроксимувати довільну періодичну форму сезонної кривої без перепідбору структури моделі під конкретний набір даних. Збільшення порядку розкладання N підвищує здатність моделі відтворювати тонкі особливості сезонного профілю, проте водночас підвищує ризик перенавчання на шум, що зумовлює необхідність обережного підбору цього гіперпараметра.

Оцінка всіх параметрів моделі - параметрів тренду, сезонних коефіцієнтів та параметрів аномалій - здійснюється одночасно за критерієм максимуму правдоподібності, що звільняє розробника від необхідності їх послідовного підбору. Ключовою практичною перевагою Prophet є відсутність вимоги до стаціонарності ряду та автоматичне виявлення сезонностей за даними, що робить її природним вибором у ролі базової лінії для порівняння з більш складними методами. Серед обмежень моделі - обмежена здатність до моделювання короткострокової інерції процесу: оскільки Prophet не використовує лагових ознак, а спирається виключно на функції часу, він не може ефективно враховувати залежність поточного споживання від значень безпосередньо попередніх годин.

Другою реалізованою моделлю є XGBoost (Extreme Gradient Boosting) - високопродуктивна реалізація алгоритму градієнтного бустингу дерев рішень, розроблена Тяньци Ченом у 2014 році. Концептуальною основою методу є послідовна побудова ансамблю слабких базових моделей - неглибоких регресійних дерев, кожне з яких навчається коригувати залишкові похибки попередніх. Підсумковий прогноз ансамблю формується як адитивна сума прогнозів окремих дерев, формальний вигляд якої було наведено у формулі (1.3). Регресійне дерево розбиває простір ознак на скінченну кількість непересічних областей (листків), у межах кожної з яких прогнозне значення є константою; саме така структура надає алгоритму здатність моделювати складні нелінійні залежності між предикторами та цільовою змінною.

Процедура побудови ансамблю є ітеративною: на кожному кроці k до існуючого ансамблю додається нове дерево f_k, яке мінімізує регуляризовану цільову функцію виду:

$$\mathcal{L}^{(k)} = \sum_{i=1}^{n} l\left(y_i, \widehat{y_i^{(k-1)}} + f_k(x_i)\right) + \Omega(f_k) \qquad (2.3)$$

де $l(\cdot, \cdot)$ функція втрат, яка для задач регресії приймає квадратичну форму; $\hat{y}_i^{(k-1)}$ - прогноз ансамблю на попередній ітерації; $\Omega(f_k)$ - регуляризаційний штраф за складністю нового дерева, що включає кількість листків та L2-норму їх

значень. Безпосередня мінімізація цієї функції є обчислювально складною задачею, тому в XGBoost застосовується апроксимація другого порядку за рядом Тейлора, що зводить задачу побудови оптимального дерева до серії аналітично розв'язуваних підзадач. Завдяки цьому процедура навчання XGBoost є на кілька порядків швидшою за класичні реалізації градієнтного бустингу, що особливо важливо при роботі з вибірками великого обсягу.

Принциповою особливістю XGBoost у контексті задач прогнозування часових рядів є відсутність вбудованого механізму роботи з послідовностями: алгоритм оперує виключно табличними даними, тому його застосування потребує попереднього конструювання матриці ознак з лаговими значеннями та закодованими часовими предикторами. Серед практичних переваг алгоритму - нечутливість до масштабу ознак, стійкість до сильних кореляцій між предикторами, можливість обробки пропущених значень без їх попереднього заповнення та наявність вбудованих механізмів оцінки важливості окремих предикторів за метриками gain та cover, що дозволяє інтерпретувати модель навіть попри її ансамблеву природу.

Третьою реалізованою моделлю є рекурентна нейронна мережа LSTM (Long Short-Term Memory) - спеціалізована архітектура глибокого навчання, призначена для роботи з послідовностями. Архітектура запропонована Зеппом Гохрейтером і Юргеном Шмідгубером у 1997 році з метою подолання проблеми зникаючого градієнта, характерної для класичних рекурентних мереж: при тренуванні на довгих послідовностях градієнт функції втрат експоненційно загасає у міру поширення назад у часі, що унеможливлює навчання довгострокових залежностей.

Ключовою архітектурною особливістю LSTM є наявність у клітинці мережі трьох керованих затворів (gates), які регулюють потоки інформації між поточним вхідним сигналом, попереднім прихованим станом та внутрішнім станом клітинки (cell state). Кожен з затворів реалізується як сигмоїдальна нейронна одиниця, що повертає значення у діапазоні [0, 1], яке трактується як коефіцієнт пропускання інформації. Затвор забування f_t, що визначає, яка

частина попереднього стану клітинки буде відкинута на поточному кроці, обчислюється за виразом:

$$f_t = \sigma\big(W_f \cdot [h_{t-1}, x_t] + b_f\big) \quad (2.4)$$

де σ(·) - сигмоїдальна функція активації; W_f та b_f - навчальні параметри затвору (матриця ваг та вектор зсуву відповідно); h_{t-1} - прихований стан клітинки на попередньому часовому кроці; x_t вхідний сигнал на поточному кроці; квадратні дужки позначають операцію конкатенації векторів. Аналогічну математичну форму, але з власними параметрами, мають затвор входу i_t (визначає, яка частина нової інформації буде записана у стан клітинки) та затвор виходу o_t (визначає, яка частина оновленого стану буде передана як вихід поточного кроку).

Саме адитивна - а не мультиплікативна, як у класичних рекурентних мережах - структура оновлення внутрішнього стану клітинки забезпечує LSTM здатність зберігати інформацію впродовж сотень часових кроків без катастрофічного загасання градієнта. Це є ключовою алгоритмічною властивістю, яка робить LSTM придатною для моделювання довгострокових сезонностей у часових рядах - зокрема, тижневих та сезонних патернів у задачах прогнозування енергоспоживання.

Теоретичною перевагою LSTM перед методами на основі ручного конструювання ознак є здатність до автоматичного виявлення довгострокових нелінійних залежностей у даних без необхідності їх явного програмування. Проте на практиці ця перевага реалізується лише за умови достатнього обсягу навчальних даних та належно підібраної архітектури: за обмеженого обсягу вибірки нейромережа схильна до перенавчання та може поступатися значно простішим табличним методам. Окремою практичною характеристикою є висока обчислювальна вартість навчання, що пов'язано з послідовною природою рекурентного обчислення та неможливістю його повної паралелізації, на відміну від градієнтного бустингу або згорткових нейронних мереж.

Розглянуті три моделі репрезентують принципово різні алгоритмічні парадигми: Prophet формалізує адитивне статистичне розкладання часового ряду

на інтерпретовані компоненти; XGBoost реалізує ансамблеву комбінацію великої кількості простих деревоподібних моделей з градієнтною оптимізацією; LSTM застосовує рекурентну нейромережеву архітектуру з керованою пам'яттю для безпосереднього моделювання послідовностей. Принципова теоретична різниця між цими підходами зумовлює очікувано різну поведінку моделей у конкретних умовах задачі, що обґрунтовує доцільність їх паралельної реалізації та порівняння на ідентичних вхідних даних. Архітектурні рішення, які забезпечують таку паралельну реалізацію у єдиній модульній системі, розглядаються у наступному підрозділі 2.2.

2.2 Загальна архітектура системи збору та обробки телеметрії

Проєктування архітектурного каркаса інтелектуального комплексу предиктивного моніторингу енергоспоживання здійснюється на основі системного інженерного підходу, що базується на принципах модульності, декомпозиції складних процесів, ізоляції компонентів та мінімізації їхньої взаємної зв'язності. У сфері розробки сучасних інтелектуальних систем підтримки прийняття рішень та обробки великих масивів інформації з пристроїв Інтернету речей (IoT) критично важливим є розділення інфраструктурних рівнів, що відповідають за фізичну взаємодію з датчиками, від високонавантажених обчислювальних модулів аналізу та моделювання.

Для забезпечення високої масштабованості, відмовостійкості та гнучкості при подальшій модернізації алгоритмів, систему було спроєктовано та реалізовано у вигляді багаторівневого конвеєра обробки даних (Data Pipeline). Кожен рівень цього конвеєра є функціонально замкненим мікросервісом, який виконує суворо визначений перелік трансформацій вхідного потоку інформації, взаємодіючи з іншими рівнями через стандартизовані програмні інтерфейси (API).

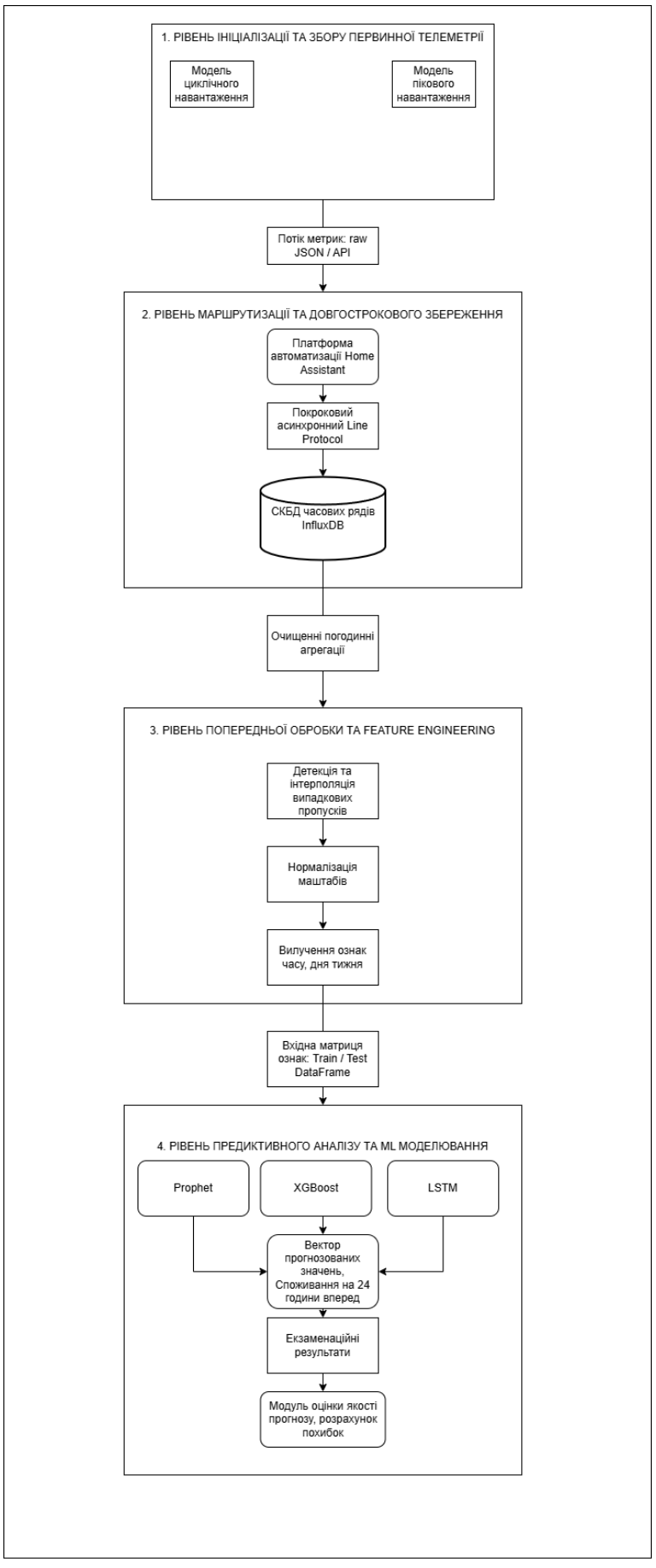

Рисунок 2.1 - Структурна схема потоків даних

Наведена архітектурна модель (рис. 2.1) складається з чотирьох послідовних технологічних рівнів:

1. Рівень ініціалізації та збору первинної телеметрії (Data Ingestion Layer). Даний шар є первинною вхідною точкою всього конвеєра обробки. Головне завдання рівня полягає у безперервному фіксуванні показників електричної потужності та формуванні сирого часового ряду телеметрії. У промислових умовах експлуатації цей рівень складається з кінцевих апаратних рішень, що транслюють метрики через бездротові протоколи. У межах поточної реалізації, з метою забезпечення репрезентативності та повноти річного аналітичного зрізу, цей рівень програмно емулюється спеціалізованим модулем. Модуль математично відтворює високодинамічні змішані патерни навантаження (базові, циклічні, пікові) та здійснює первинне пакетування точок даних.

2. Рівень маршрутизації, агрегації та довгострокового збереження (Data Storage & Routing Layer). Отриманий телеметричний потік спрямовується до інтеграційної зв'язки, що є ядром інфраструктури Розумного дому. Локальна open-source платформа автоматизації Home Assistant виконує роль інтелектуального шинного маршрутизатора. Вона перехоплює стани віртуальних або фізичних сенсорів, нормалізує їхні метадані до єдиного системного формату та здійснює короткострокову буферизацію у пам'яті. Після цього дані в потоковому асинхронному режимі експортуються до бази даних часових рядів InfluxDB[6]. Архітектура InfluxDB спеціально оптимізована під інтенсивні операції запису з високою частотою (high-frequency write tasks), що дозволяє накопичувати погодинну історію глибиною в 365 днів без зниження швидкодії та деградації дискової підсистеми.

