ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Методика автоматизації прогнозування термінів виконання завдань для малих команд з використанням машинного навчання»

на здобуття освітнього ступеня магістра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Ігор КОЛОМІЄЦЬ
(підпис)

Виконав: здобувач вищої освіти групи ПДМ-63
Ігор КОЛОМІЄЦЬ

Керівник: Ірина ЗАМРІЙ
д-р техн. наук, проф.

Рецензент: _____
науковий ступінь, вчене звання *Ім'я, ПРІЗВИЩЕ*

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Магістр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
«_____» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Коломійцю Ігорю Юрійовичу

1. Тема кваліфікаційної роботи: «Методика автоматизації прогнозування термінів виконання завдань для малих команд з використанням машинного навчання»

керівник кваліфікаційної роботи Ірина ЗАМРІЙ, доктор технічних наук, професор

затверджені наказом Державного університету інформаційно-комунікаційних технологій від « 30 » жовтня 2025 р. № 467 .

2. Строк подання кваліфікаційної роботи «19» грудня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література; параметри моделей TF-IDF, K-means, Random Forest Regressor; прототип прогнозування.

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

1. Дослідження сучасних рішень у сфері управління завданнями та визначення їхніх обмежень щодо прогнозування часу виконання завдань і пріоритизації.
2. Огляд алгоритмів машинного навчання та їх застосування у системах управління завданнями.
3. Розробка методики автоматичного прогнозування часу виконання завдань і їх пріоритизації.

5. Перелік ілюстративного матеріалу: *презентація*
   1. Існуючі алгоритми вирішення задачі, їх переваги та недоліки
   2. Математична модель комплексного методу
   3. Структура фінального вектора ознак
   4. Алгоритм роботи комплексного методу
   5. Архітектура взаємодії системи управління завданнями та сервісу прогнозування
   6. Діаграма Use-Case веб-застосунку системи управління завданнями
   7. Практичний результат
   8. Порівняльний аналіз ефективності розробленої методики
6. Дата видачі завдання « 31 » жовтня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	31.10-03.11.25	виконано
2	Розробка концептуальної моделі та загального алгоритму методики прогнозування	06.11-20.11.25	виконано
3	Проектування структури програмного прототипу та вибір програмних засобів	20.11-22.11.25	виконано
4	Програмна реалізація основних модулів: обробка даних, векторизація та модель прогнозування	23.11-25.11.25	виконано
5	Визначення метрик оцінки та розробка методики експериментального дослідження	25.11.2025	виконано
6	Проведення експериментальних досліджень та порівняльний аналіз ефективності розробленої методики з базовими підходами	26.11-30.11.25	виконано
7	Оформлення роботи: вступ, висновки, реферат	01.12-03.12.25	виконано
8	Розробка демонстраційних матеріалів	03.12-19.12.25	виконано

Здобувач вищої освіти _____ Ігор КОЛОМІЄЦЬ
(підпис)

Керівник
кваліфікаційної роботи _____ Ірина ЗАМРІЙ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 83 стор., 5 табл., 38 рис., 40 джерел.

Мета роботи – підвищення ефективності робочих процесів за рахунок автоматичного прогнозування часу та пріоритетів.

Об'єкт дослідження – процес прогнозування часу виконання та пріоритизацію в управлінні завданнями.

Предмет дослідження – методи та алгоритми машинного навчання для автоматизації прогнозування часу та пріоритизації завдань.

У роботі використано різноманітні методи, такі як системний аналіз, методи обробки природної мови, кластерний аналіз, та методи регресії.

Проведено аналіз сучасних методів для управління завданнями, виявлено їхні обмеження точності прогнозування та автоматизації пріоритизації.

Розроблено та оптимізовано метод автоматизації прогнозування термінів виконання завдань, яка поєднує етапи попередньої обробки текстових описів завдань їх кластеризації за рівнем складності та побудови регресійної моделі для прогнозування часу.

Проведено експерименти для перевірки точності розробленої методики на основі реального набору даних. Виконано порівняльний аналіз результатів прогнозування моделі з ручними експертними оцінками, що дозволило кількісно оцінити ефективність запропонованого рішення.

КЛЮЧОВІ СЛОВА: АВТОМАТИЗАЦІЯ УПРАВЛІННЯ ЗАВДАННЯМИ, МАШИННЕ НАВЧАННЯ, ПРОГНОЗУВАННЯ ЧАСУ, КЛАСТЕРИЗАЦІЯ, TF-IDF, RANDOM FOREST REGRESSOR, K-MEANS.

ABSTRACT

Text part of the master's qualification work: 83 pages, 38 pictures, 5 table, 40 sources.

The purpose of the work is to increase the efficiency of work processes by automatically predicting time and priorities.

Object of research – the process of predicting execution time and prioritizing in task management.

Subject of research – machine learning methods and algorithms for automating time forecasting and task prioritization.

Summary of the work: The work uses various methods, such as system analysis, natural language processing methods, cluster analysis, and regression methods.
An analysis of modern methods for task management was conducted, and their limitations in forecasting accuracy and prioritization automation were identified.
A method for automating task time forecasting was developed and optimized, which combines the stages of preprocessing text descriptions of tasks, clustering them by level of complexity, and building a regression model for time forecasting.
Experiments were conducted to verify the accuracy of the developed methodology based on a real dataset. A comparative analysis of the model forecasting results with manual expert assessments was performed, which allowed us to quantitatively assess the effectiveness of the proposed solution.

KEYWORDS: TASK MANAGEMENT AUTOMATION, MACHINE LEARNING, TIME FORECASTING, CLUSTERIZATION, TF-IDF, RANDOM FOREST REGRESSOR, K-MEANS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

TF-IDF - Частота слова - обернена частота документа (Term Frequency-Inverse Document Frequency)

LLM - Велика мовна модель (Large language model)

GPT - Генеративний попередньо тренований трансформер (Generative Pre-trained Transformer)

ML - Машинне навчання (Machine learning)

SVR - Метод опорних векторів (Support Vector Regressiono)

XGBoost - Екстремальне градієнтне підсилювання (eXtreme Gradient Boosting)

RBF - Мережа радіально базисних функцій (Radial basis function networks)

PHP - Гіпертекстовий препроцесор (Hypertext Preprocessor)

MYSQL - Система керування реляційними базами даних (My Structured Query Language)

HTTP - Гіпертекстовий протокол передачі даних (HyperText Transfer Protocol)

API - Інтерфейс програмування застосунків (Application programming interface)

NLP - Нейролінгвістичне програмування (Neuro-linguistic programming)

ВСТУП

В умовах стрімкого розвитку IT-індустрії та поширення гнучких методологій розробки, де робота часто ведеться в невеликих, динамічних командах, ключовим фактором успіху є продуктивне управління проєктами. Однак сучасні підходи до планування, ручне прогнозування термінів виконання завдань та визначення їх пріоритетів, часто призводять до помилок, зривів дедлайнів та неоптимального використання ресурсів. Тому застосування методів машинного навчання відкриває нові можливості для автоматизації та інтелектуалізації цих процесів.

Сучасні системи управління завданнями, такі як Jira, Trello чи Asana, є інструментами для організації роботи, проте їхній функціонал в більшості зосереджений на відстеженні прогресу, а не на інтелектуальному аналізі та прогнозуванні. Використання алгоритмів машинного навчання дозволяє перейти від простого зберігання даних до їх аналізу, виявлення прихованих закономірностей та побудови точних прогнозних моделей. Це створює передумови для створення більш надійних та адаптивних систем управління.

Дана магістерська робота присвячена розробці та дослідженню методики автоматизації управління завданнями для малих команд із застосуванням машинного навчання. У ході роботи виконано аналіз існуючих підходів, їх переваг та недоліків, а також розроблено комплексну методику, що поєднує методи обробки текстових даних, кластеризації та регресійного аналізу для підвищення точності прогнозування.

Ця робота спрямована на вирішення важливої практичної проблеми в інженерії програмного забезпечення - зниження невизначеності у плануванні та підвищення продуктивності команд розробників. Таким чином, завдання розробки методики автоматизації прогнозування термінів виконання завдань з використанням машинного навчання є сучасним та актуальним.

Мета роботи - підвищення ефективності робочих процесів за рахунок автоматичного прогнозування часу та пріоритетів.

Об’єкт дослідження - процес прогнозування часу виконання та пріоритизацію в управлінні завданнями.

Предмет дослідження - методи та алгоритми машинного навчання для автоматизації прогнозування часу та пріоритизації завдань.

Для досягнення поставленої мети вирішувалися наступні завдання:

1. Провести літературний огляд та аналіз існуючих наукових джерел для отримання повного уявлення про різноманітні методи та технології, що використовуються в управлінні проєктами, прогнозуванні та застосуванні машинного навчання.

2. Провести аналіз сучасних систем управління завданнями та існуючих методів прогнозування часу виконання, дослідити їхні переваги, недоліки та визначити можливості для вдосконалення за допомогою машинного навчання.

3. Розробити комплексну методику автоматизації прогнозування, що включає етапи попередньої обробки текстових даних завдань з використанням алгоритму TF-IDF, їх кластеризації за допомогою K-means та побудови прогнозної моделі на основі регресійного алгоритму Random Forest.

4. Створити програмний прототип системи, що реалізує розроблену методику, з використанням мови програмування Python та бібліотек для машинного навчання Scikit-Learn.

5. Провести експериментальну перевірку розробленої методики на реальному наборі даних, виконати порівняльний аналіз точності прогнозів моделі з експертними оцінками та оцінити практичність запропонованого рішення.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Огляд актуальних наукових робіт в галузі автоматизації управління

Автоматизація управління завданнями для малих команд є важливою складовою підвищення ефективності роботи, завдяки використанню алгоритмів машинного навчання. У дослідженні Мельниченко та співавторів [1] розглядається проєктування вебзастосунку для керування робочим часом. Автори пропонують інтегроване рішення, яке дозволяє автоматизувати відстеження часу виконання завдань та оптимізувати організацію робочих процесів. Запропонований підхід демонструє можливості сучасних веб-технологій для забезпечення оперативного управління, що є надзвичайно важливим для малих команд.

У роботі Сотнікова та Лугова [2] досліджується автоматизація управлінської діяльності в умовах цифрової економіки. Автори аналізують впровадження інноваційних технологій, що сприяють оптимізації бізнес-процесів, розподілу завдань та планування робочого часу. Отримані результати дозволяють відзначити позитивний вплив автоматизації на підвищення ефективності управління, що актуально при розробці систем керування завданнями для малих команд.

Лошенюк [3] у своїй роботі розглядає автоматизацію управління бізнес-процесами як основу гнучкості підприємства. Автор детально аналізує характеристику існуючих систем управління, підкреслюючи необхідність інтеграції сучасних технологій для оптимізації процесів. Результати дослідження можуть бути використані для розробки методик автоматизації управління завданнями із застосуванням машинного навчання.

Тонковид та Мілашенко [4] пропонують машинне навчання для автоматизації управління базами даних, розробку алгоритмів для автоматичного моніторингу та оптимізації. Запропоновані алгоритмічні підходи демонструють високий потенціал для інтеграції в системи управління завданнями, де автоматизація процесів дозволяє підвищити точність прогнозування часу виконання та класифікації завдань.

У статті Краснокутська та Подопрухіна [5] проводиться аналіз методологій управління проєктами в ІТ-індустрії. Автори порівнюють різні підходи до організації процесів, підкреслюючи важливість сучасних технологій для оптимізації планування, моніторингу та контролю виконання завдань. Отримані результати створюють теоретичну основу для впровадження алгоритмів машинного навчання в системи автоматизації управління завданнями для малих команд.

Криворучко та співавтори [6] розглядають інформаційно-управляючі системи автоматизації бізнес-процесів підприємства на мобільних платформах. Автори демонструють, як сучасні мобільні технології дозволяють оптимізувати управління завданнями за рахунок оперативного доступу до даних та інтеграції з іншими інформаційними системами, що є ключовим для забезпечення ефективної роботи малих команд.

Кравченко, Гришкун та Власенко [7] аналізують методи класифікації в машинному навчанні із застосуванням бібліотеки scikit-learn. Автори детально описують різні алгоритми класифікації, що можуть бути використані для побудови моделей прогнозування часу виконання завдань, що є важливим елементом автоматизації управління завданнями.

У роботі Олещенка та Мельничука [8] розглядається застосування асамблевих методів машинного навчання для виявлення неправдивого тексту. Представлений підхід базується на поєднанні кількох алгоритмів, що забезпечує високу точність класифікації текстових даних. Отримані результати свідчать про можливість адаптації таких методів для аналізу описів завдань з метою їх класифікації та прогнозування часу виконання.

Стаття Chary Deekshith та Singh [9] містить огляд сучасних моделей машинного навчання із використанням бібліотеки scikit-learn. Автори порівнюють переваги та недоліки різних алгоритмів, що дозволяє вибрати найбільш ефективні підходи для реалізації автоматизованих систем управління завданнями, для прогнозування часу виконання завдань.

У дослідженні Крежевська, Poniszewska-Maranda та Ochelska-Mierzejewska [10] здійснено систематичне порівняння методів векторизації у контексті класифікації. Автори аналізують вплив різних способів перетворення текстових даних на ефективність алгоритмів машинного навчання, що має безпосереднє значення для оптимізації процесів автоматизації управління завданнями.

У дослідженні Ольги Гарбич-Мошори та співавторів [11] розглядається розробка мобільного додатку для трекінгу та контролю роботи над завданнями, який інтегрує сучасні можливості мобільних пристроїв із системами моніторингу. Запропонований підхід дозволяє автоматизувати відстеження виконання завдань та забезпечує оперативну інформацію про продуктивність команди, що сприяє підвищенню ефективності управління завданнями.

У роботі Орхан Масімов та співавторів [12] акцент зроблено на інтеграції штучного інтелекту в системи управління завданнями. Автори демонструють, як застосування алгоритмів машинного навчання сприяє автоматичному прогнозуванню часу виконання завдань та оптимальному розподілу пріоритетів, що є важливим для підвищення гнучкості та адаптивності систем управління у малих командах.

У дослідженні Celestine Iwendi та співавторів [13] представлено TF/IDF алгоритмічну модель для аналізу даних, що обробляються у режимі потокової передачі. Запропонований підхід дозволяє виділяти ключові ознаки текстових описів завдань, що може бути адаптовано для покращення класифікації та прогнозування часу виконання завдань у системах автоматизації.

Стаття Muhammad Saad Sheikh та співавторів [14] присвячена застосуванню динамічного кластеризаційного алгоритму на основі K-means з використанням нечіткої логіки в умовах FOG-середовища. Автори досліджують можливості адаптивного розкладу завдань, що забезпечує більш точне визначення пріоритетів та прогнозування часу виконання, що є критичним для оптимізації робочих процесів у малих командах.

У дослідженні André O. Sousa та співавторів [15] аналізується застосування методів машинного навчання для оцінки витрат зусиль і тривалості виконання

окремих завдань у програмних проєктах. Запропоновані алгоритмічні моделі дозволяють більш точно прогнозувати час виконання завдань, що сприяє оптимізації розподілу ресурсів і підвищенню ефективності планування у системах автоматизації управління завданнями.

У документації Scikit-Learn наведено базові принципи роботи з алгоритмами машинного навчання, методами класифікації, регресії та кластеризації, що забезпечують теоретичну основу для практичних рішень з автоматизації управління завданнями [16].

У дослідженні Pham, Durillo та Fahringer [17] запропоновано двоетапний підхід до прогнозування часу виконання робочих завдань у хмарних середовищах із застосуванням машинного навчання. Представлений підхід дозволяє оптимізувати розподіл ресурсів завдяки точному аналізу даних робочого процесу, що є надзвичайно важливим для автоматизації процесів управління завданнями.

Монографія Raschka, Liu та Mirjalili [18] детально описує сучасні підходи до розробки моделей машинного та глибокого навчання з використанням PyTorch і Scikit-Learn. Запропоновані методи дозволяють створювати прогностичні моделі, які можуть бути адаптовані для класифікації та прогнозування часу виконання завдань у системах автоматизації.

У огляді основних алгоритмів машинного навчання із застосуванням Scikit-Learn [19] представлено характерні особливості бібліотеки, що забезпечують можливості для розробки моделей прогнозування та кластеризації. Розглянуті алгоритми сприяють підвищенню точності автоматизації управління завданнями за рахунок оптимізації обробки даних.

Дослідження Коваля [20] присвячене методам лінійної регресії та алгоритму k-means для прогнозування та кластеризації виробничих показників із застосуванням Orange Data Mining. Запропоновані підходи демонструють ефективність використання класичних методів аналізу даних для оптимізації робочих процесів, що має велике значення для розробки систем автоматизації управління завданнями.

Ткачик та Бойко [21] детально досліджують ефективність різних методів кластеризації, що сприяють оптимізації процесу аналізу даних, що є важливим для автоматизації управління завданнями у малих командах. Автори аналізують застосування сучасних алгоритмів машинного навчання для групування різнотипових даних та підвищення точності прогнозування.

Ievlanov та Черепньов [22] розглядають використання алгоритмів кластеризації як засіб для ефективного призначення завдань виконавцям проєкту. В роботі аналізується підхід, що дозволяє автоматизувати процес розподілу роботи та покращити продуктивність командної діяльності, знижуючи вплив людського фактора на прийняття рішень.

Капліна [23] досліджує сучасні підходи до впровадження систем контролю в управлінні проєктами. Аналізовані методики дозволяють швидко коригувати управлінські процеси та оптимізувати розподіл ресурсів, для автоматизації управління завданнями. Використання цих механізмів дозволяє зменшити витрати часу на управлінські процеси.

Крикун та Медяник [24] пропонують критерії вибору електронних засобів для створення системи тайм-менеджменту, для забезпечення менеджерам оптимально планувати та розподіляти завдання у сучасних інформаційних системах. Автори також наголошують на важливості адаптивних систем управління завданнями, які підтримують персоналізоване налаштування пріоритетів.

Монографія Приходченка та співавторів [25] присвячена технологіям тайм-менеджменту в управлінській діяльності державних службовців, в якій розглядаються сучасні підходи до організації робочого часу та контролю за виконанням завдань. Запропоновані технології сприяють впровадженню систематизованих методів планування, що є важливим елементом автоматизації управління завданнями.

Таранич та Пелехацький [26] аналізують роль штучного інтелекту в стратегічному управлінні підприємствами. Автори підкреслюють, що

використання AI дозволяє оптимізувати бізнес-процеси прогнозування результатів діяльності.

Хорошилова [27] розглядає питання автоматизації обліку та управління в умовах малого бізнесу. Дослідження демонструє, як впровадження електронних систем дозволить знизити витрати на адміністрування та покращити виконання завдань.

Методи векторизації текстів для задач валідації, описані Кузьмой та Мельником [28], дозволяють перетворювати текстові дані у числові вектори, що створює аналітичну базу для подальшого застосування алгоритмів машинного навчання. Такий підхід є надзвичайно корисним для класифікації та прогнозування часу виконання завдань у системах автоматизації.

Джоші та Павленко [29] аналізують методи машинного навчання для виявлення фейкових новин, що підкреслює важливість вибору оптимальних алгоритмів для аналізу текстових даних. Отримані результати можуть бути адаптовані для покращення точності класифікації завдань та розподілу пріоритетів в автоматизованих системах управління.

Михайлов [30] аналізує традиційні підходи до планування проєктів у поєднанні з методами машинного навчання. Запропонована інтеграція класичних алгоритмів з сучасними технологіями дозволяє оптимізувати розподіл ресурсів та покращити точність планування завдань, для автоматизації управління завданнями у малих командах.

1.2 Сучасні системи управління завданнями та їх функціонал

На сьогоднішній день є багато різних інструментів для управління завданнями. Ці системи мають широкі можливості для створення і контролю завдань, але вони мають певні обмеження, які стосуються прогнозування часу виконання завдань і пріоритету та інші недоліки.

