НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

«На правах рукопису»
УДК 004.9, 681.513.7

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
«___»_____2024 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-науковою програмою «Інженерія програмного забезпечення комп'ютерних та інформаційних систем»

зі спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Метод навчання з підкріпленням для управління витратами на опалення»

Виконав:
студент ІІ курсу, групи ІП-21мн
Кришталь Віктор Олегович _____

Керівник:
д.т.н., професор кафедри
інформатики та програмної інженерії
Сидоров Микола Олександрович _____

Рецензент:
к.т.н., доцент кафедри
обчислювальної техніки
Долголенко Олександр Миколайович _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2024 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – другий (магістерський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-наукова програма «Інженерія програмного забезпечення комп'ютерних та інформаційних систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Едуард ЖАРІКОВ

«___»_____2024 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Кришталю Віктору Олеговичу

1. Тема дисертації «Метод навчання з підкріпленням для управління витратами на опалення», науковий керівник дисертації Сидоров Микола Олександрович, д.т.н., професор кафедри інформатики та програмної інженерії, затверджені наказом по університету від «23» березня 2024 р. № 1445-с

2. Термін подання студентом дисертації «9» червня 2024 р.

3. Об'єкт дослідження – програмне забезпечення керування об'єктами.

4. Предмет дослідження – підходи, методи, моделі, засоби керування об'єктами, застосовуючи методи машинного навчання, що реалізовані програмно.

5. Перелік завдань, які потрібно розробити – аналіз застосування підходу навчання з підкріпленням до задачі оптимізації витрат на опалення; аналіз винагород підходу навчання з підкріпленням для задачі оптимізації витрат на опалення; розробка удосконаленого підходу навчання з підкріпленням,

що спрощує практичне застосування до задач керування об’єктами; дослідження ефективності розробленого методу; створення, налагодження та тестування серверного застосунку, що реалізує розроблений метод.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу – 1 плакат

7. Орієнтовний перелік публікацій – одна публікація
8.Дата видачі завдання «5» лютого 2023 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання	Примітка
1	Отримання завдання	05.02.2023	
2	Огляд літератури	04.03.2023	
3	Розробка удосконаленого методу навчання з підкріпленням	12.06.2023	
4	Розробка спрощеної моделі приміщення	24.09.2023	
5	Аналіз ефективності розробленого методу та моделі	02.11.2023	
6	Проектування серверного застосунку	15.01.2024	
7	Розробка серверного застосунку	17.03.2024	
8	Оформлення пояснювальної записки	10.04.2024	
9	Подання дисертації на попередній захист	19.05.2024	
10	Подання дисертації на захист	09.06.2024	

Студент Віктор КРИШТАЛЬ

Науковий керівник Микола СИДОРОВ

РЕФЕРАТ

Розмір пояснювальної записки – 126 аркушів, містить 22 ілюстрації, 6 таблиць, 3 додатки, 59 посилань на джерела.

Актуальність теми. У роботі розглянуто задачу керування опаленням та оптимізації витрат на опалення. Розглянуто застосування підходу навчання з підкріпленням до цієї задачі, його основні особливості, переваги та недоліки. Виявлено потребу в удосконаленні цього підходу для спрощення його практичного застосування до даної задачі.

Мета дослідження. Метою дослідження є мінімізація витрат на опалення приміщень шляхом створення застосунку, який керує нагрівальними панелями.

Об’єкт дослідження: програмне забезпечення керування об’єктами.

Предмет дослідження: підходи, методи, моделі, засоби керування об’єктами, застосовуючи методи машинного навчання, що реалізовані програмно.

Для реалізації поставленої мети **сформульовані наступні завдання**:

- проаналізувати застосування підходу навчання з підкріпленням до задачі оптимізації витрат на опалення, проаналізувати можливий вигляд винагород у цьому підході, порівняти їх ефективність та вибрати оптимальну;
- розробити удосконалену версію цього підходу, що використовує спрощену модель середовища, що будується на основі зібраних даних без втручання користувача, та дозволяє спростити практичне застосування;
- розробити програмну реалізацію цього підходу для даної задачі та створити серверний застосунок з веб-клієнтом, що дозволяє користувачу керувати та слідкувати за витратами на опалення.

Наукова новизна: вперше запропоновано архітектуру програмного забезпечення керування витратами на опалення із використанням

модифікованого методу навчання з підкріпленням, в якому застосовується спрощена модель приміщення, параметри якої знаходяться експериментальним шляхом.

Практичне значення отриманих результатів полягає в тому, що запропоновано метод керування опаленням, що дозволяє значно підвищити економічну ефективність опалення приміщень, враховуючи денну варіацію цін.

Зв'язок з науковими програмами, планами, темами. Робота виконувалась на кафедрі інформатики та програмної інженерії Національного технічного університету України "Київський політехнічний інститут імені Ігоря Сікорського".

Апробація. Наукові положення дисертації пройшли апробацію на VI Міжнародній науково-практичній конференції молодих вчених та студентів «Інженерія програмного забезпечення і передові інформаційні технології SoftTech-2024» – м. Київ.

Публікації. Наукові положення дисертації опубліковані в:

1) Кришталь В.О.. Метод навчання з підкріпленням для управління витратами на опалення // Матеріали науково-практичної конференції молодих вчених та студентів «Інженерія програмного забезпечення і передові інформаційні технології SoftTech-2024» – м. Київ: КПІ ім. Ігоря Сікорського, 21-23 травня 2024 р.

Ключові слова: НАВЧАННЯ З ПІДКРІПЛЕННЯМ, ГЛИБОКЕ НАВЧАННЯ, СИСТЕМИ УПРАВЛІННЯ, СИМУЛЯЦІЯ.

ABSTRACT

Explanatory note size – 126 pages, contains 22 illustrations, 6 tables, 3 appendices, 59 references.

Topicality. Examines the problem of heating control and heating expense optimization. Examines application of reinforcement learning to this problem, its benefits and drawbacks. It was determined that this approach needs improvements to simplify its practical application to this problem.

The aim of the study. The target of the study is to minimize building heating costs by creating an application, that controls heating panels.

The object of research: object control software.

The subject of research: approaches, methods and applications for object control using machine learning.

To achieve this goal, the **following tasks** were formulated:

- analyze application of reinforcement learning to the problem of heating expense optimization, analyze possible rewards in this approach, compare their effectiveness and choose the optimal one;
- develop an improved version of this approach, that uses a simplified physical model of environment, that is built from collected data without user intervention and can simplify practical usage;
- develop a software implementation of this aproach to the given task and create a server application with web-client, that allows a user to control and monitor heating expenses.

The scientific novelty of the results of the master's dissertation is that a software architecture for managing heating costs using a modified reinforcement learning method has been proposed, in which a simplified room model is used, with parameters determined experimentally.

The practical value of the obtained results is that a heating control method was proposed, which allows to increase economic efficiency of building heating, taking into accout daily price variation.

Relationship with working with scientific programs, plans, topics. Work was performed at the Department of Informatics and Software Engineering of the National Technical University of Ukraine «Kyiv Polytechnic Institute. Igor Sikorsky».

Approbation. The scientific provisions of the dissertation were tested at the Sixth International Scietific and Practical Conference of Young Scientists and Students "Software engineering and Advanced Information Technologies Softtech-2024" – Kyiv.

Publications. The scientific provisions of the dissertation were published in:

1) Kryshtal V.O. Reinforcement learning method for heating cost management // Proceedings of the Sixth International Scietific and Practical Conference of Young Scientists and Students "Software engineering and Advanced Information Technologies Softtech-2024" – Kyiv: NTUU "KPI them. Igor Sikorsky", May 21-23, 2024.

Keywords: REINFORCEMENT LEARNING, DEEP LEARNING, CONTROL SYSTEMS, SIMULATION.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

HVAC (heating, ventilation, air conditioning) – системи опалення, вентиляції та кондиціонування повітря;

IoT (internet of things) – інтернет речей;

DR (demand resource management) – управління ресурсами згідно з попитом;

NN, ANN ((artificial) neural networks) – (штучні) нейронні мережі;

LR (logistic regression) - логістична регресія;

LDA (linear discriminant analysis) – лінійний дискримінантний аналіз;

KNN (k-nearest neighbours) – метод k-найближчих сусідів;

CT (classification trees) – дерева класифікації;

SVM (support vector machine) – метод опорних векторів;

SEM (structural equation modelling) – метод моделювання структурних рівнянь;

RL (reinforcement learning) – навчання з підкріпленням;

DRL (deep reinforcement learning) – глибоке навчання з підкріпленням;

MDP (Markov decision process) – Марковський процес вирішування;

DQN (deep Q-networks) – метод глибоких Q-мереж;

DPG (deterministic policy gradients) – метод детерміністичного градієнта стратегії;

DDPG (deep deterministic policy gradient) – метод глибокого детерміністичного градієнта стратегії.

ВСТУП

Будівлі відповідають приблизно за 40% глобального енергоспоживання, близько половини якого використовується на опалення, вентиляцію та кондиціонування повітря (ОВК) (heating, ventilation, and air conditioning, HVAC) [1, 2], що є основними засобами контролю мікроклімату у будівлях. Крім того, будівлі відповідають за одну третину глобальних викидів парникових газів, пов'язаних з енергетикою [1]. Тому навіть невелике покращення енергоефективності будівель та систем ОВК значно сприяє створенню стійкого, економічного та енергоефективного майбутнього.

Більшість енергетичних операторів у Європі пропонують своїм клієнтам контракти, де ціна слідує за годинними змінами на ринку електроенергії Nord Pool [3]. Фактори, які впливають на ціни, включають доступну потужність виробництва, ціни на паливо, викиди парникових газів та споживання електроенергії [4]. Найбільш поширеною причиною коливань цін є погода. Тому вони можуть помітно варіюватись впродовж дня. Завдяки використанню цієї варіації, якщо прогрівати приміщення заздалегідь та вимикати нагрівач під час піку цін, можна досягти значної економії коштів, що витрачаються на електроенергію при використанні електричних обігрівачів.

Сучасні будівлі стають все більш розумними завдяки використанню пристроїв інтернету речей (internet of things, IoT). Вони підключаються до мережі та дозволяють застосовувати складні алгоритми керування для досягнення результатів, недоступних звичайним побутовим приладам. Вони дозволяють збирати дані з датчиків, оброблювати та аналізувати їх в режимі реального часу. Це відкриває можливість застосовувати алгоритми штучного інтелекту та машинного навчання.

Одним з найпоширеніших підходів машинного навчання для задач керування є навчання з підкріпленням. Це є загальний підхід з великою кількістю методів та є активною областю наукових досліджень: постійно публікуються нові методи та покращуються вже існуючі. Цей підхід намагається повторювати принцип навчання людей: за кожну дію модель отримує винагороду, яку намагається збільшити, оптимізуючи свої дії.

Проте при практичному застосуванні до задач керування фізичними об’єктами цей підхід вимагає створення моделі середовища, що моделює фізичні процеси. Це вимагає вимірювання різних фізичних параметрів приміщення, таких як фізичні розміри, товщина та матеріал стін, розміри вікон тощо. Це ускладнює використання цього підходу, тому однією з задач даної роботи є удосконалення цього методу, щоб він міг працювати без необхідності введення фізичних параметрів вручну користувачем.

Метою даної роботи є удосконалення підходу навчання з підкріпленням при застосуванні до задачі мінімізації витрат на опалення, що спрощує його практичне застосування, та розробка серверного застосунку, що його реалізує.

1 АНАЛІЗ МЕТОДІВ НАВЧАННЯ З ПІДКРІПЛЕННЯМ У КОНТЕКСТІ КЕРУВАННЯ ОПАЛЕННЯМ

1.1 Огляд існуючих рішень оптимізації опалення

Системи опалення, вентиляції та кондиціонування повітря (ОВК) є одними з найбільш широко використовуваних і споживають найбільше енергії у будівлях. Відповідно, оптимальне керування системами ОВК може покращити споживання електроенергії, знизити ціни на електроенергію та одночасно зменшити викиди парникових газів. Оптимізація функцій ОВК не є новою областю досліджень. Вона широко вивчається як частина управління відповідно до попиту (demand resource management, DR management), яке також включає підходи до зміни використання електроенергії та динамічного контролю цін. Існуючі методи покращення енергоефективності ОВК будівель можна категоризувати таким чином: традиційні математичні методи на основі правил (ruled-based methods), методи на основі моделей (model-based methods) та методи на основі даних (data-driven methods).

Методи на основі правил - це прості евристичні методи. Зазвичай вони базуються на відомих даних і покладаються на моніторинг певного «вимикаючого» параметра (наприклад, температура в кімнаті), для якого фіксується порогове значення для керування системою відповідно до попередньо визначеної стратегії. Наприклад, дослідження, проведені в [5], досліджували алгоритми керування на основі правил DR в кількох типах будівель в Фінляндії, що базувалися на цінах на електроенергію для керування заданою температурою приміщень. Розглядався алгоритм, що використовує погодинні значення ціни на електроенергію в реальному часі та алгоритм, що використовує ціну за минулу та наступну години. Було визначено, що алгоритм керування на основі попередніх годинних цін на

електроенергію є найефективнішим алгоритмом у більшості досліджених випадків. Порівняно з референтним випадком (встановлення постійної внутрішньої температури обігріву на рівні 21 ◦C), максимальна загальна енергія, яка була заощаджена за допомогою алгоритмів керування, становила приблизно 3%, а економія коштів - від 6 до 14%, залежно від типу будинку, системи розподілу тепла та параметрів, що використовували алгоритми. Хоча стратегії керування на основі правил мають перевагу в простоті, вони мають кілька недоліків, зазвичай пов'язаних з їхньою слабкою динамікою. Моделі на основі правил можуть бути складними в обслуговуванні через можливі зміни протягом життя будівлі. Незважаючи на цей недолік адаптації, динамічності та передбачуваності, стратегії керування на основі правил становлять більшість комерційних реалізацій DR [6, 7].

У керуючих алгоритмах на основі моделі деякі параметри передбачаються, що призводить до більш надійної, але складної стратегії керування. Наприклад, контрольні алгоритми систем ОВК на основі моделі для мінімізації загальних енергозатрат для кінцевих користувачів були досліджені в [8]. Однак підходи на основі моделі мають обмежену практичну придатність через складність моделі прогнозування та велику потребу в обсягу пам'яті, необхідному для оптимізації в реальному часі. Обчислювальна складність експоненційно зростає зі складністю будівлі та структури енергетичної мережі [9, 10]. Кілька досліджень вказали на те, що підходи на основі моделей долають обмеження, з якими стикаються прості правила керування, і перевершують їх за результатами [11, 12].

Натомість методи, засновані на даних та штучному інтелекті (AI data-driven methods), були продемонстровані як більш гнучкі [13] та здатні впливати на операції систем ОВК шляхом налаштування параметрів керування (наприклад, температури), використовуючи історичні дані про

роботу і зайнятість будівлі, а також дані навколишнього середовища (наприклад, погодні умови). Гнучкість полягає у можливості алгоритмів машинного навчання вчитися на основі історичних даних про роботу будівлі та відповідно налаштовувати функції систем ОВК. Крім того, порівняно з традиційними моделями на основі правил, підходи, засновані на даних, потребують менше експертних знань в області та не вимагають опису фізичної динаміки будівлі.

Багато досліджень, заснованих на даних (data-driven models), використовують навчання з учителем (supervised learning). Наприклад, автори праці [14] застосували метод глибокого детерміністичного градієнта стратегії (deep deterministic policy gradient, DDPG) для короткострокового енергоспоживання систем ОВК для опалення та охолодження в невеликих офісних приміщеннях. Вони отримали, що запропонована модель дає більш точні результати, ніж стандартні моделі навчання з учителем, такі як метод опорних векторів (support vector machine, SVM) та нейронні мережі (neural networks, NN). У роботі [3] досліджено великі комерційні будівлі, де для покращення прогнозування температури в кожній зоні для побудови енергоефективних систем ОВК було запропоновано використовувати метод моделювання структурних рівнянь (structural equation modelling, SEM).

Аналіз поведінки мешканців та їх взаємодії з системами ОВК також може допомогти краще забезпечити тепловий комфорт мешканців, водночас зберігаючи енергію. У роботі [15] розроблено модель машинного навчання для опалення приміщень, яка може визначити поведінку мешканців, яка призводить до даремної витрати енергії при роботі систем ОВК.

Вплив різних моделей прогнозування зайнятості приміщення, що використовують техніки машинного навчання, було проаналізовано в [16]. Декілька технік машинного навчання (дерева рішень, метод k-найближчих

сусідів, багатошаровий персептрон та вентильні рекурентні вузли) були використані для прогнозування типів та шаблонів зайнятості та для надання оцінки ефективності використання моделі зайнятості у системах керування ОВК. У роботі [17] були запропоновані декілька моделей навчання з учителем для прогнозування рівнів комфорту мешканців: метод опорних векторів (SVM), штучна нейронна мережа (artificial neural network, ANN), логістична регресія (logistic regression, LR), лінійний дискримінантний аналіз (linear discriminant analysis, LDA), k-найближчих сусідів (k-nearest neighbours, KNN) та дерева класифікації (classification trees, CT).

Також для вивчення оптимальних параметрів керування з використанням історичних даних використовуються еволюційні алгоритми. Наприклад, у [18] використовується еволюційний алгоритм для пошуку оптимальних параметрів керування (а саме температури та статичного тиску подачі повітря) системи ОВК на основі моделі, заснованої на даних (data-driven model).

У [19] запропоновано оптимізацію охолодження систем ОВК на основі генетичних алгоритмів для оптимізації контролера та методів навчання з учителем для моделювання системи ОВК. У [20] розглянуто оптимальне цінове керування системами ОВК в багатозонних офісних будівлях для реагування на попит. Змінні теплові уподобання мешканців моделюються за допомогою штучної нейронної мережі (ANN) і включаються до оптимального планування роботи системи ОВК. Крім того, розроблено механізм керування для визначення налаштувань термостату системи ОВК в різних зонах на основі результатів моделі прогнозування ANN.

Методи на основі алгоритмів навчання з учителем можуть потребувати великої кількості позначених даних. Відповідно, ефективність

підходів навчання з учителем залежить від якості історичних даних будівлі, які можуть бути недоступними. Крім того, у разі зміни обладнання або користувачів ці дані можуть застаріти, і ефективність навчених алгоритмів машинного навчання може зменшитися.

Для вирішення цих проблем потрібен підхід, заснований на даних (data-driven approach), який може вивчати оптимальні параметри керування з історичних даних для оптимізації роботи системи ОВК. Для вирішення проблем такого типу перспективними є підходи на основі навчання з підкріпленням (reinforcement learning, RL). В таких методах програмний агент повинен вивчити оптимальну або майже оптимальну стратегію, що максимізуватиме винагороду, визначену користувачем. Крім того, підходи на основі навчання з підкріпленням для керування опаленням та охолодженням та оптимізації прийняття рішень в реальному часі вимагають малої кількості історичних даних.

Розглянемо дослідження, що застосовують стратегію керування на основі навчання з підкріпленням для оптимізації роботи систем ОВК будівель [21]. У [22] було досліджено застосування дискретного та неперервного підходів до навчання з підкріпленням, які активно вивчають, як належним чином розпланувати температурні налаштування термостатів на основі вподобань комфорту мешканців. [23] використовували навчання з підкріпленням, зокрема Q-навчання (Q-learning), для оптимізації використання активного та пасивного запасу теплової енергії в будівлі. У [24] було досліджено інтелектуальне керування температурою в контрольованих зонах будівлі, шляхом вивчення характеристик обладнання ОВК та звичок мешканців. [25] застосували стратегії керування на основі навчання з підкріпленням для побудови системи, що реагує на попит, щоб досягти 90% математично оптимального розв'язку. [26] застосували алгоритми навчання з підкріпленням до системи ОВК з тепловим насосом,

досягнувши значних енергозбережень. [27] запропонували великомасштабне навчання з підкріпленням для прискорення процесу знаходження оптимальних стратегій керування. [28] запропонували метод керування системою ОВК на основі глибокого навчання з підкріпленням (deep reinforcement learning, DRL). У [29] була запропонована архітектура навчання з підкріпленням для ефективного планування та керування системою ОВК в комерційній будівлі з використанням реагування на попит (demand response, DR). Симуляція продемонструвала досягнення тижневого зменшення споживання енергії до 22% порівняно з базовим контролером.

[30] запропонували модель оптимізації енергії на основі навчання з підкріпленням, яка застосовується в реальному середовищі фабрик (повідомлено про час навчання приблизно декілька тижнів) та може забезпечити приблизно 25% енергозбережень порівняно з базовим контролером. Метою оптимізації системи ОВК було підтримувати температуру та (відносну) вологість в межах встановлених допустимих діапазонів для виробничих потреб, а також збалансувати це з енергозбереженням та зменшенням викидів CO2.

[31] запропонували підхід глибокого навчання з підкріпленням (DRL) для керування опаленням будівель з метою автоматизації прийняття рішень в реальному часі з мінімальною залежністю від історичних даних. У симуляціях в якості вхідних даних використовувалися реальні дані зовнішньої температури, але сталий тариф на електроенергію. Повідомлялося, що інтелектуальний контролер на основі DRL перевершує традиційний термостат, покращуючи тепловий комфорт на 15–30% та зменшуючи витрати на енергію від 5% до 12% в умовах симуляції.

1.2 Підхід навчання з підкріпленням

1.2.1 Спрощений опис підходу навчання з підкріпленням

Основна концепція навчання з підкріпленням полягає в тому, щоб навчити агента виконувати певні дії в середовищі з метою максимізації якості деякого показника результативності, який називається "винагородою".

Основна ідея полягає в тому, що агент може спробувати різні дії в своєму середовищі, і на основі результатів кожної дії він отримує зворотний зв'язок у вигляді винагороди або покарання. Цей зворотний зв’язок ще називають підкріпленням, звідси назва методу – навчання з підкріпленням. Підкріплення може бути позитивним (нагородою) або негативним (покаранням), в залежності від того, наскільки дія сприяє досягненню мети агента.

Ця концепція базується на ідеї максимізації винагороди (або мінімізації покарання) через взаємодію агента з його оточенням. Для досягнення цієї мети агент використовує різні стратегії, які вивчаються шляхом проб та помилок і використанням здобутої в процесі навчання інформації.

Основні компоненти навчання з підкріпленням (рис. 1.1):

- Агент: Суб’єкт, який взаємодіє з середовищем та вчиться приймати оптимальні дії для досягнення мети.
- Середовище: Місце, в якому агент здійснює свої дії та з яким він взаємодіє.
- Дії: Варіанти впливу агента на середовище.
- Стан: Певна конфігурація середовища, яка описує поточний контекст.

– Винагорода: Числове значення, який надається агентові за виконання певної дії в певному стані. Якщо винагорода від'ємна, її ще називають покаранням.

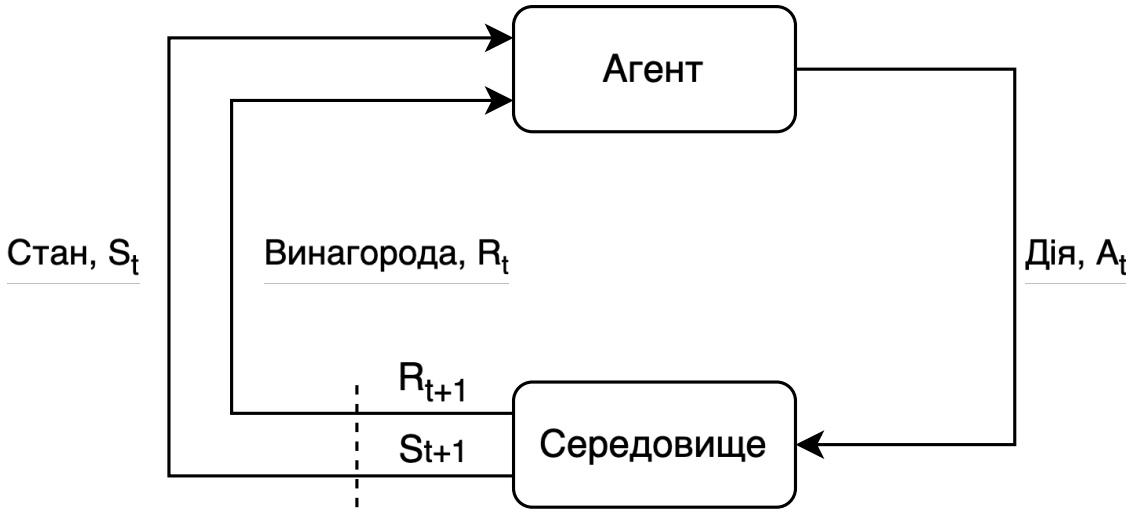


Рисунок 1.1. Високорівнева схема взаємодії агенту та середовища під час навчання з підкріпленням

Навчання з підкріпленням може бути застосоване до багатьох різних задачах. Його застосовують в системах керування, робототехніці, для знаходження стратегій в іграх (шахи, ґо, комп'ютерні ігри), самокерованих автомобілях, обробці мови та інших сферах. У табл. 1.1 наведено декілька прикладів областей, в яких застосується навчання з підкріпленням.

1.2.2 Формальний опис підходу навчання з підкріпленням

Формально середовище навчання з підкріпленням описується за допомогою Марковського процесу вирішування (МПВ або Markov decision process, MDP).

Таблиця 1.1. Приклади застосування навчання з підкріпленням

Область застосування	Стани	Дії	Винагороди
Ігри	Для настільних ігор (наприклад, шахи або го) стан ігрової дошки. Для комп'ютерних ігор – зображення, що виводиться на екран кожен кадр	Доступні у грі дії	Можуть бути як прості винагороди, наприклад, винагорода за перемогу і покарання за поразку, так і складні винагороди, засновані на механіці конкретних ігор
Робототехніка	Стани з датчиків та камер робота	Дії, які може виконати робот	Винагороди залежать від задач, яким хочуть навчити робота
Системи керування на підприємствах	Дані датчиків	Параметри, які можна регулювати	Параметри, які намагаються оптимізувати. Наприклад, якщо ціллю є електрозбереження, то агент буде отримувати покарання, пропорційне до витраченої енергії
Передбачення часових рядів, наприклад, для передбачення цін на акції	Поточні та історичні дані	Дією є передбачення наступного значення часового ряду	Агент отримує покарання за відхилення передбаченого значення від реального

MDP є дискретною моделлю стохастичної оптимальної задачі управління та класичним формулюванням процесу послідовного прийняття рішень, де розглядаються як миттєві, так і майбутні винагороди. MDP є формальною моделлю середовища для агентів, що мають вирішувати задачі планування та діяти в умовах невизначеності. Існує багато різних визначень MDP, які є еквівалентними з незначними змінами у постановці задачі. За одним з таких визначень MDP є п'ятіркою (S, A, P, γ, r), де:

- S – множина станів.
- A – множина дій.
- $P_a(s, s') = \Pr(s_{t+1} = s' | s_t = s, a_t = a)$ – ймовірність того, що система перейде в стан s' в момент часу $t + 1$, якщо вона була в стані s в момент часу t і була виконана дія a.
- $r_a(s, s')$ - безпосередня винагорода, отримана після переходу зі стану s до стану s' через дію a.
- $\gamma \in [0; 1]$ – коефіцієнт знецінення, який дозволяє врахувати майбутню невизначеність, знецінюючи майбутні винагороди порівняно з поточними.

На рис. 1.2 зображено приклад Марковського процесу вирішування з трьома станами (s_0, s_1 та s_2) та двома діями (a_0 та a_1). Числами над стрілками підписані ймовірності переходу між станами в залежності від вибраної дії, а числами біля жовтих стрілок підписані винагороди.