3. Рівень попередньої обробки та конструювання ознак (Data Preprocessing & Feature Engineering Layer). Цей шар функціонує як проміжний аналітичний модуль у середовищі виконання Python з використанням можливостей бібліотеки Pandas. Його призначення - трансформація сирих даних, вилучених із бази даних InfluxDB[7] за допомогою SQL-подібних API-запитів, у

структуровані матриці ознак. На цьому етапі реалізовано алгоритми перевірки цілісності даних, детекції та інтерполяції випадкових пропусків. З первинної часової мітки програмно вилучаються окремі ознаки (конкретна година доби, день тижня, індикатор вихідного дня), що є критично важливим для надання моделям машинного навчання контекстуальної інформації про поведінкові фактори користувачів.

4. Рівень предиктивного аналізу та моделювання (Predictive Analytics & ML Layer). Високотехнологічне обчислювальне серце системи, де зосереджено математичні алгоритми прогнозування. Рівень функціонує в ізольованому програмному контейнері та містить три паралельні аналітичні підмодулі:

- Модуль статистичного аналізу на базі адитивної моделі Prophet, що ефективно ізолює загальні тренди та сезонні коливання на основі рядів Фур'є.
- Модуль ансамблевого моделювання на базі алгоритму градієнтного бустингу XGBoost, який оперує побудованою матрицею часових ознак для знаходження складних нелінійних зв'язків.
- Модуль глибокого навчання на базі рекурентної нейронної мережі LSTM, що безпосередньо аналізує послідовності часового ряду за допомогою складних комірок пам'яті з вентильними механізмами контролю інформації.

Кожен із зазначених підмодулів приймає підготовлений на попередньому етапі масив даних, здійснює ітераційне навчання та генерує вихідний вектор прогнозних значень навантаження з глибиною у 24 години вперед. Результати предиктивного аналізу акумулюються у вигляді структурованих масивів для подальшої передачі в автоматизовані сценарії управління енергоспоживанням.

Взаємодія між усіма чотирма рівнями конвеєра побудована на базі чітко специфікованих протоколів обміну, що гарантує високу стабільність, відсутність каскадних помилок при збоях у поодиноких модулях та забезпечує повну готовність системи до роботи в умовах безперервного надходження телеметричної інформації.

2.3 Структура програмних модулів та класів

Описана у попередньому підрозділі загальна архітектура системи має своє конкретне втілення на рівні програмного коду у вигляді чітко організованої сукупності модулів, класів та функцій, які реалізують функціональність кожного з виділених архітектурних шарів. Перш ніж переходити до детального опису реалізації окремих підсистем, доцільно навести узагальнений огляд програмної структури системи та принципів, що лежать в основі взаємодії її компонентів. Такий огляд дозволяє сформувати у читача цілісне уявлення про спосіб організації коду і полегшує сприйняття технічних подробиць, які наводитимуться у наступних розділах роботи.

Кодова база системи організована за принципом функціональної декомпозиції: кожен пакет каталогу проєкту відповідає за чітко окреслену зону відповідальності. Каталог data містить модулі завантаження інформації з InfluxDB та її попередньої обробки; каталог models - реалізації трьох прогностичних моделей разом із базовим абстрактним класом; каталог evaluation - функції обчислення метрик якості; каталог visualization - модулі побудови графіків; каталог scripts - допоміжні сценарії одноразового призначення (генерація історії, імпорт у базу даних, відкат); каталог utils - службові засоби, зокрема централізоване логування. У кореневій директорії проєкту розташовано конфігураційний файл config.py, точку входу командного інтерфейсу main.py та модуль графічного інтерфейсу gui.py. Така структура відповідає прийнятим у промисловій розробці на Python практикам і забезпечує можливість легкої навігації по проєкту як для його автора, так і для потенційних співавторів або користувачів коду.

Центральним архітектурним рішенням, на якому ґрунтується підсистема машинного навчання, є застосування об'єктно-орієнтованого шаблону проєктування «Шаблонний метод» (Template Method). Цей шаблон передбачає визначення у базовому абстрактному класі загальної послідовності виконання операції з делегуванням варіативних кроків конкретним підкласам, які успадковуються від базового. У реалізованій системі роль базового класу

виконує BaseModel, оголошений у модулі models/base_model.py, що декларує єдиний інтерфейс взаємодії з прогностичною моделлю. Цей інтерфейс складається з двох обов'язкових абстрактних методів - train(df) для навчання моделі на історичній вибірці та predict(df) для формування прогнозу на горизонт 24 годин, - а також готового неабстрактного методу evaluate(df), який реалізує процедуру екзаменаційного оцінювання на основі Train-Test Split. Завдяки тому, що ця процедура повністю реалізована у базовому класі, додавання нової прогностичної моделі вимагає реалізації виключно двох методів - навчання та прогнозу, - а функція оцінки автоматично стає для неї доступною без жодних додаткових модифікацій коду.

Від базового класу BaseModel успадковуються три конкретні підкласи, кожен з яких реалізує одну з обраних прогностичних моделей. Клас ProphetModel (модуль models/prophet_model.py) інкапсулює роботу з адитивною статистичною моделлю Prophet від компанії Meta; його реалізація є найкомпактнішою з трьох завдяки тому, що бібліотека Prophet самостійно виконує конструювання сезонних ознак. Клас XGBoostModel (модуль models/xgboost_model.py) інкапсулює роботу з ансамблевим методом градієнтного бустингу на деревах рішень; його реалізація включає авторегресивну схему прогнозування на 24 кроки з використанням лагових ознак. Клас LSTMModel (модуль models/lstm_model.py) інкапсулює роботу з рекурентною нейронною мережею з пам'яттю довгої та короткої тривалості; його реалізація є найскладнішою з трьох і включає побудову мультиваріантних вхідних послідовностей, нормалізацію даних та механізм автоматичної зупинки навчання. Принципово важливим аспектом такої архітектури є те, що з точки зору решти системи всі три класи є взаємозамінними: основний код програми оперує ними через спільний інтерфейс BaseModel, що дозволяє замінювати одну прогностичну модель на іншу простою зміною одного рядка коду без необхідності модифікації клієнтського коду.

Окрім класів-моделей, у складі підсистеми машинного навчання використовуються три допоміжні модулі, що надають загальносистемну функціональність. Модуль data/preprocessor.py забезпечує підготовку вхідних

даних і містить функції конструювання часових ознак (create_time_features), формування лагових ознак (add_lag_features), створення тривимірних послідовностей для LSTM (create_sequences) та хронологічного розділення вибірки на тренувальну і тестову частини (train_test_split). Модуль evaluation/metrics.py забезпечує об'єктивну оцінку якості прогнозів і містить функцію compute_metrics, яка обчислює метрики MAE та RMSE на основі пари «факт-прогноз». Модуль config.py виконує роль єдиного джерела істини для всіх гіперпараметрів, констант, шляхів до файлів та параметрів підключення до бази даних; такий централізований підхід виключає виникнення розбіжностей між різними частинами системи та спрощує процедуру налаштування експериментів.

Описаний підхід до організації програмної структури системи забезпечує дотримання декількох фундаментальних принципів якісної програмної інженерії. Принцип єдиної відповідальності (Single Responsibility Principle) реалізується через чіткий поділ функціональності між пакетами: кожен модуль відповідає за виконання обмеженого набору пов'язаних операцій. Принцип відкритості/закритості (Open-Closed Principle) реалізується через спадковість від абстрактного класу BaseModel: система є відкритою для розширення (додавання нових моделей) і закритою для модифікації (не потребує зміни існуючого коду). Принцип інверсії залежностей (Dependency Inversion Principle) реалізується через те, що високорівнева логіка програми залежить від абстрактного інтерфейсу BaseModel, а не від конкретних реалізацій моделей. Сукупність цих принципів забезпечує тривалу підтримуваність системи та її стійкість до подальших розширень функціональності, що є важливою практичною перевагою розробленого рішення.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

Викладені у попередньому розділі теоретичні та архітектурні засади розробленої системи знаходять своє конкретне втілення на рівні програмного коду - у вигляді чітко організованих підсистем, кожна з яких відповідає за окрему функціональну зону. У межах цього розділу детально розглядається реалізація

всіх ключових компонентів системи з аналізом конкретних інженерних рішень, прийнятих на етапі розробки, та посиланнями на фрагменти вихідного коду, що ілюструють особливості реалізації.

Послідовність викладу відповідає природному потоку даних у системі та логіці взаємодії її компонентів. Підрозділ 3.1 присвячений підсистемам збору телеметрії з інтелектуальної електричної розетки, її зберігання у спеціалізованій базі даних часових рядів InfluxDB та подальшої попередньої обробки для подачі у прогностичні моделі. У підрозділі 3.2 детально описуються три реалізовані прогностичні моделі та їхні архітектурні особливості з наведенням ключових фрагментів вихідного коду. Підрозділ 3.3 присвячений опису програмного інструментарію та технологічного стека: переліку використаних сторонніх бібліотек, версій та компонентів інфраструктури. Завершує розділ підрозділ 3.4, у якому розглянуто два паралельних користувацьких інтерфейси системи: командний (CLI) та графічний (GUI).

Сукупність викладеного у цьому розділі матеріалу формує повне технічне уявлення про реалізовану систему та готує підґрунтя для її експериментальної верифікації, опис якої наведено у наступному розділі.

3.1 Підсистеми збору та підготовки даних

Підсистеми збору та підготовки даних формують вхідний рівень розробленого програмного конвеєра, від якості роботи якого безпосередньо залежить репрезентативність навчальної вибірки та, як наслідок, прогностична здатність усіх алгоритмів машинного навчання, що працюють на наступних рівнях. У промислових умовах експлуатації джерелом телеметрії виступає фізичний пристрій - інтелектуальна електрична розетка з вбудованим вимірювачем активної потужності, яка через бездротовий протокол передає миттєві показники до локального вузла автоматизації Home Assistant. Однак для забезпечення можливості повноцінного навчання моделей та відтворюваності експериментальних досліджень у межах дипломного проєкту необхідна тривала історична вибірка глибиною не менше одного календарного року. Збір такого

обсягу реальних даних у режимі реального часу є технічно неможливим у межах строків виконання роботи, що зумовило необхідність розробки спеціалізованого програмного модуля імітаційного моделювання.

Архітектурно об'єднана підсистема логічно поділена на три послідовні функціональні етапи: етап синтетичної генерації річної історії, етап її зберігання у спеціалізованій базі даних часових рядів InfluxDB та етап перетворення сирого ряду у структуровані матриці ознак, формат яких відповідає вимогам кожної з трьох прогностичних моделей. Усі три етапи реалізовані у вигляді незалежних програмних модулів, що відповідає принципу низького зв'язування між компонентами та дозволяє виконувати їх як в межах єдиного процесу, так і окремо - наприклад, виконати генерацію один раз і надалі багаторазово запускати лише підготовку даних з різними параметрами без повторного формування історії.

Розроблений модуль generate_history.py реалізує процедурну генерацію погодинного часового ряду активної потужності тривалістю 365 діб, що сумарно становить 8760 дискретних відліків. Алгоритм побудований на принципі суперпозиції трьох незалежних компонент електричного навантаження, кожна з яких математично відтворює окремий клас побутових споживачів та має власний характерний профіль роботи. Такий декомпозиційний підхід дозволяє отримати на виході реалістичну криву споживання, що поєднує стабільну базову складову, стохастичні циклічні стрибки та чітко детерміновані пікові навантаження.

Перша компонента - фонове навантаження - формується як константна складова потужністю 40 Вт, що присутня в кожному відліку часового ряду незалежно від години доби чи дня тижня. Ця складова математично описує сумарне споживання приладів, які перебувають у режимі очікування (роутер, ТБ-приставка, зарядні пристрої, лампи індикації), та забезпечує наявність нижньої межі споживання, нижче якої значення активної потужності не опускається у жодний момент часу. Такий підхід відповідає реальній експлуатаційній

поведінці сучасного домогосподарства, де частина приладів є постійно підключеною до електромережі.

Друга компонента - циклічне навантаження - реалізована як випадкова величина потужністю 150 Вт, що додається до загального споживання з імовірністю 40 % у межах кожної години. Така стохастична модель відповідає реальному режиму роботи побутового холодильного компресора, який вмикається періодично залежно від внутрішньої температури камери, частоти відкривання дверей та зовнішніх теплових умов. Використання ймовірнісного, а не строго періодичного підходу дозволяє внести у вибірку контрольований випадковий шум, що ускладнює задачу для прогностичних моделей та робить її ближчою до реальних умов, у яких алгоритмам машинного навчання доведеться функціонувати після впровадження.

Третя компонента - пікове навантаження - реалізована як детермінований імпульс потужністю 2000 Вт, прив'язаний до фіксованих часових інтервалів доби: ранкового вікна з 7:00 до 8:00 та вечірнього вікна з 20:00 до 21:00. Така прив'язка відображає типові поведінкові патерни користувачів, пов'язані з ранковими гігієнічними процедурами та підготовкою до сну. Саме ця компонента формує яскраво виражені сезонні піки на добовому графіку споживання, виявлення та відтворення яких є ключовою практичною задачею для всіх трьох прогностичних моделей, які надалі будуть навчатися на згенерованому наборі даних.