Trello - інструмент управління проєктами, останнім часом програма завойовує повагу професіоналів [31]. Інтерфейс інструменту можна подивитися на рисунку 1.1.

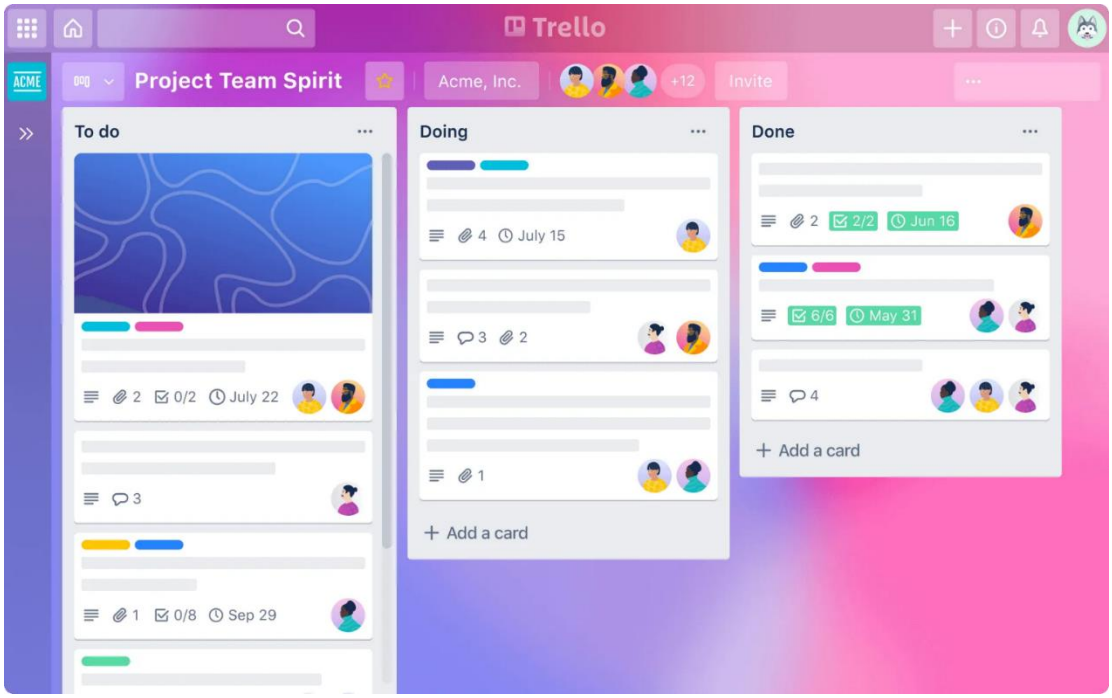


Рис. 1.1 Інтерфейс Trello

Переваги:

- має зрозумілий інтерфейс із системою дощок, карток та списків, що дозволяє легко організувати завдання;
- надає можливість командної роботи з додаванням членів команди до завдань, що підвищує прозорість і контроль за процесом;
- інтеграції з багатьма сторонніми інструментами, такими як Slack, Google Drive.

Недоліки:

- відсутність функцій для автоматичного прогнозування часу виконання завдань;
- не має можливості автоматичної пріоритезації завдань;
- обмеження щодо кількості інтеграцій та функціональності.

Відмінності від пропонованої роботи:

- у пропонованій системі будуть використовуватися методи машинного навчання для прогнозування часу виконання завдань на основі їх назви та опису завдання, а також автоматична класифікація завдань за пріоритетами.

Asana - це програма для хмарного управління проєктами, що значно полегшує проджект-менеджмент у складних комплексних реалізаціях з багатьма підзадачами чи окремими підрозділами [32]. Інтерфейс інструменту можна подивитися на рисунку 1.2.

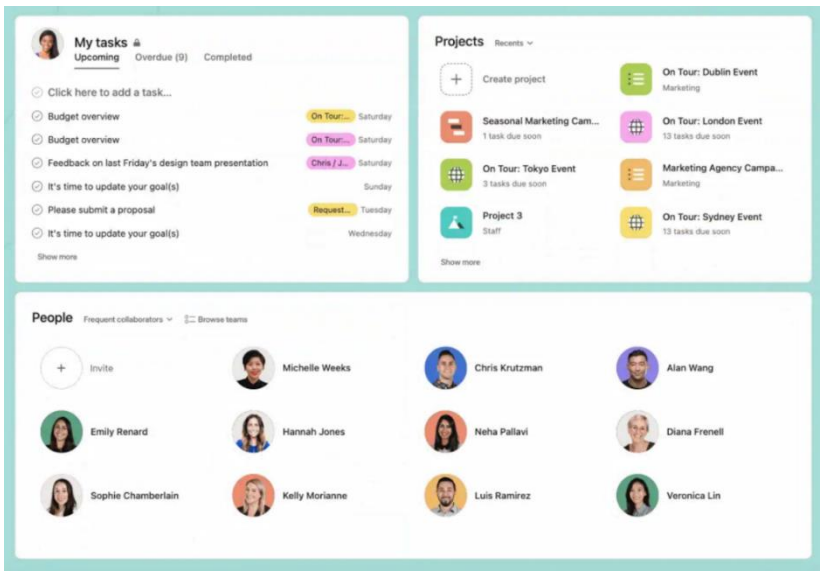


Рис. 1.2 Інтерфейс Asana

Переваги:

- підтримує складні робочі процеси з можливістю створення підзадач, контроль за дедлайнами та статус виконання задач;
- інтеграції з різними інструментами, щоб налаштувати процес під конкретні потреби команди;
- має можливість створення звітів і використання таймлайну для аналізу прогресу виконання завдань.

Недоліки:

- немає автоматичного прогнозування часу виконання завдань;
- додаткові можливості автоматизації обмежені;
- пріоритизація завдань потребує ручного налаштування яке займає час.

Відмінності від пропонованої роботи:

- пропонована система буде автоматично прогнозувати час виконання завдань та визначати їх пріоритет за допомогою машинного навчання.

YouTrack створено для обробки великої кількості завдань. Він має багато функцій, які забезпечують інтуїтивно зрозумілу та легку роботу між командами [33]. Інтерфейс інструменту можна подивитися на рисунку 1.3.

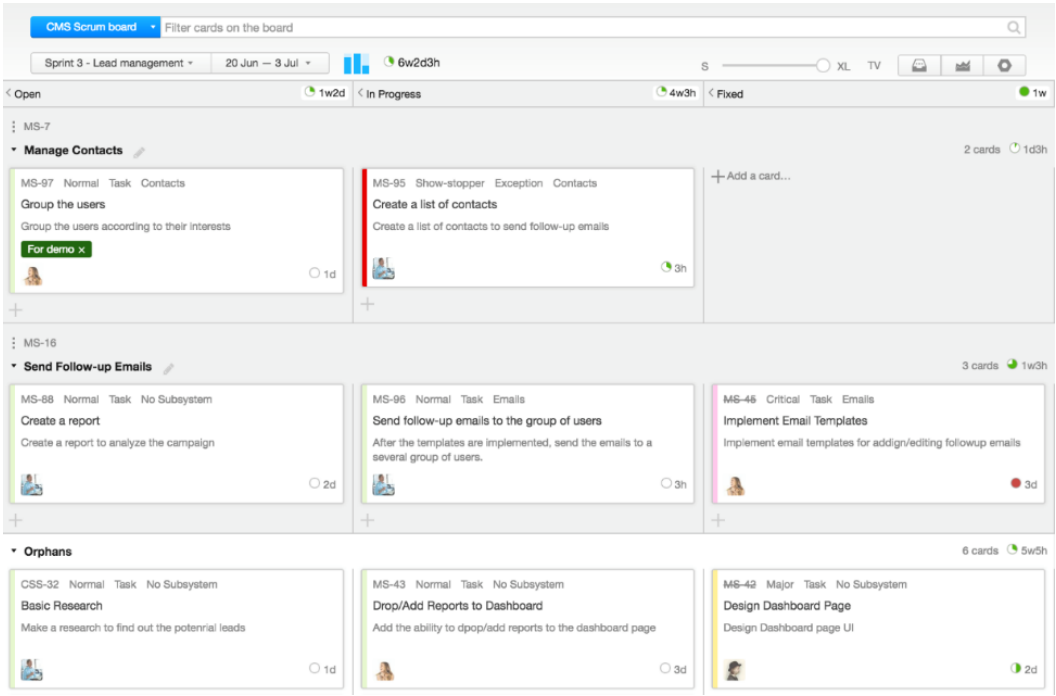


Рис. 1.3 Інтерфейс управління проєктами YouTrack

Переваги:

- має дуже зручний трекінг завдань, особливо коли команда велика або проєкт IT-шний;
- має інтеграцію з іншими продуктами JetBrains;
- має підтримку тайм-трекінгу та системи керування версіями, і можна налаштувати майже все.

Недоліки:

- складний інтерфейс для малих команд, що може ускладнити його використання;
- не пропонує автоматичного прогнозування часу виконання завдань;

- відсутня вбудована функція автоматичної класифікації завдань за пріоритетами.

Відмінності від пропонованої роботи:

- пропонована система буде простішою для малих команд та надаватиме функції машинного навчання для прогнозування часу виконання і автоматичної класифікації пріоритетів.

Microsoft Project 2010 - це рішення для керування проєктами, де можна розробляти розклади, призначати ресурси, керувати бюджетом, аналізувати навантаження та відстежувати перебіг виконання завдань. Доступний режим планувальника роботи групи, інтерфейс у вигляді стрічки, а також інші функції [34]. Інтерфейс інструменту можна подивитися на рисунку 1.4.

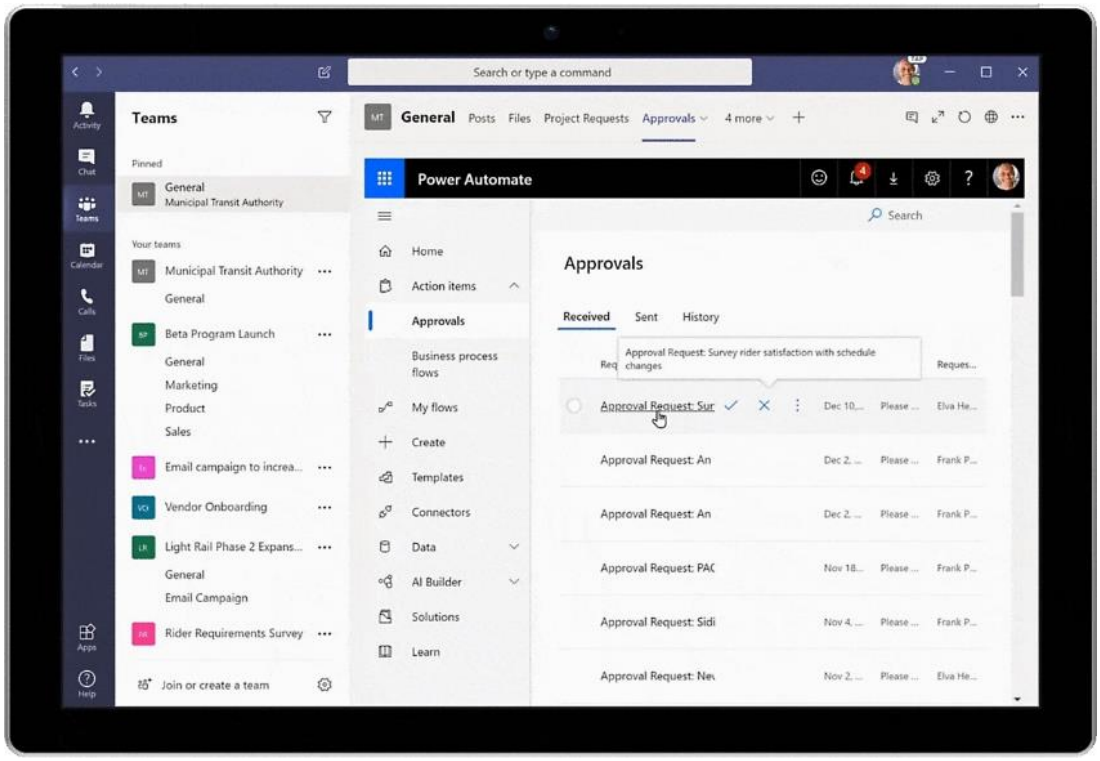


Рис. 1.4 Інтерфейс Microsoft Project

Переваги:

- має налаштування для планування, прогнозування і контролю за виконанням завдань;
- має можливість прогнозувати завантаження ресурсів і складання графіків;

- добре підходить для великих проєктів із багатьма завданнями та підзадачами.

Недоліки:

- складність використання;
- орієнтована більше на великі проєкти, що робить її важкою для адаптації у малих командах;
- немає можливостей автоматичного прогнозування часу виконання та пріоритизації.

Відмінності від пропонованої роботи:

- у порівнянні з Microsoft Project, пропонована система буде простішою і адаптованою для малих команд з акцентом на автоматизацію процесів за допомогою машинного навчання.

ClickUp – універсальний сервіс,має багато функцій, який можна використовувати як для командної роботи, так і для виконання індивідуальних завдань. Він дозволяє створювати робочі області для різних проєктів, додавати розділи завдань, підзавдань, контрольні списки, а також має широкі можливості для редагування їх та керування ними [35]. Інтерфейс інструменту можна подивитися на рисунку 1.5.

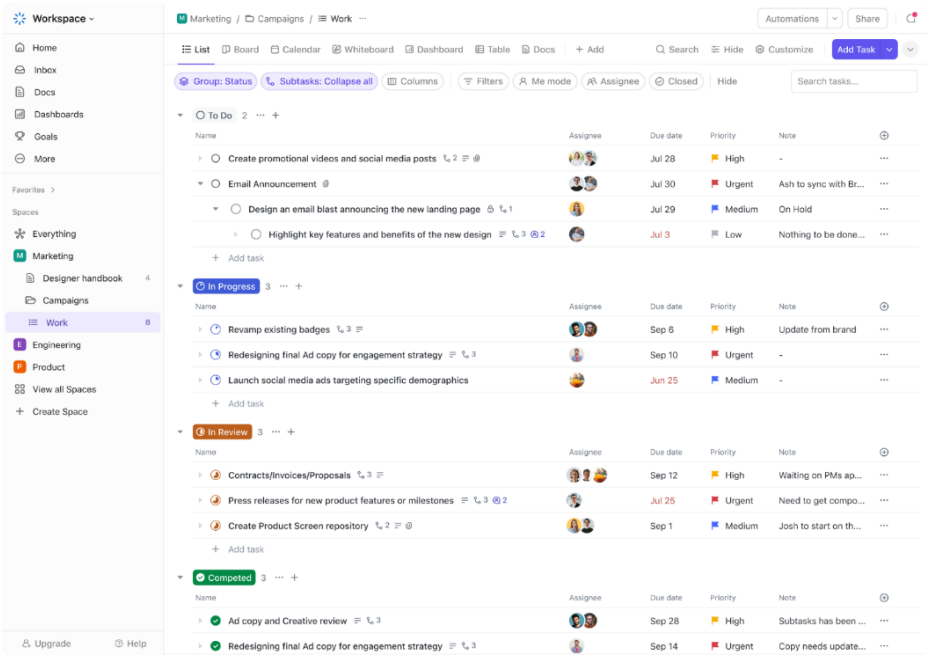


Рис. 1.5 Інтерфейс ClickUp

Переваги:

- має списки завдань, та можливість створювати підзадачі, встановлювати пріоритети і призначати членів команди для виконання задачі;
- дозволяє налаштувати робочі простори під конкретні потреби команди;
- має інтеграцію з іншими інструментами для управління завданнями, трекінгом часу і звітності.

Недоліки:

- немає вбудованих алгоритмів машинного навчання для прогнозування часу виконання завдань;
- пріоритети завдань доводиться ставити вручну, що забирає багато часу.

Відмінності від пропонованої роботи:

- пропонована система буде автоматично прогнозувати час виконання завдань і класифікувати їх за пріоритетами, використовуючи машинне навчання, тому користувачам не доведеться робити це вручну.

Wrike - це хмарна платформа для управління проєктами та співпраці, яка має на меті допомогти командам працювати краще. Вона пропонує ряд функцій для оптимізації робочих процесів, покращення комунікації та підвищення продуктивності [36]. Елемент інтерфейсу інструменту можна подивитися на рисунку 1.6.

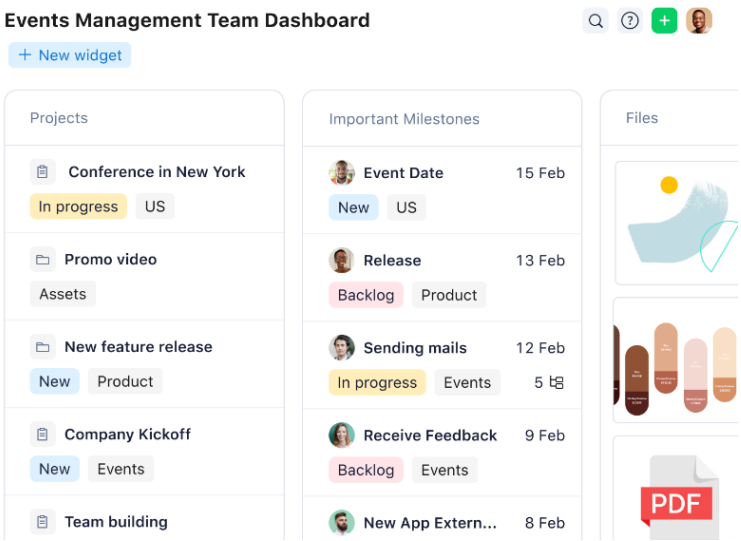


Рис. 1.6 Інтерфейс проєктів Wrike

Переваги:

- підтримка багаторівневої структури завдань, таймлайнів і календарів, що дозволяє краще керувати проєктами;
- інструменти для співпраці, обміну файлами і документами, інтеграція з Office 365, Google Workspace тощо;
- можливість створення звітів і аналізу продуктивності команд.

Недоліки:

- ніяк не прогнозує терміни виконання завдань автоматично;
- пріоритетність завдань виставляються вручну, що забирає досить багато часу.

Відмінності від пропонованої роботи:

- пропонована система інтегруватиме функції машинного навчання для автоматизації прогнозування часу виконання завдань та пріоритетизації, що є перевагою у зменшенні трудомісткості управління завданнями.

Jira - це популярна система управління проєктами та завданнями від компанії Atlassian. Вона необхідна для планування, оптимізації процесів, відстеження прогресу, командної роботи та аналітики. Jira допомагає командам залишатися на одній хвилі і досягати цілей [37]. Інтерфейс інструменту можна подивитися на рисунку 1.7.