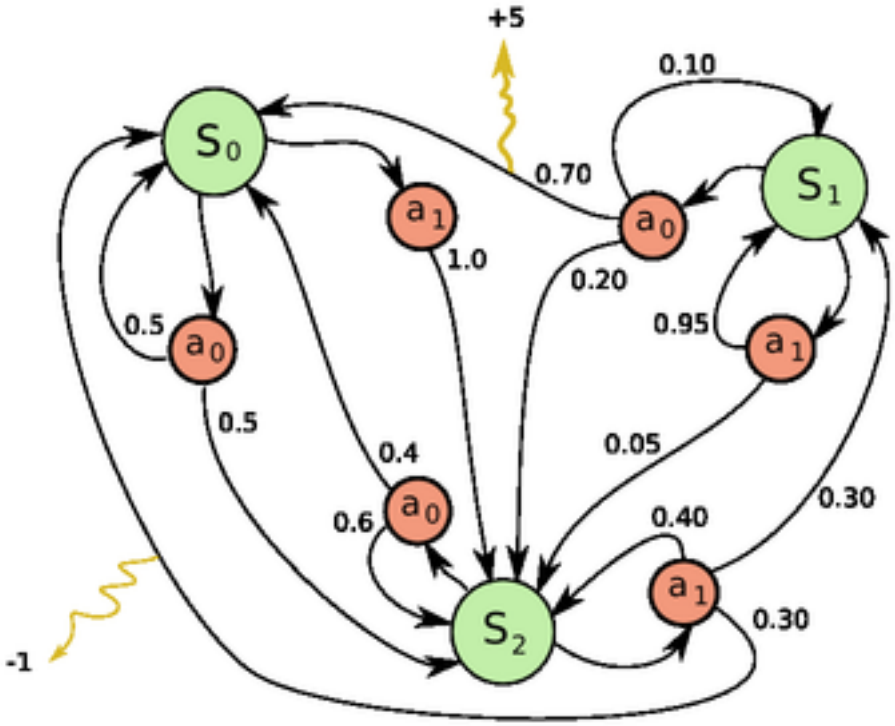


Рисунок 1.2. Приклад Марковського процесу вирішування

Процес починається з того, що агент знаходиться в деякому початковому стані s_0 в момент часу $t = 0$. В кожний момент часу $t = 0, 1, 2, 3, \ldots$ агент вибирає деяку дію a_t, базуючись на тому в якому стані s_t він знаходиться та керуючись деякою стратегією (policy) π: $a_t = \pi(s_t)$. Після цього система переходить в наступний стан s_{t+1} згідно з ймовірностями P, та агент отримує винагороду r_t. Схематично це можна зобразити як

$$s_0 \xrightarrow{a_0, r_0} s_1 \xrightarrow{a_1, r_1} s_2 \rightarrow \cdots \tag{1.1}$$

Загальну винагороду, починаючи з моменту часу t визначають як

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \tag{1.2}$$

Майбутні винагороди знецінюються на фактор γ на кожному кроці. Це, по-перше, допомагає врахувати невизначеність майбутніх винагород, а по-друге, робить загальну винагороду R_t скінченною навіть у випадку нескінченного MDP.

Задача полягає в тому, щоб знайти оптимальну стратегію π^*, яка максимізує очікуване значення загальної винагороди R_t для всіх початкових станів.

Означимо функцію цінності (value function) $V_\pi(s)$ для стратегії π та стану s як очікувану загальну винагороду, якщо рухатись зі стану s за стратегією π:

$$V_\pi(s) = \mathbb{E}[R_t | s_t = s] = \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \,|s_t = s] \tag{1.3}$$

Тоді задачу пошуку оптимальної стратегії π^* можна також сформулювати як задачу максимізації функції цінності $V_\pi(s)$.

Ще однією важливою функцією є так звана Q-функція або функція якості (Q-function, quality function). Вона визначає очікувану загальну винагороду, якщо рухатись зі стану s з початковою дією a:

$$Q_\pi(s,a) = \mathbb{E}[R_t | s_t = s, a_t = a] = \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \,|s_t = s, a_t = a] \tag{1.4}$$

Різниця між функцією цінності та Q-функцією, полягає в тому, що Q-функція враховує, яку дію потрібно виконати, порівняно з функцією цінності. Вони обидві залежать від стратегії π. Також часто говорять про просто функцію цінності та Q-функцію, без вказання стратегії, маючи на увазі ці функції для оптимальної стратегії:

$$V(s) = V_{\pi^*}(s) = \max_\pi V_\pi(s) \tag{1.6}$$

$$Q(s,a) = Q_{\pi^*}(s,a) = \max_{\pi} Q_{\pi}(s,a) \tag{1.7}$$

Ці дві функції містять одну й ту саму інформацію і кожну з них можна відновити з іншої:

$$V(s) = \max_{a} Q(s,a) \tag{1.8}$$

$$Q(s,a) = \mathbb{E}[r_a(s,s') + \gamma V(s')] = \sum_{s'} P_a(s,s')[r_a(s,s') + \gamma V(s')] \tag{1.9}$$

Знаючи будь-яку з цих функцій, можна відновити оптимальну стратегію:

$$\pi^*(s) = \operatorname*{argmax}_{a} Q(s,a) = \operatorname*{argmax}_{a} \mathbb{E}[r_a(s,s') + \gamma V(s')] \tag{1.10}$$

Таким чином, задачу пошуку оптимальної стратегії можна ставити або як задачу максимізації функції цінності, або як задачу оптимізації Q-функції. Той чи інший підхід може бути зручнішим в залежності від методу, що використовується для розв’язання задачі.

Виходячи з їх означення, можна довести, що функція цінності та Q-функція задовольняють наступні рекурсивні рівняння:

$$V(s) = \max_{a} \mathbb{E}[r_a(s,s') + \gamma V(s')] = \max_{a} \sum_{s'} P_a(s,s')[r_a(s,s') + \gamma V(s')] \tag{1.11}$$

$$\begin{aligned} Q(s,a) &= \mathbb{E}\left[r_a(s,s') + \gamma \max_{a'} Q(s',a')\right] \\ &= \sum_{s'} P_a(s,s')\left[r_a(s,s') + \gamma \max_{a'} Q(s',a')\right] \end{aligned} \tag{1.12}$$

Ці рівняння використовуються в методах навчання з підкріпленням для ітеративного знаходження цих функцій.

1.3 Класифікація методів навчання з підкріпленням

Алгоритми навчання з підкріпленням можна класифікувати за декількома ознаками. Один з варіантів такої класифікації наведений на рис. 1.3.

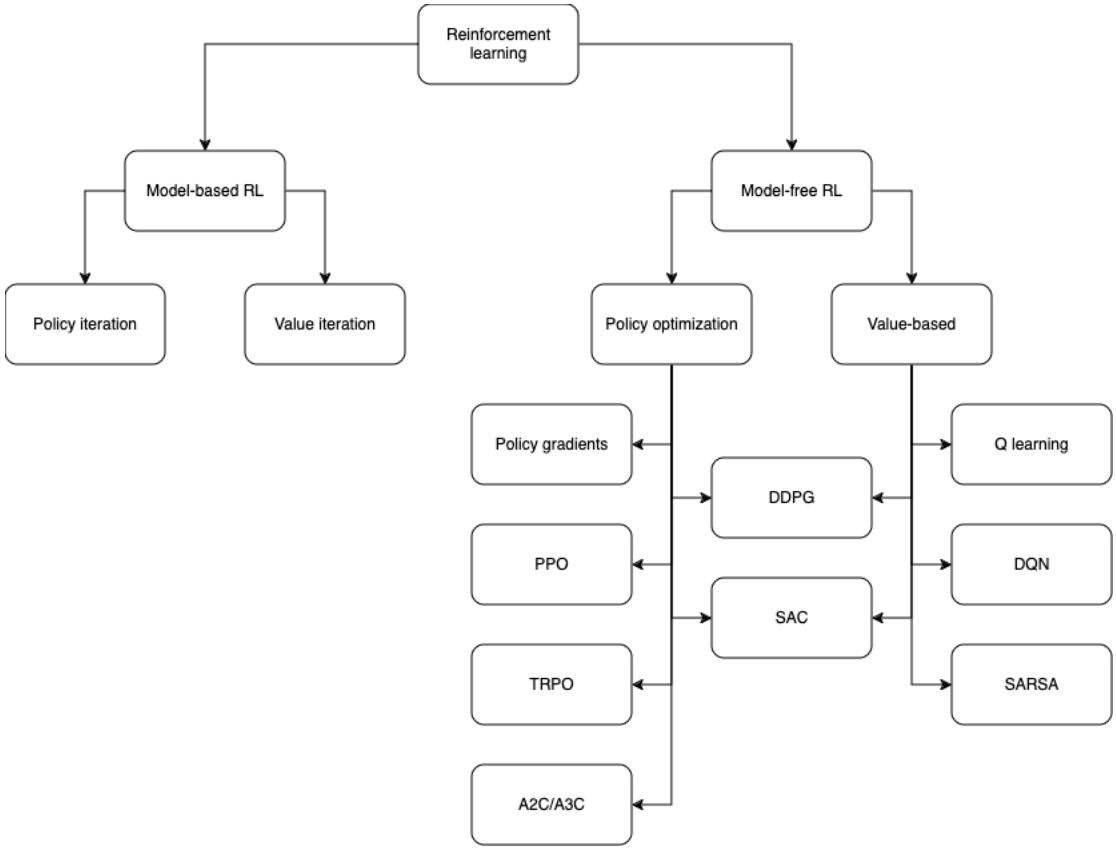


Рисунок 1.3. Один з варіантів класифікації методів навчання з підкріпленням. Наведено найбільш популярні методи

Одна з головних ознак, за якою можна класифікувати алгоритми навчання з підкріпленням - це те, чи має агент доступ до моделі середовища (або може її вивчати). Під моделлю середовища розуміється функція, яка передбачає переходи між станами та винагороди. За цією ознакою алгоритми поділяються на такі, що використовують модель (model-based algorithms) та такі, що її не використовують (model-free algorithms).

Головна перевага наявності моделі полягає в тому, що вона дозволяє агенту планувати, думаючи наперед, бачити, що відбудеться при різних можливих діях, і явно вибирати між різними доступними варіантами. Потім агенти можуть узагальнити результати планування у вивчену стратегію. Одним з найвідоміших прикладів такого підходу є AlphaZero [32], модель для гри в ґо. Коли це працює, це може призвести до значного покращення ефективності використання вибірки порівняно з методами, які не мають моделі.

Головний недолік полягає в тому, що точна модель середовища зазвичай не доступна агенту. Якщо агент хоче використовувати модель у цьому випадку, він повинен вивчити модель виключно з досвіду, що створює кілька викликів. Найбільша проблема полягає в тому, що агент може експлуатувати упередженість у моделі, що призводить до того, що агент працює добре з врахуванням вивченої моделі, але поводиться неоптимально (або дуже погано) в реальному середовищі. Вивчення моделі є фундаментально складним, тому витрачені на це зусилля (час і обчислювальні ресурси) можуть не виправдатися.

Хоча методи, які не використовують модель, відмовляються від потенційних виграшів у використанні вибірки за допомогою моделі, вони зазвичай є простішими для впровадження та налаштування. Тому ці методи є більш популярними.

1.4 Методи, що використовують модель (model-based methods)

Основними методами, що використовують модель, є методи динамічного програмування (dynamic programming, DP). Вони дозволяють знайти оптимальний розв’язок, проте вимагають наявності повної моделі середовища і великих обчислювальних витрат (часто практично неможливих) для нетривіальних задач. Через це вони використовуються

відносно нечасто. Два найбільш популярних таких метода, це ітерація стратегій (policy iteration) та ітерація цінності (value iteration).

1.4.1 Ітерація стратегій (policy iteration)

Загальна ідея ітерації стратегій полягає в пошуку оптимальної стратегії шляхом ітеративного перебору багатьох стратегій та вибору з них найкращої. Коли загальні винагороди більше не можна покращити, алгоритм знайшов оптимальну стратегію.

Агент починає з довільної стратегії, після чого оновлює її ітераційно. Кожна ітерація складається з двох кроків.

Першим кроком, використовуючи поточну стратегію π , розраховується функція цінності $V_\pi(s)$. Це робиться ітеративно. Функція цінності ініціалізується нулями, після чого проводиться така ітерація:

$$V_{0,\pi}(s) = 0 \tag{1.13}$$

$$V_{k+1,\pi}(s) = \mathbb{E}\left[r_{a=\pi(s)}(s,s') + \gamma V_{k,\pi}(s')\right] \tag{1.14}$$

Коли $k \to \infty, V_{k,\pi}(s) \to V_\pi(s)$.

Другим кроком, покращуємо стратегію, вибираючи для кожного стану таку дію, що максимізує цінність наступного стану разом з поточною винагородою, тобто

$$\pi(s) = \underset{a}{\operatorname{argmax}}(r_a(s,s') + \gamma V(s')) \tag{1.15}$$

Ітерацію стратегії можна схематично зобразити так

$$\pi_0 \overset{E}{\to} v_{\pi_0} \overset{I}{\to} \pi_1 \overset{E}{\to} v_{\pi_1} \overset{I}{\to} \pi_2 \overset{E}{\to} \dots \overset{I}{\to} \pi \overset{E}{\to} v \tag{1.16}$$

де $\overset{E}{\rightarrow}$ зображають кроки знаходження функції цінності зі стратегії, а $\overset{I}{\rightarrow}$ зображають кроки покращення стратегії, використовуючи функцію цінності.

1.4.2 Ітерація цінності (value iteration)

Ітерація цінності знаходить оптимальну стратегію шляхом знаходження оптимальної функції цінності, замість того, щоб перебирати багато стратегій. Після знаходження функції цінності, оптимальну стратегію можна отримати за допомогою формули (1.10).

Функція цінності ініціалізується нулями, після чого ітерується за допомогою рівняння

$$V_{k+1}(s) = \max_a \mathbb{E}[r_a(s, s') + \gamma V_k(s')] \tag{1.17}$$

1.4.3 Практичне застосування

На практиці, при застосуванні як ітерації стратегій так і ітерації цінностей потрібно на кожному кроці проходитись по всій множині станів S. Це робить ці методи незастосовними до великих багатовимірних задач, оскільки там простір станів має надто великий розмір. Проте для задач з малим простором станів вони дозволяють знайти оптимальний розв’язок, тоді як методи, що не використовують модель, знаходять лише наближений розв’язок.

1.5 Методи, що не використовують модель (model-free methods)

Методи, що не використовують модель можна умовно поділити на два класи: методи, засновані на оптимізації цінності (value-based methods), та методи, засновані на оптимізації стратегії (policy optimization methods) [33]. До перших відносяться такі алгоритми як Q-навчання (Q-learning), глибокі Q-мережі (deep Q networks, DQN) та SARSA (state-action-reward-state-action,

стан-дія-винагорода-стан-дія) та інші. До других: (deterministic policy gradients, DPG, детерміністичні градієнти стратегій), A2C/A3C (asynchronous advantage actor-critic), PPO (proximal policy optimization, проксимальна оптимізація стратегії), TRPO (trust region policy optimization, оптимізація стратегії з областю довіри) та інші. Також є алгоритми, що комбінують обидва підходи, такі як наприклад DDPG (deep deterministic policy gradients, глибокі детерміністичні градієнти стратегій) та SAC (soft actor-critic, м'який актор-критик). Далі розглянемо детальніше найбільш поширені з цих методів.

1.5.1 Q-навчання

Як розглядалося вище, Q-функція для оптимальної стратегії повинна задовольняти рівнянню (1.12):

$$Q(s,a) = \mathbb{E}\left[r_a(s,s') + \gamma \max_{a'} Q(s',a')\right] \tag{1.18}$$

Або для детерміністичної системи, де дія a в стані s завжди приводить систему в один і той же стан s':

$$Q(s,a) = r_a(s,s') + \gamma \max_{a'} Q(s',a') \tag{1.19}$$

Тобто максимальна загальна винагорода рівна поточній винагороді разом з максимальною винагородою за наступну дію, помноженою на коефіцієнт знецінення.

Ідея Q-навчання полягає в тому, щоб знаходити Q-функцію ітеративно, використовуючи це рекурсивне рівняння. Для цього спочатку її ініціалізують якимись початковими значеннями, наприклад нулями. Після цього агента запускають досліджувати середовище.

Після кожного кроку дослідження, Q-функцію оновлюють:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r_a(s,s') + \gamma \max_{a'} Q(s',a') - Q(s,a) \right] \qquad (1.20)$$

Де α – коефіцієнт, що відповідає за швидкість навчання (learning rate). Вираз у дужках – різниця правої та лівої частин рівняння (1.19). Додаючи цю різницю, помножену на деякий коефіцієнт до Q-функції, метод ітеративно приходить до оптимального розв'язку, який задовольняє це рівняння.

При дослідженні середовища потрібно враховувати такі два факти:

- З одного боку, агенту потрібно рухатись відповідно до вже вивченої стратегії, тобто вибирати дію, яка максимізує винагороду відповідно до поточної Q-функції.
- З іншого боку, таким чином агент може легко застрягти в локальному мінімумі, тому потрібно також досліджувати нові, ще не перевірені стратегії.

Щоб збалансувати ці два підходи, зазвичай використовують ϵ -жадібну стратегію (ϵ-greedy policy). Відповідно до неї, агент з імовірністю $\epsilon \in [0; 1]$ виконує випадково вибрану дію, а з імовірністю $1 - \epsilon$ виконує дію, що максимізує винагороду відповідну до поточної Q-функції. Значення параметра ϵ при цьому не залишається постійним. Його ініціалізують значенням, близьким або рівним одиниці, на початку навчання, щоб агент більше досліджував. Після цього його поступово (зазвичай за експоненційним законом) зменшують до значення, близького або рівного нулю, щоб ближче до кінця навчання агент покращував вже знайдену стратегію.

1.5.2 Методи глибокого навчання з підкріпленням (deep reinforcement learning)

Рис. 1.4 показує зацікавленість у навчанні з підкріпленням по всьому світу з 2007 по 2022 рік згідно з Google Trends.

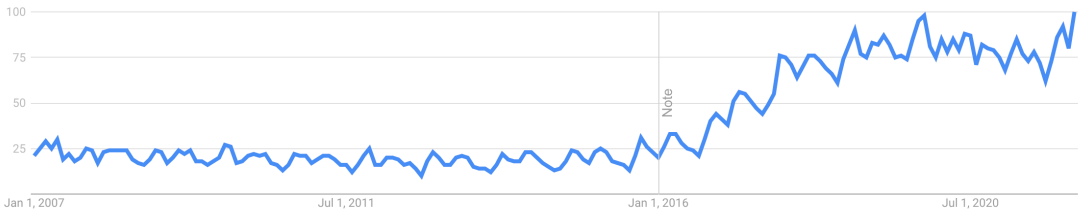


Рисунок 1.4. Зацікавленість у навчанні з підкріпленням, згідно з Google Trends [34]

Можна побачити, що зацікавленість у навчанні з підкріпленням стагнувала до 2016 року, після чого почався значний ріст. Першим чинником, який сприяв цьому зростанню, була публікація [35] у 2013 році, де автори представили алгоритм глибокого навчання з підкріпленням, названий глибока Q-мережа (deep Q-learning network, DQN), який міг грати в багато ігор ігрової приставки Atari 2600 (на рис. 1.5 зображено знімки екрану деяких з цих ігр) лише на основі вхідних зображень, використовуючи один і той самий алгоритм для різних ігр (хоча агент повинен був бути перенавчений для кожної гри). Продуктивність агента порівнювалася з продуктивністю людських гравців після тренування на 10 мільйонах кадрів (приблизно 56 годин безперервної гри). Щоб забезпечити справедливе порівняння, агент враховував людські обмеження, такі як затримки у відгуку, і він міг бачити лише інформацію, що відображалася на екрані. DQN перевищив людський рівень у половині ігор. DQN використовував Q-навчання з глибокими нейронними мережами для апроксимації функцій. У DQN можна використовувати неперервні стани,

але дії були дискретними, оскільки ігри Atari зазвичай мали дискретні вхідні дані.

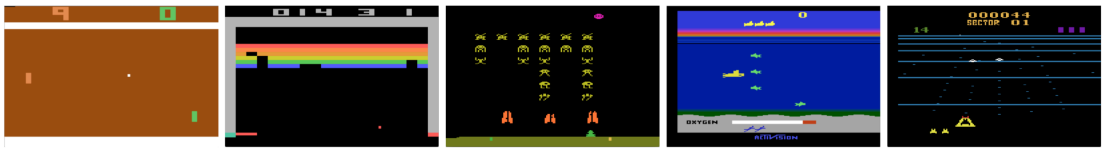


Рисунок 1.5. Знімки екрану п’яти ігор Atari [35]. Зліва направо: Pong, Breakout, Space Invaders, Seaquest, Beam Rider

Основна відмінність DQN від класичного Q-навчання полягає у використанні нейронної мережі для передбачення значень Q-функції (рис. 1.6). У класичному Q-навчанні використовують таблицю, де кожній парі стан-дія відповідає значення Q-функції. Проте розмір цієї таблиці може ставати дуже великим для задач з великими просторами станів. DQN же використовує нейронну мережу, на вхід якої подається поточний стан. Ця нейронна мережа має стільки виходів, скільки є доступних дій, і видає на кожному виході значення Q-функції, якщо вибрати цю дію в поточному стані. Навчання полягає в навчанні нейронної мережі, щоб отримати гарну апроксимацію оптимальної Q-функції.

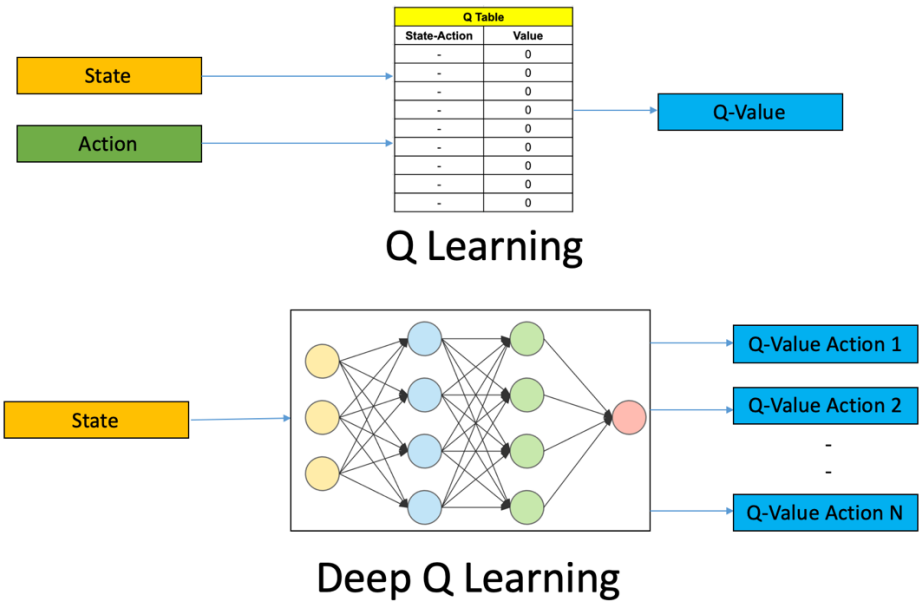


Рисунок 1.6. Порівняння класичного Q-навчання та DQN

Перший алгоритм навчання з підкріпленням, який може природно обробляти як неперервні стани, так і дії, був розроблений у роботі [36] (2014). Цей алгоритм отримав назву детермінований градієнт стратегії (deterministic policy gradient, DPG), та є методом ітерації політики, навченим за допомогою методів Монте-Карло. DPG детерміновано відображає неперервні стани на неперервні дії через нейронну мережу, яка апроксимує оптимальну стратегію. Хоча DPG може обробляти як неперервні стани, так і дії і є ще одним значним внеском, він є дуже обчислювально витратним, має високу варіативність і неприродний для неперервних завдань через метод навчання Монте-Карло.

[37] об'єднали ідеї [35] (DQN) і [36] (DPG) у один алгоритм навчання з підкріпленням з використанням підходу актор-критик, відомий як глибокий детермінований градієнт стратегії (deep deterministic policy gradient, DDPG). Загальна архітектура показана на рис. 1.7. Тут DPG використовується для відображення станів у дії і є актором. В той же час, DQN є критиком, який оцінює дії актора, і використовується для ідентифікації значень дій для оновлення DPG без використання методів Монте-Карло. Спочатку як DQN, так і DPG мали високу упередженість і високу варіативність відповідно. Однак, об'єднавши два алгоритми, можна значно зменшити як упередженість, так і варіативність [37]. Крім того, методи градієнту стратегії зазвичай навчаються за допомогою методів Монте-Карло в кінці епізодів [38]; однак у DDPG це обмеження подолано за допомогою навчання DPG за градієнтом критика (тобто похідної значень Q-функції відносно ваг моделі) за допомогою градієнтного підйому. Інтуїтивно ваги DPG оновлюються для максимізації значення дій, Q(x, u). У DDPG для ідентифікації оптимальної стратегії використовуються чотири нейронні мережі колективно. Крім того, нейронні мережі неявно

навчаються за вагами один одного, що потенційно викликає помилки апроксимації функцій і призводить до субоптимальних політик. У [39] автори ввели новий спосіб мінімізації помилок, відкладаючи оновлення стратегії. Новий алгоритм також був протестований на різних тестових середовищах та показав кращу продуктивність у кожному випадку.

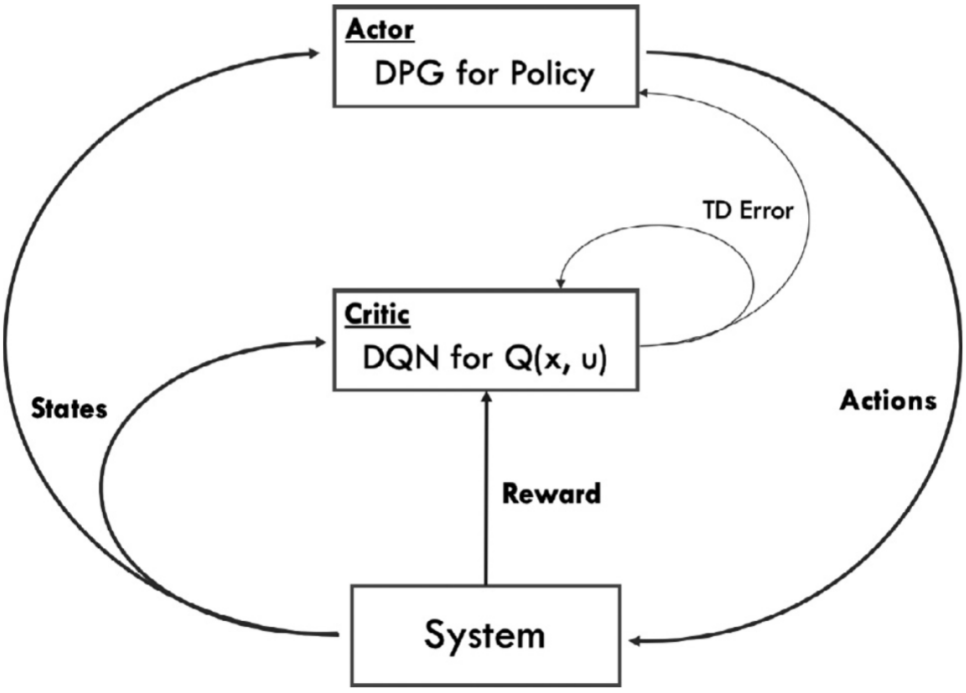


Рисунок 1.7. Архітектура методу DDPG

Приблизно в той же час [40] запропонували інший метод глибокого навчання з підкріпленням, використовуючи оптимізацію стратегії. Цей алгоритм, відомий як оптимізація стратегії з областю довіри (trust region policy optimization, TRPO), гарантує монотонні покращення стратегії на кожній ітерації за допомогою спеціально підібраних оновлень параметрів при оновленні стратегії. Зокрема, обмеження оновлення гарантує, що нова політика знаходиться в довірчому регіоні: підпросторі, в якому локальні апроксимації функцій є надійними. На відміну від DDPG, де існують актор і критик, TRPO ідентифікує політики безпосередньо. Таким чином, [37] вважають, що TRPO потребує набагато менше даних. Проте, оновлення

параметрів повинні бути виконані з допомогою методу спряжених градієнтів через наявність обмежень, що може бути важко впровадити.

Для вдосконалення попередніх недоліків TRPO [41] (2017) опублікували новий алгоритм навчання з підкріпленням у 2017 році, який називається проксимальна оптимізація стратегії (proximal policy optimization, PPO). Порівняно з TRPO, PPO є набагато простішим у впровадженні, більш загальним і має покращену ефективність використання даних. Зокрема, PPO реалізує обмеження оновлення безпосередньо в цільовій функції як штраф. Це дозволяє використовувати стандартний метод стохастичного градієнтного спуску (stochastic gradient descent, SGD) замість методу спряжених градієнтів. PPO здатний досягти кращих результатів порівняно з TRPO після тренування протягом того ж часу на різних завданнях неперервного керування.

У 2018 році [42] представили новий алгоритм типу актор-критик для поліпшення ефективності використання вибірки та факторів збіжності раніше введених методів під назвою м’який актор-критик (soft actor-critic, SOA). У цьому алгоритмі очікувана винагорода та варіативність дій максимізуються разом під час пошуку оптимальної стратегії. Загалом цей новий алгоритм показав кращі результати у порівнянні з іншими існуючими методами на різних тестових середовищах, залишаючись стабільним під час навчання. Історію популярних методів глибокого навчання з підкріпленням представлено на рис. 1.8.