Підсумкове значення активної потужності для кожної години розраховується як адитивна сума трьох описаних компонент із подальшим записом отриманої точки даних у вихідний CSV-файл з колонками timestamp (мітка часу у форматі ISO 8601) та power_w (значення потужності у ватах). Сформований у такий спосіб набір даних має чітко виражену добову сезонність (зумовлену роботою бойлера), тижневу однорідність (профіль ідентичний для всіх днів тижня), а також контрольований стохастичний шум (внесений роботою холодильника), що в сукупності робить його репрезентативним полігоном для

коректного порівняння точності адитивних статистичних моделей, ансамблевих методів та рекурентних нейронних мереж.

Після завершення процедури синтетичної генерації сформований CSV-файл піддається процедурі імпорту до бази даних часових рядів InfluxDB, що реалізована окремим програмним модулем import_to_influx.py. Така архітектурна декомпозиція - розділення процесів генерації та збереження - забезпечує можливість багаторазового повторного імпорту тих самих даних без необхідності повторної генерації, а також спрощує процедури відлагодження та контролю цілісності вибірки на проміжному CSV-етапі. Підключення до бази даних здійснюється через офіційну Python-бібліотеку influxdb, яка реалізує клієнт для роботи з InfluxDB версії 1.x за протоколом HTTP. Параметри підключення (IP-адреса сервера, порт 8086, ім'я цільової бази home_assistant, ідентифікатор вимірювання W та назва сутності sensor.roz_svroom_power) централізовано винесені в конфігураційний файл config.py, що дозволяє адаптувати систему до різних розгортань без модифікації основного коду.

Процес імпорту реалізовано у вигляді пакетного запису (batch insert), за якого набір точок даних формується у вигляді списку Python-словників з полями measurement, tags, time та fields, після чого передається до клієнта InfluxDB одним викликом методу write_points. Такий підхід є на декілька порядків продуктивнішим за поелементний запис, оскільки мінімізує кількість мережевих транзакцій та використовує внутрішні оптимізації пакетного зберігання, реалізовані у движку TSM (Time-Structured Merge Tree) бази даних. Перед записом кожна часова мітка приводиться до формату UTC та серіалізується відповідно до вимог протоколу InfluxDB Line Protocol, а значення активної потужності зберігається у полі value як число з плаваючою комою. Сутність-ідентифікатор entity_id записується у вигляді тегу, що дозволяє у подальшому ефективно фільтрувати дані за конкретною розеткою у разі розширення системи на множинні точки моніторингу.

Для забезпечення можливості безперервного функціонування системи у режимі, наближеному до реальної експлуатації, реалізовано додатковий модуль щоденного дописування даних daily_generator.py. Цей сценарій при кожному запуску формує 24 нові погодинні точки за тим самим математичним алгоритмом, та дописує їх до InfluxDB безпосередньо, без проміжного CSV-файлу. Запуск модуля може бути автоматизований засобами планувальника операційної системи (cron у Linux або Task Scheduler у Windows), що дозволяє підтримувати актуальність вибірки без втручання оператора. Окремо реалізовано службовий сценарій rollback_data.py, який забезпечує можливість видалення останніх N днів з бази у разі необхідності повторного експериментування або корекції алгоритму генерації.

Завантаження накопиченої історії з InfluxDB до робочого середовища Python для подальшої обробки моделями машинного навчання здійснюється спеціалізованим модулем data/influx_loader.py. Модуль формує InfluxQL-запит виду SELECT value FROM W WHERE entity_id = 'sensor.roz_svroom_power' із додатковим обмеженням за часовим діапазоном, виконує його через клієнт InfluxDB та повертає результат у вигляді об'єкта pandas.DataFrame з колонками ds (часова мітка у форматі datetime) та y (значення потужності у ватах). Така уніфікована форма даних відповідає вимогам формату вхідних даних бібліотеки Prophet, а для решти моделей (XGBoost та LSTM) використовується після додаткового проходження через підсистему підготовки даних. Для забезпечення стійкості системи у разі тимчасової недоступності бази даних модуль додатково реалізує механізм резервного завантаження безпосередньо з CSV-файлу, що дозволяє продовжувати роботу аналітичного ядра навіть в умовах відсутності з'єднання з InfluxDB.

Сирий часовий ряд, отриманий з InfluxDB у вигляді двоколонкового DataFrame (ds, y), є придатним для безпосереднього використання лише адитивною моделлю Prophet. Для двох інших моделей - XGBoost та LSTM - він потребує суттєвої додаткової обробки: формування предикторних ознак з часових міток, побудови лагових змінних та (у випадку LSTM) приведення

значень до єдиного числового діапазону. Виконання цих перетворень покладено на спеціалізований модуль data/preprocessor.py, який концептуально виконує три послідовні етапи обробки: конструювання часових ознак, формування лагових змінних та нормалізацію числових значень. Загальний потік перетворення сирих даних у формат, готовий до подачі в моделі, наведено на рис. 3.1.

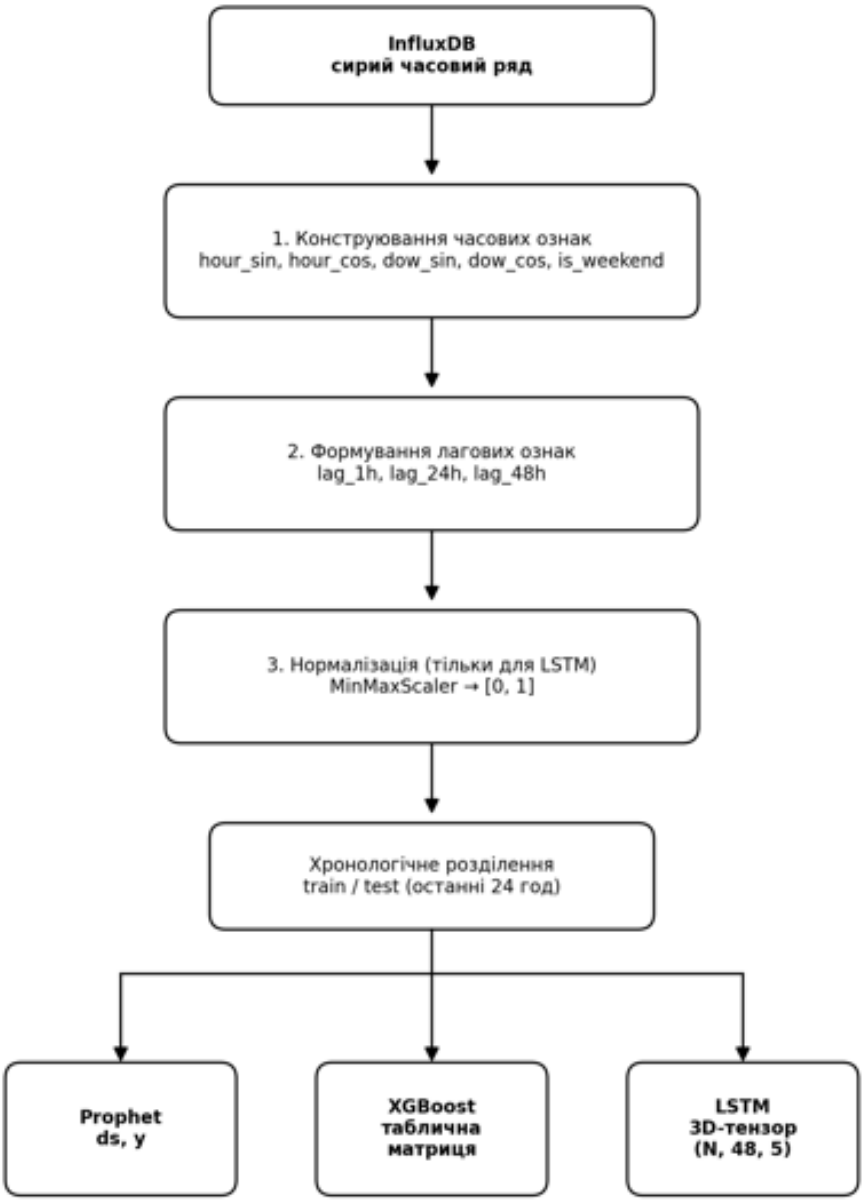

Рисунок 3.1 - Конвеєр підготовки даних для трьох прогностичних моделей

Перший етап - конструювання часових ознак - є фундаментальним для подальшої роботи моделей, що оперують табличними даними. На цьому етапі з кожної часової мітки програмно вилучаються нові предикторні змінні, які

надають моделі контекстуальну інформацію про положення відліку в межах добового та тижневого циклу. Зокрема, формуються бінарна ознака is_weekend (індикатор вихідного дня) та чотири синусоїдально-косинусоїдальні ознаки циклічного кодування - hour_sin, hour_cos, dow_sin, dow_cos. Використання саме циклічного, а не лінійного кодування часових величин є принципово важливим методичним рішенням: воно дозволяє моделі сприймати години 23:00 та 0:00 як часово близькі точки, а не як максимально віддалені одна від одної значення, що неминуче відбувалося б при простому лінійному поданні номера години у діапазоні від 0 до 23.

Математично циклічне кодування реалізовано шляхом проектування дискретної часової змінної на одиничне коло за допомогою двох тригонометричних функцій. Для години доби h пара ознак обчислюється за виразом:

$$\text{hour\_sin} = \sin\frac{2\pi h}{24}, \quad \text{hour\_cos} = \cos\frac{2\pi h}{24} \tag{3.1}$$

де h - поточна година доби у діапазоні [0, 23]. Аналогічна процедура з періодом 7 застосовується для дня тижня. У результаті кожному значенню часової змінної ставиться у відповідність унікальна пара координат на одиничному колі, при цьому евклідова відстань між сусідніми точками (наприклад, між 23-ю та 0-ю годинами) залишається мінімальною. Такий підхід суттєво підвищує здатність ансамблевих алгоритмів та нейронних мереж коректно розпізнавати періодичні закономірності у даних та є загальноприйнятою практикою у задачах прогнозування часових рядів.

Другий етап обробки полягає у формуванні лагових ознак - значень цільової змінної, зміщених у часі на фіксований інтервал назад. У межах розробленої системи реалізовано три лагові ознаки: lag_1h (споживання годину тому), lag_24h (споживання добу тому) та lag_48h (споживання дві доби тому). Вибір саме такого набору лагів обґрунтований структурою аналізованого часового ряду: лаг у 1 годину передає короткострокову інерцію навантаження

(ймовірність того, що активний прилад продовжить роботу у наступну годину), лаг у 24 години фіксує добову сезонність (повторюваність ранкових та вечірніх пікових споживань), а лаг у 48 годин дозволяє моделі розрізнити випадкові аномалії одного дня від стійких закономірностей. Слід зазначити, що формування лагових ознак неминуче призводить до втрати перших 48 рядків матриці даних, оскільки для них значення лагів є невизначеними; ці рядки виключаються з тренувальної вибірки на етапі очищення даних.

Третій етап - нормалізація числових значень - застосовується вибірково лише для моделі LSTM, що зумовлено особливостями функціонування нейронних мереж. Алгоритми оптимізації, що використовуються при навчанні рекурентних мереж (зокрема, Adam), демонструють суттєво кращу збіжність та стабільність у випадку, коли вхідні ознаки приведені до спільного числового діапазону. Для нормалізації застосовано клас MinMaxScaler з бібліотеки scikit-learn, який лінійно масштабує значення до діапазону [0, 1] за формулою:

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}} \quad (3.2)$$

де x - поточне значення ознаки; x_min та x_max - мінімальне та максимальне значення ознаки, обчислені за тренувальною вибіркою. Принципово важливим методологічним аспектом є те, що параметри масштабування розраховуються виключно за тренувальною вибіркою, після чого ті самі параметри застосовуються до тестової вибірки. Такий підхід дозволяє уникнути ефекту витоку інформації з майбутнього у минуле (data leakage) - поширеної помилки при підготовці даних для задач прогнозування часових рядів, яка полягає у випадковому використанні майбутньої інформації для оцінки моделі. Якщо параметри масштабування обчислювати за всією вибіркою (включно з тестовою частиною), то модель неявно отримує доступ до статистичних характеристик майбутніх даних, що призводить до невиправдано оптимістичних оцінок точності, які не відтворюються при реальній експлуатації системи.

Окрему функціональну роль у підсистемі підготовки даних відіграє процедура розділення вибірки на тренувальну та тестову частини. На відміну від класичних задач машинного навчання, де вибірка зазвичай розділяється випадковим чином, для часових рядів застосовується виключно хронологічне розділення: тестова вибірка завжди формується з найбільш пізніх відліків часового ряду. У межах розробленої системи кількість тестових відліків винесена у конфігураційний параметр TEST_SIZE_HOURS = 24, що відповідає горизонту короткострокового прогнозування. Таким чином, з річної історії у 8760 відліків останні 24 години відрізаються як екзаменаційний набір, а решта 8736 відліків використовуються для навчання моделей. Подібна організація процесу валідації максимально точно відтворює реальний сценарій експлуатації системи, у якому модель навчається на всій доступній історії та прогнозує наперед невідому добу.