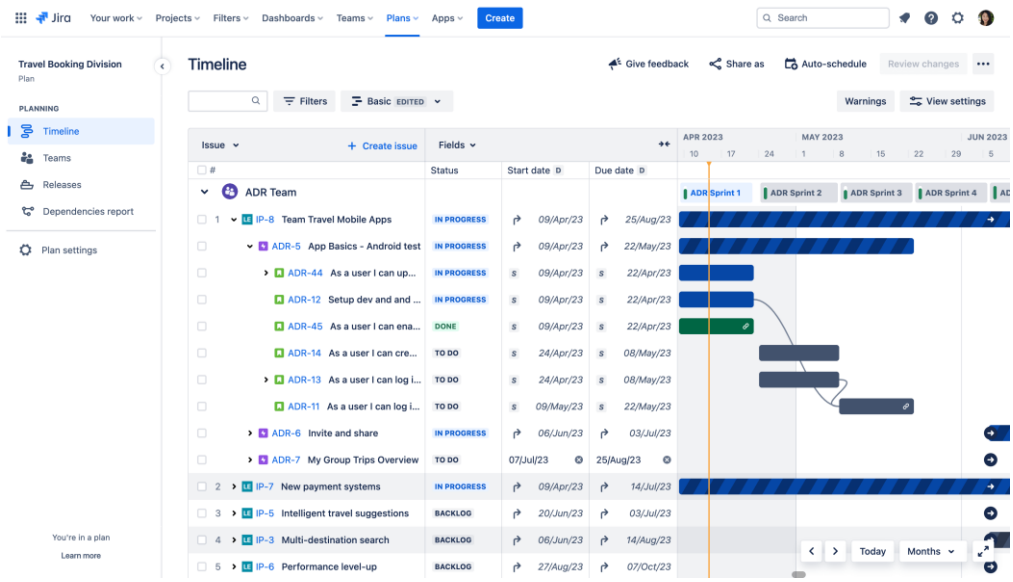


Рис. 1.7 Інтерфейс Jira

Переваги:

- спеціалізований інструмент для команд розробки, підтримує методології Agile, Scrum, Kanban;
- дозволяє дуже гнучко налаштувати для побудови робочих процесів і відстеження прогресу по спринтах і завданнях;
- має систему звітності, трекінг часу, аналіз продуктивності та оцінку завдань.

Недоліки:

- висока складність налаштувань та навчання для малих команд або команд без технічних фахівців;
- відсутність автоматичного прогнозування часу виконання завдань на основі машинного навчання;
- основний акцент на технічних командах, що ускладнює його адаптацію для загальних бізнес-процесів.

Відмінності від пропонованої роботи:

- у пропонованій системі акцент робиться на автоматизації процесів прогнозування часу виконання і пріоритизації завдань для спрощення роботи малих команд.

Notion - це універсальний додаток для організації, який поєднує таблиці, списки, бази даних та дошки в одному просторі, для легкого керування завданнями та проєктами на iPhone і Mac. За допомогою цього додатку можна зберігати текстові документи, відео та зображення, та синхронізувати дані між пристроями для легкої роботи і планування [38]. Інтерфейс інструменту можна подивитися на рисунку 1.8.

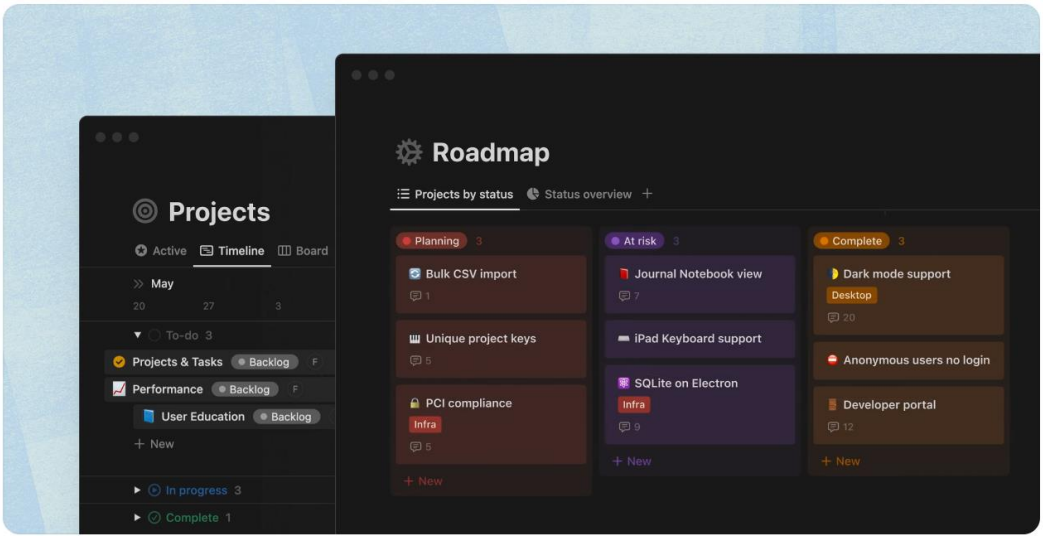


Рис. 1.8 Інтерфейс Notion

Переваги:

- є універсальною платформою для створення та управління завданнями, для організації завдання, документів, бази даних та нотатків в одному місці;
- є простим та має інтуїтивний інтерфейс, який добре підходить для невеликих команд;
- дуже гнучкий для налаштування під будь-який робочий процес завдяки можливостям розширення функціоналу.

Недоліки:

- немає розвинених можливостей для прогнозування часу виконання завдань та управління пріоритетами;
- він більше орієнтований на створення і зберігання інформації, ніж на специфічні функції управління проєктами.

Відмінності від пропонованої роботи:

- пропонована система включатиме можливості прогнозування часу та автоматичної пріоритизації завдань за допомогою машинного навчання, чого Notion не має.

Basecamp - це платформа для управління проєктами, яка допомагає невеликим командам рухатися швидше та досягати більшого прогресу, ніж вони

коли-небудь вважали можливим [39]. Інтерфейс інструменту можна подивитися на рисунку 1.9.

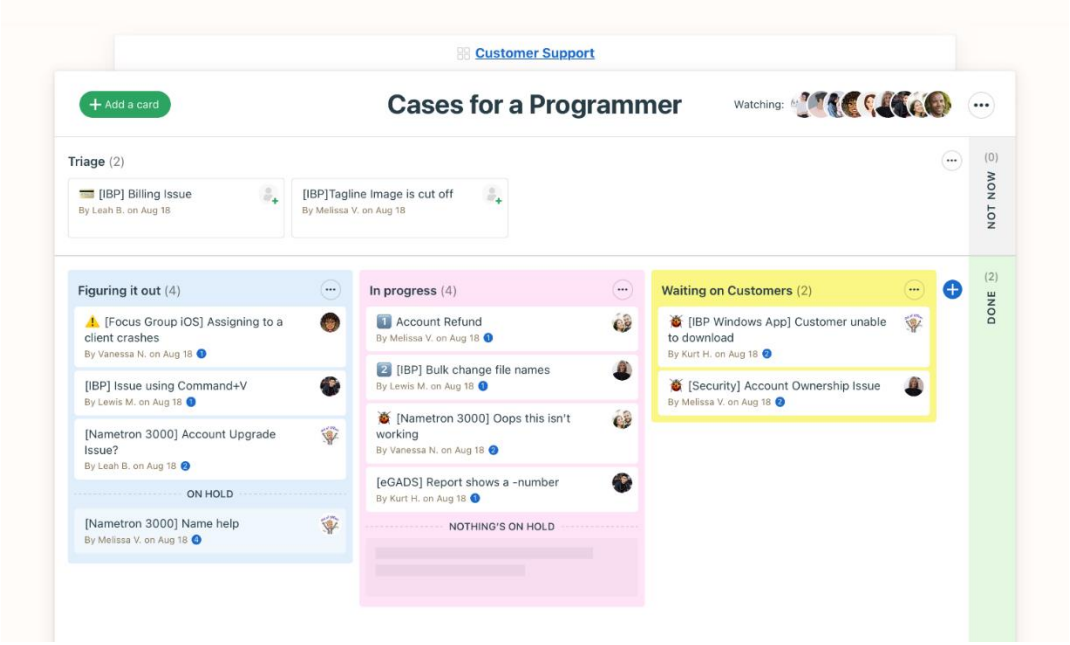


Рис. 1.9 Інтерфейс Basecamp

Переваги:

- дуже простий і зрозумілий інструмент, ідеально підходить для маленьких команд;
- має підтримку функцій відстеження прогресу, обміну файлами та календарів;
- легкий в роботі, підходить для малих команд без технічних спеціалістів.

Недоліки:

- немає машинного навчання для прогнозування часу виконання завдань або автоматизації класифікації завдань за пріоритетом;
- мало гнучкості для складних робочих процесів або великої кількості інтеграцій.

Відмінності від пропонованої роботи:

- пропонована система націлена на вдосконалення автоматизації управління завданнями, включаючи прогнозування часу виконання та класифікацію пріоритетів, що не передбачено в Basecamp.

Monday.com - це хмарна платформа управління роботою, яка допомагає командам оптимізувати свої процеси, проєкти та щоденну роботу. Це платформа з низьким рівнем кодування, яка дозволяє користувачам створювати спеціалізовані додатки та робочі процеси для задоволення своїх конкретних потреб [40]. Інтерфейс інструменту можна подивитися на рисунку 1.10.

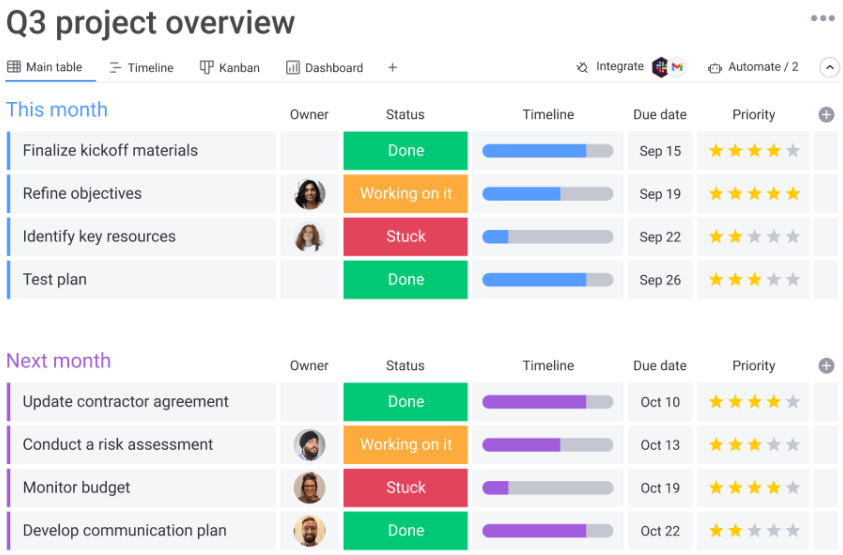


Рис. 1.10 Інтерфейс управління завданнями Monday.com

Переваги:

- пропонує гнучкі функції для налаштування та відстеження прогресу завдань, підходить для широкого спектра завдань від маркетингу до розробки;
- має зручний інтерфейс, який легко налаштовується, та має можливість інтеграції з іншими сервісами;
- має автоматизацію робочих процесів.

Недоліки:

- немає прогнозування часу виконання та автоматичного визначення пріоритетів завдання.

Відмінності від пропонованої роботи:

- пропонована система буде використовувати машинне навчання для прогнозування часу та автоматизації пріоритетів.

На основі проведеного аналізу, можна зробити висновок, що більшість існуючих рішень зосереджені на управлінні завданнями, але вони не мають автоматичного прогнозування часу виконання завдань або автоматичної класифікації пріоритетів на основі машинного навчання. Основними проблемами, які вирішуватиме пропонована система, це:

- автоматизація прогнозування часу виконання завдань;
- автоматична класифікація завдань за пріоритетами для оптимізації робочого процесу;
- система буде спрямована на малі команди з врахуванням простоти використання та мінімальної потреби у налаштуванні.

Ці функції надаватимуть малим командам перевагу, дозволяючи зосередитися на виконанні завдань без необхідності витрачати додатковий час на ручне управління пріоритетами та плануванні.

2 РОЗРОБКА МЕТОДИКИ АВТОМАТИЗОВАНОГО ПРОГНОЗУВАННЯ НА ОСНОВІ МАШИННОГО НАВЧАННЯ

З розвитком технологій та змінами в бізнес-середовищі, автоматизація процесів обліку та планування робочого часу стає критично важливою [1]. У даному розділі розглядається розробка методики автоматизованого прогнозування. Метою є створення системи, що поєднує переваги аналізу текстових даних та ансамблевих моделей машинного навчання для подолання недоліків існуючих рішень. Для досягнення цієї мети, в розділі послідовно описано ключові етапи розробки:

- проєктування концептуальної моделі та архітектури системи;
- опис методики підготовки та інженерії ознак;
- обгрунтування вибору;
- проєктування моделі прогнозування.

2.1 Визначення існуючих методів автоматизованого прогнозування та пріоритизації завдань та їхні недоліки

2.1.1 Огляд існуючих підходів до оцінки завдань та їхні недоліки

Точне прогнозування часу виконання завдань є фундаментальною проблемою в управлінні проєктами, вирішення якої безпосередньо впливає на дотримання термінів та бюджету. В процесі еволюції методологій розробки було запропоновано низку підходів до оцінки, які можна умовно поділити на дві великі групи:

- спираються на людську експертизу;
- намагаються використовувати історичні дані для побудови прогнозів.

У цьому підрозділі проведено їх аналіз з метою виявлення обмежень, які стимулюють пошук більш досконалих рішень.

Ручні методи оцінки (Effort) є найчастіше використовуваними в сучасних Agile-командах. Замість оцінки в годинах, команди використовують абстрактні

одиниці складності, Story Points або відносні розміри. Ці оцінки грунтуються на колективному обговоренні та інтуїтивному розумінні завдання членами команди, що враховує не лише обсяг роботи, а й складність, ризики та невизначеність. Ключовим поняттям тут є Effort (Зусилля) – це комплексна експертна оцінка, що відображає загальні витрати ресурсів на виконання завдання. Цей метод зображено на рисунку 2.1.

Рис. 2.1 Вплив людського фактора на ручні методи оцінки

Проте, незважаючи на широке розповсюдження, цей підхід має декілька недоліків. По-перше, він характеризується високою суб'єктивністю. Оцінка залежить від особистого досвіду розробника (junior та senior оцінять одне й те ж завдання по-різному), його поточного настрою, рівня втоми та навіть особистого оптимізму чи песимізму. По-друге, результати такої оцінки залежали від рівня втоми команди, тому така сама команда може оцінити однакове завдання по-різному на початку та в кінці робочого тижня. Через це прогнози, побудовані на таких оцінках вони є нестабільними та не завжди надійними, особливо коли проєкт довгостроковий.

Щоб позбутися суб'єктивності експертних оцінок з’явилися статистичні підходи, які базуються на аналізі історичних даних. Найпростіші з них полягають у розрахунку середнього часу виконання для певного типу завдань. Більш складні методи, наприклад, лінійна регресія на нетекстових даних, намагаються знайти математичну залежність між часом виконання та набором формальних характеристик завдання, таких як його тип (Bug, Feature), пріоритет (Low, Medium, High), кількість підзавдань тощо.

Рис. 2.2 Статистичний підхід до прогнозування часу виконання завдань та його ключовий недолік

На перший погляд, такий підхід виглядає значно більш науковим та об'єктивним, однак у ході аналізу було виявлено декілька обмежень. По-перше, це потрібна величезна кількість обсягу даних. Для навчання надійної моделі необхідна велика кількість завершених завдань з правильно заповненими полями, а в нових проєктах цих даних немає. По-друге, такі моделі повністю ігнорують зміст

завдання, описаний природною мовою. Модель не бачить різниці між завданням "виправити помилку на головній сторінці" та "виправити витік пам'яті в ядрі системи", якщо обидва мають однакову мітку завдання "Bug" та пріоритет "High". Цю проблему схематично зображено на рисунку 2.2. Це призводить до низької точності для нетипових задач і робить модель нездатною адекватно реагувати на унікальні особливості кожного нового завдання.

Використання генеративного ШІ (великих мовних моделей). Коли з’явилися великі мовні моделі (LLM), типу GPT-3, виникла потреба використовувати їх для розв'язання практично будь-яких інтелектуальних завдань, та для прогнозування термінів розробки. Процес виглядає просто, користувач надає моделі текстовий опис завдання і ставить пряме запитання "Скільки часу займе виконання цього завдання для досвідченого розробника?".

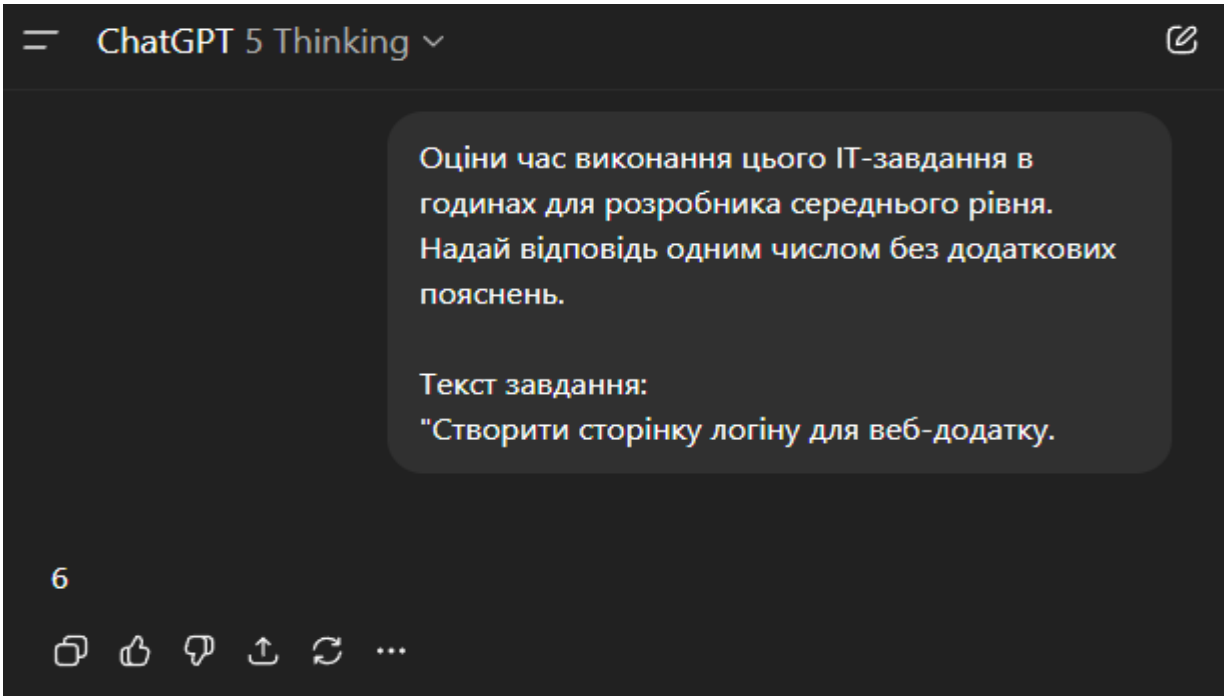


Рис. 2.3 Приклад взаємодії з LLM для отримання оцінки часу виконання завдання

На рисунку 2.3 наведено приклад, де користувач надає детальний опис типового завдання (створення сторінки логіну) і отримує у відповідь конкретний числовий прогноз. Цей приклад наочно ілюструє, як модель, спираючись на свої узагальнені знання з Інтернету, видає правдоподібну, але відірвану від реальності

оцінку. На перший погляд, такий підхід є привабливим. По-перше, модель здатна розуміти семантику тексту, тобто вловлювати суть завдання з його опису. По-друге це доступно і може використовувати будь хто. Та не потрібно розгортати складну інфраструктуру.

Однак, після аналізу зроблено висновок, що такий метод є непридатним для професійного управління проєктами через такий недолік:

- повна відсутність проєктного контексту.

Модель не знає нічого про архітектуру проєкту, наявний технічний борг, специфічні технології, які використовуються, а головне про реальну продуктивність та навички вашої команди. Її відповідь - це не прогноз, а, по суті, "галюцинація", заснована на усереднених знаннях з мільярдів текстів в Інтернеті. Вона може оцінити "створення форми логіну" в 6 годин, не знаючи, що у вашому проєкті це вимагає роботи з застарілою бібліотекою і займе два дні.

Це ручний, неінтегрований процес. "Автоматизація" тут умовна. Кожен раз потрібно вручну копіювати опис завдання, вставляти його в інтерфейс чату, формулювати запит і переносити відповідь назад. Це не є системою, інтегрованою в робочий процес (наприклад, в Jira, чи Trello). Модель не навчається на даних вашого проєкту і не стає точнішою з часом.