Рисунок 1.8. Історія популярних методів глибокого навчання з підкріпленням

1.6 Практичні обмеження навчання з підкріпленням

Незважаючи на усі його переваги, метод навчання з підкріпленням має значний недолік при застосуванні його на практиці до задач керування опаленням. Агент завжди навчається у конкретному середовищі. Тому після навчання його не можна застосовувати в іншому середовищі. Це означає, що для кожного випадку потрібно будувати нове середовище та заново проводити навчання агента. Проте це призводить до технічних труднощів. Є два основних підходи:

1. Навчання агента напряму у фізичному середовищі, де він далі буде працювати. Цей підхід має ту перевагу, що агент під час навчання має доступ до справжнього середовища роботи, в якому відсутні будь-які неточності, пов’язані з неточністю симуляцій та фізичних моделей. Проте цей метод зазвичай займає відносно багато часу. Наприклад, для задачі оптимізації енергії, витраченої на опалення, у роботі [33] навчання зайняло декілька тижнів.
2. Альтернативою є побудова симуляції фізичного середовища та тренування агента в ній. Це дозволяє провести навчання набагато швидше, ніж у справжньому середовищі. Проте, для задачі оптимізації витрат на опалення, побудова достатньо точної моделі фізичного середовища вимагає вимірювання багатьох фізичних параметрів приміщення, таких як його розмір, товщина стін, матеріал стін, кількість та розмір вікон та ін. Це може бути відносно довго та складно на практиці.

Таким чином, обидва варіанти практичного застосування наштовхуються на практичні труднощі, що спонукає до розробки нового покращеного методу.

1.7 Постановка задачі

У світлі викладених досліджень можна зробити висновок, що найбільш ефективними методами оптимізації систем ОВК є методи навчання з підкріпленням. Вони потребують відносну малу кількість історичних даних, є гнучкими та їх можна донавчати у випадку зміни параметрів будівлі. У літературі було проаналізовано різні приклади застосування цих методів. Більшість досліджень ставить перед собою завдання зменшення витрат електроенергії з урахуванням комфорту мешканців. Це сприяє покращенню екологічної стійкості та енергоефективності в цілому, однак для окремих споживачів та підприємств більш пріоритетним є зниження грошових витрат на опалення.

Ці два показники пропорційні один одному лише у випадку фіксованого тарифу на електроенергію. Проте на практиці частіше використовуються багатозонні тарифи, що змінюються протягом дня або навіть кожну годину. Наприклад, у багатьох країнах Європи використовуються тарифи, що базуються на цінах Nord Pool. Ці ціни задані погодинно і відомі на день вперед. Аналізуючи ціни Nord Pool [43] можна побачити доволі сильну денну варіацію. Наприклад, 18.01.2023 ціна змінювалась від 38.69 євро/МВт до 166.35 євро/МВТ, тобто у 4.3 рази (рис. 1.9).

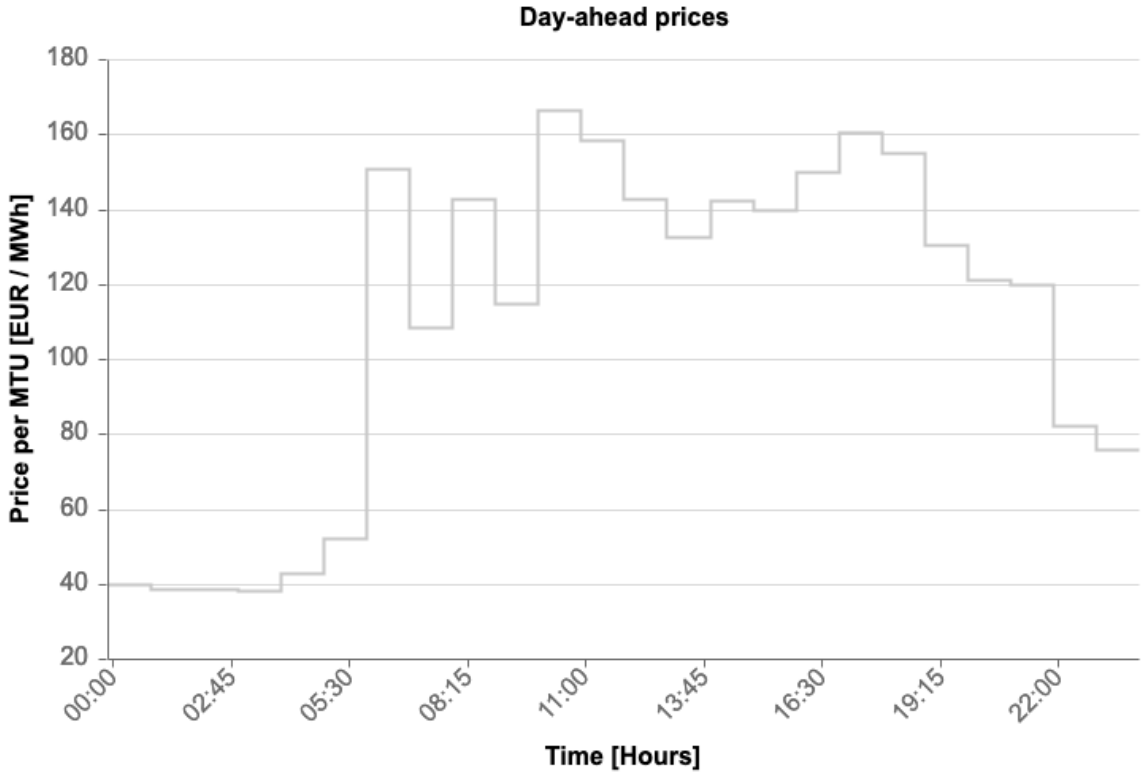


Рисунок 1.9. Ціни Nord Pool 18.01.23 згідно з [43]

Відзначена денна варіація цін створює можливості для значних заощаджень у витратах на опалення, оскільки алгоритм керування може підібрати графік нагріву так, щоб більшість нагрівання відбувалась під час періоду часу, коли ціна є малою.

Задачами даної роботи є:

а) Проаналізувати застосування підходу навчання з підкріпленням до задачі оптимізації витрат на опалення. Проаналізувати можливий вигляд винагород у цьому підході, порівняти їх ефективність та вибрати оптимальну.

б) Розробити удосконалену версію цього підходу, що використовує спрощену модель середовища, що будується на основі зібраних даних без втручання користувача, та дозволяє спростити практичне застосування.

в) Розробити програмну реалізацію цього підходу для даної задачі та створити серверний застосунок з веб-клієнтом, що дозволяє користувачу керувати та слідкувати за витратами на опалення.

Висновки до розділу

У даному розділі було проведено аналіз літератури за темою використання методів машинного навчання для керування опаленням. Встановлено, що найбільш оптимальними для цієї задачі є методи, що відносяться до підходу навчання з підкріпленням.

Розглянуто підхід навчання з підкріпленням. Наведено приклади його практичного застосування. Описано суть підходу та наведено його математичне формулювання.

Розглянуто класифікацію методів, що відносяться до цього підходу та розібрано деякі з цих методів. Розглянуто методи глибокого навчання з підкріпленням, їх історію, типи та особливості.

Описано підходи до практичного застосування навчання з підкріпленням до даної задачі та наведено їх обмеження, що створює необхідність до їх покращення.

Встановлено, що добре досліджені методи, що оптимізують витрати електроенергії, проте оптимізація грошових витрат з врахуванням змінного графіку цін є малодослідженою. Обґрунтовано можливість заощадження за рахунок врахування цього графіку та його практична доцільність. У кінці розділу поставлено задачі, що будуть розв’язані у даній роботі.

2 РОЗРОБКА УДОСКОНАЛЕНОГО ПІДХОДУ НАВЧАННЯ З ПІДКРІПЛЕННЯМ ДЛЯ КЕРУВАННЯ ОПАЛЕННЯМ

2.1 Удосконалений метод

Як було описано в першому розділі, при практичному застосуванні підходу навчання з підкріпленням до задачі керування опаленням існує два методи: навчання у справжньому середовищі, яке є відносно довгим, або навчання на моделі фізичного середовища, проте є складність з її побудовою.

Аналізуючи описані вище два методи, можна зробити висновок, що більш швидким є другий метод, проте він наштовхується на проблему побудови моделі середовища. Якщо спростити цю побудову, то метод може бути одночасно швидким та зручним до застосування на практиці.

У даній роботі пропонується удосконалений метод, який полягає в наступному:

а) Нагрівач встановлюється в приміщенні, де він буде працювати та за декілька днів проводить декілька циклів нагрівання та охолодження. Весь цей час він збирає метрики, а саме залежність його потужності та температури в приміщенні від часу.

б) Використовуючи отримані дані, будується спрощена модель приміщення. Вона детально описана в наступному підрозділі. Ця модель має малу кількість параметрів, тому для їх знаходження достатньо відносно невеликої кількості даних. При цьому, зрозуміло, що вона буде мати неточності через її простоту, проте це компенсується в кінцевих кроках методу.

в) Агент навчання з підкріпленням тренується на спрощеній моделі приміщення. Оскільки тренування відбувається з використанням симуляції, то його можна провести достатньо швидко.

г) Натренований агент підключається до справжнього нагрівача та починає працювати, одночасно продовжуючи тренування.

Знаходячись у справжньому фізичному середовищі, він має змогу донавчитись на виправити неточності, пов’язані із спрощеністю моделі.

Таким чином, запропонований підхід є модифікацією підходу з використанням симуляції середовища, проте використовує спрощену симуляцію, отриману з невеликої кількості даних, що дозволяє значно спростити її побудову, та зробити цю побудову автоматичною, не вимагаючи ніяких дій від користувача.

Покроковий опис методу та результатів кожного кроку наведений у табл. 2.1.

2.2 Спрощена модель приміщення

У даному підрозділі розглянемо вибір спрощеної моделі приміщення. Для теплового моделювання приміщень є велика кількість різних моделей, що мають різну точністю та ступінь складності.

З однієї сторони знаходяться пакети моделювання високого ступеню точності, такі як EnergyPlus [44]. Вони будують тривимірну модель приміщення та враховують багато факторів, таких як розміщення нагрівачів у приміщенні, його геометрію, рух повітря, вентиляцію та багато інших. Це дозволяє отримувати дуже точні результати, проте вимагає попереднє вимірювання параметрів приміщення, що не підходить для даної роботи.

З іншої сторони є більш прості моделі з невеликою кількістю параметрів, такі як 5R1C [45]. Вони моделюють приміщення в цілому з використанням декількох рівнянь з невеликою кількістю параметрів, не вдаючись в деталі, такі як конкретна геометрія приміщення або рух повітря. Правильно підібравши параметри таких моделей можна отримати симуляцію, яка буде достатньо точною для наших потреб, хоч, звичайно, і буде поступатись симуляціям першого виду. Тому далі розглядаємо саме цей вид моделей.

Таблиця 2.1. Опис запропонованого удосконаленого підходу до навчання з підкріпленням

Крок методу	Виконувані дії	Результат
1	Проведення декілька циклів нагрівання та охолодження на нагрівачі у справжньому приміщенні та збір даних	Зібрані дані залежностей потужності та температури від часу
2	Підбір параметрів спрощеної моделі приміщення, використовуючи дані, отримані на минулому кроці	Спрощена модель приміщення
3	Тренування агента навчання з підкріпленням на спрощеній моделі приміщення	Натренований агент (попередня версія, що має деякі неточності, через спрощеність моделі)
4	Підключення агента до справжньої моделі та продовження тренування	Натренований агент (остаточна версія, що враховує всі нюанси справжнього приміщення)

Оскільки ми маємо відносно малу кількість даних (декілька циклів нагрівання та охолодження будинку впродовж декількох днів), то необхідно вибрати модель з якнайменшим числом параметрів, щоб їх можна було надійно знайти з наявних даних. Найпростішою такою моделлю, є модель, що описується наступним диференційним рівнянням [46]:

$$C\frac{dT}{dt} = P_h u(t) - K\big(T - T_{out}(t)\big), \qquad (2.1)$$

де t – час (с), T – температура всередині приміщення (К), $T_{out}(t)$ – температура ззовні приміщення (К), C – теплоємність приміщення (Дж/К),

K – теплопровідність стін та вікон приміщення (Вт/К), P_h - потужність нагрівача (Вт), $u(t) \in \{0,1\}$ – змінна, що задає, чи увімкнутий нагрівач.

Після введення нових параметрів $p = P_h/C$ та $k = K/C$, рівняння приймає вигляд

$$\frac{dT}{dt} = pu(t) - k\big(T - T_{out}(t)\big), \tag{2.2}$$

та має всього два незалежних параметри.

Розглядаючи невеликі проміжки часу, можна наближено замінити диференціали dT та dt у рівнянні на відповідні скінченні прирости ΔT та Δt. Тоді маємо лінійну залежність величини $\frac{\Delta T}{\Delta t}$ від різниці внутрішньої та зовнішньої температур $T - T_{out}(t)$. Використовуючи зібрані дані $T(t)$ та відому залежність $T_{out}(t)$, яку можна отримати з сервісів погоди, проводимо лінійну регресію і таким чином знаходимо невідомі параметри p та k.

Тестування та обґрунтування коректності застосування даної спрощеної моделі наведено у підрозділі 4.1.

2.3 Винагороди для навчання з підкріпленням

При використанні підходу навчання з підкріпленням, після кожної дії в середовищі агент отримує винагороду. Він підбирає свої дії, намагаючись максимізувати цю винагороду. Тому її вибір є дуже важливим та напряму впливає на отриманий результат. Для задачі мінімізації витрат на опалення є дві величини, які потрібно оптимізувати. Перша, це, звичайно, економія витрат. Другою величиною є відхилення температури від заданої. Таким чином, сумарна винагорода r буде складатись з двох частин: r_E, що відповідає витратам, та r_T, що відповідає відхиленню температури.

Винагороду, що відповідає витратам, вибираємо у найпростішому можливому вигляді: $r_E = -p\Delta E$, де p – ціна на електроенергію, ΔE – енергія, витрачена на опалення, за один крок тренування. Знак мінус

необхідний для того, щоб агент, намагаючись максимізувати винагороду, тим самим мінімізував витрати.

Для винагороди, що відповідає відхиленню температури, є декілька можливих виглядів. Найпростішими є лінійне або квадратичне відхилення: $r_{TL} = -|T - T_0|\Delta t$ та $r_{TQ} = -(T - T_0)^2\Delta t$, де T – температура у приміщенні, T_0– задана користувачем цільова температура, Δt – часовий крок симуляції. Також розглянемо запропоновану у [47] винагороду з «обрізанням», яка рівна нулю при $T \geq T_0$:

$$r_{TLC} = \begin{cases} -|T - T_0|\Delta t, & T < T_0 \\ 0, & T \geq T_0 \end{cases} \tag{2.3}$$

$$r_{TQC} = \begin{cases} -(T - T_0)^2\Delta t, & T < T_0 \\ 0, & T \geq T_0 \end{cases} \tag{2.4}$$

Мотивацією введення цього «обрізання» є те, що при збільшенні температури автоматично збільшуються витрати, тому немає сенсу додатково включати область температур $T \geq T_0$ у винагороду r_T, бо вона вже буде врахована в r_E.

Щоб отримати повну винагороду, яку буде використовувати агент, необхідно об’єднати ці дві винагороди в одну. Для цього розглянуто два способи: лінійна комбінація та мультиплікативний метод з [47]:

$$r_L = (1 - \beta)r_E + \beta r_T, \tag{2.5}$$

$$r_M = r_E(1 + \beta|r_T|). \tag{2.6}$$

В обох випадках присутній додатковий параметр β. У лінійній винагороді r_L він приймає значення $0 \leq \beta \leq 1$, а в мультиплікативній винагороді r_M – значення $\beta \geq 0$. Він контролює пріоритетність складових винагород. При значеннях близьких до нуля, пріоритет має винагорода r_E, тому агент буде намагатись зменшити витрати, нехтуючи відхиленням температури. Навпаки, для значень, далеких від нуля, пріоритет буде мати винагорода r_T, і агент буде намагатись зменшити відхилення температури, нехтуючи витратами.

Таким чином, маємо один варіант вигляду винагороди r_E, чотири варіанти вигляду винагороди r_T (r_{TL}, r_{TQ}, r_{TLC} та r_{TQC}), та два способи їх об'єднати (r_L та r_M). Тобто всього будемо мати 8 винагород:

$$r_{LL} = -(1-\beta)pE - \beta|T - T_0|, \tag{2.7}$$

$$r_{LQ} = -(1-\beta)pE - \beta(T - T_0)^2, \tag{2.8}$$

$$r_{ML} = -pE(1 + \beta|T - T_0|), \tag{2.9}$$

$$r_{MQ} = -pE(1 + \beta(T - T_0)^2), \tag{2.10}$$

$$r_{LLC} = \begin{cases} -(1-\beta)pE - \beta|T - T_0|, T < T_0 \\ -(1-\beta)pE, T \geq T_0 \end{cases}, \tag{2.11}$$

$$r_{LQC} = \begin{cases} -(1-\beta)pE - \beta(T - T_0)^2, T < T_0 \\ -(1-\beta)pE, T \geq T_0 \end{cases}, \tag{2.12}$$

$$r_{MLC} = \begin{cases} -pE(1 + \beta|T - T_0|), T < T_0 \\ -pE, T \geq T_0 \end{cases}, \tag{2.13}$$

$$r_{MQC} = \begin{cases} -pE(1 + \beta(T - T_0)^2), T < T_0 \\ -pE, T \geq T_0 \end{cases}. \tag{2.14}$$

Перша літера індексу означає, як були скомбіновані складові (L (linear) – лінійно, M (multiplicative) – мультиплікативно), друга літера – який використовувався вигляд складової, що відповідає за відхилення температури (L (linear) – лінійний, Q (quadratic) – квадратичний). Остання літера C (cutoff) означає, що використовувалася винагорода з обрізанням.

Порівняння цих винагород та вибір найкращої наведено у підрозділі 4.2.

Висновки до розділу

У даному розділі запропоновано удосконалений підхід навчання з підкріпленням, що використовує спрощену модель приміщення та створює її за допомогою експериментально отриманих даних, без ручного вводу

параметрів користувачем. Запропоновано спрощену модель приміщення та метод підбору її параметрів.

Розглянуто винагороди для навчання з підкріпленням для задачі оптимізації витрат на опалення при умові підтримання температури, якомога ближчої до заданої. Наведено 8 різних винагород.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Функціональні вимоги до системи

Перед тим, як переходити до розробки програмного забезпечення, сформулюємо вимоги до системи, що буде розроблятися. Користувач повинен мати можливість керувати системою через графічний інтерфейс. При цьому, він повинен мати можливість:

- Створення та видалення акаунту.
- Додавання нагрівача до акаунту.
- Перегляду списку нагрівачів у своєму акаунті.
- Перегляд останніх метрик нагрівача, включаючи поточну температуру в кімнаті, температуру на вулиці, потужність та споживання.
- Побудову графіку (статистики) за заданий період для температури, потужності, споживання та витрат.
- Можливість почати збір даних для побудови спрощеної моделі приміщення (калібрування).
- Можливість відслідковувати прогрес калібрування.
- Перегляд доступних для нагрівача моделей машинного навчання.
- Створення та тренування нової моделі машинного навчання.
- Активація заданої моделі машинного навчання.

Типовий приклад використання системи повинен виглядати наступним чином:

а) Користувач створює акаунт

б) Користувач додає нагрівач до акаунту. При додаванні нагрівача, щоб підтвердити право доступу, він вказує пароль, що відповідає даному нагрівачу.

в) Користувач починає процес калібрування. Нагрівач декілька днів працює в циклах нагрівання та охолодження, збираючи дані для побудови спрощеної моделі приміщення. Користувач при цьому має змогу відслідковувати прогрес калібрування.

г) Після завершення калібрування, користувач може створити модель машинного навчання. При створенні, він задає цільову температуру, а також мінімальну та максимальну температуру, яких повинна дотримуватися модель. Модель одразу починає навчання, використовуючи спрощену модель приміщення, отриману в минулому кроці.

ґ) Після завершення навчання, створена модель починає керувати поведінкою нагрівача, задаючи розклад роботи.

д) За необхідності, користувач може створювати моделі з іншими параметрами, тренувати, та активувати їх.

3.2 Архітектура системи та вибір технологій

Було розроблено архітектури систему, зображену на рис. 3.1. Вона складається з таких компонент:

а) Бекенд. Центральний компонент системи, що координує взаємодію всіх інших компонентів. Для його розробки було вибрано мову програмування Kotlin [48] та фреймворк Spring [49].

б) Реляційна база даних, що використовується для зберігання даних користувачів та нагрівачів. Було обрано базу даних PostgreSQL [50].

в) База даних InfluxDB [51]. Ця база даних спеціалізується для зберігання часових рядів. Багато даних, що використовуються у системі мають вигляд часових рядів: метрики від нагрівача (температура, потужність, споживання), зовнішня температура, ціни на електроенергію. Тому має сенс використати цю спеціалізовану базу даних, бо вона є більш оптимізованою саме для цього застосування та має зручний функціонал для розрахунку різного роду статистики.

г) Сервіс машинного навчання. Його було відділено від бекенду в окремий сервіс, щоб, по-перше, не обмежуватись мовою та стеком технологій, вибраними для бекенду, та, по-друге, мати можливість незалежно його масштабувати. Для цього сервісу було вибрано мову Python [52] та бібліотеку машинного навчання PyTorch [53]. Також для прискорення навчання для моделей середовища використовувалася мова Cython [54]. Вона дозволяє використовувати синтаксис Python, та вільно взаємодіяти з кодом, написаним на Python, проте компілюється в C, що значно пришвидшує виконання.

ґ) Брокер повідомлень Apache Kafka [55]. Використовується для комунікації між бекендом та сервісом машинного навчання. Бекенд відправляє команди в один топік та отримує відповіді назад з іншого топіку. Використання брокера замість прямої комунікації через REST API, дозволяє сервісам не залежати напряму один від одного та незалежно масштабуватися у разі необхідності.

д) Веб-клієнт. Підключається до бекенду та дозволяє користувачу моніторити та керувати нагрівачем. Було використано мову TypeScript [56] та бібліотеки React, Redux, Tailwind CSS, MUI, Chart.js та Tremor. Комунікація з бекендом відбувається за допомогою REST API, а також дані нагрівача надходять в режимі реального часу за допомогою протоколів WebSocket [57] та STOMP [58]. Було вибрано саме веб-клієнт, оскільки він є найбільш універсальним та кросплатформеним, на відміну від десктопного або мобільного клієнта.

е) Черга повідомлень MQTT [59]. Використовується для отримання метрик від нагрівачів. Була обрана, оскільки вона спеціально оптимізована для використання з пристроями інтернету речей, наприклад в умовах повільних, нестабільних мереж або мереж з високою затримкою. Нагрівачі встановлюються в приміщеннях користувачів, над якими ми не маємо контролю, і не можемо

гарантувати гарне інтернет-з’єднання, тому необхідно забезпечети протокол комунікації, що може стабільно працювати в несприятливих умовах.

є) Нагрівач (або нагрівальна панель). Встановлюється в приміщенні користувача та має мікроконтроллер, що комунікує з рештою системи. Отримує режим роботи та розклад від бекенду через REST API, та постійно відправляє метрики до InfluxDB через MQTT.

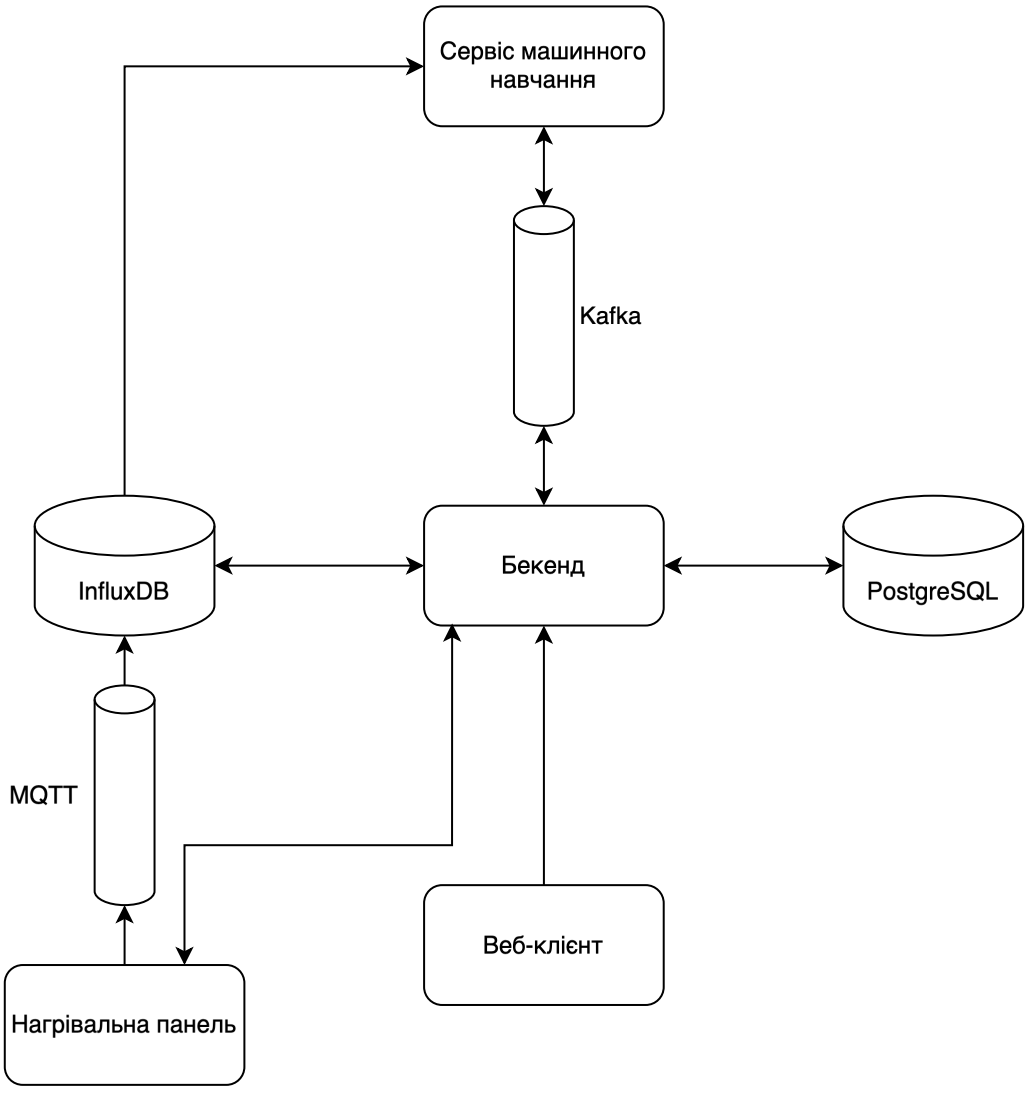


Рисунок 3.1. Високорівнева архітектура системи

3.3 Проектування баз даних

3.3.1 Реляційна БД PostgreSQL

У застосунку використовується схема бази даних з наступними таблицями (додаток А):

- users – таблиця, що зберігає дані користувачів. Користувачами є кінцеві користувачі та адміністратори. Також, для кожного нагрівача теж створюється користувач. Ця таблиця зберігає логін користувача, його електронну адресу, хеш пароля та дату створення.
- authority - ролі користувачів. Користувачі можуть мати ролі ROLE_USER, ROLE_ADMIN або ROLE_HEATER.
- user_authority – таблиця, що ставить у відповідність користувачам їх ролі.
- heater – таблиця з даними нагрівача. Зберігає серійний номер нагрівача, його потужність, користувача, за допомогою якого нагрівач отримує доступ до сервера, власника нагрівача та його локацію. Також в цій таблиці зберігаються дані про калібрацію нагрівача: статус калібрації, дати початку та кінця, та відсоток, що показує прогрес. Крім цього, в таблиці присутні поля, які відповідають за тип роботи нагрівача (вимкнений, калібрація або використання моделі), активна модель, якщо вона присутня та дата останнього порахованого розкладу.
- model – відповідає за моделі навчання з підкріпленням. Кожна модель відноситься до конкретного нагрівача, тому присутнє поле, яке вказує на відповідний нагрівач. Також є поля, що задають параметри моделі: цільову, мінімальну та максимальну температуру. Останнім є статус моделі: створена, тренується або натренована.
- location – місцезнаходження нагрівача. Складається з довготи, широти та ідентифікатора країни. Місцезнаходження використовується для

отримання прогнозу погоди. Воно також містить поле, що вказує на дату останнього оновлення погодних даних для цієї локації.