Реалізація підсистеми підготовки даних дотримується принципу єдиного джерела істини: усі константи та гіперпараметри (розмір вікна LSTM, перелік лагових ознак, цільовий діапазон нормалізації, розмір тестової вибірки) централізовано визначені у файлі config.py та імпортуються до модуля підготовки за необхідності. Такий підхід забезпечує відтворюваність експериментів, спрощує процедуру налаштування гіперпараметрів та виключає виникнення розбіжностей між параметрами, що використовуються різними прогностичними модулями системи.

3.2 Підсистема машинного навчання

Підсистема машинного навчання є центральним аналітичним ядром розробленої системи та реалізує безпосередньо функцію короткострокового прогнозування електроспоживання на горизонт 24 годин. Архітектурною особливістю підсистеми є паралельне використання трьох принципово різних алгоритмічних підходів - адитивної статистичної моделі Prophet, ансамблевого методу градієнтного бустингу XGBoost та рекурентної нейронної мережі LSTM, - що дозволяє провести коректне порівняльне дослідження їх ефективності на

ідентичних вхідних даних та обрати найбільш точну модель за об'єктивними числовими метриками.

Для забезпечення архітектурної однорідності та можливості легкого розширення системи новими прогностичними алгоритмами всі три моделі реалізовані як підкласи спільного абстрактного класу BaseModel, визначеного у модулі models/base_model.py. Базовий клас декларує два обов'язкових абстрактних методи - train(df) для навчання моделі на історичній вибірці та predict(df) для формування прогнозу на 24 години вперед, - а також надає готову реалізацію методу evaluate(df), яка є спільною для всіх трьох моделей та реалізує єдину процедуру Train-Test Split. Такий підхід відповідає шаблону проєктування «Template Method», за якого незмінна структура алгоритму визначається у базовому класі, а варіативні частини делегуються конкретним підкласам. Ключовий фрагмент реалізації базового класу наведено в лістингу 3.1.

Лістинг 3.1 - Абстрактний клас BaseModel

```
class BaseModel(ABC):

    name: str = "base"

    @abstractmethod
    def train(self, df: pd.DataFrame) -> None: ...

    @abstractmethod
    def predict(self, df: pd.DataFrame) -> pd.DataFrame: ...

    def evaluate(self, df: pd.DataFrame) -> dict:
        train_df, test_df = train_test_split(df)
        self.train(train_df)
        forecast = self.predict(train_df)
        metrics = compute_metrics(test_df["y"].values,
                                  forecast["prediction"].values)
        return {"model": self.name, **metrics, ...}
```

Метод evaluate() реалізує екзаменаційну процедуру Train-Test Split: спочатку викликає функцію train_test_split() з модуля підготовки даних, яка відрізає останні 24 години як тестову вибірку; потім навчає модель на тренувальній частині; виконує прогноз на горизонт 24 години; нарешті порівнює

прогнозні значення з реальними з тестової вибірки та обчислює метрики MAE і RMSE через функцію compute_metrics() з модуля evaluation/metrics.py. Завдяки тому, що ця процедура повністю реалізована у базовому класі, додавання нової прогностичної моделі вимагає реалізації виключно двох методів - train() та predict(), - а екзаменаційна логіка автоматично стає для неї доступною без жодних модифікацій.

Усі гіперпараметри моделей, перелік ознак, шляхи до файлів та параметри навчання централізовано винесені у конфігураційний файл config.py, що забезпечує відтворюваність експериментів та полегшує процедуру налаштування системи. Кожна модель реалізована як окремий програмний модуль у каталозі models/, що відповідає принципу низького зв'язування між компонентами та дозволяє модифікувати будь-який алгоритм без впливу на решту системи. Для централізованого журналювання процесу навчання та прогнозування у всіх модулях використовується спеціалізований модуль utils/logger.py, що надає налаштований об'єкт логера за викликом get_logger(__name__).

Перша з реалізованих моделей - Prophet - інкапсульована у класі ProphetModel модуля models/prophet_model.py. Реалізація є найбільш компактною з трьох (34 рядки коду) завдяки тому, що бібліотека Prophet самостійно виконує конструювання сезонних ознак та не потребує попередньої підготовки матриці предикторів. Метод train() створює екземпляр класу Prophet з гіперпараметрами, що зберігаються у словнику PROPHET_PARAMS конфігураційного файлу, та викликає його метод fit() на двоколонковому DataFrame (ds, y). Метод predict() використовує допоміжну функцію make_future_dataframe() для формування діапазону майбутніх часових міток та повертає результат у стандартизованому форматі з колонками timestamp та prediction. Ключовий фрагмент реалізації наведено в лістингу 3.2.

Лістинг 3.2 - Реалізація моделі Prophet

```
class ProphetModel(BaseModel):
```

```
    name = "prophet"

    def train(self, df: pd.DataFrame) -> None:
        self.model = Prophet(**config.PROPHET_PARAMS)
        self.model.fit(df[["ds", "y"]])

    def predict(self, df: pd.DataFrame) -> pd.DataFrame:
        future = self.model.make_future_dataframe(
            periods=config.FORECAST_HORIZON, freq="h"
        )
        forecast = self.model.predict(future)
        forecast["yhat"] = forecast["yhat"].clip(lower=0.0)
        result = forecast[["ds", "yhat"]].tail(config.FORECAST_HORIZON)
        return result.rename(columns={"ds": "timestamp", "yhat":
"prediction"})
```

У реалізації застосовано фіксований набір гіперпараметрів: daily_seasonality=True (увімкнено добову сезонність як ключову періодичну складову енергоспоживання); weekly_seasonality=True (увімкнено тижневу сезонність для врахування потенційної різниці між робочими днями та вихідними); yearly_seasonality=False (річна сезонність вимкнена, оскільки вибірка обмежена одним календарним роком і її виявлення не є статистично обґрунтованим); changepoint_prior_scale=0.3 (підвищена відносно стандартного значення 0.05 гнучкість виявлення точок зламу тренду, що обґрунтовано стабільною природою синтетичних патернів). Окремим практично важливим моментом реалізації є обрізання негативних прогнозів через виклик .clip(lower=0.0): оскільки активна потужність електроспоживання за фізичним змістом не може бути від'ємною, але адитивна модель Prophet теоретично здатна повертати негативні значення внаслідок поєднання тренду та сезонностей, такі значення примусово замінюються на нуль перед поверненням результату.

Друга реалізована модель - XGBoost - інкапсульована у класі XGBoostModel модуля models/xgboost_model.py. На відміну від Prophet, XGBoost потребує суттєвої попередньої обробки даних, тому метод train() послідовно викликає функції create_time_features() та add_lag_features() з модуля підготовки даних для формування повної матриці ознак з дев'ятьма предикторами: чотирма

циклічними кодами часу (hour_sin, hour_cos, dow_sin, dow_cos), бінарним індикатором вихідного дня (is_weekend) та трьома лаговими значеннями (lag_1h, lag_24h, lag_48h). Перелік цих ознак централізовано визначений у списку XGBOOST_FEATURES конфігураційного файлу.

Принципово важливою архітектурною особливістю реалізованого модуля є авторегресивна схема прогнозування на 24 кроки, продемонстрована у лістингу 3.3. Оскільки безпосередньо у момент формування прогнозу значення лагових ознак для майбутніх годин невідомі, обчислення виконується ітеративно у циклі для кожної з 24 цільових годин: на першому кроці прогнозується перша година з використанням актуальних значень з історії; отримане прогнозне значення додається до робочого DataFrame history як новий запис; на наступних ітераціях лагові ознаки беруться вже з оновленої історії, що включає попередні прогнози. Такий підхід дозволяє моделі поширювати знання про короткострокову інерцію навантаження на весь горизонт прогнозу, однак має зворотний бік - поступове накопичення похибки на пізніх кроках, оскільки кожне нове передбачення базується на потенційно неточних попередніх.

Лістинг 3.3 - Авторегресивний прогноз XGBoost на 24 години

```
class XGBoostModel(BaseModel):
    name = "xgboost"

    def predict(self, df: pd.DataFrame) -> pd.DataFrame:
        history = df[["ds", "y"]].copy()
        future_times = future_index(df["ds"].max())
        predictions = []

        for t in future_times:
            row = create_time_features(pd.DataFrame({"ds": [t]}))
            row["lag_1h"]  = history["y"].iloc[-1]
            row["lag_24h"] = history["y"].iloc[-24]
            row["lag_48h"] = history["y"].iloc[-48]
            pred = max(0.0, float(self.model.predict(
                row[config.XGBOOST_FEATURES])[0]))
            predictions.append(pred)
            history = pd.concat([history,
                pd.DataFrame({"ds": [t], "y": [pred]})],
ignore_index=True)

        return pd.DataFrame({"timestamp": list(future_times),
```

```
                "prediction": predictions})
```

Налаштування моделі здійснюється з використанням таких гіперпараметрів, винесених у словник XGBOOST_PARAMS: n_estimators=100 (кількість дерев в ансамблі), learning_rate=0.1 (темп навчання, що визначає внесок кожного нового дерева в сумарний прогноз), max_depth=5 (максимальна глибина окремого дерева, що обмежує його здатність до перенавчання) та random_state=42 (фіксація початкового стану генератора псевдовипадкових чисел для забезпечення відтворюваності результатів). Підбір саме таких значень забезпечує баланс між апроксимаційною здатністю моделі та її стійкістю до перенавчання на наявному обсязі вибірки.

Особливою практичною деталлю реалізації є запобіжний механізм для випадків недостатньої історії: умовні вирази if n >= 24 else history["y"].iloc[0] забезпечують коректну роботу моделі навіть тоді, коли довжина історії менша за необхідний лаг (така ситуація може виникнути при роботі з обрізаним набором даних під час відлагодження). Аналогічно до Prophet, прогнозні значення обмежуються знизу нулем через виклик max(0.0, ...), що гарантує фізичну осмисленість результату.

Третя реалізована модель - рекурентна нейронна мережа LSTM - інкапсульована у класі LSTMModel модуля models/lstm_model.py та є найбільш складною з трьох з точки зору обсягу коду та внутрішньої логіки. Допоміжна функція _build_features() формує мультиваріантну матрицю ознак форми (n, 5), у якій перший стовпець відповідає значенню активної потужності, а наступні чотири - циклічним кодам часу. Такий формат необхідний для подальшого формування трьохвимірних тензорів навчальних послідовностей через функцію create_sequences() модуля підготовки даних, яка генерує ковзне вікно довжиною 48 годин та формує вхідний тензор форми (n, 48, 5).

Принциповою архітектурною деталлю реалізації є вибіркова нормалізація вхідних даних: масштабуванню за допомогою MinMaxScaler піддається

виключно перша колонка (значення потужності), тоді як циклічні часові ознаки залишаються без змін, оскільки вони за побудовою лежать у діапазоні [-1, 1]. Така вибіркова нормалізація економить обчислювальні ресурси та зберігає об'єкт scaler лише для одної змінної, що спрощує процедуру оберненого перетворення прогнозних значень. Ключовий фрагмент методу train() з побудовою архітектури мережі та налаштуванням процесу навчання наведено в лістингу 3.4.

Лістинг 3.4 - Побудова та навчання нейронної мережі LSTM

```
class LSTMModel(BaseModel):
    name = "lstm"

    def train(self, df: pd.DataFrame) -> None:
        features = _build_features(df)
        self.scaler = MinMaxScaler(feature_range=(0, 1))
        features[:, 0:1] = self.scaler.fit_transform(features[:, 0:1])
        X, y = create_sequences(features, config.LSTM_LOOK_BACK)

        self.model = Sequential([
            LSTM(config.LSTM_UNITS, activation="tanh",
                 input_shape=(config.LSTM_LOOK_BACK, X.shape[2])),
            Dropout(config.LSTM_DROPOUT),
            Dense(1),
        ])
        self.model.compile(optimizer="adam", loss="mse")
        early_stop = EarlyStopping(monitor="val_loss",
            patience=config.LSTM_PATIENCE, restore_best_weights=True)
        self.model.fit(X, y, epochs=self.epochs,
            batch_size=self.batch_size,
            validation_split=config.LSTM_VALIDATION_SPLIT,
            callbacks=[early_stop])
```

Архітектура нейронної мережі є навмисно компактною та складається з одного шару LSTM з 50 прихованими нейронами (параметр LSTM_UNITS=50), шару регуляризації Dropout зі швидкістю відключення 0.2 (LSTM_DROPOUT=0.2) та повнозв'язного вихідного шару Dense з одним нейроном. Шар Dropout під час навчання випадковим чином обнуляє 20 % виходів попереднього шару, що ефективно протидіє перенавчанню та підвищує узагальнюючу здатність мережі. Така мінімалістична конфігурація обрана свідомо: за обмеженого обсягу вибірки (8760 годин) глибші архітектури з

більшою кількістю параметрів схильні до перенавчання та погіршення точності на тестових даних.