Непередбачуваність та невідтворюваність результатів. Відповіді генеративного ШІ можуть різнитися навіть при повторному запиті. Немає жодної гарантії стабільності, а головне - неможливо перевірити, на чому базується оцінка. Це "чорна скринька", результатам якої не можна довіряти при фінансовому та часовому плануванні. На мою думку, використання таких моделей для оцінки є вкрай ризикованим і може призвести до хибних управлінських рішень.

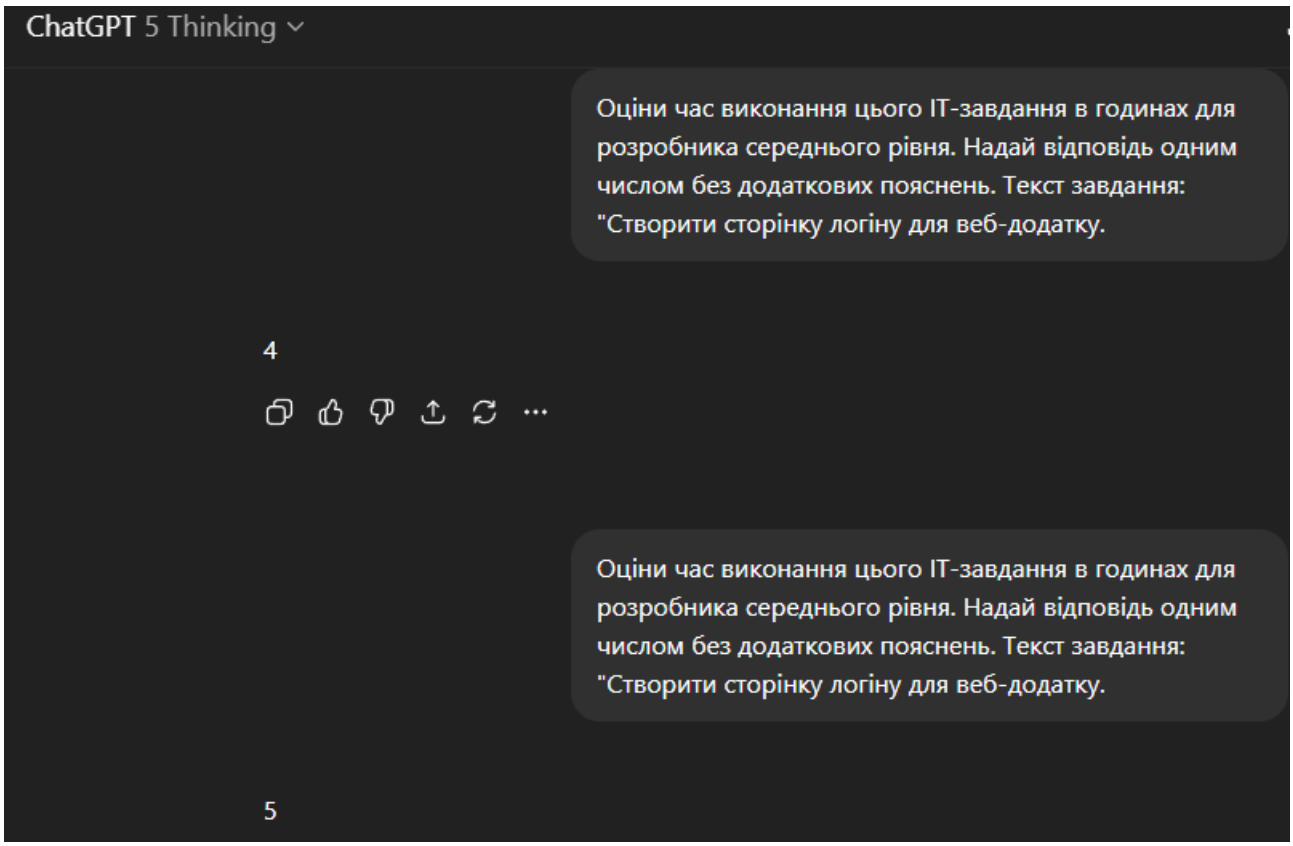


Рис. 2.4 Ілюстрація стохастичної природи LLM при оцінці завдань

На рисунку 2.4 наведено експеримент, в ході якого було двічі надіслано ідентичний запит з однаковим описом завдання. Як видно з результатів, у першому випадку модель надала оцінку "4 години", а в другому - "5 годин". Цей простий тест наочно демонструє стохастичну (випадкову) природу великих мовних моделей. Для цілей планування такий інструмент є непридатним, оскільки його прогнозам не можна довіряти, відсутня гарантія, що завтра на той самий запит не буде отримано кардинально іншу відповідь.

Узагальнюючи проведений аналіз експертних, статистичних підходів та використання генеративних мовних моделей, можна побачити, що кожен із них розв’язує проблему прогнозування часу виконання завдань лише частково. Одні методи забезпечують гнучкість і врахування контексту, але залишаються повністю ручними, інші дають формально коректні числові оцінки, проте ігнорують зміст завдання або демонструють нестабільність результатів. Для наочного зіставлення сильних і слабких сторін цих підходів було сформовано порівняльну таблицю, узагальнений вигляд якої наведено в таблиці 2.1.

Таблиця 2.1

Порівняння існуючих підходів до прогнозування часу виконання завдань

Підхід	**Короткий опис і вхідні дані**	**Переваги**	**Недоліки**	**Чому недостатньо для нашої мети**
Експертна оцінка (Effort, Story Points/Planning Poker)	Колективна/індивідуальна оцінка у балах, без прямої прив’язки до годин/днів	Швидко; враховує неявний досвід команди	Суб’єктивність, низька відтворюваність; залежить від складу/втоми; важко масштабувати	Не дає стабільної, відтворюваної числової оцінки в днях; не автоматизується
Статистичний метод (лінійна регресія на простих ознаках)	Прогноз з використанням історичних числових полів	Простий, інтерпретований; працює на малих обсягах	Ігнорує зміст тексту; лінійність зв’язків; потребує чистих історичних даних	Недостатня точність для нетипових задач; не розуміє опис завдання
LLM (ChatGPT)	Відповідь за текстом опису в чаті	Розуміє природну мову; швидкий старт	Відсутність проєктного контексту; невідтворюваність;	Непридатний для продукційного планування; довідковий інструмент

2.1.2 Аналіз компонентів для побудови спеціалізованих ML-систем прогнозування

Проведений в попередньому підрозділі аналіз показав, що існуючі готові рішення для оцінки завдань є або занадто суб'єктивними, або нездатними враховувати ключовий контекст завдання. Це свідчить про те, що для досягнення високої точності прогнозування необхідний більш гнучкий та спеціалізований підхід. Тому наступним логічним кроком мого дослідження є аналіз не цілісних систем, а окремих компонентів та алгоритмів машинного навчання, з яких, наче з конструктора, можна побудувати власну, більш досконалу методику.

Будь-яка ML-система для аналізу тексту складається з кількох послідовних етапів, що утворюють так званий "конвеєр обробки даних" (pipeline). У цьому підрозділі буде розглянуто ключові алгоритми для кожного з цих етапів:

- векторизація текстових даних;
- кластеризація;
- побудови регресійної моделі для прогнозування.

Існуючі алгоритми розв’язання задачі прогнозування часу виконання завдань, їх переваги та недоліки у вигляді порівняльної таблиці подано в таблиці 2.2.

Таблиця 2.2

Існуючі алгоритми вирішення задачі, їх переваги та недоліки

Алгоритм	**Переваги**	**Недоліки**
TF-IDF	Простий у використанні, швидкий, легко виділяє ключові слова.	Не враховує семантичний контекст, що може призводити до неточностей при аналізі неоднозначних описів завдань.

Продовження таблиці 2.2

Існуючі алгоритми вирішення задачі, їх переваги та недоліки

Алгоритм	**Переваги**	**Недоліки**
K-Means	Простий і масштабований, швидко працює з великими наборами даних, допомагає автоматизувати розподіл завдань між членами команди.	Вимагає заздалегідь визначити кількість кластерів, чутливий до початкової ініціалізації центрів, що може впливати на стабільність результатів.
Random Forest Regressor	Висока точність, стійкість до перенавчання, працює з великою кількістю ознак, справляється з нелінійними залежностями.	Високі обчислювальні витрати, складність у налаштуванні параметрів, менш інтуїтивна інтерпретація результатів.
Support Vector Regression (SVR)	Добре працює з високовимірними даними, може моделювати нелінійні залежності за допомогою ядрових трюків, забезпечує високу точність для невеликих наборів даних.	Обчислювально інтенсивний для великих наборів даних, потребує ретельного налаштування параметрів, чутливий до вибору ядра.
Gradient Boosting (XGBoost)	Висока точність прогнозування, здатність враховувати складні взаємозв’язки між ознаками, автоматичне оброблення пропущених значень, хороша гнучкість у налаштуванні.	Вимагає ретельного налаштування параметрів, може бути чутливим до переобучення, тривалий час тренування на великих наборах даних.

Першим і фундаментальним етапом при роботі з текстовою інформацією є її перетворення у числовий формат, який можуть «зрозуміти» алгоритми машинного навчання. Цей процес називається векторизацією. Одним з найпоширеніших та класичних підходів до векторизації є метод TF-IDF.

TF-IDF - це статистичний метод, що дозволяє оцінити важливість слова в контексті документа, який є частиною колекції документів. Його основна ідея - надати вищу вагу тим словам, які часто зустрічаються в конкретному документі (описі завдання), але рідко зустрічаються в усіх інших документах. Метод складається з двох компонентів:

- TF (Term Frequency - частота слова) яка показує, наскільки часто слово зустрічається в конкретному документі.
- IDF (Inverse Document Frequency- обернена частота слова) яка зменшує вагу слів, які є загальновживаними і часто зустрічаються у всьому корпусі (наприклад, сполучники «і», «але», «в» або загальні терміни «завдання», «система»).

Головною перевагою TF-IDF є його простота у використанні та висока швидкість обчислень. Він не потребує складних налаштувань чи великих обчислювальних ресурсів, він є чудовим вибором для швидкого прототипування та базових моделей. Алгоритм легко виділяє ключові слова, які найкраще характеризують документ, і є стандартною реалізацією в більшості бібліотек для машинного навчання.

Але, TF-IDF має один суттєвий недолік, він не враховує семантичний контекст та синоніми. Для цього алгоритму слова «виправити помилку» та «усунути баг» є абсолютно різними, хоча для людини вони несуть однаковий сенс. Він не розуміє зв’язків між словами та їхнього значення в реченні. Це призводить до втрати важливої семантичної інформації та може спричинити неточності при аналізі завдань з неоднозначними або синонімічними описами, що є критичним для точного прогнозування їх складності.

Після того, як текстові описи завдань перетворено на числові вектори за допомогою TF-IDF, наступним кроком є виявлення структури та закономірностей у цих даних. Одним із корисних підходів є кластеризація - процес автоматичного групування схожих об’єктів. В контексті цього дослідження, це дозволяє автоматично розділити всі завдання на групи за схожістю їхнього змісту, що може опосередковано вказувати на їхній тип або рівень складності (наприклад, кластери

«прості завдання з інтерфейсом», «складні завдання з базою даних», «завдання з рефакторингу» тощо). Отриманий номер кластера для кожного завдання може потім слугувати новою, важливою ознакою для фінальної моделі прогнозування.

Одним з найпопулярніших алгоритмів для вирішення цієї задачі є K-Means (метод k-середніх).

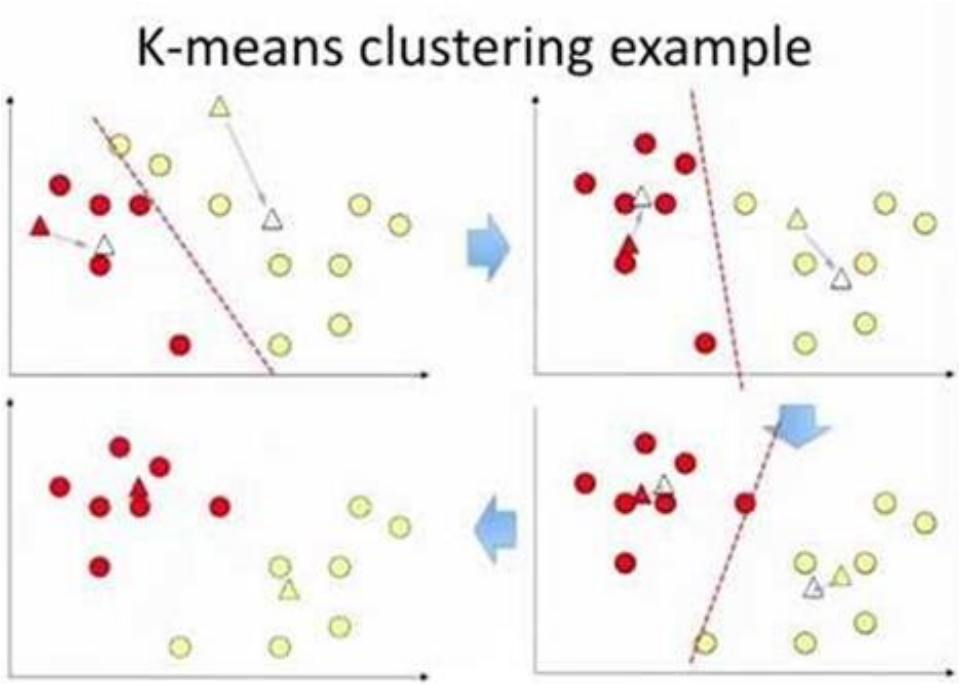


Рис. 2.5 Приклад кластеризації K-means

K-Means - це ітераційний алгоритм некерованого навчання, який розділяє набір даних на *K* заздалегідь визначених, непересічних кластерів. Алгоритм працює наступним чином, спочатку випадково обираються *K* точок, які стають початковими центрами кластерів (центроїдами). Потім кожен об’єкт (вектор завдання) присвоюється до найближчого центроїда. Після цього центроїди перераховуються як середнє арифметичне всіх об’єктів у кластері. Цей процес повторюється поки центроїди не перестануть змінювати своє положення.

K-Means є досить популярним через свою простоту в реалізації та інтерпретації. Він є швидким і добре масштабується для роботи з великими наборами даних. В даній роботі, він допомагає автоматизувати процес виявлення груп схожих завдань, для подальшого аналізу та розподілу роботи.

Але цей алгоритм має два суттєвих обмеження. По-перше, він вимагає заздалегідь визначати кількість кластерів (K). Вибір неправильного значення K може призвести до неякісного поділу даних. Для визначення оптимального K часто доводиться використовувати додаткові методи. По-друге, випадковий вибір початкових центроїдів може призвести до збіжності локального мінімуму і як наслідок, до нестабільних та неоптимальних результатів кластеризації. Хоча ця проблема частково вирішується за допомогою більш просунутих методів ініціалізації, вона залишається властивістю базового алгоритму.

Завершальним і найважливішим етапом у побудові комплексного метода є вибір та застосування регресійної моделі, яка візьме на вхід підготовлені числові ознаки (TF-IDF вектори, номер кластера з K-Means, а також інші формальні характеристики завдання) і видасть фінальний прогноз - очікуваний час виконання в годинах. Існує велика кількість регресійних алгоритмів, і в рамках цього дослідження буде доцільним розглянути кілька з них, що відрізняються за складністю та принципом роботи.

Random Forest Regressor. Це один з найпотужніших та найпопулярніших алгоритмів для задач регресії. Random Forest є ансамблевим методом, що базується на ідеї «колективного розуму», замість того, щоб покладатися на одну модель, він будує велику кількість (ансамбль) незалежних дерев рішень під час навчання і усереднює їхні прогнози для отримання фінального результату. Кожне дерево навчається на випадковій підвибірці даних та випадковому наборі ознак, що робить їх різноманітними і зменшує загальну помилку ансамблю.

Головною перевагою Random Forest є його висока точність та узагальнююча здатність. Крім того, алгоритм демонструє високу стійкість до перенавчання завдяки механізму усереднення, що є критично важливим при роботі зі складними даними. Він чудово працює з великою кількістю ознак, що є особливо важливим у цьому дослідженні, оскільки TF-IDF вектори можуть мати тисячі вимірів. Алгоритм може автоматично визначати важливість кожної ознаки.

Але, швидкість алгоритму має деякі недоліки. По-перше, це високі обчислювальні витрати як під час навчання, так і під час прогнозування, оскільки

необхідно обробити сотні окремих дерев. По-друге, Random Forest вимагає складності у налаштуванні гіперпараметрів (кількість дерев в лісі n_estimators, максимальна глибина кожного дерева max_depth та ін.) для досягнення оптимальної продуктивності. По-третє, однією з головних проблем є менш інтуїтивна інтерпретація результатів. На відміну від простої лінійної регресії чи одного дерева рішень, зрозуміти, чому ансамбль із 500 дерев ухвалив те чи інше рішення, практично неможливо, що робить модель «чорною скринькою».

Support Vector Regression (SVR - Регресія Опорних Векторів). Також розглянемо альтернативний підхід, представлений методом регресії опорних векторів. SVR є регресійним подібним до популярного класифікаційного алгоритму Support Vector Machine (SVM). На відміну від класичних регресійних методів, які намагаються мінімізувати помилку для всіх точок даних, SVR працює за іншим принципом. Він намагається знайти гіперплощину (лінію регресії), яка б проходила найближче до максимальної кількості точок, дозволяючи певний рівень похибки.

Основна ідея SVR полягає у визначенні «коридору» або «трубки» (з шириною, що визначається гіперпараметром ε) навколо лінії регресії. Модель не штрафує точки, що потрапляють всередину цього коридору, і намагається мінімізувати помилку лише для точок, які опинилися за його межами. Ті точки даних, які лежать на межі коридору або за нею, називаються опорними векторами, оскільки саме вони визначають положення фінальної гіперплощини. Геометрично роботу моделі SVR зручно інтерпретувати як пошук лінії регресії з ε-коридором навколо неї, у межах якого помилки не штрафуються.