- country – країна. Використовується для отримання цін на електроенергію. Має поля для імені, коду, часової зони та дати останнього оновлення цін.
- schedule – розклад роботи нагрівача. Містить ідентифікатор нагрівача, до якого відноситься розклад, дату розклад та дані розкладу.
- latest_metrics – зберігає останні метрики від нагрівача. Ця таблиця існує для того, щоб зменшити кількість запитів до InfluxDB, де зберігаються метрики. Містить ідентифікатор нагрівача, метрики та час їх отримання.

3.3.2 БД для часових рядів InfluxDB

Дані в InfluxDB організовані не в таблиці а у виміри (measurement) – це є аналогом таблиці в звичайній реляційній базі даних. Кожен вимір складається з записів, що є аналогом рядків. Синтаксис запису:

<measurement>[,<tag_key>=<tag_value>[,<tag_key>=<tag_value>]] <field_key>=<field_value>[,<field_key>=<field_value>] [<timestamp>].

Кожен запис має вимір, до якого він відноситься, набір тегів, набір полів та час. Поля відповідають даним, що змінюються з часом (часові ряди). Теги – це метадані до полів, які можна потім використовувати для фільтрування.

Для даної системи в InfluxDB зберігаються три типи даних: метрики нагрівачів, погодні дані та дані цін на електроенергію. Відповідно, будемо мати 3 виміри:

- Вимір heater-sensors. Зберігає метрики нагрівачів. Має тег serial для збереження серійного номера нагрівача. Поля: room_t (температура в кімнаті), el_co (споживання енергії), est_cost (розраховані витрати). Поля споживання та витрат зберігають значення з початку роботи

нагрівача, тобто ці величини монотонно зростають. Щоб порахувати споживання або витрати за деякий період, необхідно розрахувати різницю значення в кінці та на початку періоду.

- Вимір out-temp. Зберігає зовнішню температуру. Має тег loc_id, що відповідає ідентифікатору локації з таблиці location в PostgreSQL. Має єдине поле temp, що зберігає температуру.
- Вимір price. Зберігає ціни на електроенергію. Має тег country_id, що відповідає ідентифікатору країни з таблиці country в PostgreSQL. Має єдине поле price, що зберігає ціну.

3.4 Бекенд

У табл. 3.1 наведено список основних ендпоінтів, що надає бекенд. При побудові ендпоінтів використовувалася методологія REST.

При розробці бекенду використовувались такі типи класів:

а) Класи-сутності (табл. 3.2), що відповідають таблицям базу даної та використовуються при відображенні реляційної моделі бази даних в об’єктну модель бекенду (ORM, object-relational mapping).

б) Репозиторії (табл. 3.3), що дають можливість виконувати операції над сутностями.

в) Сервіси (табл. 3.4) - класи, що відповідають за бізнес-логіку.

г) Контролери – класи, що описують ендпоінти, доступні користувачу, та викликають відповідні методи сервісів.

Таблиця 3.1. Список головних ендпоінтів

Ендпоінт	Опис	Параметри та тіло запиту
GET /api/heaters	Повертає список нагрівачів користувача	Відсутні
GET /api/heaters/{serial}	Повертає нагрівач за заданим серійним номером	serial – серійний номер нагрівача
POST /api/heaters	Створює новий нагрівач. Доступний лише адміністраторам	Тіло запиту – дані нового нагрівача, а саме серійний номер, пароль та потужність
POST /api/heaters/add	Додає нагрівач до акаунту користувача	Тіло запиту включає назву нового нагрівача, його серійний номер, пароль та локацію
POST /api/heaters/{serial}/start-calibration	Запускає калібрацію нагрівача	serial – серійний номер нагрівача
GET /api/heaters/{serial}/models	Повертає моделі, доступні для даного нагрівача	serial – серійний номер нагрівача

Продовження таблиці 3.1

Ендпоінт	Опис	Параметри та тіло запиту
POST /api/heaters /{serial}/models	Створює нову модель для даного користувача	serial – серійний номер нагрівача. Тіло запиту – параметри моделі: назва, цільова, мінімальна та максимальна температури, та чи потрібно одразу активувати цю модель після створення
GET /api/countries	Повертає список доступних країн	Відсутні
GET /api/heater	Повертає дані даного нагрівача. Цей ендпоінт доступний лише користувачам, що мають роль ROLE_HEATER, тобто тим, якими користуються нагрівачі для комунікації з сервером	Відсутні
GET /api/heater/schedule	Повертає розклад даного нагрівача для заданої дати. Доступний лише користувачам з роллю ROLE_HEATER	date – дату, для якої необхідно повернути розклад

Продовження таблиці 3.1

Ендпоінт	Опис	Параметри та тіло запиту
POST /api/statistics/calculate	Розраховує статистику для нагрівача. Використовується клієнтом для відображення графіків	Тіло запита – параметри для розрахунку статистики: серійний номер нагрівача, поля для розрахунку, час початку та кінця періоду, часова зона та розмір вікна для агрегації даних

Таблиця 3.2. Сутності

Сутність	Відповідна таблиця в базі даних	Опис
User	users	Користувач
Authority	authority	Роль користувача
Heater	heater	Нагрівач
Model	model	Модель машинного навчання для нагрівача
Schedule	schedule	Розклад роботи нагрівача
LatestMetrics	latest_metrics	Остання метрика нагрівача
Location	location	Місцезнаходження нагрівача
Country	country	Країна

Таблиця 3.3. Репозиторії

Репозиторій	Функціонал
HeaterRepository	Пошук нагрівача за власником. Пошук нагрівача за серійним номером. Пошук нагрівача, що відповідає поточному користувачу (для користувачів з роллю ROLE_HEATER). Збереження нового нагрівача.
LatestMetricsRepository	Збереження метрик. Пошук останніх метрик за нагрівачем.
LocationRepository	Збереження та пошук локацій.
CountryRepository	Збереження та пошук країн
ModelRepository	Збереження та пошук моделей.
ScheduleRepository	Збереження розкладу. Пошук розкладу за нагрівачем та датою.

Таблиця 3.4. Сервіси

Сервіс	Функціонал
MQTTQueueService	Підписка на топіки MQTT. Публікація повідомлень в топіки MQTT.
MetricsService	Отримання метрик від нагрівача через MQTT та їх подальша обробка та збереження. Пошук останніх метрик нагрівача.
InfluxDBService	Робота з InfluxDB. Збереження метрик в InfluxDB. Збереження прогнозу погоди в InfluxDB. Збереження цін на електроенергію в InfluxDB. Побудова запитів у InfluxDB для розрахунку статистики для нагрівача.

Продовження таблиці 3.4

Сервіс	Функціонал
KafkaService	Відправка повідомлень в топіки Kafka. Підписка на топіки Kafka. Відправка команд на сервіс машинного навчання. Отримання відповідей на команди від сервісу машинного навчання.
UpdateSchedulerService	Періодичне оновлення погодних даних локацій. Періодичне оновлення даних цін на електроенергію країн.
WeatherService	Отримання погодних даних для локацій із використанням відповідного API.
PriceService	Отримання цін на електроенергію для країн із використанням відповідного API.
StatisticsService	Розрахунок статистики для нагрівача для відображення графіків у веб-клієнті.

3.5 Сервіс машинного навчання

Сервіс машинного навчання займається побудовою спрощеної моделі приміщення, створенням та тренуванням моделей навчання з підкріплення та створенням розкладів для нагрівачей, використовуючи натреновані моделі.

Для навчання з підкріпленням, було використано метод глибокого Q-навчання (DQN), проте систему можна розширити й іншими методами за потреби. Цей метод використовує нейронну мережу, на вхід якої подаються температура в приміщенні, графік зовнішньої температури на добу вперед та графік цін на електроенергію на добу вперед. Мережа має стільки виходів,

скільки є можливих температур з кроком $1\,°C$ у проміжку, що задав користувач. Тобто кожен вихід відповідає одній можливій дії і після навчання повертає значення Q-функції для цієї дії в поточному стані. Модель вибирає дію з максимальним значенням Q-функції. У якості архітектури нейронної мережі було вибрано багатошаровий перцептрон, оскільки це найпростіша архітектура, що пришвидшує розрахунки, проте вона все ж дозволяє апроксимувати будь-яку функцію.

Основні файли сервісу машинного навчання та їх функціонал:

а) Файл environment.pyx. Містить клас LinearEnvironment, що реалізовує середовище для навчання з підкріпленням з використанням спрощеної моделі приміщення. Містить логіку фізичної симуляції. Написаний на Cython для прискорення його роботи.

б) Файл reward.pyx. Містить класи, що реалізовують різні винагороди навчання з підкріпленням. Для цього заданий базовий клас RewardPolicy, а від нього наслідуються інші класи, що реалізовують винагороди, описані у даній роботі.

в) Файл agent.py. Містить моделі машинного навчання. Основним класом є RLAgent, що реалізовує підхід навчання з підкріпленням, а саме метод глибокого Q-навчання.

г) Файл simulate.py. Містить логіку, що, симулює модель навчання з підкріпленням на заданому середовищі.

ґ) Файл train.py. Містить логіку, що тренує модель машинного навчання на заданому середовищі.

д) Файл phys_fit.py. Містить логіку, що підбирає параметри спрощеної моделі приміщення за заданими даними.

е) Файл influxdb.py. Містить логіку отримання даних з БД InfluxDB.

є) Файл kafka.py. Містить логіку отримання команд та відправлення відповідей за допомогою брокеру повідомлень Apache Kafka.

3.6 Веб-клієнт

Веб-клієнт має наступні сторінки та функціонал.

Сторінка списку нагрівачів (рис. 3.2) показує нагрівачі, що знаходяться в акаунті даного користувача. Для кожного нагрівача показана назва, серійний номер, країна та місцезнаходження.

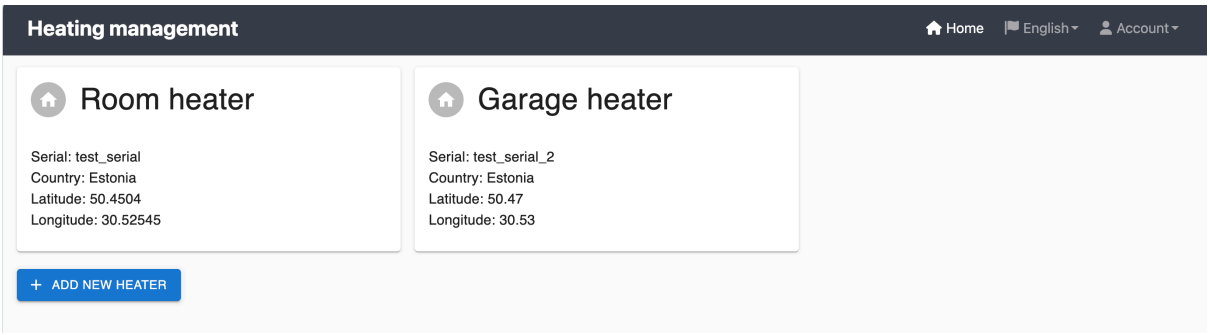


Рисунок 3.2. Сторінка списку нагрівачів

Одразу під списком нагрівачів знаходиться кнопка для додавання нового нагрівача. При її натисканні відкривається діалогове вікно (рис. 3.3), де користувач повинен ввести дані нового нагрівача.

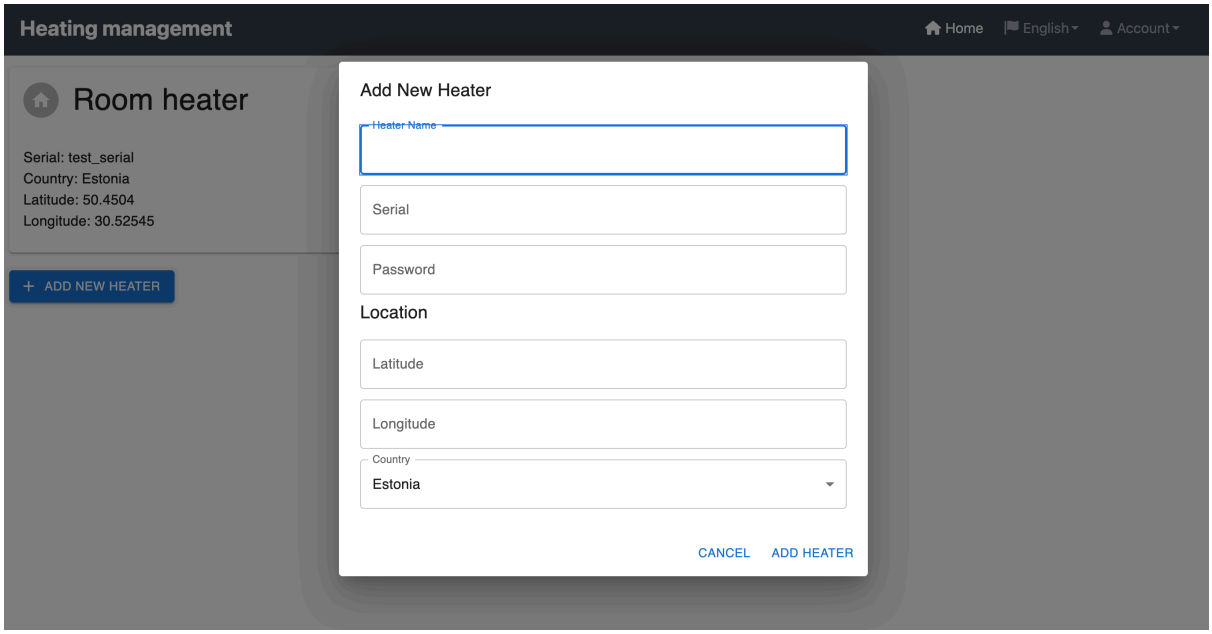


Рисунок 3.3. Діалогове вікно додавання нового нагрівача

При натисканні на будь-який з нагрівачів зі списку, користувача переходить на сторінку нагрівача (рис. 3.4). На цій сторінці він може побачити останні метрики нагрівача, такі як температура в кімнаті,

зовнішня температура, потужність нагрівача та споживання електроенергії за тиждень. Також користувач бачить список моделей, створених для цього нагрівача, разом з їх параметрами, та окремо параметри поточної моделі разом з розрахунком збережених коштів.

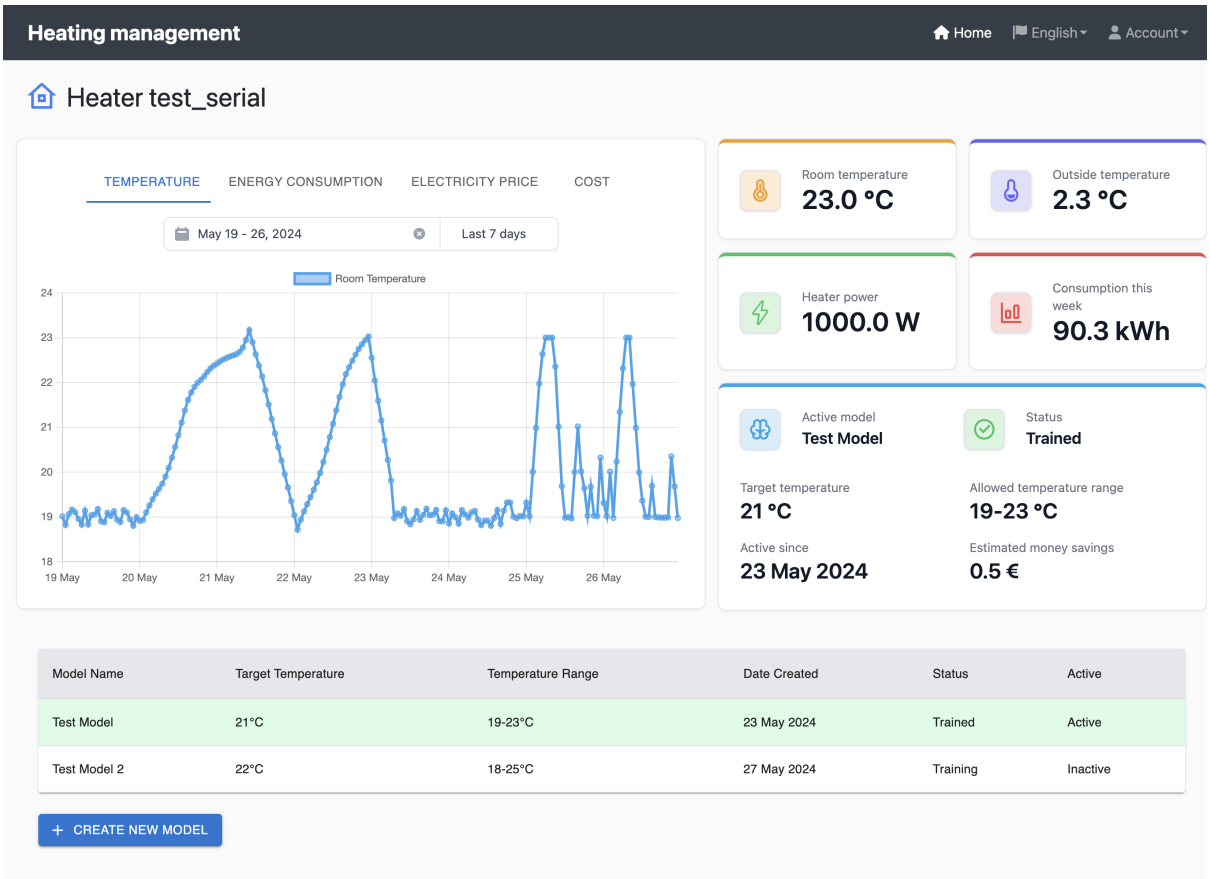


Рисунок 3.4. Сторінка нагрівача

Крім цього, користувач може передивлятись графіки цих величин. Для цього він обирає тип величини (температура, споживання електроенергії, ціна на електроенергію або витрати) та проміжок часу, за який його цікавить графік. Графік оновлюється в режимі реального часу, коли приходять нові дані з нагрівача. Приклади різних графіків зображені на рис. 3.4–3.6.

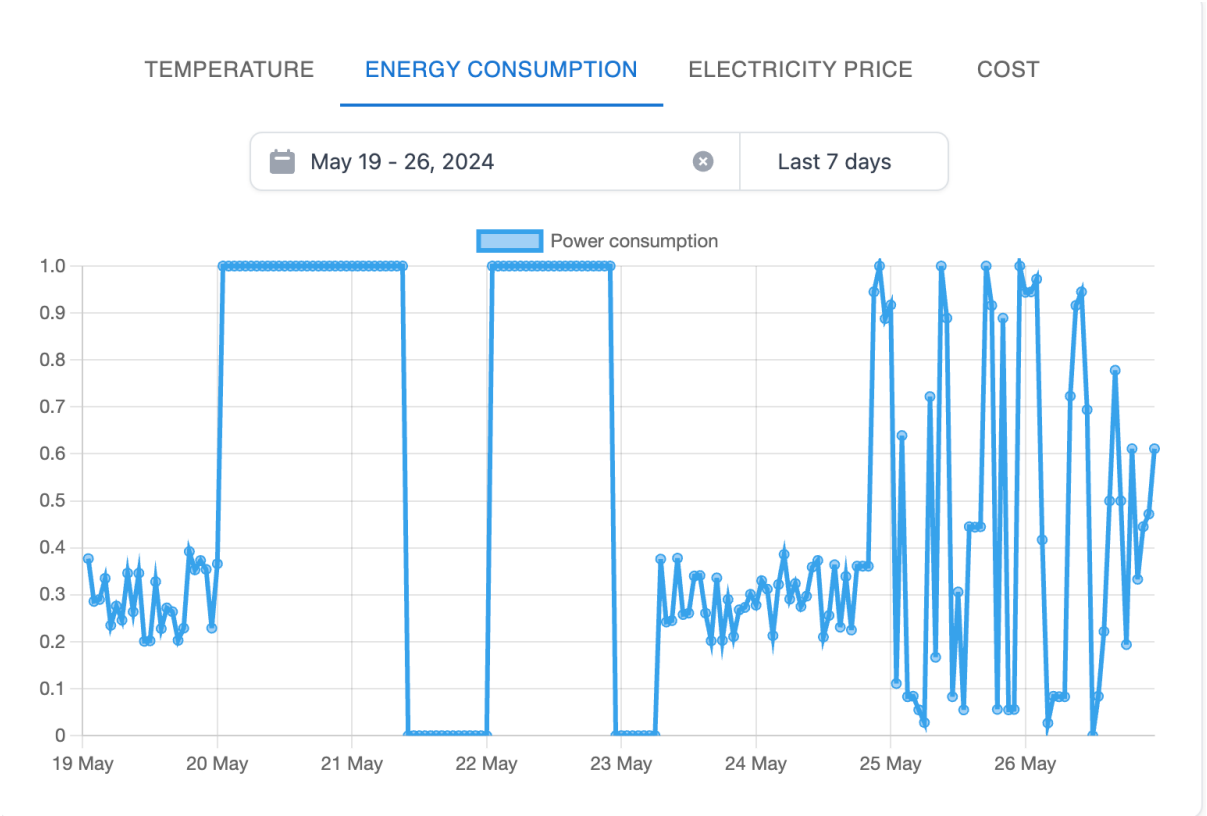


Рисунок 3.5. Приклад графіку потужності нагрівача

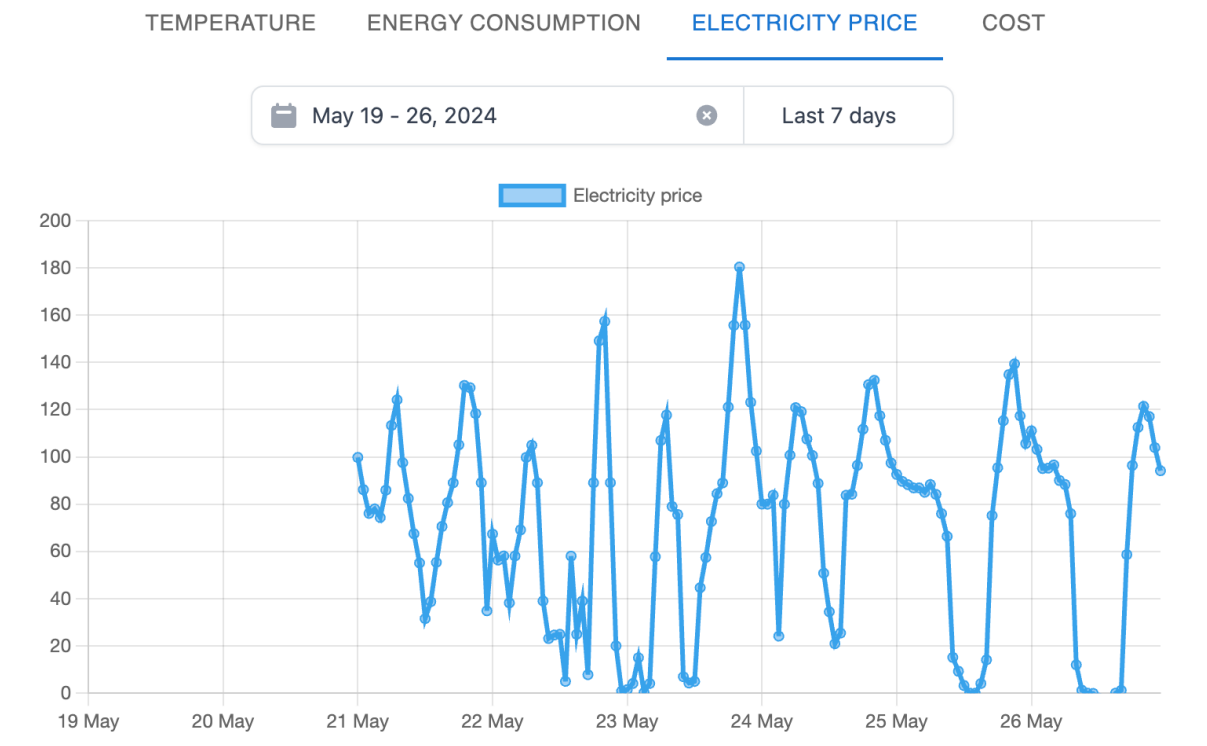


Рисунок 3.6. Приклад графіку ціни на електроенергію

Калібрування та створення моделі відбувається наступним чином. Коли користувач вперше відкриває сторінку нового нагрівача, він бачить повідомлення про те, що нагрівач не відкалібрований (рис. 3.7) та кнопку для початку калібрування. Після її натискання, починається процес калібрування, і користувач може бачити його прогрес (рис. 3.8).

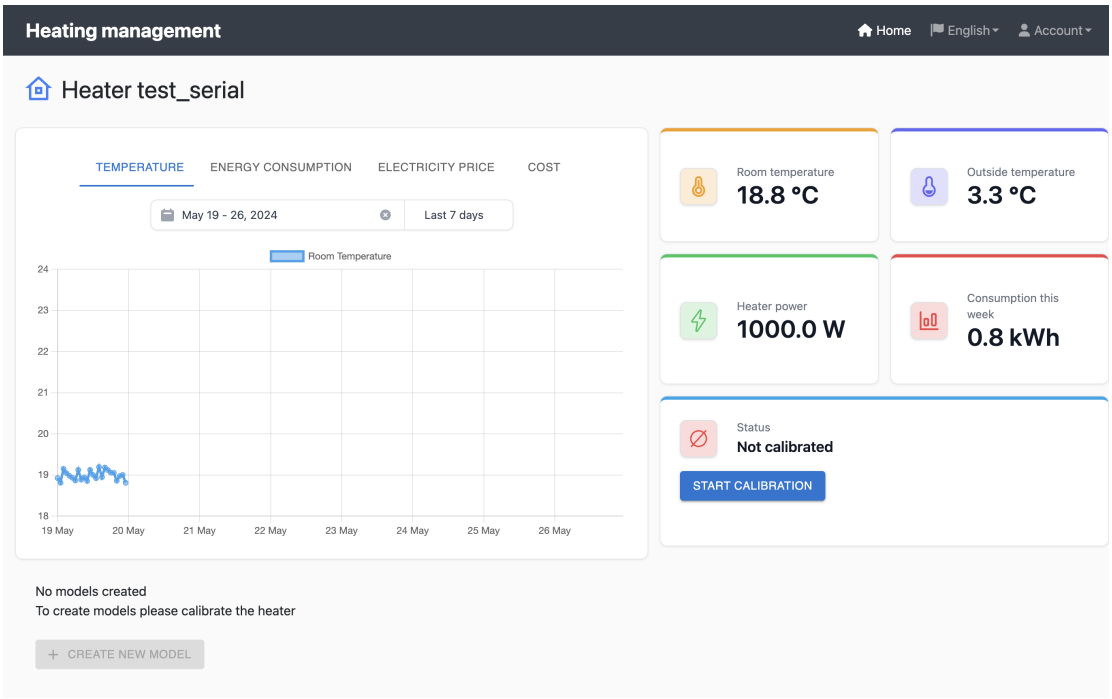


Рисунок 3.7. Сторінка невідкаліброваного нагрівача

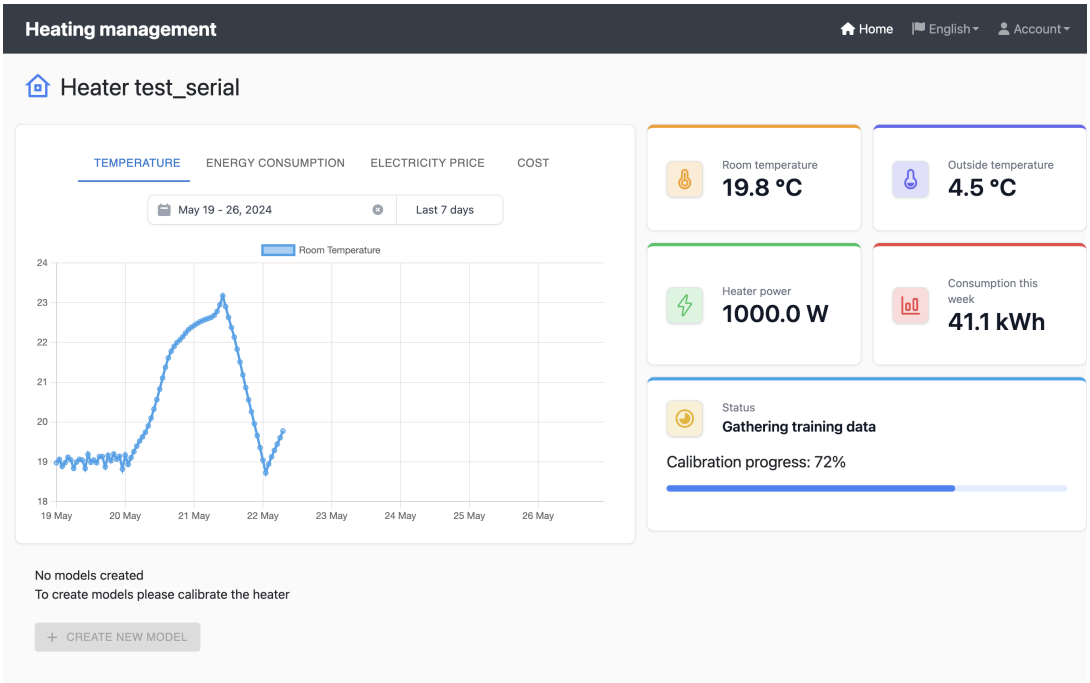


Рисунок 3.8. Сторінка нагрівача під час калібрування

Після завершення процесу калібрування, користувачу стає доступна кнопка створення нової моделі. При створенні моделі він вказує її назву, температурні параметри та чи потрібно її активувати негайно після створення (рис. 3.9).