Навчання мережі здійснюється з використанням адаптивного оптимізатора Adam, що поєднує переваги методів моментуму та RMSProp, та функції втрат MSE (середньоквадратична похибка), яка є стандартним вибором для задач регресії. Для запобігання перенавчанню та автоматичного визначення оптимального моменту зупинки навчання реалізовано механізм EarlyStopping з параметрами patience=7 та restore_best_weights=True: процес навчання припиняється автоматично, якщо протягом семи послідовних епох не спостерігається покращення значення функції втрат на валідаційній вибірці, причому до фінальної моделі відновлюються ваги тієї епохи, на якій спостерігалось найкраще значення метрики. Це звільняє розробника від необхідності ручного підбору оптимальної кількості епох та забезпечує стабільність процедури навчання за різних умов. Розмір валідаційної вибірки задається параметром LSTM_VALIDATION_SPLIT=0.1, що означає виділення останніх 10 % тренувальних даних для контролю якості під час навчання.

Параметри навчання дещо відрізняються залежно від режиму використання модуля: у режимі прогнозу встановлюється LSTM_FORECAST_EPOCHS=50 та LSTM_FORECAST_BATCH=32, тоді як у режимі екзаменаційного тестування використовується той самий максимум епох, але збільшений до 64 розмір батчу (LSTM_EXAM_BATCH=64) для прискорення процедури Train-Test Split. Прогнозування на 24 години реалізоване, як і у випадку XGBoost, за авторегресивною схемою: модель послідовно генерує по одному прогнозному значенню, яке потім додається до вхідного вікна через операцію vstack та використовується при формуванні наступного прогнозу. Після завершення процедури прогнозні значення піддаються зворотному масштабуванню за допомогою збереженого об'єкта MinMaxScaler через виклик inverse_transform() для повернення у фізично інтерпретовану шкалу ватів, а від'ємні значення відсікаються через np.clip(preds, a_min=0.0, a_max=None).

Описана реалізація трьох прогностичних моделей дотримується єдиного набору архітектурних принципів, що забезпечують цілісність, відтворюваність та розширюваність системи. Принцип уніфікованого інтерфейсу через абстрактний клас BaseModel дозволяє основному коду програми працювати з усіма моделями за єдиним сценарієм виклику, що проявляється у можливості замінити одну модель на іншу простою зміною одного рядка коду. Принцип централізованої конфігурації через файл config.py гарантує, що всі гіперпараметри, шляхи та параметри підключення задаються у єдиному місці, що виключає неузгодженості між різними частинами системи. Принцип відокремлення підготовки даних від моделювання, реалізований через використання спільного модуля data/preprocessor.py, забезпечує консистентність формату вхідних даних для всіх моделей та виключає дублювання коду конструювання ознак. Сукупність цих принципів робить розроблену систему прикладом промислово якісної архітектури машинного навчання, придатної як для дослідницьких цілей у межах дипломного проєкту, так і для подальшого розширення у напрямку реального використання в інфраструктурі розумного дому.

3.3 Опис програмного інструментарію та технологічного стека

Розроблена система прогнозування електроспоживання реалізована як модульний Python-додаток, що інтегрується з відкритою екосистемою домашньої автоматизації Home Assistant через спеціалізовану базу даних часових рядів InfluxDB. Технологічний стек системи сформований із пріоритетом відкритості, локальної обробки даних та можливості розгортання на побутовому обчислювальному обладнанні без вимог до спеціалізованих графічних прискорювачів або хмарних сервісів.

Основною мовою реалізації обрано Python версії 3.13.13 - найновіший на момент розробки стабільний випуск, що поєднує тривалу історію підтримки бібліотек машинного навчання з низкою архітектурних вдосконалень останніх років (зокрема, покращеною обробкою винятків, пришвидшеним

інтерпретатором та удосконаленою системою анотацій типів). Усі модулі системи написані з дотриманням стандарту PEP 8, використовують анотації типів для критичних інтерфейсів та організовані у чітку пакетну структуру, що відповідає сучасним практикам промислової розробки на Python.

Перелік сторонніх бібліотек, від яких залежить функціонування системи, централізовано задекларований у файлі requirements.txt. Цей файл забезпечує можливість відтворення робочого середовища на будь-якому сумісному комп'ютері за допомогою однієї команди пакетного менеджера pip. Повний перелік залежностей з мінімально допустимими версіями наведено в табл. 3.1.

Таблиця 3.1 - Перелік сторонніх бібліотек системи

Бібліотека	Мінімальна версія	Призначення
pandas[10]	2.0	Робота з табличними даними та часовими рядами
numpy	1.24	Обчислення над багатовимірними масивами
scikit-learn	1.3	Метрики MAE/RMSE, MinMaxScaler для нормалізації
xgboost	2.0	Реалізація ансамблевого алгоритму градієнтного бустингу
prophet	1.1	Адитивна статистична модель прогнозування
tensorflow	2.13	Платформа глибокого навчання, основа для LSTM
matplotlib	3.7	Побудова графіків прогнозів та порівняльних діаграм
influxdb	5.3	Python-клієнт для взаємодії з InfluxDB версії 1.x

Кожна з перерахованих бібліотек обрана з обґрунтованих причин. Бібліотека pandas є де-факто стандартом для роботи з часовими рядами у Python-

екосистемі та надає зручні засоби індексації за часом, групування та зсуву (shift), які активно використовуються при формуванні лагових ознак. Бібліотека numpy забезпечує високопродуктивні операції над масивами і є обов'язковою залежністю для більшості бібліотек машинного навчання. Бібліотека scikit-learn використовується для двох конкретних цілей: обчислення метрик якості (mean_absolute_error, mean_squared_error) та нормалізації вхідних даних LSTM (MinMaxScaler). Бібліотеки xgboost, prophet і tensorflow[9] є безпосередніми реалізаціями трьох прогностичних моделей системи. Бібліотека matplotlib забезпечує побудову графіків порівняння прогнозів та екзаменаційних результатів. Нарешті, бібліотека influxdb надає інтерфейс взаємодії з InfluxDB версії 1.x за протоколом HTTP.

Окремим вузлом інфраструктури є платформа автоматизації Home Assistant, що виконує функцію джерела телеметрії від інтелектуальної електричної розетки. У межах розгортання, на якому виконувалась експериментальна частина роботи, використано Home Assistant Core версії 2026.5.4 з операційною системою Home Assistant OS версії 17.3. Платформа налаштована на дублювання усіх показників датчиків до бази даних часових рядів InfluxDB версії 1.8.x з пакету Home Assistant, доступної на порту 8086 за стандартним протоколом HTTP. Параметри підключення (IP-адреса сервера, ім'я бази даних, ідентифікатор сутності) централізовано задано у конфігураційному файлі config.py розробленої системи, що дозволяє адаптувати її до різних розгортань без модифікації основного коду.

Для адміністративного управління InfluxDB та візуального контролю накопиченої телеметрії використовується веб-інтерфейс Chronograf версії 1.10.2, який поставляється у складі того ж пакету та доступний за тією ж адресою сервера. Chronograf не задіяний у безпосередньому функціональному ланцюгу прогнозування - він використовується виключно для перевірки цілісності збережених даних та діагностики стану бази у разі виникнення проблем з підключенням.

Розробка та виконання експериментальної частини здійснювались на обчислювальному вузлі з такими технічними характеристиками: процесор AMD Ryzen 5 5600X (шестиядерна архітектура Zen 3, 12 логічних потоків, базова тактова частота 3.7 ГГц із можливістю динамічного підвищення); оперативна пам'ять обсягом 16 ГБ DDR4; операційна система Microsoft Windows 11 Pro. Зазначена конфігурація є типовою для сучасного робочого комп'ютера середнього класу, без спеціалізованих графічних прискорювачів або серверного обладнання. Це є принциповим практичним підтвердженням того, що розроблена система не потребує спеціалізованої апаратної інфраструктури і може бути впроваджена на побутовому обладнанні кінцевого користувача.

Слід окремо зазначити, що навчання нейронної мережі LSTM здійснюється виключно на центральному процесорі (CPU) без використання графічних прискорювачів. Бібліотека TensorFlow за замовчуванням здатна задіяти GPU за наявності, однак з огляду на компактний розмір мережі (один шар LSTM з 50 нейронами) та помірний обсяг навчальної вибірки (8760 відліків), використання GPU не дає суттєвого виграшу в швидкості та лише ускладнює процедуру розгортання. Повне навчання моделі LSTM з активованим механізмом EarlyStopping на наведеній апаратній конфігурації займає у середньому 30-60 секунд, що є цілком прийнятним для практичних сценаріїв використання.

Архітектура файлової системи проєкту дотримується класичного принципу поділу за функціональним призначенням: каталог models/ містить реалізації прогностичних моделей; data/ - модулі завантаження та підготовки даних; evaluation/ - модулі обчислення метрик; visualization/ - модулі побудови графіків; utils/ - допоміжні модулі (зокрема, налаштування логування); scripts/ - допоміжні сценарії одноразового призначення (генерація історії, імпорт у базу, відкат). У кореневій директорії розташовані точка входу main.py, конфігураційний файл config.py, файл залежностей requirements.txt та модуль графічного інтерфейсу gui.py. Така чітка структуризація спрощує орієнтацію в проєкті, полегшує підтримку коду та забезпечує сумісність із сучасними інструментами статичного аналізу та автоматизованого тестування.

Уся програмна система є самодостатньою та не потребує постійного підключення до зовнішніх веб-сервісів або хмарних API. Єдиною мережевою залежністю є локальне HTTP-з'єднання з сервером InfluxDB, який розгортається у тій самій локальній мережі або безпосередньо на тому самому обчислювальному вузлі. Така архітектура повністю відповідає принципу пріоритету приватності даних, що є фундаментальною властивістю екосистеми Home Assistant та однією з ключових переваг розробленої системи відносно комерційних рішень з обов'язковою хмарною обробкою.

3.4 Інтерфейс системи та основні характеристики реалізації

Для забезпечення зручності використання розробленої системи реалізовано два паралельних користувацьких інтерфейси, орієнтованих на різні сценарії взаємодії: командний інтерфейс (CLI) - для запуску у пакетному режимі, автоматизації через планувальники операційної системи та інтеграції з іншими програмними засобами; графічний інтерфейс (GUI)[12] - для інтерактивної роботи з можливістю наочної візуалізації результатів прогнозування. Обидва інтерфейси спираються на ті самі прогностичні модулі, описані у Розділі 2, і повністю взаємозамінні з функціональної точки зору.

Командний інтерфейс реалізовано у модулі main.py, що слугує єдиною точкою входу до системи у пакетному режимі. Розбір аргументів командного рядка виконується стандартною бібліотекою argparse, яка автоматично формує довідкове повідомлення (--help), валідує введені параметри та перевіряє їх несумісність. Логічна структура парсера побудована на принципі взаємовиключних режимів: користувач обов'язково вказує один з трьох сценаріїв запуску (--model, --exam або --forecast), при цьому одночасне використання двох сценаріїв заборонено механізмом add_mutually_exclusive_group(required=True).

Перший режим, --model, призначений для індивідуального запуску однієї або всіх моделей з подальшим формуванням прогнозу та збереженням результату у CSV-файл. Він приймає чотири можливі значення: prophet, xgboost, lstm або all. У трьох перших випадках навчається лише одна обрана модель та

зберігається відповідний файл прогнозу (forecast_prophet.csv, forecast_xgboost.csv або forecast_lstm.csv); у режимі all послідовно навчаються всі три моделі, формуються три CSV-файли та додатково генерується порівняльний графік comparison_result.png, що відображає прогнози всіх моделей на одній координатній сітці. Другий режим, --exam, реалізує процедуру Train-Test Split для всіх трьох моделей одночасно: вибірка ділиться хронологічно, моделі навчаються на тренувальній частині, формують прогноз на тестових 24 годинах, після чого обчислюються метрики MAE і RMSE та формується підсумкова порівняльна таблиця. Третій режим, --forecast, виконує автоматичний вибір найкращої моделі: спочатку запускається той самий екзаменаційний цикл, що й у режимі --exam, після чого з трьох моделей обирається та, що показала мінімальне значення MAE, і саме вона повторно навчається на повній історичній вибірці для формування фінального прогнозу.

На рис. 3.2 наведено приклад виводу команди python main.py --exam у консолі. Видно, що журналювання процесу здійснюється централізовано через модуль utils/logger.py з виведенням повідомлень рівня INFO у форматі HH:MM:SS [LEVEL] module: message. Усі етапи обчислення відображаються послідовно - від підключення до InfluxDB та завантаження історичної вибірки до формування прогнозу кожною моделлю та виведення підсумкової таблиці метрик. Підсумкова таблиця оформлюється функцією log_comparison_table() з модуля evaluation/metrics.py та містить значення MAE і RMSE у ватах для трьох моделей.