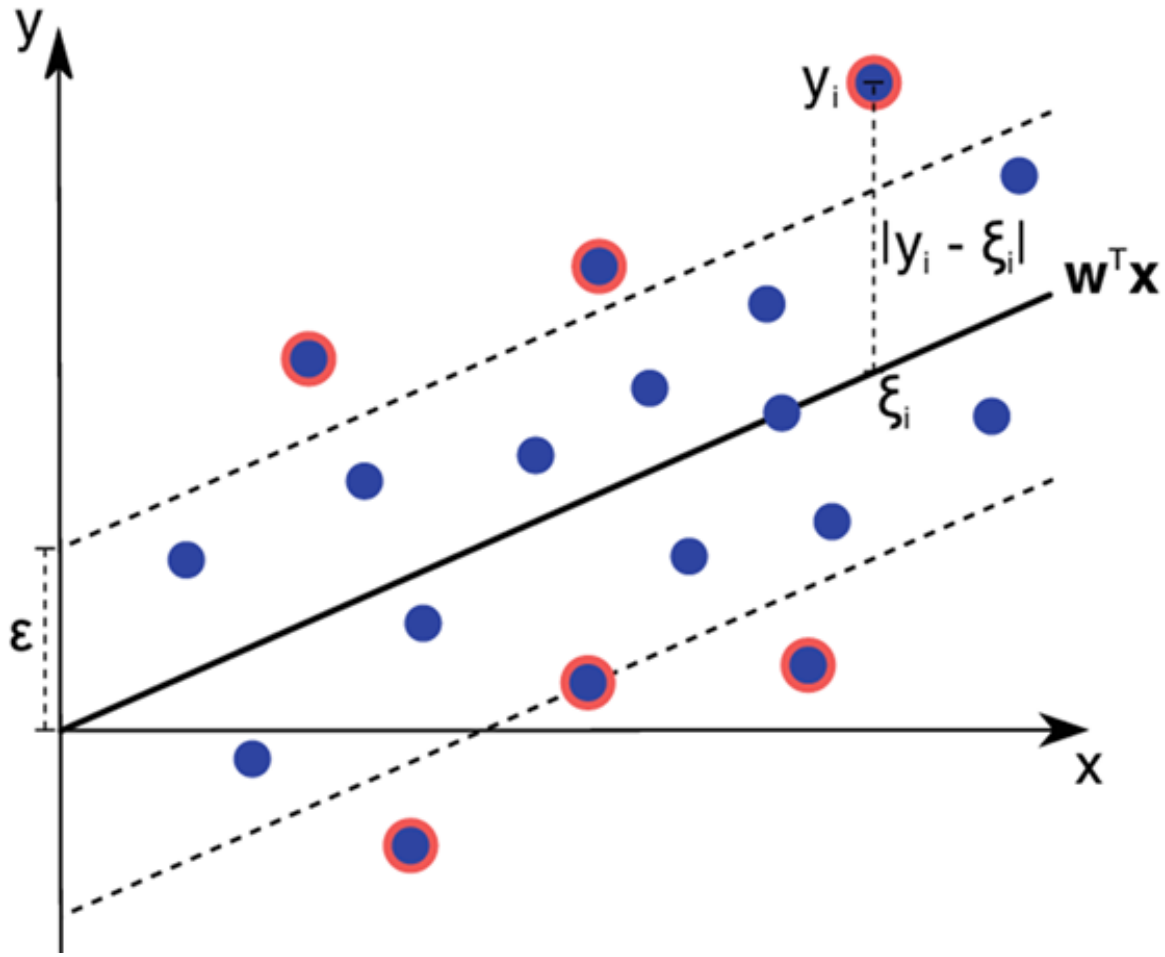


Рис. 2.6 Геометрична інтерпретація регресії опорних векторів (SVR) з ε-коридором та опорними векторами

Як видно з рисунку 2.6, тільки точки, що виходять за межі ε-коридору, стають опорними векторами і безпосередньо впливають на положення гіперплощини регресії.

Однією з ключових переваг SVR є його швидкість при роботі з високовимірними даними, що робить його кандидатом для аналізу векторів, отриманих після TF-IDF. Завдяки використанню так званих «ядрових трюків» (kernel tricks), SVR може швидко моделювати складні нелінійні залежності між ознаками та цільовою змінною, навіть не переходячи явно у простір вищої розмірності. Це дозволяє досягати високої точності навіть на відносно невеликих наборах даних.

Проте, використання SVR також пов’язане з певними перевагами та недоліками. Алгоритм є обчислювально інтенсивним для великих наборів даних,

оскільки його складність може зростати квадратично зі збільшенням кількості зразків. Крім того, SVR потребує ретельного налаштування параметрів для досягнення хорошої продуктивності. Вибір правильного типу ядра (лінійного, поліноміального, RBF), а також налаштування гіперпараметрів, таких як коефіцієнт регуляризації C та ширина коридору ε, є нетривіальною задачею і може суттєво впливати на кінцевий результат. Неправильний вибір ядра або параметрів призводить до перенавчання або, навпаки, до надто спрощеної моделі, нездатної розуміти складні залежності в даних.

Завершуючи огляд регресійних моделей, необхідно проаналізувати Gradient Boosting (XGBoost) – це ще один ансамблевий підхід, який демонструє найкращі результати в порівнянні з іншими моделями машинного навчання. Як і Random Forest, він використовує дерева рішень, але його принцип роботи інший. Замість того, щоб будувати дерева паралельно і незалежно, градієнтний бустинг будує їх послідовно.

Основна ідея полягає в тому, що кожна наступна модель в ансамблі виправляє помилки попередньої. Перше дерево робить початковий, зазвичай неточний, прогноз. Друге дерево навчається не на самих даних, а на помилках першого дерева. Третє дерево навчається на помилках, що залишилися після роботи перших двох, і так далі. Як показано на рисунку 2.7, кожне наступне дерево навчається на помилках попередніх, і намагається виправити помилки попередньої.

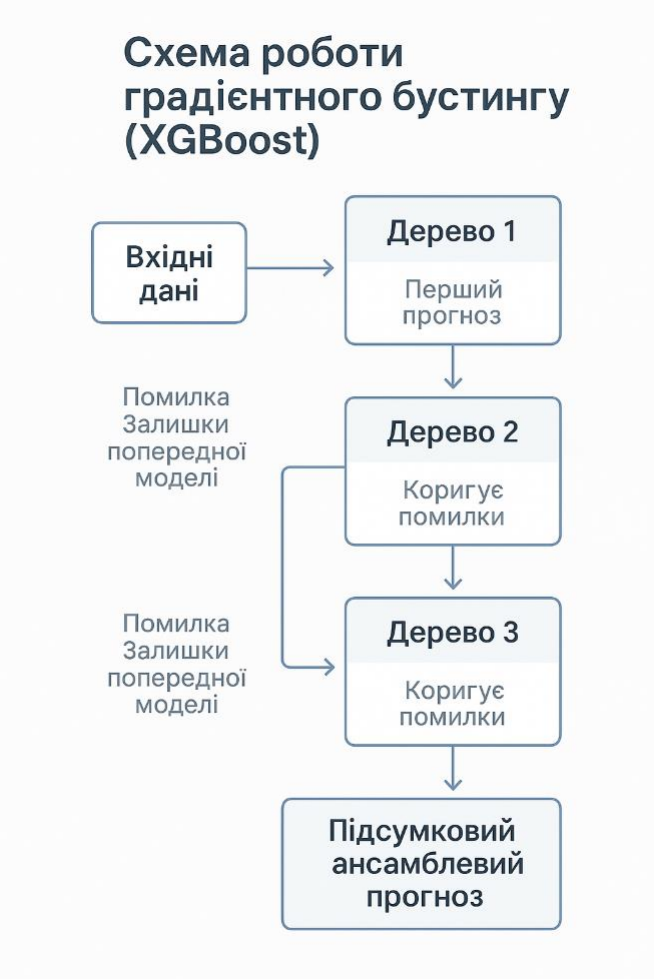


Рис. 2.7 Схема роботи градієнтного бустингу XGBoost

Цей ітераційний процес допомагає ансамблю поступово, крок за кроком, наближатися до оптимального прогнозу, приділяючи увагу найскладнішим для моделювання випадкам. XGBoost є одним з найкращих реалізацій оптимізованого для швидкості та точності.

Головною перевагою XGBoost є його дуже висока точність прогнозування, яка часто краща ніж інші алгоритми. Завдяки послідовному навчанню, модель може враховувати складні нелінійні взаємозв’язки між ознаками, які можуть бути пропущені іншими методами. Крім цього, реалізація XGBoost містить в собі вбудовані механізми регуляризації для боротьби з перенавчанням та оптимізовані алгоритми для паралельної обробки, через це він є швидким порівняно з класичними реалізаціями градієнтного бустингу.

Але, така висока швидкість вимагає значної відповідальності при використанні. По-перше, XGBoost вимагає ретельного налаштування параметрів. В порівнянні від Random Forest, який може давати хороші результати «з коробки», XGBoost має велику кількість гіперпараметрів (швидкість навчання learning_rate, кількість дерев n_estimators, глибина дерев max_depth та ін.), неправильний вибір яких може кардинально погіршити результат. По-друге, алгоритм є чутливим до перенавчання, особливо якщо використовувати занадто багато дерев або високу швидкість навчання без належної регуляризації. Нарешті, послідовна природа алгоритму означає, що час тренування на великих наборах даних може бути значним, оскільки кожне наступне дерево не може бути побудоване, доки не завершиться робота попереднього.

2.2 Розробка концептуальної моделі системи автоматизованого прогнозування

Першим кроком у розробці є створення її концептуальної моделі. Цей етап є невід’ємною частиною інженерії програмного забезпечення, який дозволяє розбити загальну задачу на менші, керовані компоненти та визначити високорівневу архітектуру майбутнього рішення. Сучасні системи на базі розвинутого машинного навчання мають можливість виконувати такі завдання прогнозування [3]. Розробка такої моделі забезпечує чітке розуміння взаємозв’язків між окремими функціональними блоками та потоками даних, що є основою для подальшої деталізації методики та її програмної реалізації. У даному підрозділі було описано запропоновану концептуальну модель, починаючи з визначення її ключових модулів.

2.2.1 Визначення основних функціональних модулів системи

Для реалізації поставленої задачі, було обрано мікросервісну архітектуру, яка є сучасним стандартом для побудови гнучких та масштабованих систем. Такий підхід дозволяє чітко розділити відповідальність. Основна система управління

завданнями, написана на PHP, займається бізнес-логікою та взаємодією з користувачем, тоді як вся складна логіка машинного навчання винесена в окремий, незалежний сервіс, реалізований на Python.

Ця архітектура складається з трьох ключових компонентів, взаємодію яких показано на рисунку 2.8.

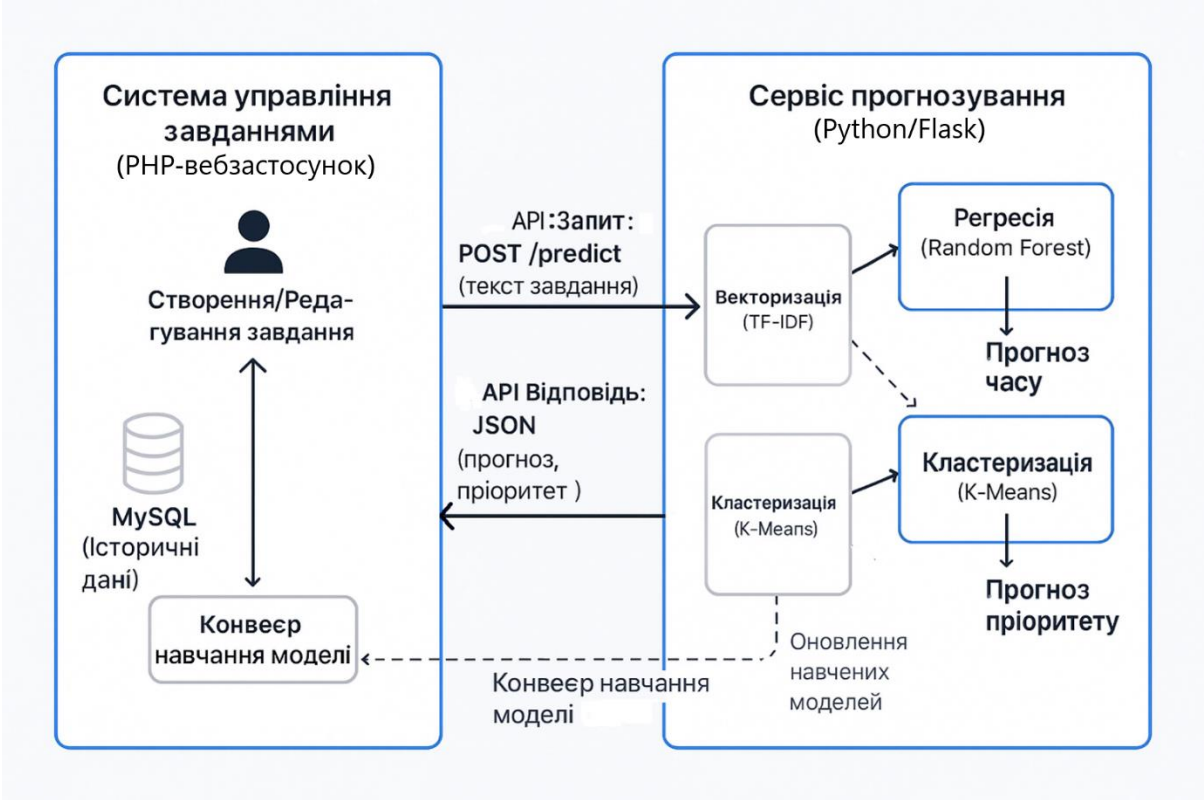


Рис. 2.8 Архітектура взаємодії системи управління завданнями та сервісу прогнозування

Розглянемо компоненти архітектури детальніше. Основна система управління завданнями (веб-додаток). Це ядро системи, з яким безпосередньо взаємодіє користувач. Її функціонал, реалізований на PHP та MySQL, включає створення та редагування дошок, секцій та завдань. Та має вдосконалення системи з урахуванням змін в бізнес-середовищі та технологічних можливостей [6]. В контексті даної роботи, цей компонент виконує дві основні ролі:

- джерело даних;
- клієнт.

Зберігає в базі даних MySQL всю інформацію про завдання, включаючи їх текстовий опис та, після виконання, фактичний час. Ці історичні дані використовуються для періодичного навчання моделей.

У момент створення або редагування завдання, PHP-додаток формує API-запит до сервісу прогнозування, передаючи йому текстовий опис.

Сервіс прогнозування (Python/Flask Microservice). Це автономний сервіс, що реалізує всю логіку машинного навчання. Він не зберігає дані про завдання, а лише виконує обчислення прогнозування часу та пріоритетів завдань. Застосовує різні алгоритми класифікації та модулі машинного навчання за допомогою бібліотеки Scikit-Learn [7]. Сервіс приймає HTTP-запити через API і складається з наступних внутрішніх модулів:

- модуль векторизації (TF-IDF) який приймає текстовий опис завдання із запиту та перетворює його на числовий вектор;
- модуль регресійного аналізу (Random Forest) який використовує навчену модель RandomForestRegressor для прогнозування очікуваного часу виконання на основі вхідного вектора;
- модуль кластеризації (K-Means) який паралельно використовує навчену модель Kmeans для визначення кластера складності/пріоритетності завдання;
- арі-контролер який формує та повертає відповідь у форматі JSON, що містить прогнозований час та рівень пріоритету.

Кластеризація - це завдання групування подібних об’єктів у групи, які називаються кластерами [17].

Конвеєр навчання моделі (Offline Training Pipeline). На основі зібраних даних з різних комерційних баз даних, моделі будуть навчені, протестовані та валідовані з високою точністю [4]. Цей компонент виконує періодичне оновлення моделей машинного навчання. Він працює в офлайн-режимі та виконує такі дії:

- виконує запит до бази даних MySQL основної системи для отримання історичних даних про виконані завдання;

- проводить повний цикл навчання моделей (TF-IDF, K-Means, Random Forest) на цих даних;
- зберігає оновлені, навчені моделі, які потім завантажуються та використовуються сервісом прогнозування. (У прототипі цей етап виконується один раз при запуску сервісу, але в повноцінній системі він буде періодичним).

Така архітектура є досить гнучкою, бо дозволяє незалежно оновлювати та масштабувати систему, а також легко замінювати моделі машинного навчання без втручання в основну логіку системи управління завданнями.

Структуру варіантів використання веб-застосунку зображено на рисунку 2.9.

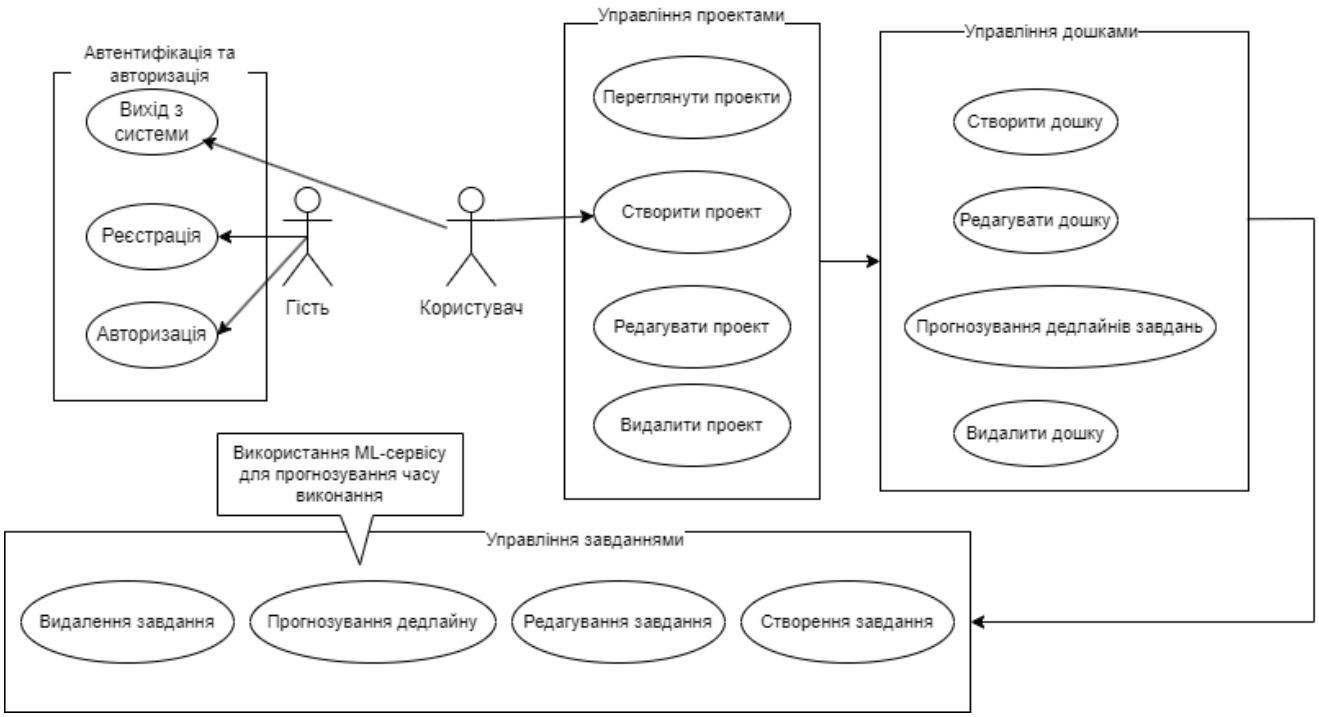


Рис. 2.9 Діаграма Use-Case веб-застосунку системи управління завданнями

На діаграмі відображено акторів Гість і Користувач, блоки прецедентів автентифікації/авторизації, управління проєктами, управління дошками та управління завданнями. Окремим елементом показано використання ML-сервісу прогнозування при створенні/редагуванні завдання та при розрахунку дедлайну.

2.2.2 Побудова схеми конвеєра обробки даних (data pipeline)

Після визначення основних функціональних модулів системи, наступним кроком є деталізація послідовності операцій, які виконуються над даними. Це конвеєр обробки даних (data pipeline), який описує повний шлях даних від їх отримання до генерації фінального прогнозу. Існує три основні кроки автоматичної обробки тексту, підготовка тексту, векторизація та остаточна обробка, така як класифікація [10].

В контексті розробленої системи необхідно розрізняти два основних конвеєри, які виконуються на різних етапах життєвого циклу системи:

- конвеєр навчання моделі (офлайн-процес);
- конвеєр прогнозування (онлайн-процес).

Загальну схему цих процесів представлено на рисунку 2.10.