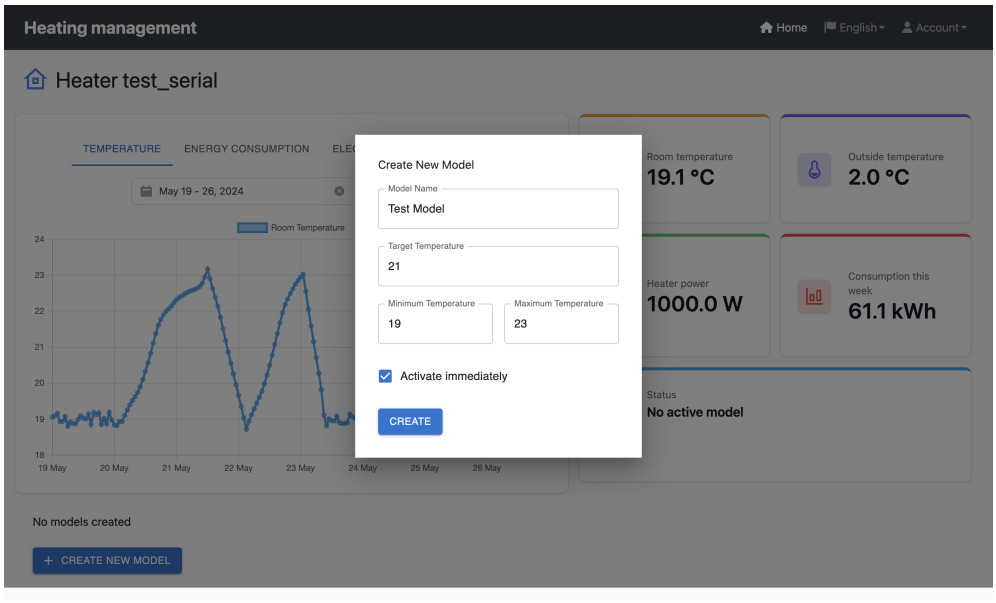


Рисунок 3.9. Діалогове вікно створення нової моделі

Після створення, модель одразу починає навчатися (рис. 3.10). Після завершення навчання, модель активується, та починає керувати роботою нагрівача (рис. 3.4).

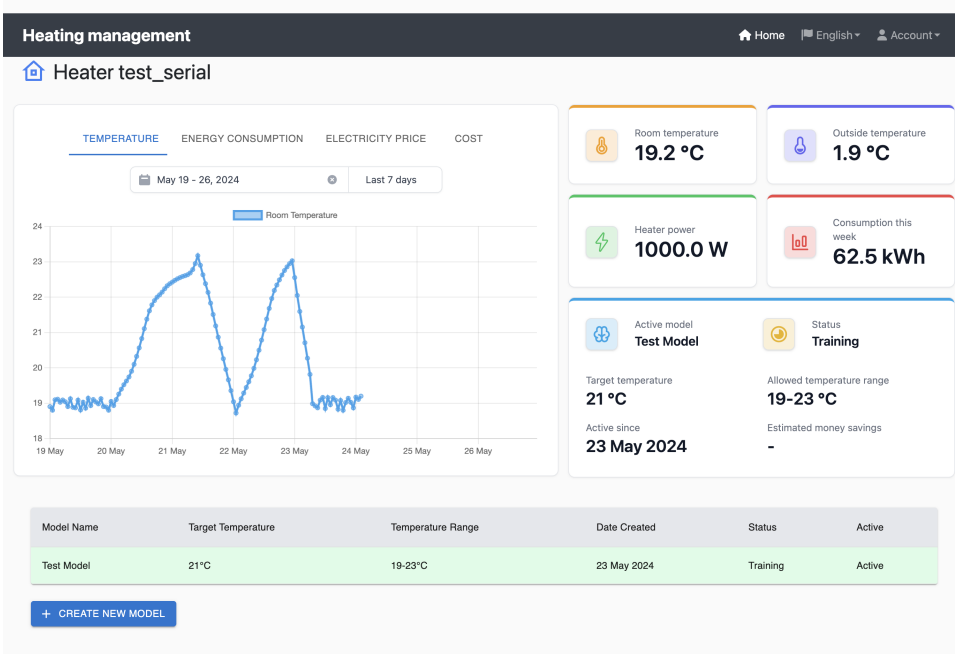


Рисунок 3.10. Сторінка нагрівача під час тренування моделі

Висновки до розділу

У даному розділі розглянуто програмну реалізацію запропонованого методу у вигляді серверного застосунку. Наведено архітектуру застосунку, що складається з сервісу машинного навчання, нагрівача, веб-клієнту, баз даних та черг повідомлень для комунікації. Наведено список технологій, що застосовувались для реалізації застосунку.

Описано схеми бази даних. Наведено список сутностей та їх опис. Також розглянуто використання нереляційної бази даних InfluxDB для зберігання даних, що мають вигляд часових рядів.

Описано архітектуру бекенду, список його класів, їх функціонал, та взаємодію один з одним. Описано функціонал веб-клієнту, наведено знімки екрану різних сторінок в різних станах системи та пояснено приклад використання клієнту користувачем.

Розглянуто сервіс машинного навчання, вибір методу в рамках підходу навчання з підкріпленням та вигляд нейронної мережі. Наведено список модулів, з яких складається програмна реалізація, та їх функціонал.

4 ОЦІНКА ЗАПРОПОНОВАНОГО РІШЕННЯ

4.1 Тестування роботи спрощеної моделі приміщення

Для застосування запропонованої спрощеної моделі приміщення, необхідно довести, що вона є достатньо точною та може передати суть фізичного явища нагрівання приміщень без суттєвих спотворень. Для цього її було порівняно з більш точною та широко використовуваною моделлю 5R1C [47]. Для цього порівняння, було виконано симуляцією більш точною моделлю, на отриманих даних підібрано параметри спрощеної моделі, після чого проведена симуляція вже спрощеної моделі. Для симуляції використовувались погодні дані Києва. Фрагмент графіку отриманих даних зображено на рис. 4.1.

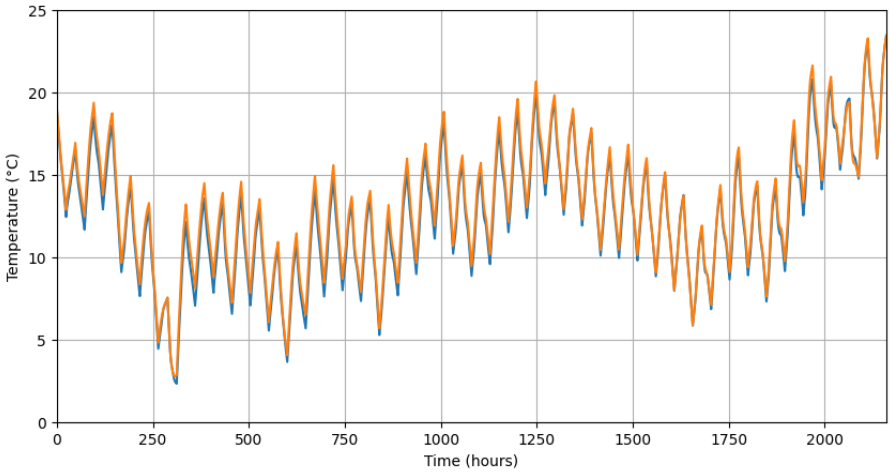


Рисунок 4.1. Порівняння моделі 5R1C (синя лінія) та спрощеної моделі, використаної в даній роботі (жовта лінія)

Для чисельного порівняння, було використано метрики MAE (mean absolute error, середнє абсолютне відхилення) та MAPE (mean absolute percentage error, середнє абсолютне відсоткове відхилення) та отримано такі результати:

$$MAE = 0.53°C, \quad MAPE = 3.1\%.$$

Оскільки при практичному використанні користувач буде задавати температури з точністю до одного градуса, то отриману точність в

половину градуса можна вважати достатньою для застосування даної спрощеної моделі приміщення.

4.2 Порівняння винагород навчання з підкріпленням

Перейдемо до порівняння винагород, розглянутих у підрозділі 2.3. Для цього було використано дві метрики: економію коштів, порівняно з підтримкою постійної заданої температури та середню температуру в приміщенні. Під час тестування цільова температура була встановлена в $21°C$, а дозволені межі температур – 19-23$°C$. Використовувались дані цін та погоди для Естонії за 2019-2023 роки.

Оскільки розглянуті винагороди мають вільний параметр β, то насправді ми маємо набір винагород для різних значень β. Щоб їх порівняти, було розраховано метрики для набору значень цього параметру, після чого побудовано залежність збережених коштів від середньої температури в приміщенні (рис. 4.2).

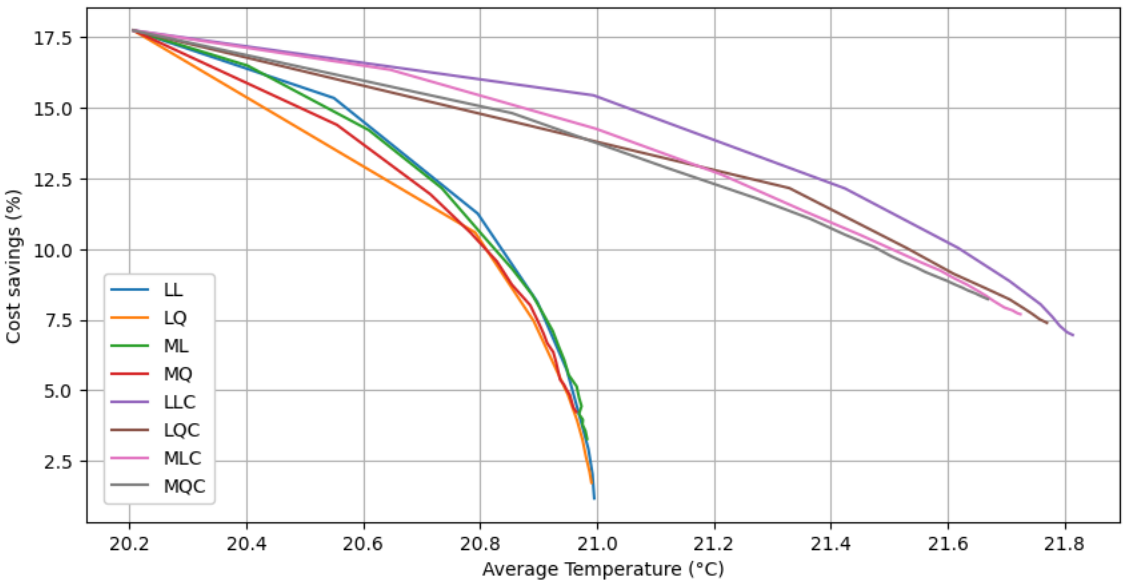


Рисунок 4.2. Порівняння різних винагород. По осі x відкладено середню температуру у приміщенні, по осі y – збережені кошти, порівняно з підтриманням постійної температури 21°C

Тут кожна лінія відповідає одній винагороді, а різні точки відповідають різним значенням β . Малим значенням β відповідають точки лівіше на графіку, оскільки такі значення пріоритизують економію коштів за рахунок більш низької температури в приміщенні, тоді як великим значенням β відповідає пріоритизація більш високої температури, і відповідно правіші точки.

Винагорода тим краще, чим вище знаходиться лінія на графіку, оскільки це означає більшу економію при тих же значеннях середньої температури.

Аналізуючи графік, можемо зробити висновок, що дуже сильний вплив має обрізання винагороди r_T при $T > T_0$. Відповідні винагороди дають кращі показники для всього доступного проміжку значень. Це можна пояснити тим, що при такому обрізанні, модель не отримує таке сильне покарання при знаходженні в області $T > T_0$, як при його відсутності. Таким чином, модель має більшу ефективно доступну область температур для використання, що й домагає їй отримати кращі результати.

Найкращою з розглянутих є винагорода r_{LLC}, що задається формулою (2.11). Вона дає помітно кращі результати, ніж інші винагороди з обрізанням. Також варто відмітити, що серед винагород без обрізання найкращою є r_{LL}, що також підтверджує те, що цей вигляд винагороди є найоптимальнішим.

4.3 Тестування роботи серверного застосунку

Для перевірки роботоздатності та коректності роботи серверного застосунку, було проведено його тестування. Замість фізичного нагрівача використовувався віртуальний нагрівач, що моделював фізичні процеси за допомогою комп’ютерної симуляції. Це дозволяє значно спростити тестування, а також пришвидшити його. Пришвидшення відбувається за рахунок того, що комп’ютерну симуляцію можна виконувати швидше, ніж відповідний реальний фізичний процес. Під час тестування, було

розглянуто проміжок часу обсягом в тиждень. Проте, завдяки використанню симуляції, відповідне тестування можна провести за хвилини.

Оскільки у системі, що тестується, теж використовується модель середовища, то, щоб тестування мало сенс, віртуальний нагрівач повинен використовувати більш точну модель. В даному випадку використовувалась модель 5R1C, що вже використовувалась у розділі 2.1 та є досить поширеною для симуляції приміщень та навіть зафіксована в міжнародному стандарті ISO 13790 [47].

Під час тестування було виконано наступну послідовність дії:

а) Відкрито сторінку керування панеллю у веб-клієнті.

б) Запущено віртуальний нагрівач разом із симуляцією приміщення. При цьому було перевірено, що дані симуляції відображаються на графіку температури та постійно оновлюються.

в) Запущено процес калібрування нагрівача.

г) Очікувався кінець калібрування.

ґ) Після закінчення калібрування, створено нову модель машинного навчання, та відразу активовано.

Результати тестового запуску системи зображені на рис. 4.3.

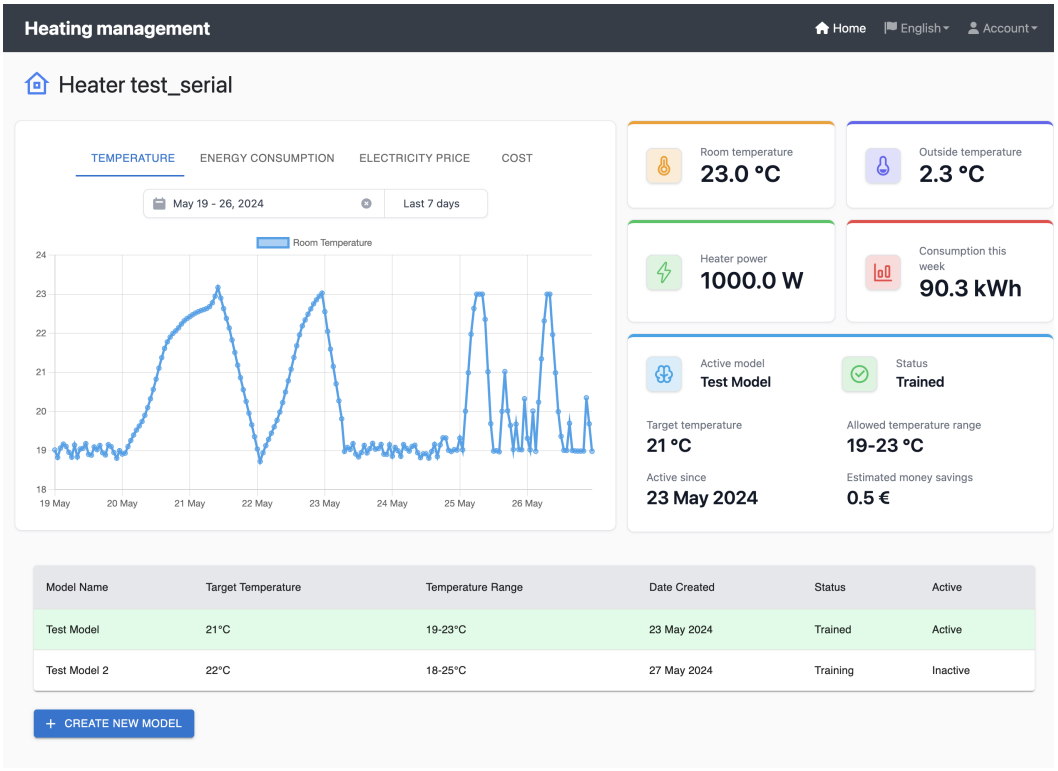


Рис. 4.3. Тестовий запуск системи разом із віртуальним нагрівачем

Графік залежності температури від часу складається з таких частин:

а) Одну добу нагрівач підтримував постійну температуру 19 °C. Це відповідає режиму роботи перед початком калібрування.

б) Після початку калібрування, нагрівач працював циклічно, спочатку нагріваючись до 23 °C, після чого охолоджуючись назад до 19 °C. Таким чином він збирав дані для побудови спрощеної моделі приміщення. Він встиг виконати два повні цикли за трохи більше, ніж 3 доби. Можна звернути увагу, що нагрівання приміщення відбувається з різною швидкістю (різний нахил графіку) в різні моменти часу. Це пов’язано з варіацією температури ззовні приміщення.

в) Після закінчення калібрування, нагрівач повернувся до підтримання постійної температури 19 °C. Цей проміжок відповідає тренуванню моделі машинного навчання. В даній симуляції це зайняло 1.5 доби, і це пов’язано з тим, що фізична симуляція була пришвидшена, проте

тренування моделі машинного навчання відбувалося зі звичайною швидкістю. В реальному часі це тренування займає порядку хвилин.

г) Після закінчення тренування настає період зі складною залежністю температури від часу. Це відповідає проміжку, коли нагрівачем почала керувати модель машинного навчання. Аналізуючи графік, можемо побачити, що вона сильніше прогріває приміщення вночі, коли ціни менші, що має сенс.

Таким чином, аналізуючи проведене тестування, можна дійти висновку, що розроблена система працює коректно та очікувано, без втручання користувача проводить збір даних, будує модель приміщення, модель машинного навчання та застосовує її для керування нагрівачем з метою зменшення витрат.

Висновки до розділу

У даному розділі обґрунтовано доцільність застосування запропонованої спрощеної моделі приміщення без значних розбіжностей.

Проведено порівняння запропонованих винагород для навчання з підкріпленням для задачі оптимізації витрат. У якості метрик було використано зекономлені кошти та середнє відхилення температури. Знайдено оптимальну винагороду для даної задачі.

Було проведено тестування розробленого серверного застосунку із використанням віртуального нагрівача. Розглянуто повний цикл роботи, починаючи зі збору даних і закінчуючи побудовою спрощеної моделі приміщення, побудовою моделі машинного навчання, та її застосуванням до керування нагрівачем. Проаналізовано отриману залежність температури від часу, та отримано висновок, що система працює так, як очікувалось.

ВИСНОВКИ

У ході виконання магістерської дисертації було розглянуто задачу керування опаленням та оптимізації витрат за рахунок врахування змінного графіка цін на електроенергію. Оглянуті існуючі методи її розв’язання. Було зроблено детальний огляд підходу навчання з підкріпленням, що є найпоширенішим для розв’язку цієї задачі. Описано обмеження цього підходу при практичному застосуванні.

Сформульовано задачу по удосконаленню підходу навчання з підкріпленням при застосуванні до задачі керування опаленням. Розроблено модифікацію цього підходу, що значно спрощує його застосування на практиці за рахунок побудови спрощеної моделі приміщення. Побудована спрощена модель приміщення та проведено її порівняння з більш точними моделями для обґрунтування доцільності застосування.

Було запропоновано ряд винагород для навчання з підкріпленням для даної задачі та проведено їх детальний аналіз та порівняння. Виявлено винагороду, що дозволяє отримати найкращі результати в рамках задачі опалення.

Розроблено серверний застосунок, що реалізує запропонований метод та надає користувачу доступ до веб-клієнту для моніторингу та управління опаленням. При розробці використовувались мови Kotlin, Python, Cython та Typescript, бази даних PostgreSQL та InfluxDB, черги повідомлень Apache Kafka та MQTT та бібліотеки Spring, React, Redux, Tailwind CSS, NumPy та PyTorch. Усі наведені засоби є широко вживаними та перевіреними спільнотою розробників, постійно оновлюються та мають відкритий вихідний код.

Розроблений застосунок складається з трьох модулей: сервісу машинного навчання, що реалізує запропонований удосконалений метод навчання з підкріпленням, бекенду, що керує панеллю, збирає дані та

координує дії сервісу машинного навчання, та веб-клієнту, що дозволяє керувати системою.

Наукова новизна полягає в тому, що вперше запропонована архітектура програмного забезпечення керування витратами на опалення, в якій для управління нагрівачами застосовується модель приміщення, параметри якої знаходяться експериментальним шляхом. Таким чином вона не потребує ручного введення параметрів приміщення користувачем.

Практична значимість виконаної роботи полягає у тому, що запропоновано метод керування опаленням, що дозволяє значно підвищити економічну ефективність опалення приміщень, враховуючи денну варіацію цін.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Nejat P., Jomehzadeh F., Taheri M.M., Gohari M., Majid M.Z..A. A global review of energy consumption, co2 emissions and policy in the residential sector (with an overview of the top ten co2 emitting countries) – *Renewable and Sustainable Energy Reviews*, 2015; 43 – 843–862 p.
2. Wei T., Wang Y., Zhu Q.. Deep reinforcement learning for building HVAC control – *Proceedings of the 54th annual design automation conference,* 2017, ACM – 22 p.
3. Nord Pool. [Електронний ресурс] // Режим доступу до ресурсу: https://www.nordpoolgroup.com/.
4. Knapik, O. Modeling and Forecasting Electricity Price Jumps in the Nord Pool Power Market - *CREATES Research Paper* 2017-7 - Department of Economics and Business Economics, Aarhus University: Aarhus, Danmark, 2017.
5. Alimohammadisagvand, B.; Alam, S.; Ali, M.; Degefa, M.; Jokisalo, J.; Sirén, K. Influence of Energy Demand Response Actions on Thermal Comfort and Energy Cost in Electrically Heated Residential Houses - *Indoor and Built Environment.* 2017, *26* – 298–316 p.
6. Behl, M.; Jain, A.; Mangharam, R. Data-Driven Modeling, Control and Tools for Cyber-Physical Energy Systems - *Proceedings of the 2016 ACM/IEEE 7th International Conference on Cyber-Physical Systems (ICCPS)*, 11–14 April 2016 – Vienna, Austria.;
7. Péan, T.Q.; Salom, J.; Costa-Castelló, R. Review of Control Strategies for Improving the Energy Flexibility Provided by Heat Pump Systems in Buildings - *Journal of Process Control* 2019, *74* – 35–49 p.
8. Avci, M.; Erkoc, M.; Rahmani, A.; Asfour, S. Model Predictive HVAC Load Control in Buildings Using Real-Time Electricity Pricing – *Energy and Buildings* 2013, *60* – 199–209p.

9. Li, X.; Malkawi, A. Multi-Objective Optimization for Thermal Mass Model Predictive Control in Small and Medium Size Commercial Buildings under Summer Weather Conditions – *Energy* 2016, *112* – 1194–1206 p.
10. Zhang, H.; Seal, S.; Wu, D.; Bouffard, F.; Boulet, B. Building Energy Management with Reinforcement Learning and Model Predictive Control: A Survey - *IEEE Access* 2022, *10* - 27853–27862 p.
11. Fischer, D.; Bernhardt, J.; Madani, H.; Wittwer, C. Comparison of Control Approaches for Variable Speed Air Source Heat Pumps Considering Time Variable Electricity Prices and PV - *Applied Energy* 2017, *204* - 93–105 p.
12. Vandermeulen, A.; Vandeplas, L.; Patteeuw, D.; Sourbron, M.; Helsen, L. Flexibility Offered by Residential Floor Heating in a Smart Grid Context: The Role of Heat Pumps and Renewable Energy Sources in Optimization towards Different Objectives - *Proceedings of the IEA Heat Pump Conference*, 15–18 May 2017 - Rotterdam, The Netherlands.
13. Ala'raj, M.; Radi, M.; Abbod, M.F.; Majdalawieh, M.; Parodi, M. Data-Driven Based HVAC Optimisation Approaches: A Systematic Literature Review - *Journal of Building Engineering* 2022, *46*.
14. Liu, T.; Xu, C.; Guo, Y.; Chen, H. A Novel Deep Reinforcement Learning Based Methodology for Short-Term HVAC System Energy Consumption Prediction - *International Journal of Refrigeration* 2019, *107* - 39–51 p.
15. Raza, R.; Hassan, N.U.; Yuen, C. Determination of Consumer Behavior Based Energy Wastage Using IoT and Machine Learning - *Energy and Buildings* 2020, *220*.
16. Esrafifilian-Najafabadi, M.; Haghighat, F. Impact of Occupancy Prediction Models on Building HVAC Control System Performance: Application of Machine Learning Techniques - *Energy and Buildings* 2022, *257*.

17. Chaudhuri, T.; Soh, Y.C.; Li, H.; Xie, L. Machine Learning Based Prediction of Thermal Comfort in Buildings of Equatorial Singapore - *Proceedings of the 2017 IEEE International Conference on Smart Grid and Smart Cities (ICSGSC)*, Singapore, 23–26 July 2017 - 72–77 p.
18. Kusiak, A.; Tang, F.; Xu, G. Multi-Objective Optimization of HVAC System with an Evolutionary Computation Algorithm - *Energy* 2011, *36* - 2440–2449 p.
19. Nassif, N. Modeling and Optimization of HVAC Systems Using Artifificial Neural Network and Genetic Algorithm - *Building Simulation* 2014, *7* - 237–245 p.
20. Amin, U.; Hossain, M.J.; Fernandez, E. Optimal Price Based Control of HVAC Systems in Multizone Offifice Buildings for Demand Response - *Journal of Cleaner Production,* 2020, *270*.
21. Yuan, X.; Pan, Y.; Yang, J.; Wang, W.; Huang, Z. Study on the Application of Reinforcement Learning in the Operation Optimization of HVAC System - *Building Simulation.* 2021, *14* - 75–87 p.
22. Fazenda, P.; Veeramachaneni, K.; Lima, P.; O'Reilly, U.-M. Using Reinforcement Learning to Optimize Occupant Comfort and Energy Usage in HVAC Systems - *Journal of Ambient Intelligence and Smart Environments.* 2014, *6* - 675–690 p.
23. Liu, S.; Henze, G.P. Experimental Analysis of Simulated Reinforcement Learning Control for Active and Passive Building Thermal Storage Inventory: Part 1. Theoretical Foundation - *Energy and Buildings* 2006, *38* - 142–147 p.
24. Barrett Enda and Linder, S. Autonomous HVAC Control, A Reinforcement Learning Approach - *Proceedings of the Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD,* 2015, Porto, Portugal, 7–11 September 2015; Springer International Publishing: Cham, Germany, 2015 - 3–19 p.