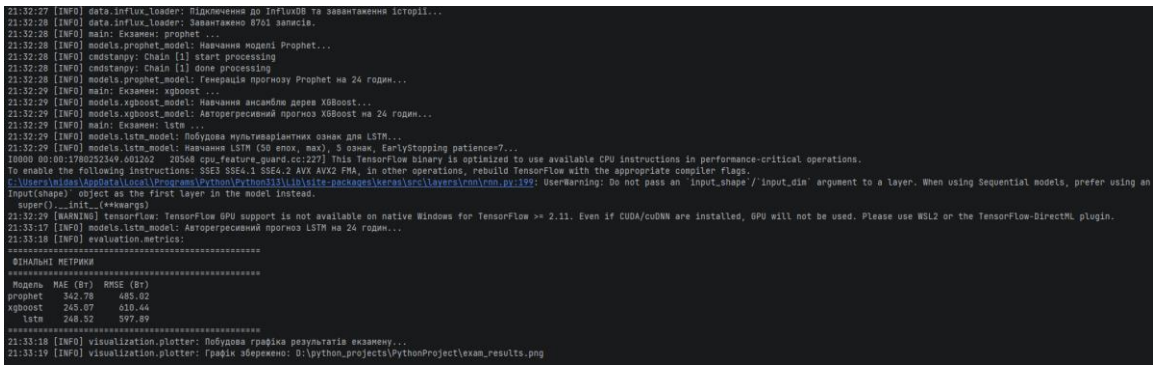

Рисунок 3.2 - Вивід команди python main.py --exam у консолі

Окремо слід відзначити декілька діагностичних повідомлень TensorFlow, що виводяться на етапі ініціалізації моделі LSTM: повідомлення про оптимізацію збірки під інструкції процесора (SSE, AVX) та попередження про недоступність прискорення на GPU під Windows. Ці повідомлення не впливають на коректність роботи системи та свідчать лише про використання центрального процесора для обчислень, що, як обґрунтовано в підрозділі 3.1, є цілком достатнім для розглядуваної задачі.

Графічний інтерфейс реалізовано у модулі gui.py із застосуванням стандартної бібліотеки tkinter, доповненої сучасними віджетами модуля ttk для прогрес-індикатора та роздільників. Візуалізація прогнозів вбудована безпосередньо у вікно програми через клас FigureCanvasTkAgg з бібліотеки matplotlib, що дозволяє створити інтегрований досвід без необхідності переключення між вікнами. Кольорова схема інтерфейсу реалізована у світлих відтінках палітри slate з акцентом синього кольору для активних кнопок, що відповідає сучасним стандартам оформлення desktop-додатків та забезпечує комфортне сприйняття при тривалій роботі.

Загальний вигляд головного вікна графічного інтерфейсу після запуску та завершення завантаження даних з InfluxDB наведено на рис. 3.3. Вікно структурно поділене на дві основні зони: ліву бічну панель «Керування» шириною 240 пікселів з елементами управління та праву центральну зону з полем візуалізації matplotlib та текстовим журналом виконання у нижній частині.

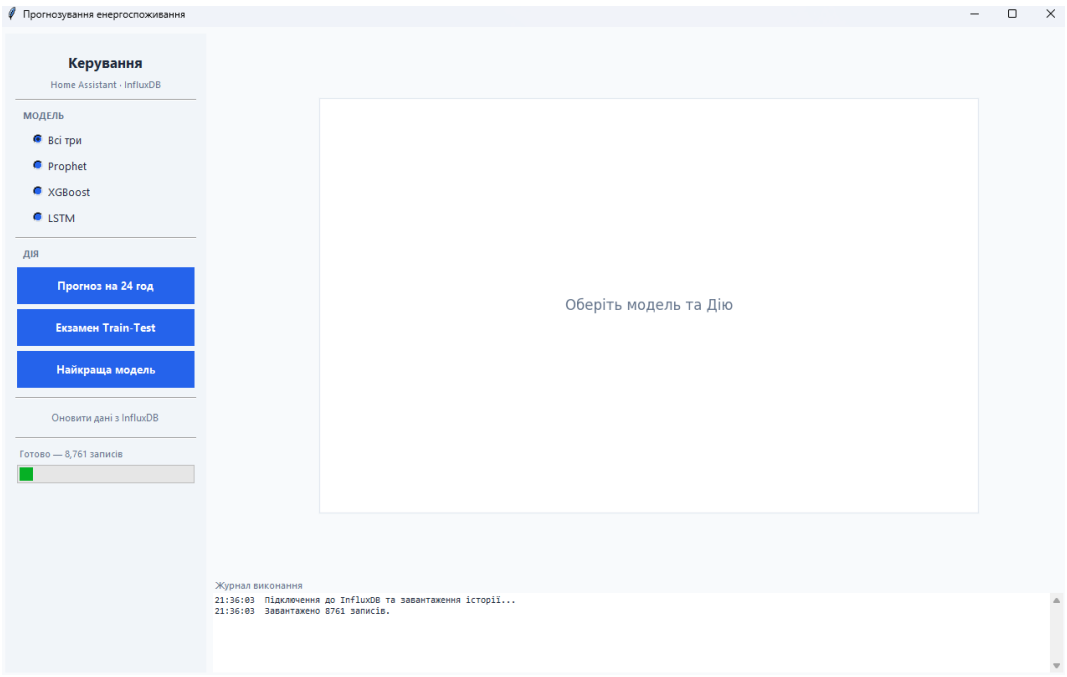

Рисунок 3.3 - Головне вікно графічного інтерфейсу після завантаження даних

Бічна панель «Керування» функціонально поділена на три блоки. Перший блок «МОДЕЛЬ» містить групу радіокнопок для вибору активної моделі: «Всі три» (значення за замовчуванням), «Prophet», «XGBoost», «LSTM». Поточний вибір зберігається у Tkinter-змінній _model_var, що дозволяє надалі звертатися до нього з обробників кнопок. Другий блок «ДІЯ» містить три головні функціональні кнопки інтерфейсу: «Прогноз на 24 год» - навчає обрану модель (або всі три, якщо обрано «Всі три») та відображає прогноз на 24 години вперед на фоні останніх 72 годин історії; «Екзамен Train-Test» - запускає процедуру Train-Test Split для всіх трьох моделей з відображенням прогнозів проти реальних даних тестової вибірки; «Найкраща модель» - автоматично визначає найточнішу модель за метрикою MAE на тестовій вибірці та виконує прогноз нею на повній історії. Третій блок містить кнопку «Оновити дані з InfluxDB», яка повторно завантажує історичну вибірку з бази даних без необхідності перезапуску програми.

Внизу бічної панелі розташовано смугу статусу з текстовим індикатором поточного стану системи (наприклад, «Готово - 8 761 записів» після успішного завантаження або «Train-Test Split для всіх моделей...» під час виконання) та

прогрес-індикатор у режимі неперервної анімації (ttk.Progressbar з параметром mode="indeterminate"), який візуально сигналізує користувачу про активне виконання фонового обчислення.

Принциповою архітектурною особливістю реалізації графічного інтерфейсу є використання багатопотокового виконання (threading.Thread) для всіх обчислювально вибагливих операцій - завантаження даних, навчання моделей та формування прогнозу. Це є необхідною умовою коректної роботи інтерфейсу, оскільки бібліотека Tkinter є однопотоковою за своєю архітектурою, і виконання тривалої обчислювальної операції у головному потоці призводить до повного «зависання» вікна на час її виконання. У реалізованому рішенні всі обробники кнопок негайно повертають управління циклу подій після створення фонового потоку, тоді як саме обчислення виконується у відокремленому потоці; після його завершення результат повертається у головний потік через виклик self.after(0, callback, result), що гарантує безпечне оновлення віджетів інтерфейсу.

Окремим інженерним рішенням є інтеграція системи логування з текстовим віджетом інтерфейсу через спеціалізований обробник _LogHandler, що успадковується від базового класу logging.Handler стандартної бібліотеки. Обробник перехоплює всі повідомлення кореневого логера програми і передає їх у віджет ScrolledText нижньої панелі через метод widget.after(0, ...), що забезпечує потокобезпечне оновлення відображення. Завдяки цьому користувач у режимі реального часу бачить всі етапи виконання процесу - від підключення до бази даних до прогнозу кожної моделі та виведення підсумкових метрик, - без необхідності переключатися до окремого вікна консолі.

На рис. 3.4 наведено приклад вікна графічного інтерфейсу одразу після завершення процедури екзаменаційного тестування (натискання кнопки «Екзамен Train-Test» з вибраним пунктом «Всі три»). У центральній зоні відображено графік порівняння реальних значень тестової вибірки (сірою суцільною лінією) з прогнозами трьох моделей (штриховими лініями у кольорах, що відповідають радіокнопкам бічної панелі: Prophet - синій, XGBoost - зелений,

LSTM - червоний). У легенді поряд з назвою кожної моделі наведено отримане значення MAE для зручності візуального порівняння. У нижньому журналі виконання продубльовано фінальну таблицю метрик, ідентичну тій, що виводиться у консольному режимі. Статусний рядок бічної панелі автоматично оновлюється, відображаючи назву моделі-переможця та її значення метрики MAE.

Рисунок 3.4 - Результати екзаменаційного тестування у графічному інтерфейсі

Сукупність описаних інженерних рішень - багатопотоковість, інтеграція matplotlib з tkinter, перенаправлення логів у графічний журнал, продуманий поділ на бічну панель та центральну зону візуалізації - забезпечує функціонально повноцінний десктопний додаток, придатний як для демонстраційних цілей, так і для повсякденного використання кінцевим користувачем без необхідності роботи з командним рядком.

4. ВЕРИФІКАЦІЯ СИСТЕМИ ПРОГНОЗУВАННЯ ЕНЕРГОСПОЖИВАННЯ

4.1 Методика експериментальних досліджень

Достовірність кількісних оцінок ефективності прогностичних моделей безпосередньо залежить від коректності експериментальної методики, відповідно до якої ці оцінки отримано. У межах розробленої системи застосовано класичну для задач машинного навчання процедуру екзаменаційного тестування за схемою Train-Test Split, адаптовану з урахуванням специфіки часових рядів. У даному підрозділі сформульовано загальні методологічні принципи проведення експерименту, які забезпечують об'єктивність та відтворюваність отриманих результатів; конкретні числові показники точності трьох прогностичних моделей наведено у наступному підрозділі.

Першим методологічним принципом є хронологічна організація розділення вибірки. На відміну від класичних задач регресії, у яких вибірка типово розділяється випадковим чином, для часових рядів допустимим є виключно хронологічне розділення: тестова вибірка завжди формується з найбільш пізніх відліків. Це принципово важлива умова, оскільки порушення хронологічного порядку призвело б до ефекту витоку інформації з майбутнього у минуле (data leakage) та необґрунтовано оптимістичних оцінок точності, які не відтворювалися б у реальних умовах експлуатації системи. У розробленій системі цю вимогу реалізовано на рівні функції train_test_split модуля підготовки даних, яка детерміновано відрізає останні TEST_SIZE_HOURS = 24 годин від вибірки незалежно від моделі чи запуску.

Другим методологічним принципом є ідентичність вхідних даних для всіх моделей, що порівнюються. Для забезпечення коректного порівняння всі три моделі отримують однакову тренувальну та тестову вибірки; будь-які попередні перетворення (конструювання ознак, нормалізація) виконуються після розділення і виключно на тренувальній частині, після чого ті самі параметри застосовуються до тестової. Такий підхід гарантує, що відмінності у показниках

точності між моделями зумовлені виключно властивостями самих алгоритмів, а не випадковими розбіжностями у вхідних даних.

Третім методологічним принципом є відтворюваність експерименту. Усі стохастичні компоненти системи (генератор синтетичних даних, ініціалізація ваг нейронної мережі LSTM, внутрішні процедури ансамблю XGBoost) ініціалізуються фіксованим значенням random_state = 42, що дозволяє при повторних запусках на тому самому наборі даних отримувати ідентичні результати. Винятком є процедура навчання нейронної мережі, у якій залишається невелика залишкова варіація на рівні десятих часток вата, зумовлена особливостями паралельних обчислень у бібліотеці TensorFlow. Параметри усіх етапів експерименту централізовано задані у конфігураційному файлі config.py, що додатково гарантує відсутність неконтрольованих відмінностей між запусками.

Четвертим методологічним принципом є об'єктивність числових оцінок. Як показники якості прогнозу використано дві поширені метрики регресії - середню абсолютну похибку (MAE) та квадратний корінь з середньої квадратичної похибки (RMSE)[11]. Спільне використання двох метрик дозволяє отримати повнішу картину якості прогнозу: MAE характеризує типову помилку, тоді як RMSE додатково сигналізує про наявність окремих критичних відхилень. Обчислення обох метрик реалізовано через функцію compute_metrics модуля evaluation/metrics.py з використанням стандартних реалізацій бібліотеки scikit-learn, що виключає можливі помилки власної реалізації та забезпечує сумісність результатів з прийнятими у науковій спільноті стандартами оцінювання.

Дотримання сформульованих принципів дозволяє розглядати отримані експериментальні результати як об'єктивну характеристику ефективності реалізованих прогностичних моделей у заданих умовах задачі. Конкретні числові показники, отримані за описаною методикою, та їх ґрунтовний аналіз наведено у підрозділі 4.2.

4.2 Експериментальні дослідження точності прогнозування

Експериментальні дослідження є ключовою практичною складовою верифікації розробленої системи, оскільки саме на цьому етапі формуються об'єктивні кількісні оцінки ефективності реалізованих прогностичних моделей у конкретних умовах задачі. У межах підрозділу спочатку розглядаються теоретичні засади метрик оцінки точності прогнозу, потім описується методика проведення експерименту, а на завершення подаються отримані числові результати з їх інтерпретацією та порівняльним аналізом.