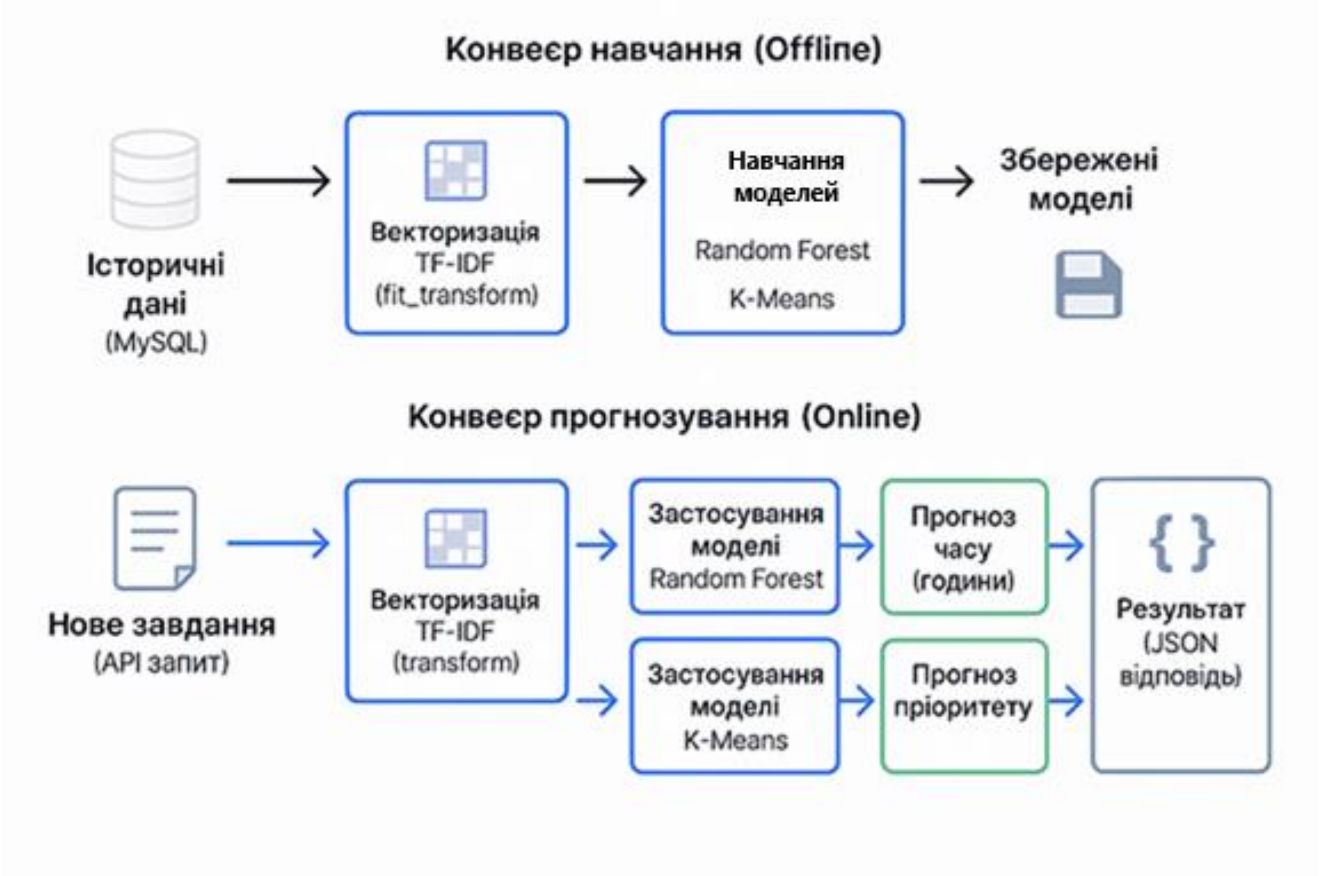


Рис. 2.10 Схема конвеєрів навчання та прогнозування

Конвеєр навчання (Training Pipeline)

Цей процес виконується періодично для створення та оновлення моделей машинного навчання на основі накопичених історичних даних. Він складається з наступних кроків:

1. збір даних шляхом вивантаження з бази даних MySQL всі виконані завдання, що містять текстовий опис та фактичний час виконання;
2. весь корпус текстових описів використовується для навчання векторизатора TF-IDF, який створює словник та перетворює тексти на числові вектори;
3. дані розділяються на тренувальний та тестовий набори для подальшої валідації якості моделі;
4. на тренувальних даних паралельно навчаються дві моделі:
   - регресійна модель RandomForestRegressor для прогнозування часу;
   - кластеризаційна модель Kmeans для визначення груп пріоритетності/складності;
5. навчені об’єкти векторизатора та моделей серіалізуються та зберігаються для подальшого використання сервісом прогнозування.

Конвеєр прогнозування (Prediction Pipeline). Цей процес активується щоразу, коли система отримує API-запит на оцінку нового завдання. Він є значно швидшим, оскільки використовує вже навчені моделі:

1. отримання запиту від користувача, який приймає текстовий опис нового завдання;
2. текст нового завдання перетворюється на числовий вектор за допомогою вже навченого векторизатора TF-IDF(це важливо, оскільки забезпечує використання того самого словника, що й при навчанні);
3. отриманий вектор паралельно подається на вхід двом навченим моделям:
   - застосовується RandomForestRegressor який видає прогноз часу виконання;
   - виконується кластеризація Kmeans, яка визначає, до якого кластера належить завдання, видаючи прогноз пріоритету;

4. отримані прогнози об’єднуються та повертаються основному додатку у вигляді JSON-відповіді.

2.3 Методика попередньої обробки даних та векторизації описів завдань з використанням TF-IDF

Після визначення високорівневої архітектури системи, наступним кроком є розробка конкретної методики перетворення необроблених вхідних даних на інформативні числові ознаки, придатні для подачі в модель машинного навчання. Цей процес є дуже важливим, оскільки якість вхідних даних безпосередньо визначає якість фінального прогнозу.

Запропонована методика складається з двох послідовних етапів:

1. попередня обробка тексту (його очищення і нормалізація);
2. векторизація (для перетворення очищеного тексту на числа).

2.3.1 Алгоритм попередньої обробки текстових даних

Текстові описи завдань, отримані з систем управління, містять велику кількість «шуму», знаки пунктуації, стоп-слова, слова в різних відмінках та регістрах. Метою попередньої обробки є усунення цього шуму та приведення тексту до стандартизованого вигляду. Для цього було розроблено алгоритм, що складається з наступних кроків:

1. всі символи тексту переводяться в нижній регістр, щоб уникнути розрізнення слів, написаних з великої та малої літери (наприклад, «Сервер» та «сервер» будуть вважатися одним і тим же словом);
2. з тексту видаляються всі символи, що не є літерами або цифрами, оскільки вони не несуть семантичного навантаження для моделі;
3. суцільний текст розбивається на окремі елементи - токени. Токенізація замінює опис завдання великою кількістю навчальних ознак (по одній на слово) [15];

4. зі списку токенів видаляються стоп-слова або найбільш вживані слова, що не мають значущого лексичного сенсу (напр., «і», «в», «на», «але», «як»), для цього використовується заздалегідь визначений словник стоп-слів;
5. використовується стеммінг, де кожне слово приводиться до його базової форми (основи) шляхом відкидання закінчень. Наприклад, слова «прогнозування», «прогнозувати», «прогнозований» будуть приведені до основи «прогноз». Це дозволяє групувати слова зі спільним коренем.

Процес перетворення сирого тексту на нормалізований набір ілюструє схема, наведена на рисунку 2.11.

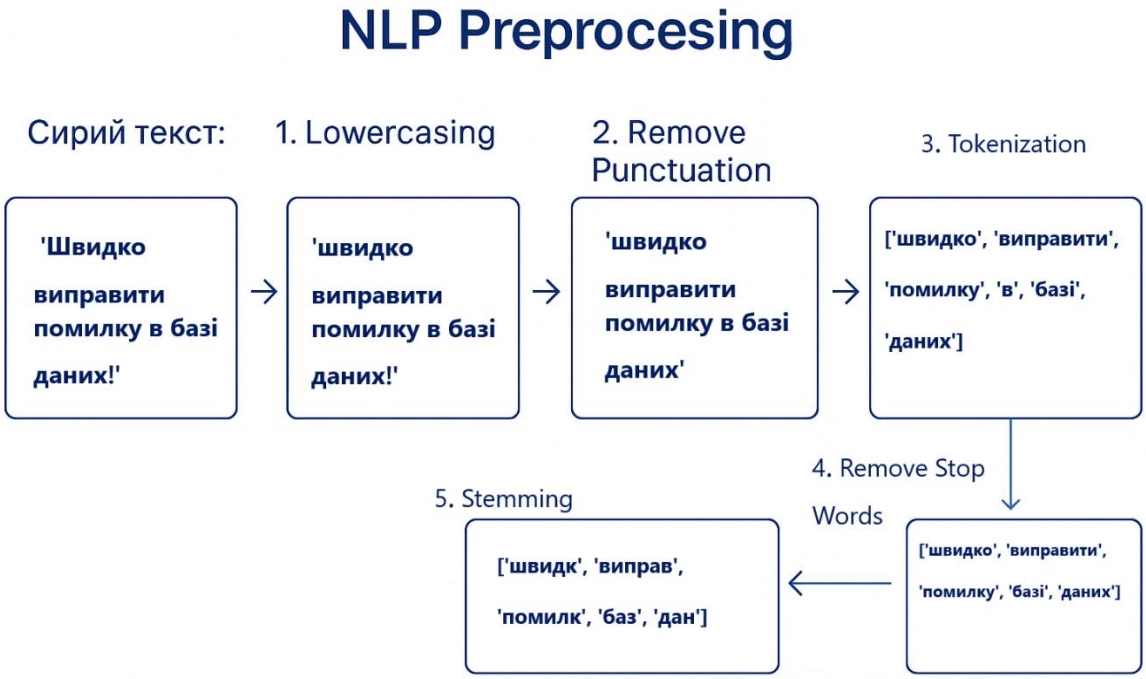


Рис. 2.11 Схема конвеєра попередньої обробки тексту

2.3.2 Векторизація тексту за методом TF-IDF

Після отримання очищеного списку токенів, наступним завданням є перетворення його на числовий вектор. Для реалізації прототипу було обрано метод TF-IDF, що забезпечує хороший баланс між простотою реалізації та швидкістю.

Метод TF-IDF (Term Frequency-Inverse Document Frequency) оцінює важливість кожного слова (терміна) в документі відносно всього корпусу документів. Вага кожного слова розраховується як добуток двох компонент: TF та IDF.

Показник TF (Term Frequency) для терміна t в документі d розраховується за формулою:

$$TF(t,d) = \frac{f(t,d)}{|d|}, \quad (2.1)$$

де f(t, d) - кількість разів, яку термін t з'являється в документі d;

|d| - загальна кількість термінів у документі d.

Велике значення TF - IDF досягається, коли значення TF є великим і коли вага в знаменнику у формулі IDF не має великого значення [8]. Розмір колекції ділиться на кількість відповідей, в яких вживається слово [28]. Таким чином, найбільшу вагу матиме слово, що зустрічається лише в одній відповіді [28].

Показник IDF (Inverse Document Frequency) для терміна t розраховується за формулою:

$$IDF(t,D) = log(\frac{N}{df(t)}), \quad (2.2)$$

де N - загальна кількість документів у корпусі;

df(t) - кількість документів з корпусу, які містять термін t.

Кінцева вага TF-IDF є добутком цих двох величин:

$$TF - IDF(t,d,D) = TF(t,d) \times IDF(t,D) \quad (2.3)$$

Результатом застосування цього методу є розріджена матриця. Приклад перетворення речення на таку матрицю показано в таблиці 2.3.

Таблиця 2.3

Приклад перетворення тексту на TF-IDF вектор

Слово (токен)	f(t,d) - Частота	TF	IDF	TF-IDF (TF*IDF)
виправ	1	1/3=0.33	log(100/10)=1.0	0.33
помилк	1	1/3=0.33	log(100/20)=0.7	0.23
баз	1	1/3=0.33	log(100/5)=1.3	0.43

2.4 Розробка моделі прогнозування часу виконання на основі методу Random Forest Regressor

Після того, як вхідні дані було очищено та перетворено на інформативні числові ознаки, фінальним етапом розробленої методики є побудова регресійної моделі, здатної на основі цих ознак прогнозувати час виконання завдання. Цей підрозділ містить обгрунтування вибору моделі, формування фінального вектора ознак та опису процесу навчання. Метод випадкового лісу - це ансамблевий алгоритм машинного навчання, який використовується для задач класифікації та регресії [29].

Для реалізації прототипу було обрано модель Random Forest Regressor. Цей вибір обґрунтований кількома ключовими перевагами даного алгоритму:

1. Random Forest є ансамблевим методом, який усереднює прогнози великої кількості незалежних дерев рішень, що зменшує дисперсію та отримує більш точний та стабільний результат порівняно з одним деревом;
2. завдяки використанню технік беггінгу та випадкового вибору ознак на кожному вузлі, алгоритм демонструє високу стійкість до перенавчання, що є особливо важливим при роботі зі складними та зашумленими даними;
3. модель досить швидко працює з високовимірними даними, що можуть містити тисячі ознак. Алгоритм здатен автоматично визначати важливість кожної ознаки для полегшення аналізу.

Принцип роботи алгоритму схематично зображено на рисунку 2.12.

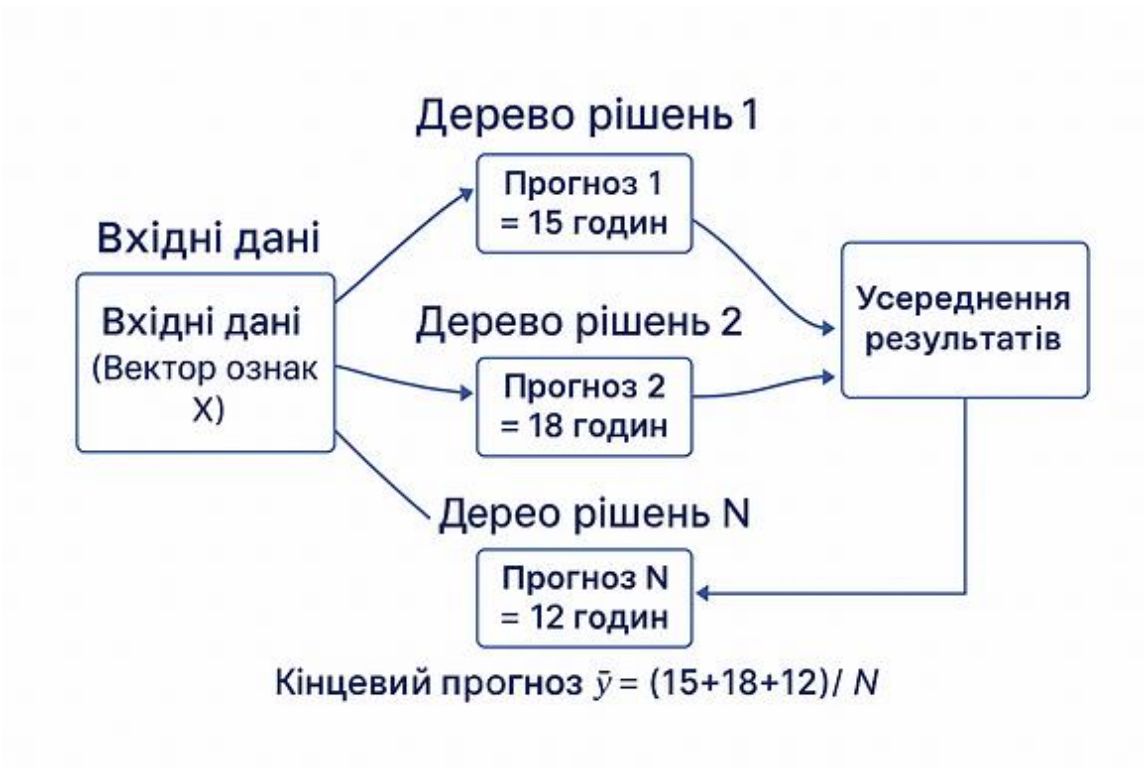


Рис. 2.12 Схема роботи алгоритму Random Forest Regressor

Фінальний прогноз моделі ŷ є усередненим значенням прогнозів N окремих дерев рішень, що входять до її складу. Математично це можна виразити наступною формулою:

$$\hat{y} = \frac{1}{N}\sum_{i=1} \hat{y}_i(X) \quad (2.4)$$

де $\hat{y}$ - фінальний прогноз ансамблю;

N - кількість дерев рішень в ансамблі (гіперпараметр n_estimators);

$\hat{y}_i(X)$ – прогноз i-го дерева рішень для вхідного вектора ознак X.

Для того, щоб модель могла навчатися, кожне завдання необхідно представити у вигляді єдиного числового вектора, що об’єднує всі підготовлені на попередніх етапах ознаки. Цей процес називається конкатенацією ознак. У рамках розробленої методики, фінальний вектор ознак X для одного завдання має наступну структуру:

X = [TF-IDF_vector | K-Means_feature | Categorical_features]

де TF-IDF_vector - це розріджений вектор, отриманий після векторизації текстового опису завдання;

K-Means_feature - це одна числова ознака, що відповідає номеру кластера, до якого

було віднесено завдання [14]; Categorical_features - це набір бінарних ознак, отриманих після One-Hot Encoding формальних атрибутів завдання (наприклад, «Тип: Bug», «Пріоритет: High»).

Алгоритм K-means дозволяє провести кластеризацію підприємств за ключовими характеристиками [20]. Структуру фінального вектора ознак, що подається на вхід моделі, візуалізовано на рисунку 2.13.

Рис. 2.13 Структура фінального вектора ознак

Процес навчання моделі RandomForestRegressor відбувається на підготовленому наборі історичних даних, де вхідними даними X є набір фінальних векторів ознак, а цільовою змінною Y - масив відповідних значень фактичного часу виконання завдань. Машинне навчання забезпечує точне прогнозування, оскільки дозволяє робити передбачення на основі аналізу великих обсягів історичних даних [30]. Після завершення навчання, модель зберігається і готова до використання в конвеєрі прогнозування для оцінки нових завдань. Процес отримання прогнозу для нового завдання полягає у послідовному виконанні всіх кроків з підготовки даних та застосуванні методу прогнозування навченої моделі до отриманого вектора ознак.

3 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ТА АНАЛІЗ ЕФЕКТИВНОСТІ РОЗРОБЛЕНОЇ МЕТОДИКИ

3.1 Програмна реалізація прототипу системи прогнозування

Розглянемо практичну реалізацію розробленої методики та зробимо перевірку працездатності та оцінку ефективності запропонованого підходу на реальних даних. Для досягнення цієї мети, у цьому розділі послідовно оглянемо всі етапи експериментального дослідження, починаючи з опису програмної реалізації прототипу системи та використаних інструментів, проведення експерименту, і завершуючи порівняльним аналізом отриманих результатів. Фінальною задачею є кількісне підтвердження гіпотези про те, що розроблена комплексна методика перевершує базові та традиційні підходи до оцінки завдань.

3.1.1 Опис використаних програмних засобів

Програмна реалізація розробленої методики потребувала вибору відповідного технологічного стеку. Для створення прототипу було використано набір програмних засобів та бібліотек, основні інструменти, що були задіяні в роботі, наведено в таблиці 3.1.

Таблиця 3.1

Програмні засоби, використані для реалізації прототипу

Інструмент / бібліотека	**Призначення**	**Роль у системі**
Python	Високорівнева мова програмування загального призначення	Основна мова реалізації серверної частини та усіх алгоритмів навчання

Продовження таблиці 3.1

Програмні засоби, використані для реалізації прототипу

Інструмент / бібліотека	**Призначення**	**Роль у системі**
Flask	Легкий веб-фреймворк для створення REST API	Побудова HTTP-інтерфейсу для взаємодії клієнта із моделлю прогнозування
Pandas	Робота з табличними даними, попередня обробка та аналіз	Зберігання та обробка датасету завдань, формування вибірок для навчання
Scikit-learn	Набір класичних алгоритмів машинного навчання	Реалізація TF-IDF, K-Means, RandomForestRegressor та конвеєрів навчання
NLTK	Опрацювання природної мови, токенізація, стоп-слова	Попередня обробка текстових описів завдань
ukrainian-stemmer	Стемінг слів українською мовою	Нормалізація українських токенів перед векторизацією TF-IDF

Було обрано стек на основі мови Python. Python є стандартом у сфері машинного навчання та обробки текстів, для нього існує велика кількість бібліотек, які закривають більшість потреб, що виникають під час розробки прототипів інтелектуальних систем. Завдяки зрозумілому синтаксису та великій кількості прикладів і документації, можна зосередитися не стільки на низькорівневій реалізації алгоритмів, скільки на експериментах з моделями та якості їх роботи.