25. Costanzo, G.T.; Iacovella, S.; Ruelens, F.; Leurs, T.; Claessens, B.J. Experimental Analysis of Data-Driven Control for a Building Heating System - *Sustainable Energy, Grids and Networks,* 2016, *6* - 81–90 p.
26. Ruelens, F.; Iacovella, S.; Claessens, B.; Belmans, R. Learning Agent for a Heat-Pump Thermostat with a Set-Back Strategy Using Model-Free Reinforcement Learning - *Energy* 2015, *8* - 8300–8318 p.
27. Li, B.; Xia, L. A Multi-Grid Reinforcement Learning Method for Energy Conservation and Comfort of HVAC in Buildings - *Proceedings of the 2015 IEEE International Conference on Automation Science and Engineering (CASE)*, Gothenburg, Swede, 24–28 August 2015 - 444–449 p.
28. Wei, T.; Wang, Y.; Zhu, Q. Deep Reinforcement Learning for Building HVAC Control - *Proceedings of the 54th Annual Design Automation Conference*, 2017, Austin, TX, USA, 18–22 June 2017 - 1–6 p.
29. Azuatalam, D.; Lee, W.-L.; de Nijs, F.; Liebman, A. Reinforcement Learning for Whole-Building HVAC Control and Demand Response. *Energy AI* 2020, *2*, 100020.
30. Biswas, D. Reinforcement Learning Based HVAC Optimization in Factories - *Proceedings of the Eleventh ACM International Conference on Future Energy Systems*, Online, 22–26 June 2020 - 428–433 p.
31. Gupta, A.; Badr, Y.; Negahban, A.; Qiu, R.G. Energy-Effificient Heating Control for Smart Buildings with Deep Reinforcement Learning - *Journal of Building Engineering* - 2021, *34*.
32. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm *arXiv preprint: 1712.01815.*
33. A review on reinforcement learning algorithms and applications in supply chain management - *International Journal of Production Research*, 2023, vol. 61, no. 20 - 7151–7179 p.

34. Trends, Google. The growth in search of reinforcement learning. [Електронний ресурс] // Режим доступу до ресурсу: https://trends.google.com/trends/explore?date=2007-01-01%202022-01-01&q=reinforcement%20learning&hl=en-US.
35. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Ried-miller, M., 2013. Playing atari with deep reinforcement learning. NIPS.
36. Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., Riedmiller, M., 2014. Deterministic policy gradient algorithms. ICML.
37. Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint:1509. 02971*.
38. Sutton, R.S., McAllester, D., Singh, S., Mansour, Y., 1999. Policy gradient methods for reinforcement learning with function approximation. NIPS 1057–1063.
39. Fujimoto, S., Hoof, H., Meger, D., 2018. Addressing function approximation error in actor-critic methods. ICML *arXiv preprint:1802.09477*.
40. Schulman, J., Levine, S., Moritz, P., Jordan, M.I., Abbeel, P., 2015. Trust region policy optimization. ICML *arXiv preprint:1502.05477*.
41. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O., 2017. Proximal policy optimization algorithms. Mach. Learn. *arXiv preprint:1707.06347*.
42. Haarnoja, T., Zhou, A., Abbeel, P., Levine, S., 2018. Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor. ICML *arXiv preprint:1801.01290*.
43. ENTSO-E Transparency Platform. [Електронний ресурс] // Режим доступу до ресурсу: https://transparency.entsoe.eu/

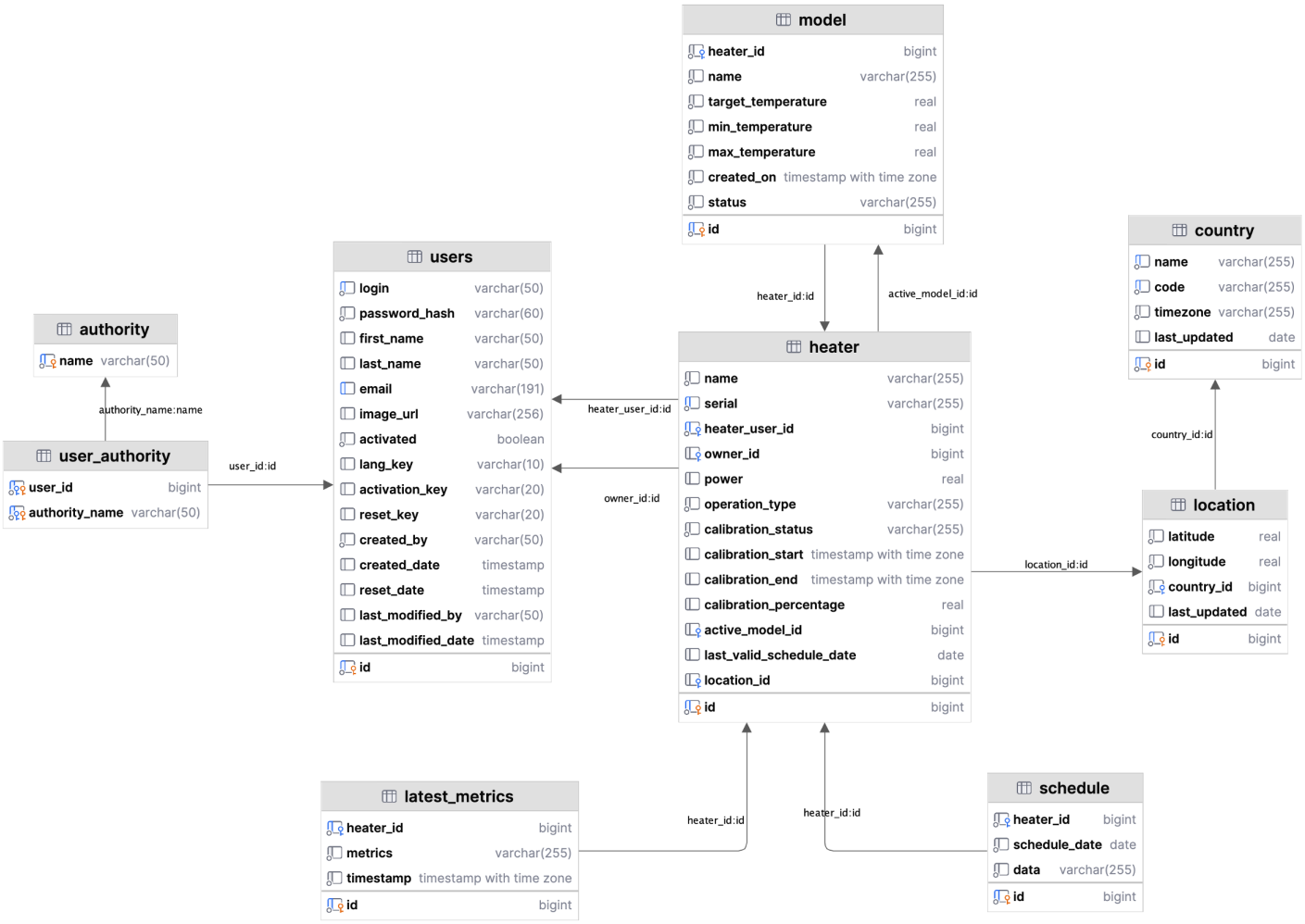
44. Energy Plus. [Електронний ресурс] // Режим доступу до ресурсу: https://energyplus.net
45. ISO 13790:2008; Energy Performance of Buildings—Calculation of Energy Use for Space Heating and Cooling.
46. Ashkan Haji Hosseinloo, Alexander Ryzhov, Aldo Bischi, Henni Ouerdane, Konstantin Turitsyn, Munther A. Dahleh, Data-driven control of micro-climate in buildings: An event-triggered reinforcement learning approach - *Applied Energy*, Volume 277, 2020.
47. Kannari, L.; Kantorovitch, J.; Piira, K.; Piippo, J. Energy Cost Driven Heating Control with Reinforcement Learning - *Buildings* 2023, 13 – 427 p.
48. Kotlin. [Електронний ресурс] // Режим доступу до ресурсу: https://kotlinlang.org
49. Spring. [Електронний ресурс] // Режим доступу до ресурсу: https://spring.io
50. PostgreSQL. [Електронний ресурс] // Режим доступу до ресурсу: https://www.postgresql.org
51. InfluxDB. [Електронний ресурс] // Режим доступу до ресурсу: https://www.influxdata.com
52. Python. [Електронний ресурс] // Режим доступу до ресурсу: https://www.python.org
53. PyTorch. [Електронний ресурс] // Режим доступу до ресурсу: https://pytorch.org
54. Cython. [Електронний ресурс] // Режим доступу до ресурсу: https://cython.org
55. Apache Kafka. [Електронний ресурс] // Режим доступу до ресурсу: https://kafka.apache.org
56. TypeScript. [Електронний ресурс] // Режим доступу до ресурсу: https://www.typescriptlang.org

57. WebSocket. [Електронний ресурс] // Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API
58. STOMP. [Електронний ресурс] // Режим доступу до ресурсу: https://stomp.github.io
59. MQTT. [Електронний ресурс] // Режим доступу до ресурсу: https://mqtt.org

ДОДАТКИ

ДОДАТОК А ГРАФІЧНИЙ МАТЕРІАЛ

СХЕМА БАЗИ ДАНИХ



ДОДАТОК Б ТЕКСТИ ПРОГРАМНОГО КОДУ

UserHeaterResource.kt

```
package org.kry.thesis.web.rest

import org.kry.thesis.domain.*
import org.kry.thesis.security.ADMIN
import org.kry.thesis.service.facade.*
import org.springframework.security.access.prepost.PreAuthorize
import org.springframework.web.bind.annotation.*

@RestController
@RequestMapping("/api/heaters")
class UserHeaterResource(
    private val heaterFacade: HeaterFacade,
) {
    @GetMapping
    fun getCurrentUserHeaters(): List<Heater> =
        heaterFacade.getCurrentUserHeaters()

    @GetMapping("/{serial}")
    fun getHeater(@PathVariable serial: String): HeaterDTO =
        heaterFacade.getBySerialForCurrentUser(serial)

    @PostMapping
    @PreAuthorize("hasAuthority(\"$ADMIN\")")
    fun createHeater(@RequestBody newHeaterDTO: NewHeaterDTO): Unit =
        heaterFacade.createHeater(newHeaterDTO)

    @PostMapping("/add")
    fun addHeater(@RequestBody addHeaterDTO: AddHeaterDTO): Unit =
        heaterFacade.addHeaterToCurrentUser(addHeaterDTO)

    @PostMapping("/{serial}/start-calibration")
    fun startCalibration(@PathVariable serial: String) {
        heaterFacade.startCalibration(serial)
    }

    @GetMapping("/{serial}/models")
    fun getHeaterModels(@PathVariable serial: String): List<Model> =
        heaterFacade.getHeaterModels(serial)

    @PostMapping("/{serial}/models")
    fun createModel(@PathVariable serial: String, @RequestBody
newModelDTO: NewModelDTO) {
        heaterFacade.createModel(serial, newModelDTO)
    }
}
```

HeaterResource.kt

```
package org.kry.thesis.web.rest

import org.kry.thesis.security.HEATER
import org.kry.thesis.service.facade.*
import org.springframework.security.access.prepost.PreAuthorize
import org.springframework.web.bind.annotation.*
import java.time.LocalDate
```

```
@RestController
@RequestMapping("/api/heater")
@PreAuthorize("hasAuthority(\"$HEATER\")")
class HeaterResource(private val heaterFacade: HeaterFacade) {
    @GetMapping
    fun getCurrentHeater(): HeaterDTO =
        heaterFacade.getCurrentHeater()

    @GetMapping("/schedule")
    fun getSchedule(@RequestParam date: LocalDate): ScheduleDTO =
        ScheduleDTO(
            schedule =
heaterFacade.getScheduleForCurrentHeater(date)?.data
        )
}

data class ScheduleDTO(
    val schedule: String?
)
```

CountryResource.kt

```
package org.kry.thesis.web.rest

import org.kry.thesis.domain.Country
import org.kry.thesis.service.CountryService
import org.springframework.web.bind.annotation.*

@RestController
@RequestMapping("/api/countries")
class CountryResource(private val countryService: CountryService) {
    @GetMapping
    fun getCountries(): List<Country> =
        countryService.findAll()
}
```

HeaterFacade.kt

```
package org.kry.thesis.service.facade

import org.kry.thesis.domain.*
import org.kry.thesis.service.*
import org.kry.thesis.service.dto.AdminUserDTO
import org.kry.thesis.service.metrics.*
import org.kry.thesis.service.metrics.HeaterMetric.*
import org.springframework.context.ApplicationEventPublisher
import org.springframework.context.event.EventListener
import org.springframework.http.HttpStatus
import org.springframework.messaging.simp.SimpMessagingTemplate
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder
import org.springframework.stereotype.Component
import org.springframework.transaction.annotation.*
import org.springframework.transaction.event.*
import org.springframework.web.server.ResponseStatusException
import java.time.*

@Component
```

```
@Transactional
class HeaterFacade(
    private val heaterService: HeaterService,
    private val userService: UserService,
    private val metricsService: MetricsService,
    private val statisticsService: StatisticsService,
    private val websocketPublisher: SimpMessagingTemplate,
    private val modelService: ModelService,
    private val applicationEventPublisher: ApplicationEventPublisher,
    private val passwordEncoder: BCryptPasswordEncoder,
    private val locationService: LocationService,
    private val countryService: CountryService,
    private val kafkaService: KafkaService,
    private val scheduleService: ScheduleService
) {
    fun getCurrentUserHeaters(): List<Heater> {
        val user = userService.getCurrentUser()
        return heaterService.findByOwnerId(user.id!!)
    }

    fun getBySerialForCurrentUser(serial: String): HeaterDTO {
        val user = userService.getCurrentUser()
        val heater = heaterService.findBySerial(serial)
        if (heater.owner != user) {
            throw HeaterAccessForbiddenException()
        }

        return heater.toHeaterDTO()
    }

    fun getCurrentHeater(): HeaterDTO =
        heaterService.findCurrentHeater().toHeaterDTO()

    private fun Heater.toHeaterDTO(): HeaterDTO {
        val metrics =
metricsService.findLatestMetrics(this)?.metrics?.let
{ parseMetrics(it) } ?: emptyMap()
        val lastWeekConsumption =
statisticsService.calculateLastWeekConsumption(this)

        return HeaterDTO(
            id = this.id!!,
            name = this.name,
            serial = this.serial,
            roomTemperature = metrics[ROOM_TEMPERATURE],
            outsideTemperature = metrics[OUTSIDE_TEMPERATURE],
            heaterPower = this.power,
            weekConsumption = lastWeekConsumption,
            operationType = this.operationType,
            calibrationStatus = this.calibrationStatus,
            calibrationStart = this.calibrationStart,
            calibrationEnd = this.calibrationEnd,
            calibrationPercentage = this.calibrationPercentage,
            activeModelId = this.activeModel?.id,
            savings = metrics[SAVINGS],
            country = this.location?.country?.name,
            latitude = this.location?.latitude,
            longitude = this.location?.longitude
```

```
        )
    }

    fun createHeater(newHeaterDTO: NewHeaterDTO) {
        val encodedPassword =
passwordEncoder.encode(newHeaterDTO.password)
        val user = userService.createUser(
            AdminUserDTO(
                login = newHeaterDTO.serial,
                authorities = mutableSetOf("ROLE_HEATER")
            )
        )

        user.password = encodedPassword
        heaterService.createHeater(
            Heater(
                name = newHeaterDTO.serial,
                serial = newHeaterDTO.serial,
                power = newHeaterDTO.power,
                heaterUser = user
            )
        )
    }

    fun addHeaterToCurrentUser(addHeaterDTO: AddHeaterDTO) {
        val user = userService.getCurrentUser()
        val heater = heaterService.findBySerial(addHeaterDTO.serial)

        if (!passwordEncoder.matches(addHeaterDTO.password,
heater.heaterUser.password)) {
            throw WrongHeaterPasswordException()
        }

        val country =
countryService.findById(addHeaterDTO.location.country_id)

        val location = locationService.createLocation(
            Location(
                latitude = addHeaterDTO.location.latitude,
                longitude = addHeaterDTO.location.longitude,
                country = country
            )
        )

        heater.name = addHeaterDTO.name
        heater.owner = user
        heater.location = location
    }

    fun getScheduleForCurrentHeater(date: LocalDate): Schedule? {
        val heater = heaterService.findCurrentHeater()
        return scheduleService.findByHeaterAndScheduleDate(heater,
date)
    }

    @TransactionalEventListener(phase = TransactionPhase.AFTER_COMMIT)
    @Transactional(propagation = Propagation.REQUIRES_NEW)
    fun publishHeaterUpdate(heaterUpdatedEvent: HeaterUpdatedEvent) {
```

```
        val heater =
heaterService.findBySerial(heaterUpdatedEvent.serial)

        websocketPublisher.convertAndSendToUser(
            heater.owner!!.login!!,
            "/queue/heater",
            "\"${heaterUpdatedEvent.serial}\""
        )
    }

    fun startCalibration(serial: String) {
        val heater = heaterService.findBySerial(serial)
        heater.calibrationStatus =
CalibrationStatus.CALIBRATION_IN_PROGRESS
        heater.calibrationStart =
metricsService.findLatestMetrics(heater)!!.timestamp
        heater.calibrationEnd = heater.calibrationStart!! +
Duration.ofHours(78)
        heater.calibrationPercentage = 0f
        heater.operationType = OperationType.CALIBRATING

        publishHeaterUpdate(HeaterUpdatedEvent(heater.serial))
    }

    @EventListener
    fun onMetricReceived(event: MetricReceivedEvent) {
        val heater = heaterService.findBySerial(event.serial)

        updateCalibrationStatus(heater, event.lastMetricTimestamp)
        updateSchedules(heater, event.lastMetricTimestamp)
    }

    fun updateCalibrationStatus(heater: Heater, lastMetricTimestamp:
Instant) {
        if (heater.calibrationStatus ==
CalibrationStatus.CALIBRATION_IN_PROGRESS) {
            if (lastMetricTimestamp > heater.calibrationEnd!!) {
                heater.calibrationStatus = CalibrationStatus.CALIBRATED
                heater.calibrationPercentage = 100f
                heater.operationType = OperationType.IDLE
                return
            }

            heater.calibrationPercentage =
                Duration.between(heater.calibrationStart!!,
lastMetricTimestamp).seconds.toFloat() /
                Duration.between(heater.calibrationStart!!,
heater.calibrationEnd!!).seconds
        }
    }

    fun updateSchedules(heater: Heater, lastMetricTimestamp: Instant) {
        if (heater.operationType != OperationType.MODEL) {
            return
        }

        val locationLastUpdated = heater.location?.lastUpdated
        val countryLastUpdated = heater.location?.country?.lastUpdated
```

```
        if (locationLastUpdated != null && countryLastUpdated != null)
{
            val dataAvailableUntil = maxOf(locationLastUpdated,
countryLastUpdated)
            val lastValidScheduleDate = heater.lastValidScheduleDate
            if (lastValidScheduleDate == null || lastValidScheduleDate
< dataAvailableUntil) {
                val currentDate =

lastMetricTimestamp.atZone(ZoneId.of(heater.location?.country?.timezone
)).toLocalDate()

                var date = if (lastValidScheduleDate == null) {
                    currentDate
                } else {
                    maxOf(currentDate,
lastValidScheduleDate.plusDays(1))
                }

                while (date <= dataAvailableUntil) {
                    calculateSchedule(heater.activeModel!!, date,
heater.location!!.country.timezone)
                    date = date.plusDays(1)
                }
            }
        }
    }

    fun calculateSchedule(model: Model, date: LocalDate, timezone:
String) {
        if (model.status != ModelStatus.Trained) {
            return
        }

        kafkaService.sendMLServiceCommand(
            CalculateScheduleCommand(
                modelId = model.id!!,
                date = date,
                timezone = timezone
            )
        )
    }

    fun getHeaterModels(serial: String): List<Model> =
        heaterService.findBySerial(serial).models

    fun createModel(serial: String, newModelDTO: NewModelDTO) {
        val heater = heaterService.findBySerial(serial)
        val latestMetrics = metricsService.findLatestMetrics(heater)
        val model = modelService.createNewModel(heater, newModelDTO,
latestMetrics?.timestamp ?: Instant.now())
        if (newModelDTO.activateImmediately) {
            heater.activeModel = model
        }

        kafkaService.sendMLServiceCommand(
            CreateModelCommand(
                modelId = model.id!!,
```

```
                serial = heater.serial,
                targetTemperature = newModelDTO.targetTemperature,
                minTemperature = newModelDTO.minTemperature,
                maxTemperature = newModelDTO.maxTemperature,
                calibrationDataStart = heater.calibrationStart!!,
                calibrationDataEnd = heater.calibrationEnd!!
            )
        )


applicationEventPublisher.publishEvent(HeaterUpdatedEvent(serial))
    }

    @EventListener
    fun modelTrainingFinished(event: ModelTrainingFinishedEvent) {
        val model = modelService.findById(event.modelId)

        if (model.heater.activeModel == model) {

applicationEventPublisher.publishEvent(HeaterUpdatedEvent(model.heater.
serial))
            model.heater.operationType = OperationType.MODEL
        }

        model.status = ModelStatus.Trained
    }

    @EventListener
    fun scheduleCalculated(event: ScheduleCalculatedEvent) {
        val model = modelService.findById(event.modelId)
        val heater = model.heater

        if (heater.activeModel != model) {
            return
        }

        val schedule =
scheduleService.findByHeaterAndScheduleDate(heater, event.date)
            ?: Schedule(heater = heater, scheduleDate = event.date,
data = event.schedule)
        schedule.data = event.schedule
        val lastValidScheduleDate = heater.lastValidScheduleDate
        heater.lastValidScheduleDate = if (lastValidScheduleDate ==
null) {
            schedule.scheduleDate
        } else {
            maxOf(lastValidScheduleDate, schedule.scheduleDate)
        }

        scheduleService.save(schedule)
    }
}

data class HeaterDTO(
    val id: Long,
    val name: String,
    val serial: String,
    val roomTemperature: Float?,
```

```
    val outsideTemperature: Float?,
    val heaterPower: Float,
    val weekConsumption: Float?,
    val operationType: OperationType,
    val calibrationStatus: CalibrationStatus,
    val calibrationStart: Instant?,
    val calibrationEnd: Instant?,
    val calibrationPercentage: Float?,
    val activeModelId: Long?,
    val savings: Float?,
    val country: String?,
    val latitude: Float?,
    val longitude: Float?
)

data class NewHeaterDTO(
    val serial: String,
    val password: String,
    val power: Float
)

data class AddHeaterDTO(
    val name: String,
    val serial: String,
    val password: String,
    val location: LocationDTO
)

data class LocationDTO(
    val latitude: Float,
    val longitude: Float,
    val country_id: Long
)

data class NewModelDTO(
    val name: String,
    val targetTemperature: Float,
    val minTemperature: Float,
    val maxTemperature: Float,
    val activateImmediately: Boolean
)

data class HeaterUpdatedEvent(val serial: String)

data class MetricReceivedEvent(
    val serial: String,
    val lastMetricTimestamp: Instant
)

data class ModelTrainingFinishedEvent(val modelId: Long)

data class ScheduleCalculatedEvent(
    val modelId: Long,
    val date: LocalDate,
    val schedule: String
)
```

```
class WrongHeaterPasswordException :
ResponseStatusException(HttpStatus.UNAUTHORIZED, "Heater password is
incorrect")
```

StatisticsFacade.kt

```
package org.kry.thesis.service.facade

import org.kry.thesis.service.HeaterService
import org.kry.thesis.service.StatisticsService
import org.kry.thesis.service.StatisticsWindowData
import org.kry.thesis.service.UserService
import org.kry.thesis.web.rest.StatisticsRequestDTO
import org.springframework.http.HttpStatus
import org.springframework.stereotype.Component
import org.springframework.transaction.annotation.Transactional
import org.springframework.web.server.ResponseStatusException
import java.time.ZoneId

@Component
@Transactional(readOnly = true)
class StatisticsFacade(
    private val heaterService: HeaterService,
    private val userService: UserService,
    private val statisticsService: StatisticsService
) {
    fun calculateStatistics(statisticsRequestDTO:
StatisticsRequestDTO): List<StatisticsWindowData> {
        val heater =
heaterService.findBySerial(statisticsRequestDTO.serial)
        val user = userService.getCurrentUser()
        if (heater.owner != user) {
            throw HeaterAccessForbiddenException()
        }

        val timeZone = try {
            ZoneId.of(statisticsRequestDTO.timeZone)
        } catch (e: Exception) {
            throw TimeZoneNotFoundException()
        }

        return statisticsService.calculateStatistics(
            target = heater,
            fields = statisticsRequestDTO.fields,
            startTime = statisticsRequestDTO.startTime,
            endTime = statisticsRequestDTO.endTime,
            aggregationPeriod = statisticsRequestDTO.aggregationPeriod,
            timeZone = timeZone
        )
    }
}

class HeaterAccessForbiddenException :
ResponseStatusException(HttpStatus.FORBIDDEN, "You don't have access to
this heater")

class TimeZoneNotFoundException :
ResponseStatusException(HttpStatus.NOT_FOUND, "Timezone not found")
```

StatisticsService.kt

```
package org.kry.thesis.service

import org.kry.thesis.domain.Heater
import org.kry.thesis.service.influxdb.InfluxDBService
import org.slf4j.LoggerFactory
import org.springframework.stereotype.Service
import java.time.*

@Service
class StatisticsService(
    private val influxDBService: InfluxDBService
) {
    private val log =
LoggerFactory.getLogger(StatisticsService::class.java)

    fun calculateLastWeekConsumption(target: Heater): Float? =
        influxDBService.calculateLastWeekConsumption(target)

    fun calculateStatistics(
        target: Heater,
        fields: Set<StatisticsField>,
        startTime: Instant,
        endTime: Instant,
        aggregationPeriod: StatisticsAggregationPeriod,
        timeZone: ZoneId
    ): List<StatisticsWindowData> {
        val statisticsData: Map<StatisticsField,
List<StatisticsDataPoint>> = fields.associateWith { field ->
            influxDBService.calculateStatistics(
                target,
                field,
                startTime,
                endTime,
                aggregationPeriod,
                timeZone
            )
        }
        return combineStatisticsData(
            data = statisticsData,
            startTime = startTime,
            endTime = endTime,
            aggregationPeriod = aggregationPeriod,
            timeZone = timeZone
        )
    }

    private fun combineStatisticsData(
        data: Map<StatisticsField, List<StatisticsDataPoint>>,
        startTime: Instant,
        endTime: Instant,
        aggregationPeriod: StatisticsAggregationPeriod,
        timeZone: ZoneId
    ): List<StatisticsWindowData> {
        val result = mutableListOf<StatisticsWindowData>()
        val fields: Set<StatisticsField> = data.keys
```

```
        val rangeStart: ZonedDateTime = startTime.atZone(timeZone)
        val rangeEnd: ZonedDateTime = endTime.atZone(timeZone)

        var currentWindowStart: ZonedDateTime =
startTime.atZone(timeZone)
            .roundDownToAggregationPeriodStart(aggregationPeriod)
        var currentWindowEnd: ZonedDateTime =
currentWindowStart.addAggregationPeriod(aggregationPeriod)

        while (
            rangesIntersect(
                currentWindowStart, currentWindowEnd,
                rangeStart, rangeEnd
            )
        ) {
            val currentWindowFieldData: Map<StatisticsField, Float?> =
fields.associateWith { field ->
                val fieldData = data[field]

                val dataPoint: Float? =
fieldData?.filterPointsIntersectingWithRange(
                    currentWindowStart.toInstant(),
                    currentWindowEnd.toInstant()
                )?.extractSinglePointOrLogError(
                    fieldData, timeZone, currentWindowStart,
currentWindowEnd
                )?.data

                dataPoint
            }

            result += StatisticsWindowData(
                periodStart =
currentWindowStart.coerceAtLeast(rangeStart).toInstant(),
                periodEnd =
currentWindowEnd.coerceAtMost(rangeEnd).toInstant(),
                data = currentWindowFieldData
            )

            val newWindowEnd =
currentWindowEnd.addAggregationPeriod(aggregationPeriod)
            currentWindowStart = currentWindowEnd
            currentWindowEnd = newWindowEnd
        }

        return result
    }

    private fun
List<StatisticsDataPoint>.filterPointsIntersectingWithRange(
        start: Instant,
        end: Instant
    ): List<StatisticsDataPoint> = filter {
        rangesIntersect(
            start, end,
            it.periodStart, it.periodEnd
        )
    }
```

```
    private fun <T> List<T>.extractSinglePointOrLogError(
        statisticsData: List<StatisticsDataPoint>,
        timeZone: ZoneId,
        currentWindowStart: ZonedDateTime,
        currentWindowEnd: ZonedDateTime
    ): T? = firstOrNull().also {
        if (size > 1) {
            log.error(
                MisalignedIntervalsException(
                    statisticsData = statisticsData,
                    timeZone = timeZone,
                    currentWindowStart = currentWindowStart,
                    currentWindowEnd = currentWindowEnd
                ).toString()
            )
        }
    }

    private fun ZonedDateTime.roundDownToAggregationPeriodStart(period:
StatisticsAggregationPeriod): ZonedDateTime =
        when (period) {
            StatisticsAggregationPeriod.HOUR -> ZonedDateTime.of(
                this.year,
                this.monthValue,
                this.dayOfMonth,
                this.hour,
                0,
                0,
                0,
                this.zone
            )

            StatisticsAggregationPeriod.DAY ->
this.toLocalDate().atStartOfDay(this.zone)
            StatisticsAggregationPeriod.WEEK -> {
                val date = this.toLocalDate()
                date.minusDays(date.dayOfWeek.value.toLong() -
1).atStartOfDay(this.zone)
            }

            StatisticsAggregationPeriod.MONTH ->
LocalDate.of(this.year, this.month, 1).atStartOfDay(this.zone)
        }

    private fun ZonedDateTime.addAggregationPeriod(period:
StatisticsAggregationPeriod): ZonedDateTime =
        when (period) {
            StatisticsAggregationPeriod.HOUR -> this.plusHours(1)
            StatisticsAggregationPeriod.DAY -> this.plusDays(1)
            StatisticsAggregationPeriod.WEEK -> this.plusWeeks(1)
            StatisticsAggregationPeriod.MONTH -> this.plusMonths(1)
        }

    companion object {
        const val KWH_TO_J = 3_600_000
    }
}
```

```
enum class StatisticsField {
    ELECTRIC_CONSUMPTION,
    ROOM_TEMPERATURE,
    OUTSIDE_TEMPERATURE,
    ELECTRICITY_PRICE
}

enum class StatisticsAggregationPeriod {
    HOUR,
    DAY,
    WEEK,
    MONTH
}

data class StatisticsDataPoint(
    val periodStart: Instant,
    val periodEnd: Instant,
    val data: Float
)

data class StatisticsWindowData(
    val periodStart: Instant,
    val periodEnd: Instant,
    val data: Map<StatisticsField, Float?>
)

fun <T : Comparable<T>> rangesIntersect(
    firstStart: T,
    firstEndExclusive: T,
    secondStart: T,
    secondEndExclusive: T
): Boolean =
// no intersection:       <---second--->   <---first--->     or      <-
--first--->   <---second-->
// intersection:    not ( <---second--->   <---first---> ) and not ( <-
--first--->   <---second--> )
    !(firstStart >= secondEndExclusive) && !(secondStart >=
firstEndExclusive)

class MisalignedIntervalsException(
    statisticsData: List<StatisticsDataPoint>,
    timeZone: ZoneId,
    currentWindowStart: ZonedDateTime,
    currentWindowEnd: ZonedDateTime
) : RuntimeException(
    """
    Misaligned statistics intervals: two InfluxDB intervals intersect
one server interval.
    Time zone: $timeZone
    Statistics data: $statisticsData
    Current interval: $currentWindowStart - $currentWindowEnd
    """.trimIndent()
)
```