Для кількісної оцінки точності прогнозів реалізовано дві найпоширеніші у задачах регресії метрики: середню абсолютну похибку (Mean Absolute Error, MAE) та квадратний корінь з середньої квадратичної похибки (Root Mean Squared Error, RMSE). Обидві метрики обчислюються на тестовій вибірці шляхом порівняння прогнозних значень з фактичними та виражаються у тих самих одиницях, що й цільова змінна (у ватах), що забезпечує їх безпосередню фізичну інтерпретацію кінцевим користувачем системи.

Середня абсолютна похибка обчислюється як середнє арифметичне модулів різниць між фактичними та прогнозними значеннями за виразом:

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \qquad (4.1)$$

де n - кількість відліків тестової вибірки (у розглядуваній задачі n=24, що відповідає горизонту прогнозу); y_i - фактичне значення активної потужності у момент часу i; ŷ_i - відповідне прогнозне значення. Принциповою властивістю метрики MAE є те, що вона надає всім похибкам однакову вагу, незалежно від їх абсолютної величини, що робить її стійкою до окремих викидів та зручною для прямої інтерпретації - значення MAE дорівнює середній очікуваній помилці прогнозу у ватах.

Квадратний корінь з середньої квадратичної похибки обчислюється за виразом:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \quad (4.2)$$

де позначення є аналогічними до формули (4.1). На відміну від MAE, метрика RMSE піддає кожну похибку квадратуванню перед усередненням, що надає непропорційно більшу вагу великим відхиленням. Як наслідок, висока різниця між значеннями RMSE і MAE для однієї і тієї самої моделі є сигналом про наявність окремих відліків з аномально великою похибкою - навіть якщо середня похибка є помірною. Спільне використання обох метрик дозволяє отримати більш повну картину якості прогнозу: MAE характеризує типову помилку, тоді як RMSE додатково сигналізує про наявність та масштаб критичних відхилень. Обчислення обох метрик у розробленій системі реалізовано через функцію compute_metrics() модуля evaluation/metrics.py із використанням відповідних функцій mean_absolute_error() та mean_squared_error() бібліотеки scikit-learn.

За методикою, сформульованою у підрозділі 4.1, у результаті виконання процедури екзаменаційного тестування командою python main.py --exam отримано числові показники точності прогнозування, що зведені у порівняльну таблицю 4.1.

Таблиця 4.1 - Результати екзаменаційного тестування трьох прогностичних моделей

Модель	**MAE (Вт)**	**RMSE (Вт)**
Prophet	342.78	485.02
XGBoost	245.07	610.44
LSTM	243.62	601.45

Графічне відображення прогнозів трьох моделей на тестових 24 годинах у порівнянні з реальними значеннями активної потужності наведено на рис. 3.4. На

графіку чітко видно характер поведінки кожної з моделей при відтворенні двох пікових інтервалів споживання - вечірнього (близько 21:00 за 23 квітня) та ранкового (близько 07:00–09:00 за 24 квітня), що відповідають режиму роботи побутового водонагрівача.

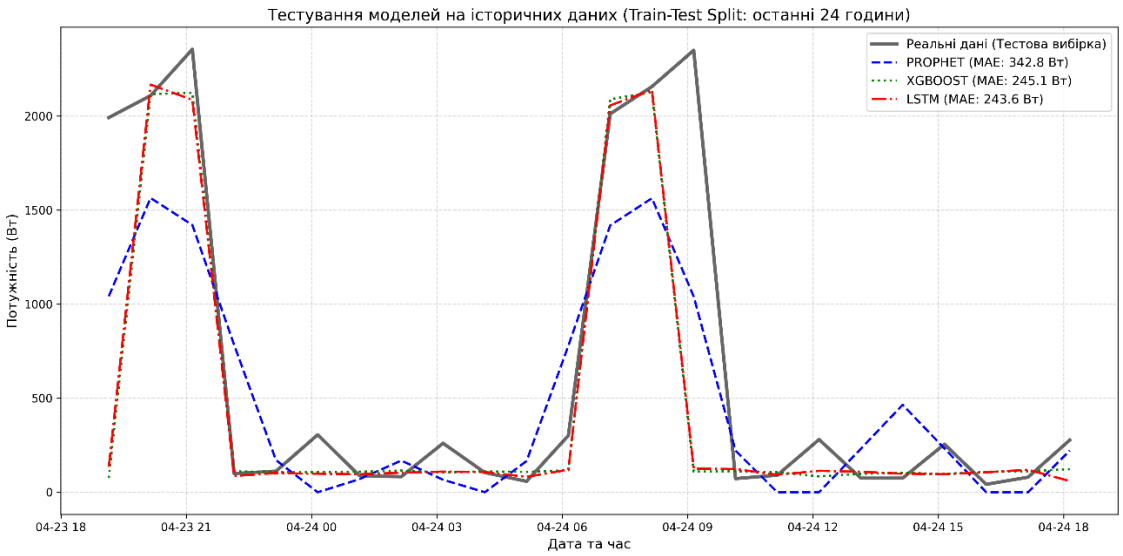

Рисунок 4.1 - Порівняння прогнозів моделей з реальними даними тестової вибірки

Аналіз отриманих результатів дозволяє сформулювати декілька важливих висновків. По-перше, моделі XGBoost та LSTM продемонстрували практично однакову точність за обома метриками з мінімальною перевагою LSTM: різниця між значеннями MAE становить лише 1.45 Вт (приблизно 0.6% від абсолютного значення), а різниця RMSE - близько 9 Вт. Така мінімальна перевага не може вважатися статистично значущою на одній реалізації тестової вибірки, оскільки знаходиться у межах випадкової варіації, зумовленої стохастичною природою процесів навчання обох моделей (зокрема, випадкової ініціалізації ваг нейронної мережі LSTM та внутрішніх процедур розбиття вибірки у XGBoost). При повторних запусках їх взаємне розташування у таблиці може змінюватися. Однак можна впевнено стверджувати, що обидва методи - ансамблеве дерево рішень та рекурентна нейронна мережа - є приблизно рівнозначними за ефективністю у розглядуваній задачі.

По-друге, модель Prophet помітно поступається обом конкурентам за метрикою MAE - її значення 342.78 Вт є приблизно на 40% вищим за показники XGBoost і LSTM. Це є очікуваним результатом і безпосередньо випливає з принципових відмінностей у природі алгоритмів: на відміну від XGBoost і LSTM, які використовують лагові ознаки минулих годин для прогнозування поточної, Prophet спирається виключно на функції часу та виявлені сезонні патерни. На графіку рис. 3.4 це проявляється в характерній закономірності - Prophet (синя штрихова лінія) систематично недооцінює амплітуду пікових споживань, прогнозуючи приблизно 1550 Вт у моменти, коли реальна потужність сягає 2200 Вт. Адитивна модель «згладжує» пікові значення, відтворюючи середній по тренувальній вибірці рівень.

По-третє, парадоксальною на перший погляд є метрика RMSE: для Prophet вона становить лише 485.02 Вт - найнижче значення серед трьох моделей, - попри найвищий показник MAE. Це пояснюється геометрією похибок: оскільки Prophet формує плавний прогноз без різких пікових значень, його похибки розподілені більш-менш рівномірно у часі без окремих катастрофічно великих відхилень. XGBoost і LSTM, навпаки, у спробі точно відтворити пікові споживання іноді промахуються з фазою або амплітудою піку, що дає окремі великі похибки, які при квадратуванні в RMSE даються взнаки. Це є цікавою методологічною ілюстрацією того, чому в задачах прогнозування часових рядів недостатньо обмежуватися однією метрикою якості - різні метрики виявляють різні аспекти поведінки моделі.

По-четверте, на графіку видно характерну особливість прогнозу Prophet, яку слід окремо відмітити: на проміжках між піками модель іноді повертає від'ємні прогнозні значення (близько 0 та нижче), що відсікаються технічно через виклик .clip(lower=0.0) у відповідній реалізації. Це підтверджує доцільність такого обмеження, реалізованого у методі predict() класу ProphetModel, оскільки активна електрична потужність за фізичним змістом не може бути від'ємною.

На завершення експериментальної частини зазначимо, що окрім графіка тестового відрізку, при запуску команди python main.py --model all система формує більш широкий порівняльний графік comparison_result.png, на якому прогнози трьох моделей побудовані на тлі останніх 72 годин історичних даних. Цей графік дозволяє візуально оцінити плавність переходу від реальної історії до прогнозованих значень та наочно демонструє, які саме закономірності реальних даних кожна з моделей відтворює у своєму прогнозі. Приклад такого графіку наведено на рис. 4.2.

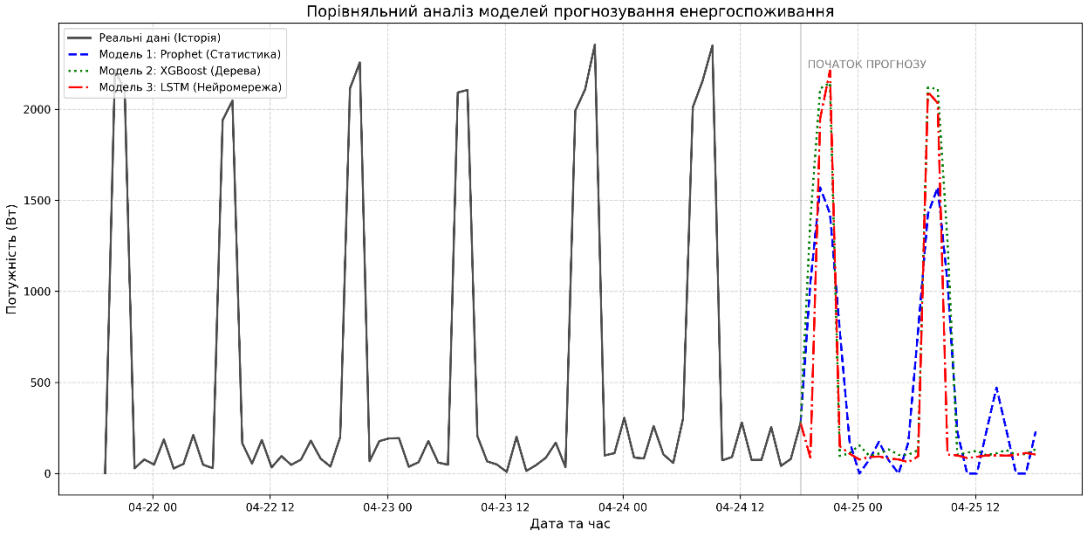

Рисунок 4.2 - Порівняльний графік прогнозів трьох моделей на тлі реальної історії

З аналізу рис. 4.2 видно, що всі три моделі коректно ідентифікували добову періодичність споживання та правильно розташували в часі пікові інтервали ранкового та вечірнього використання водонагрівача. Розбіжності полягають у амплітуді цих піків (Prophet знову демонструє занижені значення близько 1550 Вт, тоді як XGBoost і LSTM реалістично сягають 2100 Вт) та у поведінці на фонових інтервалах (Prophet іноді опускається до нульових та від'ємних значень, тоді як деревоподібний XGBoost і нейромережевий LSTM зберігають реалістичний фоновий рівень близько 100 Вт).

Сукупно отримані результати дозволяють зробити обґрунтований висновок: для задачі короткострокового прогнозування побутового

електроспоживання у заданих умовах оптимальним вибором є як XGBoost, так і LSTM з мінімальною перевагою LSTM на наявній реалізації експерименту. Адитивна модель Prophet залишається придатною як швидке базове рішення з мінімальними витратами на конструювання ознак, проте її точність на пікових значеннях помітно поступається методам, що використовують лагові ознаки. Реалізована у системі функція автоматичного вибору найкращої моделі (python main.py --forecast) забезпечує адаптивне врахування цих особливостей: у кожному окремому запуску система визначає переможця за метрикою MAE на тестовій вибірці та використовує саме цю модель для формування фінального прогнозу.

4.3 Керівництво користувача

Завершальним підрозділом розділу є практичне керівництво користувача, що описує повну послідовність дій з налаштування та використання розробленої системи для прогнозування електроспоживання. Викладені нижче кроки сформульовано у послідовності, що відповідає логічній структурі першого запуску - від конфігурації підключення до бази даних до отримання першого прогнозу - та орієнтовано на користувача із базовим рівнем володіння командним рядком, без потреби у поглибленому знанні мови Python.

Для коректного функціонування розробленої системи робоча станція користувача повинна задовольняти такі мінімальні вимоги: операційна система Microsoft Windows 10/11, macOS або одна з поширених Linux-дистрибуцій; встановлений інтерпретатор Python версії 3.10 або вищої; не менше 8 ГБ оперативної пам'яті (рекомендовано 16 ГБ для комфортної роботи з LSTM); приблизно 2 ГБ вільного дискового простору для встановлення залежностей. Окремою вимогою для користувачів Windows є наявність бібліотек виконання Microsoft Visual C++ Redistributable, які необхідні для роботи TensorFlow. Окрім самої робочої станції, у локальній мережі повинен бути доступний сервер InfluxDB версії 1.x - типово це той самий пристрій, на якому розгорнуто платформу Home Assistant.