Як основу для побудови HTTP-інтерфейсу використано Flask. Цей фреймворк є легким, не нав’язує складної структури проєкту і дуже добре підходить для невеликих сервісів-API, яким і є прототип системи прогнозування термінів виконання завдань. Через Flask було організовано єдину кінцеву точку /predict, яка приймає текстовий опис завдання, запускає повний конвеєр обробки й

повертає результат у форматі JSON. Це дозволяє у легко інтегрувати модель у будь-яку систему управління завданнями.

Для роботи з даними було обрано Pandas. На практиці опис завдання, фактичний час його виконання, додаткові атрибути (тип, статус тощо) зручно зберігати саме у табличному вигляді. За допомогою Pandas буде сформовано навчальні та тестові вибірки, виконано фільтрацію, об’єднання, перетворення типів даних. Це значно спрощує підготовку даних до подальшого навчання моделей у Scikit-learn.

Ключові алгоритми машинного навчання було реалізувано за допомогою бібліотеки Scikit-learn. Вона містить готові, добре протестовані реалізації TF-IDF векторизатора, алгоритму кластеризації K-Means та регресора RandomForestRegressor. Це дозволило не витрачати час на ручну реалізацію класичних методів, а перейти до побудови загальної методики, покращення класичних методів, налаштування гіперпараметрів і дослідження якості прогнозування після покращення методів. Крім того, Scikit-learn підтримує зручні конвеєри, щоб послідовно поєднати етапи попередньої обробки тексту, векторизації та навчання моделі.

Також було виконано попередню обробку українськомовних описів завдань. Для цього використано поєднання NLTK та саморобного Ukrainian-stemmer. NLTK застосовано для токенізації тексту й видалення стоп-слів, тоді як ukrainian-stemmer відповідає за приведення слів до стемів, тобто спільної основи. Такий підхід дозволяє зменшити розмір словника та зробити TF-IDF вектори більш узагальненими, слова "помилка", "помилки", "помилок" після стемінгу перетворюються на спільну форму, що покращує якість векторизації й підвищує стійкість моделі до варіацій у формулюваннях.

У цілому, обраний стек повністю базується на відкритому програмному забезпеченні з активною спільнотою. Це важливо для роботи, оскільки забезпечує прозорість реалізації, можливість відтворення результатів іншими дослідниками та подальшого розширення прототипу без прив’язки до закритих або комерційних рішень.

3.1.2 Опис структури проєкту

Було реалізовано мікросервісний підхід, веб-додаток на PHP відповідає за інтерфейс користувача, роботу з базою даних та бізнес-логіку управління завданнями, тоді як окремий Python-сервіс виконує обчислювально складні операції з машинного навчання. Такий поділ дозволяє незалежно розвивати клієнтську частину системи та моделі прогнозування, оновлювати їх без повного перезапуску всієї платформи й масштабувати кожен компонент окремо. На концептуальному рівні взаємодія компонентів показана на рисунку 3.1. PHP-додаток формує API-запит із текстом завдання та супутніми параметрами, надсилає його до Flask-сервісу, а у відповідь отримує прогнозований час виконання та пріоритет.

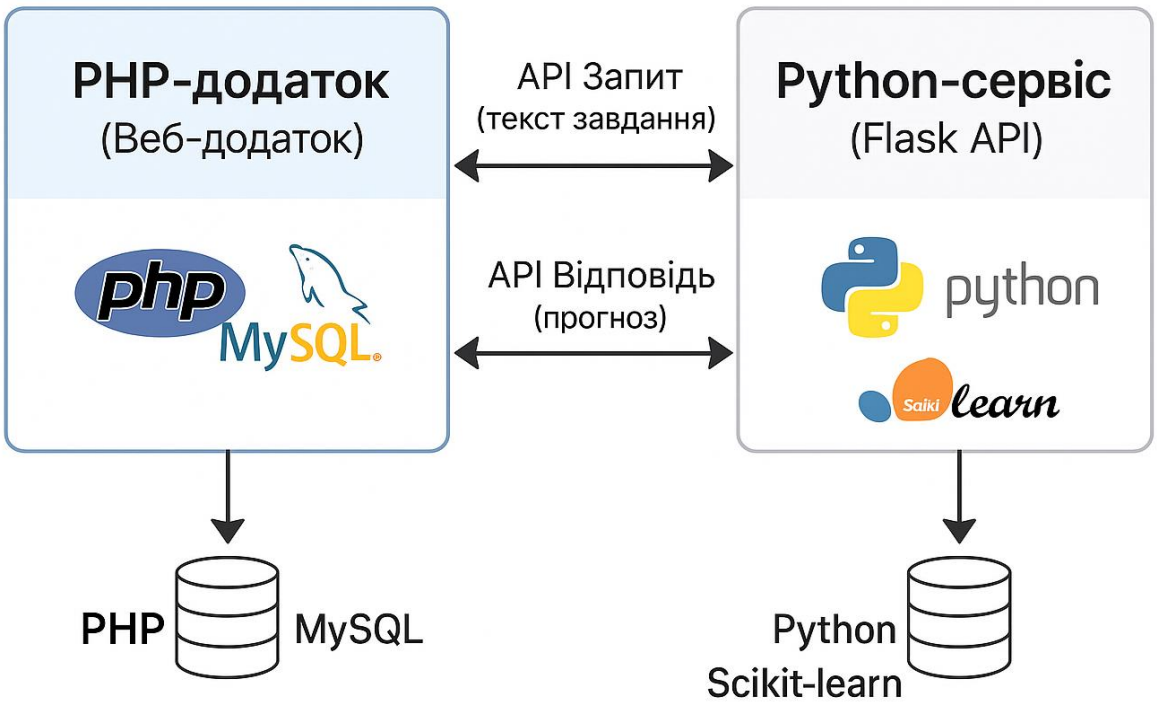


Рис. 3.1 Взаємодія сервісів

У структурі PHP-додатку було використано класичну для невеликих застосунків організацію каталогу. У корені розміщується директорія проєкту /todo.list, усередині якої є декілька логічних підрозділів. Каталог /assets містить статичні ресурси, CSS-файли для сторінок авторизації та основного інтерфейсу, а також зображення - аватари користувачів, іконки опцій тощо. Папка /config

зосереджує налаштування підключення до бази даних MySQL, що дозволяє централізовано змінювати параметри доступу. Структуру бази даних розробленого PHP-додатку зображено у вигляді ER-діаграми на рисунку 3.2.

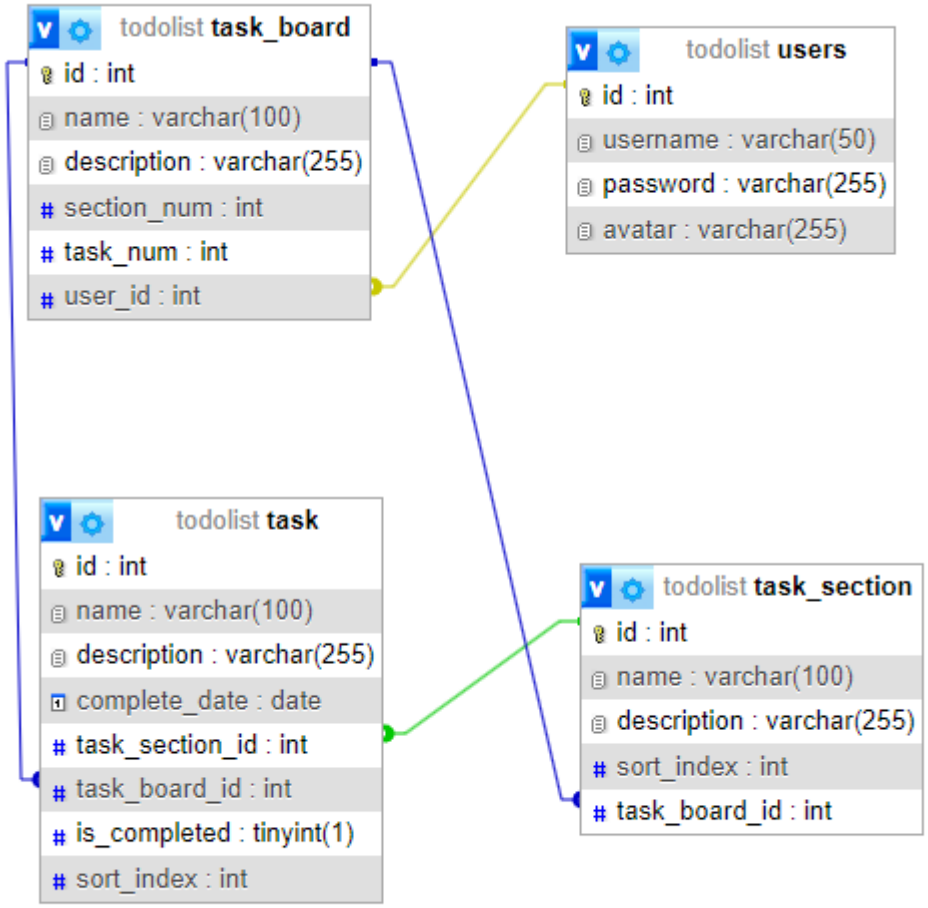


Рис. 3.2 ER-діаграма бази даних

Модель даних складається з чотирьох основних таблиць:

- users;
- task_board;
- task_section;
- task.

Між якими визначені зв’язки «один-до-багатьох». Таблиця users зберігає облікові записи користувачів системи (ідентифікатор, логін, хеш пароля, шлях до

аватара) і пов’язана з таблицею task_board за полем user_id, що відображає належність дошки конкретному користувачу.

Кожна дошка завдань описується у таблиці task_board полями name, description, а також службовими атрибутами section_num та task_num, які дозволяють швидко отримувати кількість секцій і завдань без додаткових агрегувальних запитів. З дошкою пов’язані секції та завдання. Таблиця task_section містить перелік колонок дошки (name, description, sort_index) і через зовнішній ключ task_board_id пов’язана з task_board. Таблиця task зберігає окремі завдання з полями name, description, complete_date, ознакою виконання is_completed, порядковим номером sort_index, а також зовнішніми ключами task_section_id та task_board_id, що забезпечує їх прив’язку і до секції, і до дошки.

Така схема дозволяє гнучко працювати з ієрархією «користувач → дошка → секції → завдання», та швидко формувати вибірки для відображення дошки у веб-інтерфейсі та надалі використовувати історію виконаних завдань як джерело даних для навчання моделей машинного навчання.

Основна серверна логіка рознесена по файлах у каталозі /controllers. Там розміщені обробники створення, оновлення й видалення проєктів, секцій та завдань, контролери авторизації, реєстрації та виходу користувача, а також скрипти, що повертають списки проєктів, секцій і задач у потрібному форматі. Відображення інтерфейсу побудовано у каталозі /pages, де кожна ключова сторінка (дошка, дашборд, форми логіну та реєстрації) має окремий PHP-файл, що поєднує HTML-розмітку з викликами відповідних контролерів.

Взаємодія з модульною системою машинного навчання винесена в окремий Python-проєкт, можна переглянути на рисунку 3.3.

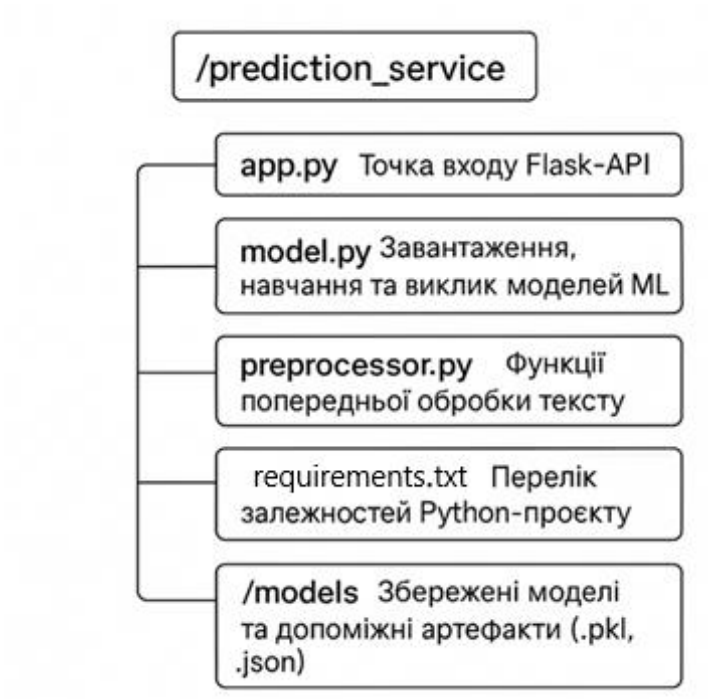


Рис. 3.3 Структура файлів Python-проєкту

Файл app.py є точкою входу сервісу, в якому налаштовано Flask-додаток, описано REST-ендпойнт /predict, приймає HTTP-запити від PHP-додатку у форматі JSON, викликає попередню обробку тексту та моделі машинного навчання, а потім формується відповідь з прогнозованим часом та рівнем пріоритету завдання. У модулі preprocessor.py зосереджені всі операції над сирим текстом, такі як:

- нормалізація регістру;
- очищення від зайвих символів;
- токенізація;
- видалення стоп-слів;
- стемінг.

Ці операції виконуються як під час навчання моделей, так і під час прогнозування в режимі реального часу, для забезпечення узгодженості вхідних даних.

Модуль model.py використовується для завантаження та використання моделей машинного навчання. Тут було реалізовано TF-IDF-векторизатор, алгоритм кластеризації K-Means і регресійна модель Random Forest Regressor. Під час ініціалізації сервісу моделі зчитуються з файлів у каталозі /models, а при

кожному запиті викликається відповідна послідовність операцій - векторизація тексту, кластеризація для оцінки "типу" задачі та регресія для розрахунку прогнозованого часу. Каталог /models використовується як сховище артефактів навчання. В ньому зберігаються серіалізовані моделі (.pkl), налаштовані векторизатори та, за потреби, конфігураційні файли з параметрами навчання або статистикою датасету.

Файл requirements.txt містить повний перелік бібліотек, необхідних для роботи сервісую. Це спрощує розгортання сервісу на новому сервері або у контейнері. Достатньо виконати встановлення залежностей за цим файлом, і середовище буде повністю відтворено. Така структуризація Python-частини проєкту дає змогу чітко розділити відповідальність між веб-рівнем (Flask-API), рівнем попередньої обробки тексту та рівнем безпосередніх моделей машинного навчання, що робить систему зрозумілою, розширюваною та зручною для подальших експериментів.

3.1.3 Опис інтерфейсу

Взаємодія користувача з розробленою системою прогнозування, а також комунікація між PHP-додатком та Python-сервісом, реалізована через два основні компоненти:

- графічний інтерфейс користувача (GUI);
- програмний інтерфейс додатку (REST API).

Інтерфейс було інтегровано в існуючу систему управління завданнями для забезпечення зручного та інтуїтивно зрозумілого доступу до функціоналу прогнозування.

На рисунку 3.4 представлено головний інтерфейс системи, де відображаються проєкти та дошки із завданнями. Кожне завдання має назву, опис, відмітку про виконання та встановлений дедлайн, що дозволяє користувачу візуально оцінювати поточний стан робіт.

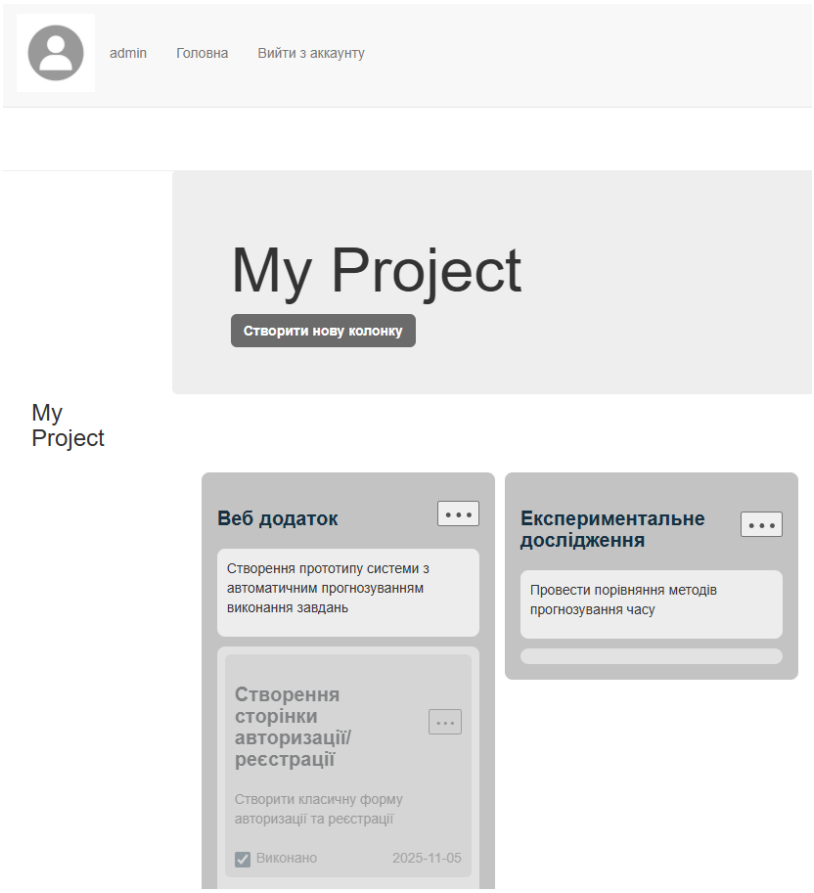


Рис. 3.4 Головний інтерфейс системи управління завданнями

Окрім базових операцій, таких як створення нових колонок (дошок), що показано на рисунку 3.5, в систему було інтегровано дві ключові функції для автоматизованого прогнозування.

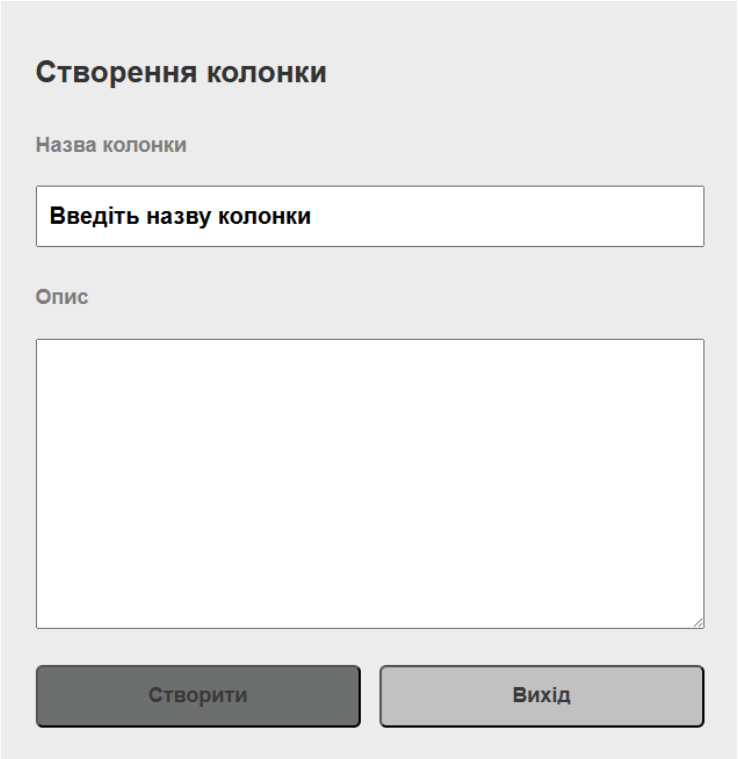


Рис. 3.5 Інтерфейс створення нової колонки

1. Пакетне прогнозування для всієї дошки. Для можливості масової переоцінки завдань, у форму оновлення колонки було додано кнопку "Перепрогнозувати дедлайни", форму оновлення колонки можна побачити на рисунку 3.6. При її натисканні система збирає описи всіх завдань у поточній колонці та надсилає їх одним запитом до сервісу прогнозування, після чого оновлює дедлайни для кожного завдання.