InfluxDBService.kt

```
package org.kry.thesis.service.influxdb
```

```
import com.influxdb.client.InfluxDBClient
import com.influxdb.client.domain.WritePrecision
import com.influxdb.query.FluxRecord
import org.kry.thesis.domain.Country
import org.kry.thesis.domain.Heater
import org.kry.thesis.domain.Location
import org.kry.thesis.service.StatisticsAggregationPeriod
import org.kry.thesis.service.StatisticsDataPoint
import org.kry.thesis.service.StatisticsField
import org.kry.thesis.service.StatisticsField.*
import org.slf4j.LoggerFactory
import org.springframework.stereotype.Service
import java.time.Instant
import java.time.ZoneId

@Service
class InfluxDBService(
    influxDBClient: InfluxDBClient
) {
    private val writeApi = influxDBClient.writeApiBlocking
    private val queryApi = influxDBClient.queryApi

    private val log =
LoggerFactory.getLogger(InfluxDBService::class.java)

    fun saveMetrics(serial: String, metricsWithTimestamp: String) {
        writeApi.writeRecord(
            WritePrecision.MS,
            assembleHeaterSensorsMeasurement(serial,
metricsWithTimestamp)
        )
    }

    fun saveOutsideTemperature(location: Location, temperatures:
List<Float>, timestamps: List<Instant>) {
        assert(temperatures.size == timestamps.size)
        temperatures.zip(timestamps).map { (temperature, timestamp) ->
            writeApi.writeRecord(
                WritePrecision.MS,
                assembleOutsideTemperatureMeasurement(location.id!!,
temperature, timestamp)
            )
        }
    }

    fun savePrices(country: Country, prices: List<Float>, timestamps:
List<Instant>) {
        assert(prices.size == timestamps.size)
        prices.zip(timestamps).map { (price, timestamp) ->
            writeApi.writeRecord(
                WritePrecision.MS,
                assemblePriceMeasurement(country.id!!, price,
timestamp)
            )
        }
    }

    private fun assembleHeaterSensorsMeasurement(serial: String,
```

```
metricsWithTimestamp: String): String =
        "$HEATER_SENSORS_MEASUREMENT,serial=$serial
$metricsWithTimestamp"

    private fun assembleOutsideTemperatureMeasurement(locationId: Long,
temperature: Float, timestamp: Instant): String =
        "$OUTSIDE_TEMPERATURE_MEASUREMENT,loc_id=$locationId
temp=$temperature ${timestamp.toEpochMilli()}"

    private fun assemblePriceMeasurement(countryId: Long, price: Float,
timestamp: Instant): String =
        "$PRICE_MEASUREMENT,country_id=$countryId price=$price
${timestamp.toEpochMilli()}"

    fun calculateLastWeekConsumption(target: Heater): Float? =
        safeInfluxDBQuery(
            fieldAggregationQuery(
                interval = "1w",
                serial = target.serial,
                measurement = HEATER_SENSORS_MEASUREMENT,
                field = ELECTRIC_CONSUMPTION,
                aggregationFunction =
InfluxDBAggregationFunction.SPREAD
            )
        ).firstOrNull()?.let { it.tryExtractSerialAndValue()?.second }

    fun calculateStatistics(
        target: Heater,
        field: StatisticsField,
        startTime: Instant,
        endTime: Instant,
        aggregationPeriod: StatisticsAggregationPeriod,
        timeZone: ZoneId?
    ): List<StatisticsDataPoint> {
        val query = windowAggregationQuery(
            startTime = startTime,
            endTime = endTime,
            idField = field.idField(),
            idValue = field.idValue(target),
            measurement = field.toMeasurement(),
            field = field.toInfluxFieldName(),
            windowPeriod = aggregationPeriod.toInfluxPeriod(),
            windowOffset = aggregationPeriod.influxWindowOffset(),
            aggregationFunction = field.aggregationFunction(),
            endOperation = if (field == ELECTRIC_CONSUMPTION) {
                "difference"
            } else {
                null
            },
            timeZone = timeZone
        )

        return safeInfluxDBQuery(query)
            .mapNotNull { it.tryExtractStatisticsDataPoint() }
            .sortedBy { it.periodStart }
    }

    private fun safeInfluxDBQuery(query: String): List<FluxRecord> =
```

```
try {
        queryApi.query(query)
            .flatMap { it.records }
    } catch (e: Exception) {
        log.error("Error executing InfluxDB query:\n{}", query)
        log.error(e.stackTraceToString())
        emptyList()
    }
}

private fun fieldAggregationQuery(
    interval: String,
    serial: String,
    measurement: String,
    field: StatisticsField,
    aggregationFunction: InfluxDBAggregationFunction
): String = """
        from(bucket: "$BUCKET")
            |> range(start: -$interval)
            |> filter(fn: (r) => r._measurement == "$measurement" and
r._field == "${field.toInfluxFieldName()}")
            |> filter(fn: (r) => r.serial == "$serial")
            |> group(columns: ["serial"])
            |> ${aggregationFunction.value}()
            |> group()
""".trimIndent()

private fun windowAggregationQuery(
    startTime: Instant,
    endTime: Instant,
    idField: String,
    idValue: String,
    measurement: String,
    field: String,
    windowPeriod: String,
    windowOffset: String?,
    aggregationFunction: InfluxDBAggregationFunction,
    endOperation: String?,
    timeZone: ZoneId?
): String {
    val windowOffsetParameter = when {
        windowOffset != null -> ", offset: $windowOffset"
        else -> ""
    }
    val query = """
        from(bucket: "$BUCKET")
            |> range(start: $startTime, stop: $endTime)
            |> filter(fn: (r) => r._measurement == "$measurement" and
r._field == "$field")
            |> filter(fn: (r) => r.$idField == "$idValue")
            |> window(every: $windowPeriod$windowOffsetParameter)
            |> ${aggregationFunction.value}()
            |> group()
           ${if (endOperation != null) "|> $endOperation()" else ""}
    """.trimIndent()

    return if (timeZone != null) {
        """
```

```
            import "timezone"
            option location = timezone.location(name: "${timeZone.id}")
        """.trimIndent() + "\n" + query
    } else {
        query
    }
}

private fun FluxRecord.tryExtractStatisticsDataPoint():
StatisticsDataPoint? {
    val startTime = values["_start"] as? Instant ?: return null
    val endTime = values["_stop"] as? Instant ?: return null
    val value = (this.value as? Double)?.toFloat() ?: return null
    return StatisticsDataPoint(
        periodStart = startTime,
        periodEnd = endTime,
        data = value
    )
}

private fun FluxRecord.tryExtractSerialAndValue(): Pair<String, Float>?
{
    val serial = values["serial"] as? String ?: return null
    val value = (this.value as? Double)?.toFloat() ?: return null
    return serial to value
}

private enum class InfluxDBAggregationFunction(val value: String) {
    MIN("min"),
    MEAN("mean"),
    SPREAD("spread")
}

private fun StatisticsField.toMeasurement(): String =
    when (this) {
        ELECTRIC_CONSUMPTION, ROOM_TEMPERATURE ->
HEATER_SENSORS_MEASUREMENT
        OUTSIDE_TEMPERATURE -> OUTSIDE_TEMPERATURE_MEASUREMENT
        ELECTRICITY_PRICE -> PRICE_MEASUREMENT
    }

private fun StatisticsField.idField(): String =
    when (this) {
        ELECTRIC_CONSUMPTION, ROOM_TEMPERATURE -> "serial"
        OUTSIDE_TEMPERATURE -> "loc_id"
        ELECTRICITY_PRICE -> "country_id"
    }

private fun StatisticsField.idValue(heater: Heater): String =
    when (this) {
        ELECTRIC_CONSUMPTION, ROOM_TEMPERATURE -> heater.serial
        OUTSIDE_TEMPERATURE -> heater.location?.id?.toString() ?:
"unknown"
        ELECTRICITY_PRICE ->
heater.location?.country?.id?.toString() ?: "unknown"
    }

private fun StatisticsField.toInfluxFieldName(): String =
```

```
    when (this) {
        ELECTRIC_CONSUMPTION -> ELECTRIC_CONSUMPTION_FIELD
        ROOM_TEMPERATURE -> ROOM_TEMPERATURE_FIELD
        OUTSIDE_TEMPERATURE -> OUTSIDE_TEMPERATURE_FIELD
        ELECTRICITY_PRICE -> PRICE_FIELD
    }

private fun StatisticsField.aggregationFunction():
InfluxDBAggregationFunction =
    when (this) {
        ELECTRIC_CONSUMPTION -> InfluxDBAggregationFunction.MIN
        ROOM_TEMPERATURE, OUTSIDE_TEMPERATURE, ELECTRICITY_PRICE ->
InfluxDBAggregationFunction.MEAN
    }

private fun StatisticsAggregationPeriod.toInfluxPeriod(): String =
    when (this) {
        StatisticsAggregationPeriod.HOUR -> "1h"
        StatisticsAggregationPeriod.DAY -> "1d"
        StatisticsAggregationPeriod.WEEK -> "1w"
        StatisticsAggregationPeriod.MONTH -> "1mo"
    }

private fun StatisticsAggregationPeriod.influxWindowOffset(): String? =
    when (this) {
        StatisticsAggregationPeriod.WEEK -> "4d" // take into account
that weeks in InfluxDB start from Thursday (Unix epoch)
        else -> null
    }

private const val BUCKET = "thesis"
private const val HEATER_SENSORS_MEASUREMENT = "heater-sensors"
private const val OUTSIDE_TEMPERATURE_MEASUREMENT = "out-temp"
private const val PRICE_MEASUREMENT = "price"

private const val ELECTRIC_CONSUMPTION_FIELD = "el_co"
private const val ROOM_TEMPERATURE_FIELD = "room_t"
private const val OUTSIDE_TEMPERATURE_FIELD = "temp"
private const val PRICE_FIELD = "price"
```

MetricsService.kt

```
package org.kry.thesis.service.metrics

import org.kry.thesis.domain.Heater
import org.kry.thesis.domain.LatestMetrics
import org.kry.thesis.repository.LatestMetricsRepository
import org.kry.thesis.service.HeaterService
import org.kry.thesis.service.facade.HeaterUpdatedEvent
import org.kry.thesis.service.facade.MetricReceivedEvent
import org.kry.thesis.service.influxdb.InfluxDBService
import org.kry.thesis.service.mqtt.MQTTMessage
import org.kry.thesis.service.mqtt.MQTTQueueService
import org.springframework.context.ApplicationEventPublisher
import org.springframework.context.event.EventListener
import org.springframework.stereotype.Service
import org.springframework.transaction.annotation.Transactional
import java.time.Instant
```

```
@Service
@Transactional
class MetricsService(
    mqttQueueService: MQTTQueueService,
    private val heaterService: HeaterService,
    private val latestMetricsRepository: LatestMetricsRepository,
    private val influxDBService: InfluxDBService,
    private val applicationEventPublisher: ApplicationEventPublisher
) {

    init {
        mqttQueueService.subscribe("/panel/metrics/#") {
            applicationEventPublisher.publishEvent(it)
        }
    }

    @EventListener
    fun processMetrics(metricsMessage: MQTTMessage) {
        val serial = parseSerialFromTopic(metricsMessage.topic)
        val metricsWithTimestamp = metricsMessage.message
        val metrics = metricsWithTimestamp.split(' ').first()
        val timestamp = parseMetricTimestamp(metricsWithTimestamp)!!
        println("Received metrics $metrics for serial $serial")

        val heater = heaterService.findBySerial(serial)
        val latestMetrics =
latestMetricsRepository.findLatestMetricsByHeater(heater)
            ?: LatestMetrics(heater = heater, metrics = metrics,
timestamp = timestamp)

        latestMetrics.metrics = metrics
        latestMetrics.timestamp = timestamp
        latestMetricsRepository.save(latestMetrics)

        influxDBService.saveMetrics(serial, metricsWithTimestamp)


applicationEventPublisher.publishEvent(MetricReceivedEvent(serial =
heater.serial, lastMetricTimestamp = timestamp))

applicationEventPublisher.publishEvent(HeaterUpdatedEvent(heater.serial
))
    }

    fun findLatestMetrics(heater: Heater): LatestMetrics? =
        latestMetricsRepository.findLatestMetricsByHeater(heater)

    private fun parseMetricTimestamp(metrics: String): Instant? =
        metrics.split(' ').lastOrNull()
            ?.toLongOrNull()
            ?.let { Instant.ofEpochMilli(it) }

    private fun parseSerialFromTopic(topic: String): String =
topic.split("/").last()
}

fun parseMetrics(metrics: String): Map<HeaterMetric, Float> =
    metrics.split(' ').firstOrNull()
```

```
        ?.split(',')?.mapNotNull { metric ->
            val code = metric.substringBefore('=', "").let
{ HeaterMetric.fromCode(it) } ?: return@mapNotNull null
            val value = metric.substringAfter('=',
"").toFloatOrNull() ?: return@mapNotNull null
            code to value
        }?.toMap() ?: emptyMap()

enum class HeaterMetric(val code: String) {
    ROOM_TEMPERATURE("room_t"),
    OUTSIDE_TEMPERATURE("out_t"),
    SAVINGS("sav");

    companion object {
        fun fromCode(code: String): HeaterMetric? =
            values().find { it.code == code }
    }
}
```

KafkaService.kt

```
package org.kry.thesis.service

import com.fasterxml.jackson.annotation.*
import com.fasterxml.jackson.databind.ObjectMapper
import org.kry.thesis.config.*
import org.kry.thesis.service.facade.*
import org.springframework.beans.factory.annotation.Qualifier
import org.springframework.cloud.stream.annotation.StreamListener
import org.springframework.context.ApplicationEventPublisher
import org.springframework.messaging.*
import org.springframework.messaging.support.GenericMessage
import org.springframework.stereotype.Service
import org.springframework.util.MimeTypeUtils
import java.time.*

@Service
class KafkaService(
    @Qualifier(KafkaSseProducer.ML_SERVICE_COMMAND_CHANNEL) private val
mlServiceCommandChannel: MessageChannel,
    private val objectMapper: ObjectMapper,
    private val applicationEventPublisher: ApplicationEventPublisher
) {
    fun sendMLServiceCommand(command: MLServiceCommand) {
        val headers = MessageHeaders(
            mapOf(MessageHeaders.CONTENT_TYPE to
MimeTypeUtils.TEXT_PLAIN_VALUE)
        )
        mlServiceCommandChannel.send(
            GenericMessage(
                objectMapper.writeValueAsString(command),
                headers
            )
        )
    }

    @StreamListener(value =
KafkaSseConsumer.ML_SERVICE_RESPONSE_CHANNEL, copyHeaders = "false")
    fun consume(message: Message<String>) {
```

```
        val response = objectMapper.readValue(message.payload,
MLServiceResponse::class.java)
        when (response) {
            is ModelTrainingFinishedResponse -> {

applicationEventPublisher.publishEvent(ModelTrainingFinishedEvent(model
Id = response.modelId))
            }

            is ScheduleCalculatedResponse -> {
                applicationEventPublisher.publishEvent(
                    ScheduleCalculatedEvent(
                        modelId = response.modelId,
                        date = response.date,
                        schedule =
objectMapper.writeValueAsString(response.schedule)
                    )
                )
            }
        }
    }
}

@JsonTypeInfo(use = JsonTypeInfo.Id.NAME, include =
JsonTypeInfo.As.PROPERTY, property = "type")
@JsonSubTypes(
    value = [
        JsonSubTypes.Type(value = CreateModelCommand::class, name =
"CREATE_MODEL"),
        JsonSubTypes.Type(value = CalculateScheduleCommand::class, name
= "CALCULATE_SCHEDULE")
    ]
)
sealed class MLServiceCommand

data class CreateModelCommand(
    val modelId: Long,
    val serial: String,
    val targetTemperature: Float,
    val minTemperature: Float,
    val maxTemperature: Float,
    val calibrationDataStart: Instant,
    val calibrationDataEnd: Instant
) : MLServiceCommand()

data class CalculateScheduleCommand(
    val modelId: Long,
    val date: LocalDate,
    val timezone: String
) : MLServiceCommand()

@JsonTypeInfo(use = JsonTypeInfo.Id.NAME, include =
JsonTypeInfo.As.PROPERTY, property = "type")
@JsonSubTypes(
    value = [
        JsonSubTypes.Type(value = ModelTrainingFinishedResponse::class,
name = "MODEL_TRAINING_FINISHED"),
```

```
        JsonSubTypes.Type(value = ScheduleCalculatedResponse::class,
name = "SCHEDULE_CALCULATED")
    ]
)
sealed class MLServiceResponse

data class ModelTrainingFinishedResponse(
    val modelId: Long
) : MLServiceResponse()

data class ScheduleCalculatedResponse(
    val modelId: Long,
    val date: LocalDate,
    val schedule: List<Float>
) : MLServiceResponse()
```

WeatherService.kt

```
package org.kry.thesis.service

import org.kry.thesis.domain.Location
import org.springframework.http.*
import org.springframework.stereotype.Service
import org.springframework.web.client.RestTemplate
import org.springframework.web.util.UriComponentsBuilder
import java.time.*
import java.time.format.DateTimeFormatter
import java.time.temporal.ChronoUnit

@Service
class WeatherService(
    private val restTemplate: RestTemplate
) {
    fun getWeather(location: Location, startDate: LocalDate,
endDateInclusive: LocalDate): List<Float> {
        val headers = HttpHeaders().apply {
            set(HttpHeaders.ACCEPT, MediaType.APPLICATION_JSON_VALUE)
        }
        val entity = HttpEntity<Any>(headers)
        val url = chooseApiUrl(endDateInclusive,
location.country.timezone)

        val urlTemplate = UriComponentsBuilder.fromHttpUrl(url)
            .queryParam("latitude", "{latitude}")
            .queryParam("longitude", "{longitude}")
            .queryParam("start_date", "{start_date}")
            .queryParam("end_date", "{end_date}")
            .queryParam("hourly", "{hourly}")
            .queryParam("timezone", "{timezone}")
            .encode()
            .toUriString()
        val params = mapOf(
            "latitude" to location.latitude.toString(),
            "longitude" to location.longitude.toString(),
            "start_date" to
startDate.format(DateTimeFormatter.ISO_LOCAL_DATE),
            "end_date" to
endDateInclusive.format(DateTimeFormatter.ISO_LOCAL_DATE),
            "hourly" to "temperature_2m",
```

```
            "timezone" to location.country.timezone
        )

        return restTemplate.exchange(
            urlTemplate,
            HttpMethod.GET,
            entity,
            WeatherResponse::class.java,
            params
        ).body!!.hourly.temperature_2m
    }

    private fun chooseApiUrl(endDateInclusive: LocalDate, timezone:
String): String {
        val currentDate = LocalDateTime.now(ZoneId.of(timezone))
        return if (ChronoUnit.DAYS.between(endDateInclusive,
currentDate) >= HISTORICAL_DATA_LAG_DAYS) {
            "https://archive-api.open-meteo.com/v1/archive"
        } else {
            "https://api.open-meteo.com/v1/forecast"
        }
    }
}

private const val HISTORICAL_DATA_LAG_DAYS: Int = 7

private data class WeatherResponse(
    val hourly: HourlyWeatherData
)

private data class HourlyWeatherData(
    val temperature_2m: List<Float>
)
```

PriceService.kt

```
package org.kry.thesis.service

import org.kry.thesis.domain.Country
import org.springframework.http.*
import org.springframework.stereotype.Service
import org.springframework.web.client.RestTemplate
import org.springframework.web.util.UriComponentsBuilder
import java.time.LocalDate
import java.time.format.DateTimeFormatter

@Service
class PriceService(
    private val restTemplate: RestTemplate
) {
    fun getPrices(country: Country, startDate: LocalDate,
endDateInclusive: LocalDate): List<Float> {
        val result = mutableListOf<Float>()
        var curDate = startDate
        while (curDate.isBefore(endDateInclusive.plusDays(1))) {
            getPrices(country, curDate)?.let { prices ->
                result += prices
            }
            curDate = curDate.plusDays(1)
```

```
        }
        return result
    }

    fun getPrices(country: Country, date: LocalDate): List<Float>? {
        val headers = HttpHeaders().apply {
            set(HttpHeaders.ACCEPT, MediaType.APPLICATION_JSON_VALUE)
        }
        val entity = HttpEntity<Any>(headers)
        val areaCode = country.code

        val urlTemplate =
UriComponentsBuilder.fromHttpUrl("https://dataportal-
api.nordpoolgroup.com/api/DayAheadPrices")
            .queryParam("date", "{date}")
            .queryParam("market", "{market}")
            .queryParam("deliveryArea", "{deliveryArea}")
            .queryParam("currency", "{currency}")
            .encode()
            .toUriString()
        val params = mapOf(
            "date" to date.format(DateTimeFormatter.ISO_LOCAL_DATE),
            "market" to "DayAhead",
            "deliveryArea" to areaCode,
            "currency" to "EUR"
        )

        return restTemplate.exchange(
            urlTemplate,
            HttpMethod.GET,
            entity,
            PriceResponse::class.java,
            params
        ).body?.multiAreaEntries?.map {
            it.entryPerArea[areaCode]!!
        }
    }
}

private data class PriceResponse(
    val multiAreaEntries: List<MultiAreaEntry>
)

private data class MultiAreaEntry(
    val entryPerArea: Map<String, Float>
)
```

UpdateSchedulerService.kt

```
package org.kry.thesis.service

import org.kry.thesis.domain.Country
import org.kry.thesis.domain.Location
import org.kry.thesis.service.influxdb.InfluxDBService
import org.springframework.boot.context.event.ApplicationReadyEvent
import org.springframework.context.event.EventListener
import org.springframework.http.HttpStatus
import org.springframework.scheduling.annotation.Scheduled
import org.springframework.stereotype.Component
```

```
import org.springframework.transaction.annotation.Transactional
import org.springframework.web.server.ResponseStatusException
import java.time.Instant
import java.time.LocalDate
import java.time.LocalDateTime
import java.time.ZoneId
import java.util.concurrent.TimeUnit

@Component
@Transactional
class UpdateSchedulerService(
    private val weatherService: WeatherService,
    private val locationService: LocationService,
    private val countryService: CountryService,
    private val influxDBService: InfluxDBService,
    private val priceService: PriceService,
) {

    @EventListener(ApplicationReadyEvent::class)
    @Scheduled(fixedDelay = 30, timeUnit = TimeUnit.MINUTES)
    fun updateLocationsWeather() {
        locationService.findAll()
            .forEach { updateLocationWeather(it) }
    }

    @EventListener(ApplicationReadyEvent::class)
    @Scheduled(fixedDelay = 30, timeUnit = TimeUnit.MINUTES)
    fun updateCountriesPrices() {
        countryService.findAll()
            .forEach { updateCountryPrices(it) }
    }

    fun updateLocationWeather(location: Location) {
        val timezone = ZoneId.of(location.country.timezone)
        val currentDate = LocalDate.now(timezone)
        val endDateInclusive = currentDate.plusDays(1)

        if (location.lastUpdated == endDateInclusive) {
            return
        }

        val startDate = location.lastUpdated ?:
currentDate.minusDays(5)
        val temperatures: List<Float> =
weatherService.getWeather(location, startDate, endDateInclusive)
        val timestamps = generateTimestamps(startDate,
endDateInclusive, timezone)
        influxDBService.saveOutsideTemperature(location, temperatures,
timestamps)

        location.lastUpdated = endDateInclusive
    }

    fun updateCountryPrices(country: Country) {
        val timezone = ZoneId.of(country.timezone)
        val currentDate = LocalDate.now(timezone)
        val endDateInclusive = currentDate.plusDays(1)
```

```
        if (country.lastUpdated == endDateInclusive) {
            return
        }

        val startDate = country.lastUpdated ?: currentDate.minusDays(5)
        val prices: List<Float> = priceService.getPrices(country,
startDate, endDateInclusive)
        val timestamps = generateTimestamps(startDate,
endDateInclusive, timezone)
        influxDBService.savePrices(country, prices, timestamps)

        country.lastUpdated = endDateInclusive
    }
}

private fun generateTimestamps(startDate: LocalDate, endDateInclusive:
LocalDate, timezone: ZoneId): List<Instant> =
    generateLocalTimestamps(startDate, endDateInclusive).map {
        it.atZone(timezone).toInstant()
    }

private fun generateLocalTimestamps(startDate: LocalDate,
endDateInclusive: LocalDate): List<LocalDateTime> {
    val result = mutableListOf<LocalDateTime>()
    var cur = startDate.atStartOfDay()
    while (cur.toLocalDate() <= endDateInclusive) {
        result += cur
        cur = cur.plusHours(1)
    }
    return result
}

class HeaterLocationNotFoundException :
ResponseStatusException(HttpStatus.NOT_FOUND, "Heater has no location")
```

ModelService.kt

```
package org.kry.thesis.service

import org.kry.thesis.domain.Heater
import org.kry.thesis.domain.Model
import org.kry.thesis.domain.ModelStatus
import org.kry.thesis.repository.ModelRepository
import org.kry.thesis.service.facade.NewModelDTO
import org.springframework.data.repository.findByIdOrNull
import org.springframework.http.HttpStatus
import org.springframework.stereotype.Service
import org.springframework.transaction.annotation.Transactional
import org.springframework.web.server.ResponseStatusException
import java.time.Instant

@Service
@Transactional
class ModelService(
    private val modelRepository: ModelRepository
) {
    fun findById(id: Long): Model =
        modelRepository.findByIdOrNull(id) ?: throw
ModelNotFoundException()
```

```
    fun createNewModel(heater: Heater, newModelDTO: NewModelDTO,
createdOn: Instant): Model =
        modelRepository.saveAndFlush(
            Model(
                heater = heater,
                name = newModelDTO.name,
                targetTemperature = newModelDTO.targetTemperature,
                minTemperature = newModelDTO.minTemperature,
                maxTemperature = newModelDTO.maxTemperature,
                status = ModelStatus.Training,
                createdOn = createdOn
            )
        )
}

class ModelNotFoundException :
ResponseStatusException(HttpStatus.NOT_FOUND, "Model not found")
```