Перед першим запуском необхідно вказати параметри підключення до сервера InfluxDB у конфігураційному файлі config.py, розташованому в кореневій директорії проєкту. Ключовими параметрами є наступні рядки:

```
INFLUX_HOST     = "192.168.1.8"       # IP-адреса сервера InfluxDB
INFLUX_PORT     = 8086                # стандартний порт InfluxDB 1.x
INFLUX_USER     = "ha_user"           # ім'я користувача
INFLUX_PASSWORD = "*****"             # пароль користувача
INFLUX_DB       = "home_assistant"    # ім'я бази даних
MEASUREMENT     = "W"                 # назва вимірювання
ENTITY_ID       = "roz_svroom_power"  # ідентифікатор сутності розетки
```

Параметри INFLUX_HOST та INFLUX_PORT повинні вказувати на сервер InfluxDB у локальній мережі. У типовій конфігурації Home Assistant з вбудованим додатком InfluxDB це IP-адреса самого пристрою з Home Assistant. Параметри INFLUX_USER та INFLUX_PASSWORD повинні відповідати обліковому запису, створеному в адміністративній панелі InfluxDB і наділеному правами читання бази home_assistant. Значення ENTITY_ID повинно точно збігатися з ідентифікатором конкретної розумної розетки у Home Assistant (зазначається без префіксу sensor.).

Якщо у момент розгортання системи накопиченої історичної вибірки в InfluxDB ще немає (наприклад, при демонстрації роботи без реального обладнання або при тестуванні системи на синтетичних даних), необхідно виконати дві допоміжні команди для генерації та імпорту синтетичної річної історії:

```
python scripts/generate_history.py
python scripts/import_to_influx.py
```

Перша команда формує файл history_365d_single_plug.csv з 8760 погодинних відліків за алгоритмом, описаним у підрозділі 3.1 (фон 40 Вт + холодильник 150 Вт з ймовірністю 40% + бойлер 2000 Вт у пікові години 7-8 та 20–21). Друга команда читає сформований CSV-файл та записує його вміст до бази InfluxDB пакетами по 1000 точок. Після успішного виконання обох команд

у консолі з'явиться повідомлення про кількість імпортованих записів, а сама база буде готова для роботи системи прогнозування.

Якщо ж користувач планує працювати з реальною телеметрією від Home Assistant, ці два кроки можна пропустити - система автоматично використає накопичені дані за умови коректного налаштування інтеграції Home Assistant з InfluxDB.

Найбільш зручним сценарієм роботи для кінцевого користувача є запуск графічного інтерфейсу, який звільняє від необхідності працювати з командним рядком після первинного розгортання. Запуск виконується однією командою:

```
python gui.py
```

Після виконання команди відкривається головне вікно програми зі світлим оформленням у відтінках палітри slate, описане раніше у підрозділі 3.4 (рис. 3.3). Безпосередньо після запуску у фоновому потоці автоматично виконується завантаження історичної вибірки з InfluxDB - це триває кілька секунд, упродовж яких у нижньому журналі виконання з'являються діагностичні повідомлення про процес підключення та кількість завантажених записів. Після завершення завантаження статусний рядок у бічній панелі змінюється з «Ініціалізація...» на «Готово - 8 761 записів», а кнопки дій стають активними.

Для виконання прогнозування користувач повинен виконати дві послідовні дії: спочатку обрати модель у блоці «МОДЕЛЬ» бічної панелі (за замовчуванням обрано «Всі три»), після чого натиснути одну з трьох кнопок блоку «ДІЯ». Кнопка «Прогноз на 24 год» формує прогноз обраної моделі та відображає його на графіку; кнопка «Екзамен Train-Test» запускає процедуру екзаменаційного тестування з виведенням метрик MAE і RMSE у журнал та порівняльного графіку у центральній зоні (рис. 3.4); кнопка «Найкраща модель» автоматично визначає найточнішу модель та виконує прогноз нею. Під час будь-якої з цих операцій кнопки тимчасово деактивуються, а прогрес-індикатор у нижній частині бічної панелі переходить у режим неперервної анімації, сигналізуючи про активне виконання.

Альтернативним сценарієм є запуск системи у пакетному режимі через командний інтерфейс. Цей режим є особливо зручним для автоматизованих сценаріїв (наприклад, періодичного виконання прогнозу через системний планувальник) та для отримання текстових результатів без графічного відображення. Перелік основних команд наведено в таблиці 4.2.

Таблиця 4.2 - Основні команди командного інтерфейсу

Команда	Дія
`python main.py --model prophet`	Навчити модель Prophet, зберегти прогноз у forecast_prophet.csv
`python main.py --model xgboost`	Навчити модель XGBoost, зберегти прогноз у forecast_xgboost.csv
`python main.py --model lstm`	Навчити модель LSTM, зберегти прогноз у forecast_lstm.csv
`python main.py --model all`	Навчити всі три моделі та сформувати порівняльний графік comparison_result.png
`python main.py --exam`	Виконати Train-Test Split для всіх моделей з виведенням таблиці метрик та графіка exam_results.png
`python main.py --forecast`	Автоматично обрати найкращу модель за MAE та сформувати прогноз нею

Усі перераховані команди супроводжуються детальним виведенням журналу у консолі, як було продемонстровано на рис. 3.2. Час виконання кожної команди залежить від обраної моделі: Prophet і XGBoost завершуються за декілька секунд, тоді як LSTM (за рахунок навчання нейронної мережі) триває від 30 секунд до однієї хвилини.

Сформовані системою файли результатів мають такі функціональні призначення. CSV-файли прогнозів (forecast_prophet.csv, forecast_xgboost.csv, forecast_lstm.csv) містять дві колонки: timestamp (часова мітка у форматі ISO 8601) та prediction (прогнозне значення активної потужності у ватах). Ці файли можуть бути імпортовані до будь-якого зовнішнього аналітичного засобу - електронної таблиці, бази даних або сценарію автоматизації Home Assistant - для подальшого використання прогнозу.

Графічний файл comparison_result.png (приклад наведено на рис. 4.2) дозволяє візуально оцінити характер прогнозу кожної моделі у контексті останніх 72 годин історичних даних. Графічний файл exam_results.png (приклад на рис. 3.4) є ключовим аналітичним інструментом для оцінки якості системи: він показує, наскільки точно кожна з трьох моделей здатна відтворити реальні дані на тестовій вибірці. У легенді цього графіка зазначено значення MAE для кожної моделі, що дозволяє швидко визначити переможця експерименту.

Інтерпретація числових метрик MAE та RMSE здійснюється у тих самих одиницях, що й сама прогнозована величина - у ватах. Менші значення відповідають кращій точності прогнозу. Як показано у попередньому підрозділі, для синтетичної вибірки розглядуваного типу значення MAE моделей XGBoost і LSTM знаходяться у діапазоні приблизно 240-250 Вт, що означає середню очікувану похибку прогнозу одиничної години у вказаних межах. Для адитивної моделі Prophet типова MAE становить близько 340 Вт.

Окрім основних сценаріїв використання, у складі проєкту реалізовано три допоміжні скрипти, які можуть знадобитися при тривалій експлуатації системи. Скрипт python scripts/daily_generator.py додає у базу InfluxDB 24 нові синтетичні погодинні точки, починаючи з моменту після останньої наявної записи; цей сценарій призначений для періодичного автоматичного запуску через планувальник операційної системи (cron у Linux або Task Scheduler у Windows) для імітації безперервного збору телеметрії. Скрипт python scripts/rollback_data.py видаляє останні ROLLBACK_DAYS діб даних з бази (за замовчуванням 3) і використовується для виправлення помилково доданих записів. Перед його запуском рекомендується перевірити поточне значення константи ROLLBACK_DAYS у файлі config.py, оскільки операція є незворотною.

Виконання наведених у керівництві кроків дозволяє користувачу самостійно налаштувати розроблену систему та використовувати її як для

дослідницьких цілей, так і для повсякденного прогнозування електроспоживання у домашній інфраструктурі.

ВИСНОВКИ

У результаті виконання дипломної роботи розроблено та програмно реалізовано модульну систему короткострокового прогнозування електроспоживання в інфраструктурі розумного дому, побудовану на базі відкритої платформи Home Assistant та бази даних часових рядів InfluxDB. Система забезпечує паралельну реалізацію трьох прогностичних моделей різної алгоритмічної природи - Prophet, XGBoost та LSTM - з можливістю об'єктивного порівняльного дослідження їх ефективності на ідентичних вхідних даних. Розроблена система функціонує локально без передачі телеметрії у хмарні сервіси, що відповідає сучасним вимогам до приватності даних.

У першому розділі виконано аналіз сучасного стану галузі прогнозування побутового енергоспоживання, оглянуто характеристики платформи Home Assistant та проведено порівняльний огляд методів прогнозування часових рядів. На основі цього аналізу обґрунтовано вибір трьох прогностичних моделей різних алгоритмічних парадигм та сформульовано перелік задач дипломної роботи.

У другому розділі викладено теоретичні засади розробленої системи, спроєктовано багаторівневу архітектуру з чітким розподілом відповідальностей між підсистемами збору даних, їх підготовки та машинного навчання, а також детально розглянуто структуру програмних модулів і класів. Усі три прогностичні моделі організовано як підкласи спільного абстрактного класу BaseModel за шаблоном проєктування «Шаблонний метод», що забезпечує уніфікований програмний інтерфейс та можливість легкого розширення системи новими алгоритмами.

У третьому розділі описано програмну реалізацію всіх ключових компонентів системи. Реалізовано модуль синтетичної генерації річної історії погодинного електроспоживання з реалістичним профілем навантаження, підсистеми збору телеметрії та її попередньої обробки, а також безпосередньо прогностичне ядро з трьома моделями машинного навчання. Розроблено два

паралельні користувацькі інтерфейси - командний (CLI) та графічний (GUI) з вбудованою візуалізацією та багатопотоковим виконанням обчислень.

У четвертому розділі виконано верифікацію розробленої системи. Сформульовано методологічні принципи експериментального дослідження, проведено об'єктивне порівняння точності трьох прогностичних моделей за процедурою Train-Test Split та проаналізовано отримані результати. Встановлено, що моделі XGBoost і LSTM показали практично однакову точність та значно перевершили Prophet через відсутність у останнього використання лагових ознак. Складено покрокове керівництво користувача, що дозволяє самостійно розгорнути систему у домашній інфраструктурі.

Поставлену мету дипломної роботи досягнуто у повному обсязі, всі сформульовані задачі послідовно виконано. Практична цінність розробленої системи полягає у можливості її безпосереднього впровадження у домашню інфраструктуру користувачів платформи Home Assistant без вимог до спеціалізованої апаратної інфраструктури. Відкритий характер реалізації та модульна архітектура відкривають можливості для подальшого розвитку у напрямах розширення на множинні точки моніторингу, реалізації гібридних прогностичних підходів та інтеграції з підсистемами автоматизації для динамічного управління навантаженням.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Hyndman R. J. Forecasting: Principles and Practice [Електронний ресурс] / R. J. Hyndman, G. Athanasopoulos. – 3rd ed. – OTexts, 2021. – Режим доступу: https://otexts.com/fpp3/ – Назва з екрана.
2. Taylor S. J. Forecasting at Scale [Електронний ресурс] / S. J. Taylor, B. Letham // The American Statistician. – 2018. – Vol. 72, No. 1. – P. 37–45. – Режим доступу: https://peerj.com/preprints/3190.pdf – Назва з екрана.
3. Chen T. XGBoost: A Scalable Tree Boosting System [Електронний ресурс] / T. Chen, C. Guestrin // Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. – 2016. – P. 785–794. – Режим доступу: https://dl.acm.org/doi/10.1145/2939672.2939785 – Назва з екрана.
4. Hochreiter S. Long Short-Term Memory [Електронний ресурс] / S. Hochreiter, J. Schmidhuber // Neural Computation. – 1997. – Vol. 9, No. 8. – P. 1735–1780. – Режим доступу: https://direct.mit.edu/neco/article/9/8/1735/6109/Long-Short-Term-Memory – Назва з екрана.
5. Home Assistant. Integrations [Електронний ресурс] / Home Assistant. – Режим доступу: https://www.home-assistant.io/integrations/ – Назва з екрана.
6. Home Assistant. InfluxDB Integration [Електронний ресурс] / Home Assistant. – Режим доступу: https://www.home-assistant.io/integrations/influxdb/ – Назва з екрана.
7. InfluxData. InfluxDB OSS v1 Documentation [Електронний ресурс] / InfluxData. – Режим доступу: https://docs.influxdata.com/influxdb/v1/ – Назва з екрана.
8. MQTT Version 5.0 [Електронний ресурс] / OASIS Standard. – 2019. – Режим доступу: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html – Назва з екрана.
9. TensorFlow. tf.keras.layers.LSTM [Електронний ресурс] / TensorFlow. – Режим доступу: https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM – Назва з екрана.

10. pandas. Time series / date functionality [Електронний ресурс] / pandas. – Режим доступу: https://pandas.pydata.org/docs/user_guide/timeseries.html – Назва з екрана.
11. scikit-learn. Metrics and scoring: quantifying the quality of predictions [Електронний ресурс] / scikit-learn. – Режим доступу: https://scikit-learn.org/stable/modules/model_evaluation.html – Назва з екрана.
12. Python Software Foundation. The Python Standard Library [Електронний ресурс] / Python Software Foundation. – Режим доступу: https://docs.python.org/3/library/ – Назва з екрана.