ScheduleService.kt

```
package org.kry.thesis.service

import org.kry.thesis.domain.Heater
import org.kry.thesis.domain.Schedule
import org.kry.thesis.repository.ScheduleRepository
import org.springframework.stereotype.Service
import org.springframework.transaction.annotation.Transactional
import java.time.LocalDate

@Service
@Transactional
class ScheduleService(
    private val scheduleRepository: ScheduleRepository,
) {
    fun findByHeaterAndScheduleDate(heater: Heater, scheduleDate:
LocalDate): Schedule? =
        scheduleRepository.findByHeaterAndScheduleDate(heater,
scheduleDate)

    fun save(schedule: Schedule): Schedule =
        scheduleRepository.save(schedule)
}
```

LocationService.kt

```
package org.kry.thesis.service

import org.kry.thesis.domain.Location
import org.kry.thesis.repository.LocationRepository
import org.springframework.stereotype.Service

@Service
class LocationService(
    private val locationRepository: LocationRepository,
) {
    fun findAll(): List<Location> =
        locationRepository.findAll()
```

```
    fun createLocation(location: Location): Location =
        locationRepository.save(location)
}
```

CountryService.kt

```
package org.kry.thesis.service

import org.kry.thesis.domain.Country
import org.kry.thesis.repository.CountryRepository
import org.springframework.data.repository.findByIdOrNull
import org.springframework.http.HttpStatus
import org.springframework.stereotype.Service
import org.springframework.web.server.ResponseStatusException

@Service
class CountryService(
    private val countryRepository: CountryRepository,
) {
    fun findById(id: Long): Country =
        countryRepository.findByIdOrNull(id) ?: throw
CountryNotFoundException()

    fun findAll(): List<Country> =
        countryRepository.findAll()
}

class CountryNotFoundException :
ResponseStatusException(HttpStatus.NOT_FOUND, "Country not found")
```

HeaterService.kt

```
package org.kry.thesis.service

import org.kry.thesis.domain.Heater
import org.kry.thesis.repository.HeaterRepository
import org.springframework.http.HttpStatus
import org.springframework.stereotype.Service
import org.springframework.transaction.annotation.Transactional
import org.springframework.web.server.ResponseStatusException

@Service
@Transactional
class HeaterService(
    private val heaterRepository: HeaterRepository,
) {
    fun findCurrentHeater(): Heater =
        heaterRepository.findCurrentHeater() ?: throw
HeaterNotFoundException()

    fun findByOwnerId(ownerId: Long): List<Heater> =
        heaterRepository.findByOwnerId(ownerId)

    fun findBySerial(serial: String): Heater =
        heaterRepository.findBySerial(serial) ?: throw
HeaterNotFoundException()

    fun createHeater(heater: Heater): Heater =
        heaterRepository.save(heater)
```

```
}

class HeaterNotFoundException :
ResponseStatusException(HttpStatus.NOT_FOUND, "Heater not found")
```

Heater.kt

```
package org.kry.thesis.domain

import com.fasterxml.jackson.annotation.*
import org.hibernate.annotations.*
import org.hibernate.annotations.Cache
import java.time.*
import javax.persistence.*
import javax.persistence.Entity
import javax.persistence.Table

@Entity
@Table(name = "heater")
@Cache(usage = CacheConcurrencyStrategy.NONSTRICT_READ_WRITE)
class Heater(

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
"sequenceGenerator")
    @SequenceGenerator(name = "sequenceGenerator")
    val id: Long? = null,

    @Column(name = "name", nullable = false)
    var name: String,

    @Column(name = "serial", nullable = false, unique = true)
    var serial: String,

    @JsonIgnore
    @OneToOne
    @JoinColumn(nullable = false, unique = true)
    val heaterUser: User,

    @Column(name = "power", nullable = false)
    val power: Float,

    @JsonIgnore
    @ManyToOne
    var owner: User? = null,

    @Enumerated(EnumType.STRING)
    @Column(name = "operation_type", nullable = false)
    var operationType: OperationType = OperationType.IDLE,

    @Enumerated(EnumType.STRING)
    @Column(name = "calibration_status", nullable = false)
    var calibrationStatus: CalibrationStatus =
CalibrationStatus.NOT_CALIBRATED,

    @Column(name = "calibration_start")
    var calibrationStart: Instant? = null,

    @Column(name = "calibration_end")
```

```
    var calibrationEnd: Instant? = null,

    @Column(name = "calibration_percentage")
    var calibrationPercentage: Float? = null,

    @OneToOne
    var activeModel: Model? = null,

    @OneToMany(mappedBy = "heater", fetch = FetchType.EAGER)
    var models: MutableList<Model> = mutableListOf(),

    @OneToOne
    var location: Location? = null,

    @OneToMany(mappedBy = "heater")
    var schedules: MutableList<Schedule> = mutableListOf(),

    @Column
    var lastValidScheduleDate: LocalDate? = null
)

enum class OperationType {
    IDLE,
    CALIBRATING,
    MODEL
}

enum class CalibrationStatus {
    NOT_CALIBRATED,
    CALIBRATION_IN_PROGRESS,
    CALIBRATED
}
```

LatestMetrics.kt

```
package org.kry.thesis.domain

import org.hibernate.annotations.*
import org.hibernate.annotations.Cache
import java.time.*
import javax.persistence.*
import javax.persistence.Entity
import javax.persistence.Table

@Entity
@Table(name = "latest_metrics")
@Cache(usage = CacheConcurrencyStrategy.NONSTRICT_READ_WRITE)
class LatestMetrics(

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
"sequenceGenerator")
    @SequenceGenerator(name = "sequenceGenerator")
    val id: Long? = null,

    @OneToOne
    val heater: Heater? = null,

    @Column(nullable = false)
```

```
    var metrics: String,

    @Column(nullable = false)
    var timestamp: Instant
)
```

Location.kt

```
package org.kry.thesis.domain

import org.hibernate.annotations.*
import org.hibernate.annotations.Cache
import java.time.*
import javax.persistence.*
import javax.persistence.Entity
import javax.persistence.Table

@Entity
@Table(name = "location")
@Cache(usage = CacheConcurrencyStrategy.NONSTRICT_READ_WRITE)
class Location(

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
"sequenceGenerator")
    @SequenceGenerator(name = "sequenceGenerator")
    val id: Long? = null,

    @Column(nullable = false)
    val latitude: Float,

    @Column(nullable = false)
    val longitude: Float,

    @ManyToOne
    @JoinColumn(nullable = false)
    val country: Country,

    @Column
    var lastUpdated: LocalDate? = null
)
```

Country.kt

```
package org.kry.thesis.domain

import org.hibernate.annotations.*
import org.hibernate.annotations.Cache
import java.time.*
import javax.persistence.*
import javax.persistence.Entity
import javax.persistence.Table

@Entity
@Table(name = "country")
@Cache(usage = CacheConcurrencyStrategy.NONSTRICT_READ_WRITE)
class Country(

    @Id
```

```
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
"sequenceGenerator")
    @SequenceGenerator(name = "sequenceGenerator")
    val id: Long? = null,

    @Column(nullable = false, unique = true)
    val name: String,

    @Column(nullable = false, unique = true)
    val code: String,

    @Column(nullable = false)
    val timezone: String,

    @Column
    var lastUpdated: LocalDate?
)
```

Model.kt

```
package org.kry.thesis.domain

import com.fasterxml.jackson.annotation.*
import org.hibernate.annotations.*
import org.hibernate.annotations.Cache
import java.time.*
import javax.persistence.*
import javax.persistence.Entity
import javax.persistence.Table

@Entity
@Table(name = "model")
@Cache(usage = CacheConcurrencyStrategy.NONSTRICT_READ_WRITE)
class Model(

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
"sequenceGenerator")
    @SequenceGenerator(name = "sequenceGenerator")
    val id: Long? = null,

    @JsonIgnore
    @ManyToOne
    @JoinColumn(nullable = false)
    val heater: Heater,

    @Column(nullable = false)
    var name: String,

    @Column(nullable = false)
    val targetTemperature: Float,

    @Column(nullable = false)
    val minTemperature: Float,

    @Column(nullable = false)
    val maxTemperature: Float,

    @Column(nullable = false)
```

```
    val createdOn: Instant,

    @Enumerated(EnumType.STRING)
    @Column(nullable = false)
    var status: ModelStatus
)

enum class ModelStatus {
    Created,
    Training,
    Trained
}
```

Schedule.kt

```
package org.kry.thesis.domain

import org.hibernate.annotations.*
import org.hibernate.annotations.Cache
import java.time.*
import javax.persistence.*
import javax.persistence.Entity
import javax.persistence.Table

@Entity
@Table(name = "schedule")
@Cache(usage = CacheConcurrencyStrategy.NONSTRICT_READ_WRITE)
class Schedule(

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
"sequenceGenerator")
    @SequenceGenerator(name = "sequenceGenerator")
    val id: Long? = null,

    @ManyToOne
    @JoinColumn(nullable = false)
    val heater: Heater,

    @Column(nullable = false)
    val scheduleDate: LocalDate,

    @Column(nullable = false)
    var data: String
)
```

environment.pyx

```
from __future__ import annotations
import numpy as np
import cython
import math

from reward cimport RewardPolicy

cdef class LinearEnvironment:

    cdef readonly float target_temperature
    cdef readonly float simulation_dt
    cdef readonly float step_dt
```

```
cdef readonly float max_time
cdef readonly int output_delta_t
cdef readonly int degree_div
cdef readonly int num_steps

cdef readonly float[:] price_data
cdef readonly float[:] outside_temperature

cdef readonly float time
cdef readonly float temp
cdef readonly int heater_state
cdef readonly double consumption
cdef readonly double cost
cdef readonly float reward
cdef readonly int price_start_index

cdef readonly float heater_power
cdef readonly float heat_capacity
cdef readonly float thermal_conductance
cdef readonly float KWH_TO_J
cdef readonly float MWH_TO_J

cdef readonly int state_count

def __init__(
        self,
        target_temperature: float,
        price_data: np.ndarray,
        outside_temperature: np.ndarray,
        simulation_dt: float,
        step_dt: float,
        max_time: float,
        output_delta_t: int,
        degree_div: int,
        price_start_index: int,
        one_step_cooling: float | None = None,
        one_step_heating: float | None = None,
        heater_power: float = 13000,  # (W)
        heat_capacity: float = 1e7,  # (J/K)
        thermal_conductance: float = 325  # (W/K)
):
    self.target_temperature = target_temperature
    self.simulation_dt = simulation_dt
    self.step_dt = step_dt
    self.max_time = max_time
    self.output_delta_t = output_delta_t
    self.degree_div = degree_div
    self.num_steps = int(math.ceil(self.max_time/self.step_dt))

    self.price_data = price_data
    self.outside_temperature = outside_temperature

    self.time = 0
    self.temp = target_temperature
    self.heater_state = 0
    self.consumption = 0
    self.cost = 0
    self.reward = 0
```

```
        self.price_start_index = price_start_index

        self.heater_power = heater_power

        self.KWH_TO_J = 3.6e6
        self.MWH_TO_J = 3.6e9

        if one_step_cooling is not None and one_step_heating is not
None:
            self.heat_capacity = heater_power * step_dt /
(one_step_heating + one_step_cooling)
            self.thermal_conductance = self.heat_capacity *
one_step_cooling / target_temperature / step_dt
        else:
            self.heat_capacity = heat_capacity
            self.thermal_conductance = thermal_conductance

        self.state_count = (2 * output_delta_t + 1) * degree_div

    cpdef LinearEnvironment copy(self):
        result = LinearEnvironment(
            target_temperature=self.target_temperature,
            price_data=self.price_data,
            outside_temperature=self.outside_temperature,
            simulation_dt=self.simulation_dt,
            step_dt=self.step_dt,
            max_time=self.max_time,
            output_delta_t=self.output_delta_t,
            degree_div=self.degree_div,
            price_start_index=self.price_start_index,
            heater_power=self.heater_power,
            heat_capacity=self.heat_capacity,
            thermal_conductance=self.thermal_conductance
        )

        result.time = self.time
        result.temp = self.temp
        result.heater_state = self.heater_state
        result.consumption = self.consumption
        result.cost = self.cost
        result.reward = self.reward

        return result

    @cython.cdivision(True)
    def reset(self, initial_state: cython.int):
        self.time = 0
        self.temp = self.target_temperature - self.output_delta_t +
initial_state * 1.0 / self.degree_div
        self.heater_state = 0
        self.consumption = 0
        self.cost = 0
        self.reward = 0

    @cython.cdivision(True)
    cpdef tuple[float, bool] step(self, next_state: cython.int,
reward_policy: RewardPolicy):
```

```
        if self.time >= self.max_time:
            return 0, True

        reward: cython.float = 0
        done = False

        cur_target_temp: cython.float = self.target_temperature -
self.output_delta_t + next_state * 1.0 / self.degree_div

        step_end: cython.float = self.time + self.step_dt
        while self.time < step_end:

            self.time += self.simulation_dt

            cur_action: cython.int = 0
            if self.temp < cur_target_temp:
                cur_action = 1

            cur_hour_index: cython.int = self.price_start_index +
<int>(self.time / 3600)
            cur_outside_temperature: cython.float =
self.outside_temperature[cur_hour_index]
            electricity_price: cython.float =
self.price_data[cur_hour_index]

            incoming_power: cython.float = self.heater_power *
cur_action
            outgoing_power: cython.float = self.thermal_conductance *
(self.temp - cur_outside_temperature)
            self.temp += (incoming_power - outgoing_power) *
self.simulation_dt / self.heat_capacity

            delta_cost: cython.double = electricity_price *
incoming_power * self.simulation_dt / self.MWH_TO_J
            self.consumption += incoming_power * self.simulation_dt /
self.KWH_TO_J
            self.cost += delta_cost

            temp_diff: cython.float = self.temp -
self.target_temperature
            delta_reward = reward_policy.reward(
                dt=self.simulation_dt,
                cost=delta_cost,
                temp_diff = temp_diff
            )
            reward += delta_reward

        if self.time >= self.max_time:
            done = True

        self.reward += reward
        return reward, done
```

reward.pyx

```
cdef class RewardPolicy:
    cdef float reward(self, float dt, float temp_diff, float cost):
        return 0
```

```
cdef class TempOnlyRewardPolicy(RewardPolicy):
    cdef float reward(self, float dt, float temp_diff, float cost):
        return - temp_diff * temp_diff * dt


cdef class CostOnlyRewardPolicy(RewardPolicy):
    cdef float reward(self, float dt, float temp_diff, float cost):
        return - cost

cdef class LLRewardPolicy(RewardPolicy):

    cdef float beta

    def __init__(self, beta):
        self.beta = beta

    cdef float reward(self, float dt, float temp_diff, float cost):
        return - cost * (1 - self.beta) - abs(temp_diff) * dt *
self.beta

cdef class LQRewardPolicy(RewardPolicy):

    cdef float beta

    def __init__(self, beta):
        self.beta = beta

    cdef float reward(self, float dt, float temp_diff, float cost):
        return - cost * (1 - self.beta) - temp_diff * temp_diff * dt *
self.beta

cdef class LLCRewardPolicy(RewardPolicy):

    cdef float beta

    def __init__(self, beta):
        self.beta = beta

    cdef float reward(self, float dt, float temp_diff, float cost):
        if temp_diff < 0:
            return - cost * (1 - self.beta) - (-temp_diff) * dt *
self.beta
        else:
            return - cost * (1 - self.beta)

cdef class LQCRewardPolicy(RewardPolicy):

    cdef float beta

    def __init__(self, beta):
        self.beta = beta

    cdef float reward(self, float dt, float temp_diff, float cost):
        if temp_diff < 0:
            return - cost * (1 - self.beta) - temp_diff * temp_diff *
dt * self.beta
        else:
            return - cost * (1 - self.beta)
```

```
cdef class MLRewardPolicy(RewardPolicy):

    cdef float beta

    def __init__(self, beta):
        self.beta = beta

    cdef float reward(self, float dt, float temp_diff, float cost):
        return - cost * (1 + self.beta * abs(temp_diff))

cdef class MQRewardPolicy(RewardPolicy):

    cdef float beta

    def __init__(self, beta):
        self.beta = beta

    cdef float reward(self, float dt, float temp_diff, float cost):
        return - cost * (1 + self.beta * temp_diff * temp_diff)

cdef class MLCRewardPolicy(RewardPolicy):

    cdef float beta

    def __init__(self, beta):
        self.beta = beta

    cdef float reward(self, float dt, float temp_diff, float cost):
        if temp_diff < 0:
            return - cost * (1 + self.beta * (-temp_diff))
        else:
            return - cost


cdef class MQCRewardPolicy(RewardPolicy):

    cdef float beta

    def __init__(self, beta):
        self.beta = beta

    cdef float reward(self, float dt, float temp_diff, float cost):
        if temp_diff < 0:
            return - cost * (1 + self.beta * temp_diff * temp_diff)
        else:
            return - cost
```

agent.py

```
from __future__ import annotations
import math
import random
from collections import deque
from itertools import count
from dataclasses import dataclass, astuple
from tqdm.notebook import trange

import numpy as np
```

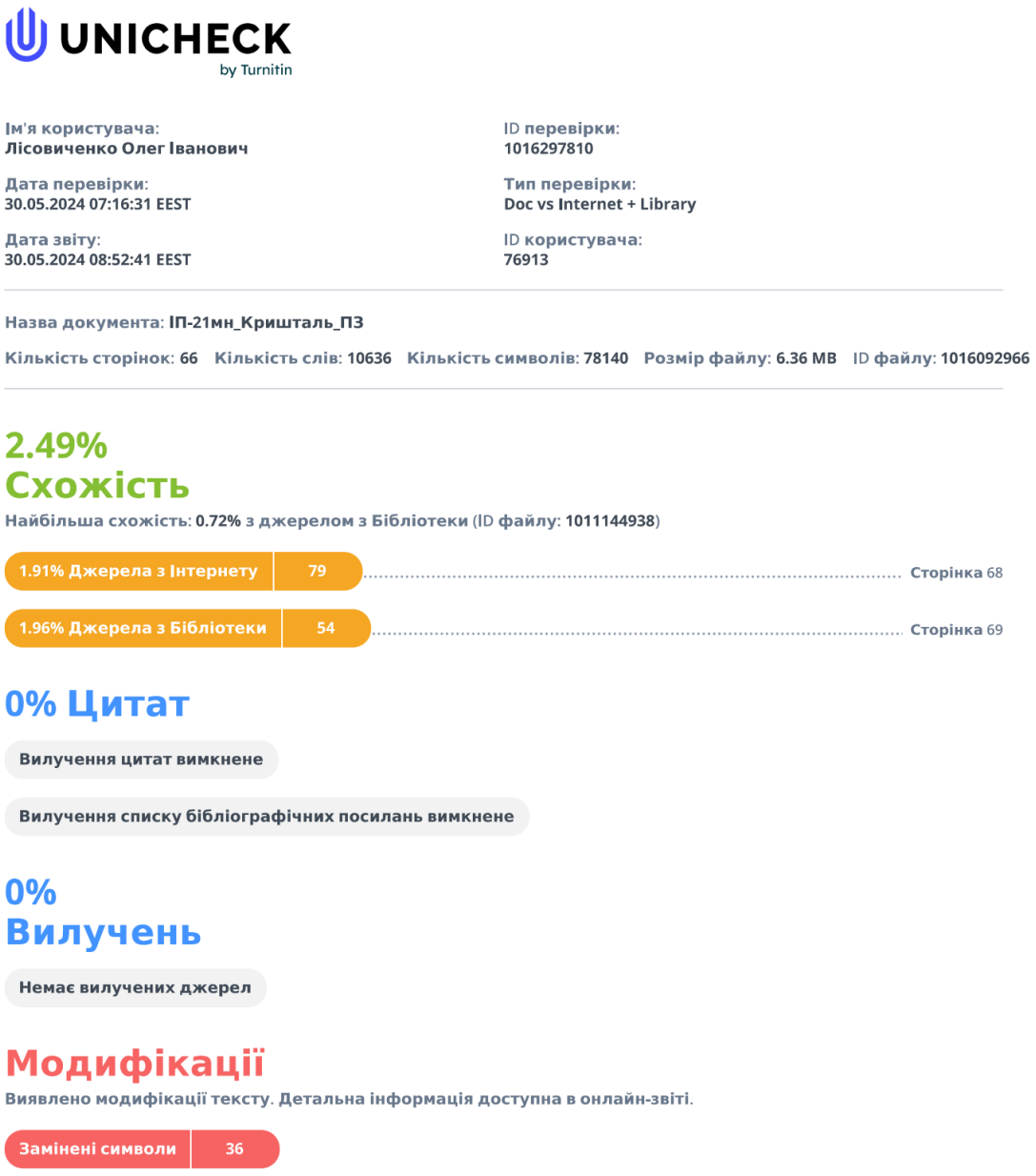
```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch import optim

from agent import Agent
from environment.environment import Environment
from reporter import Reporter


@dataclass
class Transition:
    state: list[float]
    action: int
    next_state: list[float]
    reward: float

    def __iter__(self):
        return iter(astuple(self))


class ReplayMemoryStorage:
    def __init__(self, capacity):
        self.memory: deque[Transition] = deque([], maxlen=capacity)

    def push_value(self, transition: Transition):
        self.memory.append(transition)

    def random_sample(self, batch_size: int) -> list[Transition]:
        return random.sample(self.memory, batch_size)

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


class NN(nn.Module):
    def __init__(self, input_size: int, output_size: int, hidden_size:
int):
        super().__init__()
        self.layer1 = nn.Linear(input_size, hidden_size)
        self.layer2 = nn.Linear(hidden_size, hidden_size)
        self.layer3 = nn.Linear(hidden_size, hidden_size)
        self.layer4 = nn.Linear(hidden_size, hidden_size)
        self.layer5 = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        x = F.relu(self.layer1(x))
        x = F.relu(self.layer2(x))
        x = F.relu(self.layer3(x))
        x = F.relu(self.layer4(x))
        return self.layer5(x)


class RLAgent(Agent):
    def __init__(
            self,
            state_space_size: int,
```

```
            action_space_size: int,
            hidden_layer_size: int,
            replay_memory_size: int,
            batch_size: int,
            learning_rate: float,
            gamma: float,
            tau: float,
            record_history: bool = False
    ):
        self.__state_space_size = state_space_size
        self.__action_space_size = action_space_size
        self.hidden_layer_size = hidden_layer_size
        self.batch_size = batch_size
        self.gamma = gamma
        self.tau = tau

        self.policy_net = NN(input_size=state_space_size,
output_size=action_space_size, hidden_size=hidden_layer_size)
        self.target_net = NN(input_size=state_space_size,
output_size=action_space_size, hidden_size=hidden_layer_size)
        self.target_net.eval()
        self.target_net.load_state_dict(self.policy_net.state_dict())

        self.memory = ReplayMemoryStorage(replay_memory_size)
        self.optimizer = optim.AdamW(self.policy_net.parameters(),
lr=learning_rate, amsgrad=True)

        self.history: list[RLAgent] = []
        self.record_history = record_history

    @classmethod
    def copy(cls, agent: RLAgent) -> RLAgent:
        obj = cls.__new__(cls)
        super(RLAgent, obj).__init__()

        obj.__state_space_size = agent.__state_space_size
        obj.__action_space_size = agent.action_space_size
        obj.hidden_layer_size = agent.hidden_layer_size

        obj.policy_net = NN(input_size=obj.state_space_size,
output_size=obj.action_space_size, hidden_size=obj.hidden_layer_size)
        obj.target_net = NN(input_size=obj.state_space_size,
output_size=obj.action_space_size, hidden_size=obj.hidden_layer_size)
        obj.target_net.eval()

        target_net_state_dict = agent.target_net.state_dict()
        obj.target_net.load_state_dict(target_net_state_dict)
        policy_net_state_dict = agent.policy_net.state_dict()
        obj.policy_net.load_state_dict(policy_net_state_dict)

        return obj

    def action(self, state: list[float]) -> int:
        self.policy_net.eval()

        with torch.no_grad():
            return self.policy_net(torch.tensor(state,
dtype=torch.float).unsqueeze(0)).squeeze(0).argmax().item()
```

```
    def Q_function(self, state: list[float]) -> list[float]:
        self.policy_net.eval()

        with torch.no_grad():
            return self.policy_net(torch.tensor(state,
dtype=torch.float).unsqueeze(0)).squeeze(0).tolist()

    def epsilon_greedy_action(self, state: list[float], epsilon: float)
-> int:
        if random.random() > epsilon:
            return self.action(state)
        else:
            return random.randint(0, self.__action_space_size - 1)

    def optimize(self):
        if len(self.memory) < self.batch_size:
            return
        self.policy_net.train()

        cur_batch =
Transition(*zip(*self.memory.random_sample(self.batch_size)))

        non_final_binary_mask = torch.tensor([s is not None for s in
cur_batch.next_state])
        non_final_filtered_next_states = torch.tensor([s for s in
cur_batch.next_state if s is not None])
        state_batch_tensor = torch.tensor(np.array(cur_batch.state,
dtype=np.float32))
        action_batch_tensor = torch.tensor(cur_batch.action)
        reward_batch_tensor = torch.tensor(cur_batch.reward)

        state_action_values =
self.policy_net(state_batch_tensor).gather(1,
action_batch_tensor.unsqueeze(1))

        next_state = torch.zeros(self.batch_size)
        with torch.no_grad():
            next_state[non_final_binary_mask] =
self.target_net(non_final_filtered_next_states).max(1).values
        expected_aciton_state_values = reward_batch_tensor + self.gamma
* next_state

        crit = nn.SmoothL1Loss()
        loss = crit(state_action_values.squeeze(),
expected_aciton_state_values)
        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()

    def train(self, env: Environment, num_episodes: int, eps_start:
float, eps_end: float, eps_decay: float,
              reporter: Reporter):
        if env.state_space_size != self.state_space_size:
            raise ValueError(f'Environment and agent space state sizes
don\'t match: {env.state_space_size} and {self.state_space_size}')

        if env.action_space_size != self.action_space_size:
```

```
            raise ValueError(f'Environment and agent action space sizes
don\'t match: {env.action_space_size} and {self.action_space_size}')

        for i_episode in trange(num_episodes):
            cur_state = env.reset()
            cur_epsilon = eps_end
            if eps_decay != 0:
                cur_epsilon = eps_end + (eps_start - eps_end) *
math.exp(-i_episode / eps_decay)
            for t in count():
                action = self.epsilon_greedy_action(cur_state,
epsilon=cur_epsilon)
                next_state, reward, done = env.step(action)
                if done:
                    next_state = None

                self.memory.push_value(Transition(cur_state, action,
next_state, reward))

                cur_state = next_state

                self.optimize()

                target_net_state_dict = self.target_net.state_dict()
                policy_net_state_dict = self.policy_net.state_dict()
                for key in policy_net_state_dict:
                    target_net_state_dict[key] =
policy_net_state_dict[key] * self.tau + target_net_state_dict[key] * (
                                1 - self.tau)
                self.target_net.load_state_dict(target_net_state_dict)

                if done:

reporter.report_measurements(env.get_measurements())
                    if self.record_history:
                        self.history.append(RLAgent.copy(self))
                    break

    @property
    def state_space_size(self) -> int:
        return self.__state_space_size

    @property
    def action_space_size(self) -> int:
        return self.__action_space_size
```

ДОДАТОК В РЕЗУЛЬТАТ ПЕРЕВІРКИ РОБОТИ НА СПІВПАДІННЯ



UNICHECK
by Turnitin

Ім'я користувача:
Лісовиченко Олег Іванович

ID перевірки:
1016297810

Дата перевірки:
30.05.2024 07:16:31 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
30.05.2024 08:52:41 EEST

ID користувача:
76913

Назва документа: ІП-21мн_Кришталь_ПЗ

Кількість сторінок: 66 Кількість слів: 10636 Кількість символів: 78140 Розмір файлу: 6.36 MB ID файлу: 1016092966

2.49%
Схожість

Найбільша схожість: 0.72% з джерелом з Бібліотеки (ID файлу: 1011144938)

1.91% Джерела з Інтернету 79 ..... Сторінка 68

1.96% Джерела з Бібліотеки 54 ..... Сторінка 69

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи 36