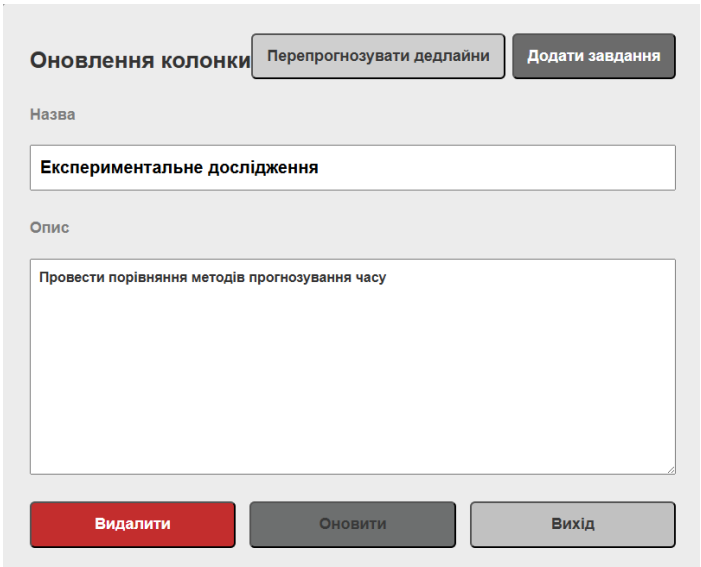


Рис. 3.6 Кнопка для пакетного прогнозування завдань у колонці

2. Прогнозування для окремого завдання. У форму створення нового завдання було інтегровано кнопку "Спрогнозувати дедлайн", цю форму можна побачити на рисунку 3.7. Це дозволяє користувачу отримати оцінку часу безпосередньо в процесі створення завдання.

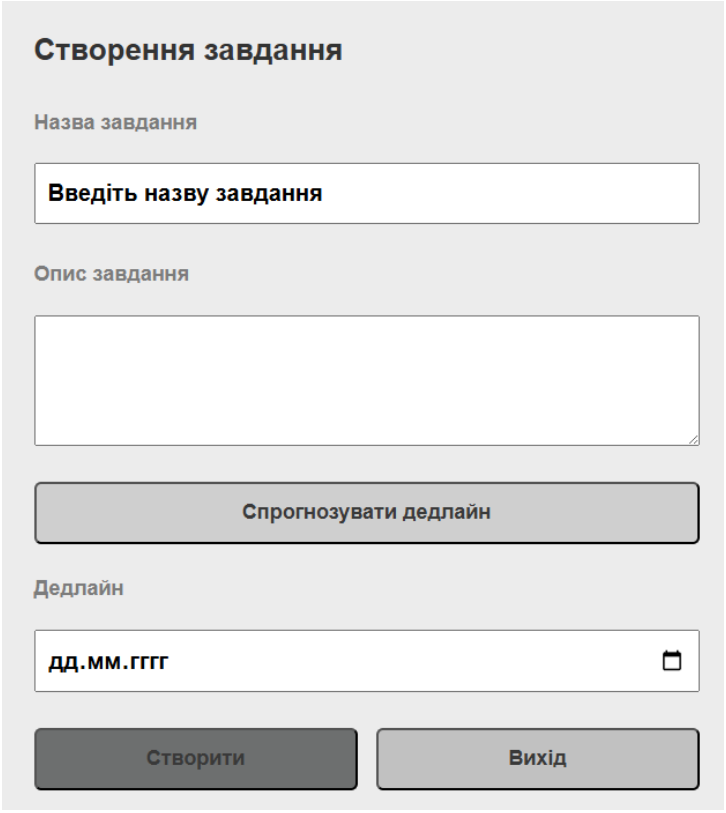


Рис. 3.7 Форма створення завдання з функцією прогнозування

Процес роботи даної функції наочно ілюструє рисунок 3.8. Користувач вводить назву та опис завдання, після чого натискає кнопку "Спрогнозувати дедлайн". Система надсилає текстовий опис до сервісу машинного навчання, отримує у відповідь прогнозовану кількість днів та автоматично заповнює поле "Дедлайн" відповідною датою. Наприклад, для прогнозу в 1 день, отриманого 05.11.2025, система встановлює дедлайн на 06.11.2025.

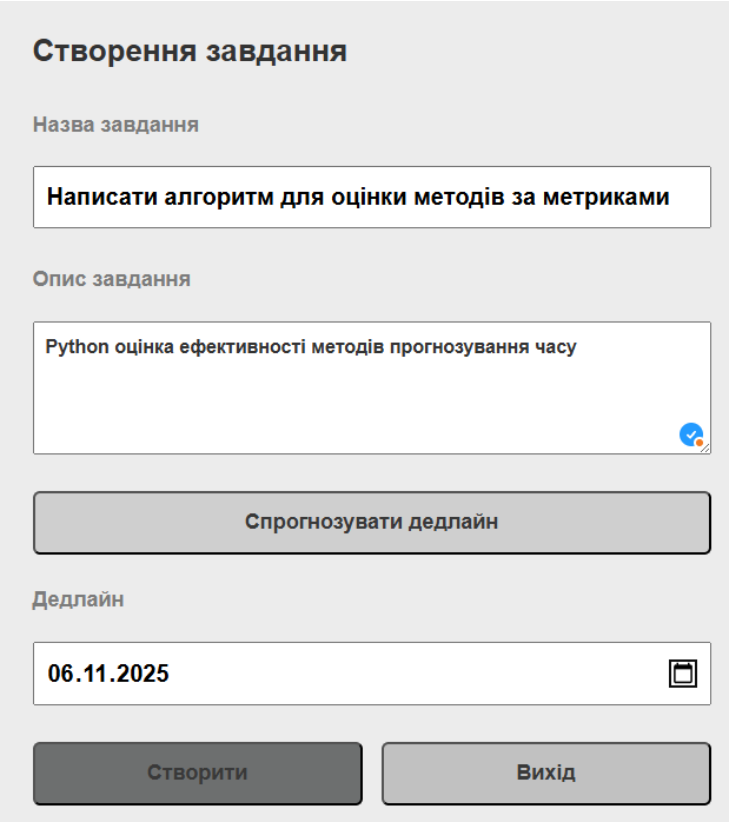


Рис. 3.8 Приклад роботи функції прогнозування для окремого завдання

Уся описана вище функціональність графічного інтерфейсу працює через REST API, що забезпечує комунікацію між PHP-додатком та Python-сервісом. Було розроблено два основні ендпоінти:

1. Ендпоінт /predict для прогнозування одного завдання:
   - метод: POST;
   - тіло запиту (Request Body) містить JSON-об'єкт з полем task, що містить текстовий опис;
   - відповідь (Response Body) містить JSON-об'єкт з полями predicted_time_days та predicted_priority;
   - приклад запиту:
     - curl -X POST http://127.0.0.1:5000/predict -H "Content-Type: application/json" -d "{\"task\": \"Реалізувати повнотекстовий пошук на сайті з індексацією\"}".

Приклад відповіді можна подивитись на рисунку 3.9.

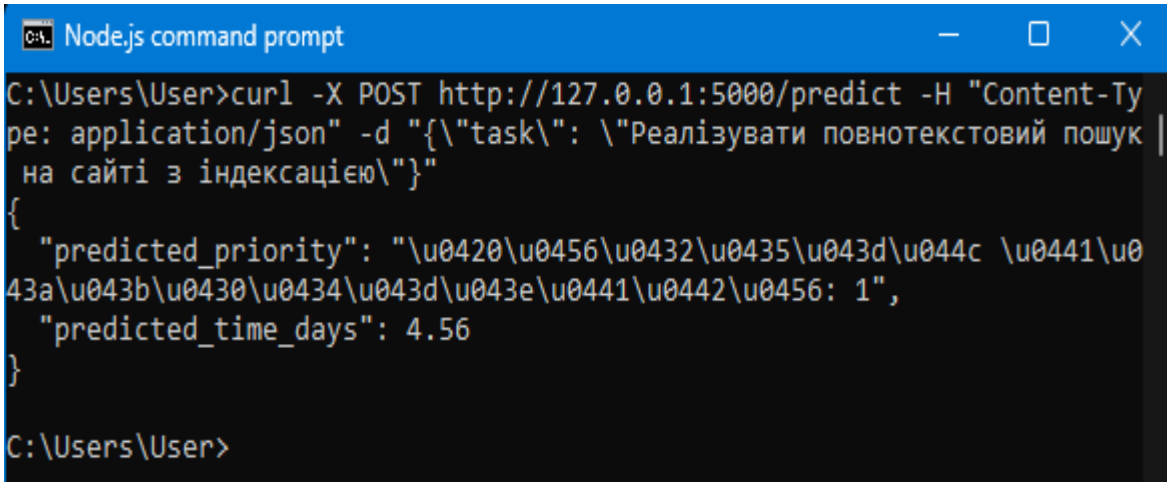


Рис. 3.9 Приклад роботи запиту на прогнозування одного завдання

2. Ендпоінт /predict-board для пакетного прогнозування завдань:

- метод: POST;
- тіло запиту (Request Body): містить JSON-об'єкт з полем tasks, що містить масив текстових описів;
- відповідь (Response Body), містить масив JSON-об'єктів, де кожен об'єкт відповідає одному завданню.

Приклад пакетного запиту можна подивитись на рисунку 3.10.

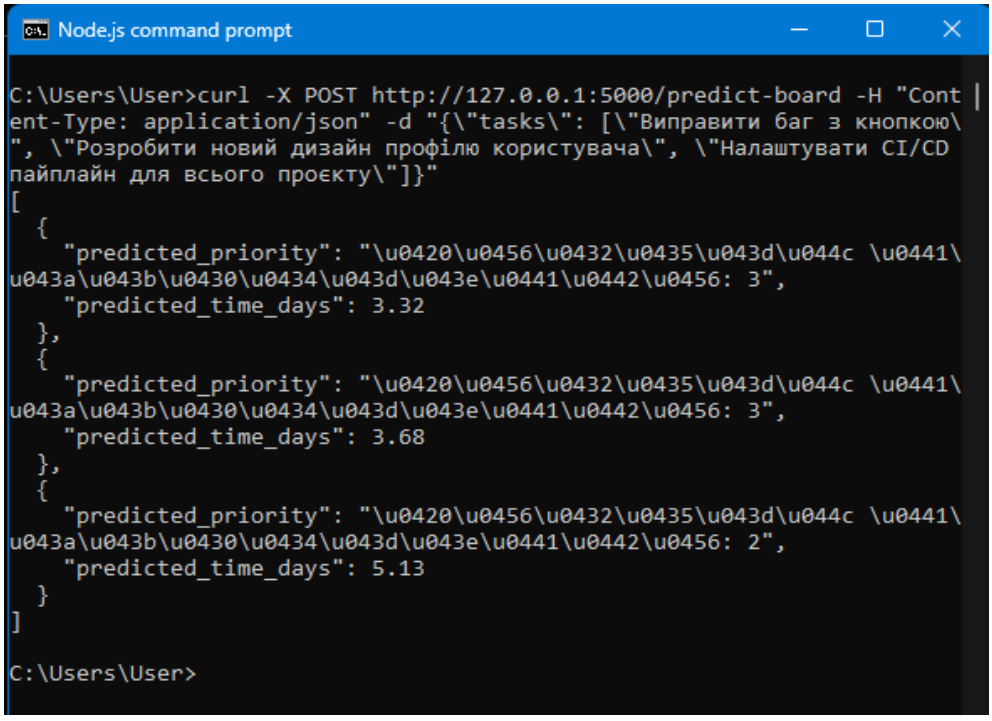


Рис. 3.10 Приклад роботи пакетного запиту на прогнозування одного завдання

Символи кирилиці у відповіді curl в консолі Windows можуть відображатися у вигляді Unicode-послідовностей (напр., \u0420...).

3.1.4 Опис розроблених класів

Архітектура програмного рішення складається з трьох ключових логічних компонентів, що відповідають за обробку даних, моделювання та надання доступу через API. Їхня взаємодія забезпечує повний цикл від отримання вхідного запиту до видачі фінального прогнозу. На рисунку 3.11 зображено UML-діаграму залежностей між основними компонентами системи.

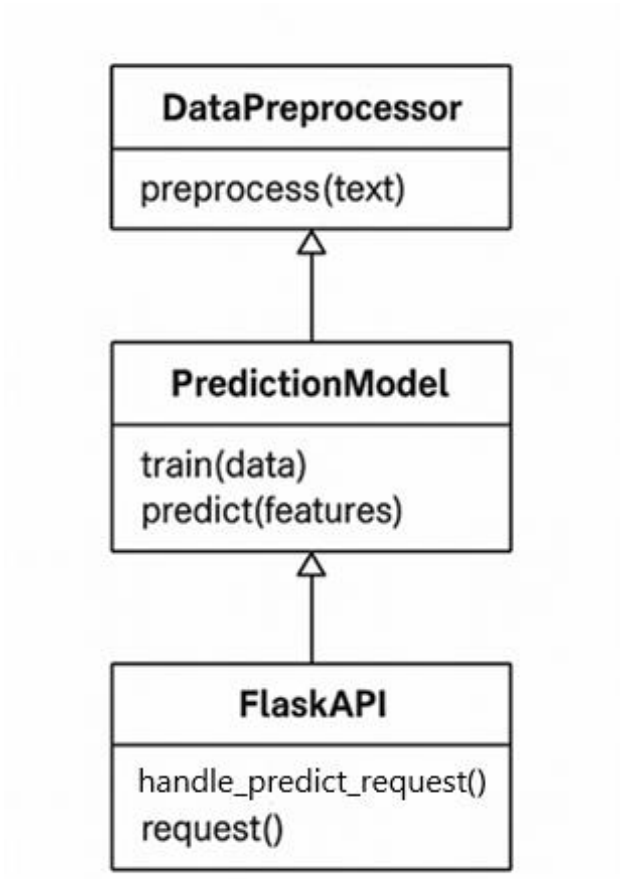


Рис. 3.11 UML-діаграма залежностей між ключовими класами

На діаграмі показано, що FlaskAPI для виконання своїх завдань залежить від PredictionModel, викликаючи його метод для прогнозування. А PredictionModel залежить від DataPreprocessor для підготовки текстових даних як на етапі навчання, так і на етапі прогнозування. Нижче наведено детальний опис кожного компонента.

Клас DataPreprocessor. Цей логічний компонент відповідає за попередню обробку текстових даних, приводячи їх до формату, придатного для векторизації та подальшого аналізу моделлю.

Метод preprocess(text) виконує повний конвеєр обробки вхідного тексту. В реалізації прототипу ця логіка інкапсульована у функції preprocess_text. Процес включає:

1. приведення тексту до нижнього регістру;
2. видалення всіх символів, окрім українських літер та пробілів;
3. токенізацію (розбиття на слова);
4. лематизацію (приведення кожного слова до його базової форми);
5. видалення стоп-слів.

Клас PredictionModel. Це компонент, що інкапсулює логіку навчання та використання моделей машинного навчання для прогнозування часу виконання завдань.

Метод train(data) відповідає за процес навчання моделей на основі наданого набору даних. Метод послідовно виконує такі кроки:

1. навчає TfidfVectorizer для перетворення текстів у числові вектори;
2. навчає модель кластеризації KMeans для групування завдань;
3. навчає регресійну модель RandomForestRegressor на основі TF-IDF векторів та міток кластерів для прогнозування цільової змінної (часу виконання).

Метод predict(features) використовується для отримання прогнозу на основі нових вхідних даних. Ця функціональність використовується в API при обробці запитів, яка приймає підготовлені ознаки (векторизований текст і кластер) та повертає прогнозований час.

Клас FlaskAPI. Це компонент, що реалізує вебсервер та надає зовнішній HTTP-інтерфейс для взаємодії з системою.

Метод handle_predict_request(), що обробляє вхідні POST-запити на прогнозування. Реалізовано як функція-обробник predict_single для ендпоінта /predict. Логіка роботи ендпоінта така:

- отримання та валідація JSON-запиту, що містить опис завдання;
- виклик preprocess_text для очищення та нормалізації тексту;
- використання заздалегідь навчених моделей TfidfVectorizer та KMeans для отримання числових ознак;
- виклик методу predict регресійної моделі RandomForestRegressor для отримання фінального прогнозу;
- повернення результату у форматі JSON.

3.2 Проведення експерименту та аналіз результатів точності прогнозування

3.2.1 Постановка експерименту

На цьому етапі перейдемо від опису методики до її практичної перевірки. Основна ідея експерименту полягала в тому, щоб на одному й тому самому наборі завдань порівняти кілька варіантів комплексного методу прогнозування, які відрізняються лише підходом до попередньої обробки тексту.

За основу було взято датасет із 1000 реалістичних завдань з галузі інженерії програмного забезпечення. Кожен запис містить ідентифікатор, коротку назву, текстовий опис завдання та фактичний час виконання у днях. Приклади завдань:

- налаштування репозиторію Git;
- реалізація форм логіну та реєстрації;
- створення REST API;
- впровадження CI/CD;
- оптимізація SQL-запитів.

Цільовою змінною у всіх моделях був час виконання завдання в днях. Вхідною ознакою виступав текстовий опис завдання (назва плюс опис), який проходив через різні варіанти попередньої обробки, після чого перетворювався на числовий вектор за допомогою TF-IDF. Додатково до ознак було додано номер кластера, отриманий алгоритмом K-means, який інтерпретувався як рівень

складності або «тип» завдання. На фінальному кроці для регресії часу виконання використовувався Random Forest Regressor.

Усі експерименти проводилися в середовищі Python з використанням бібліотек Pandas, scikit-learn, NLTK та pymorphy3. Для забезпечення відтворюваності було зафіксовано параметр random_state у функціях розбиття на вибірки та під час ініціалізації моделей. Дані поділялися на тренувальну та тестову вибірки у співвідношенні приблизно 70 % до 30 %. Навчання кожної моделі відбувалося на тих самих тренувальних даних, а оцінювання - на спільній тестовій множині.

Для порівняння було розглянуто чотири варіанти однієї й тієї самої базової архітектури TF-IDF + K-means + Random Forest, змінюючи лише блок попередньої обробки тексту. Перший варіант - базовий метод без лематизації та стемінгу Другий варіант - той самий метод, але з додаванням лематизації. Третій - варіант зі стемінгом без лематизації. Четвертий - комбінований підхід, де лематизація та стемінг використовуються разом. (У всіх окрім базового методу, було ще додано такі покращення як:

- приведення до нижнього регістру;
- очищення від зайвих символів;
- токенізація.

3.2.2 Метрики оцінювання

Щоб оцінити якість прогнозування часу виконання завдань, було використано кілька класичних регресійних метрик. Основний акцент робився саме на регресійних показниках, оскільки задача формулюється як оцінка неперервної змінної - кількості днів.

Середня абсолютна помилка (MAE) (3.1) показує, наскільки в середньому прогноз відхиляється від фактичного часу виконання в абсолютному вираженні. MAE визначається як:

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|, \quad (3.1)$$

де y_i - фактичний час виконання завдання,

ŷ - прогноз моделі,

n - кількість прикладів у тестовій вибірці.

Менше значення MAE відповідає кращій якості прогнозу.

Середня квадратична помилка (MSE) (3.2) підсилює вплив великих помилок, оскільки використовує квадрат різниці між прогнозом та істинним значенням:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \quad (3.2)$$

Коренева середньоквадратична помилка (RMSE) (3.3) є квадратним коренем із MSE:

$$RMSE = \sqrt{MSE} \quad (3.3)$$

її зручно інтерпретувати в тих самих одиницях, що й цільова змінна (дні).

Коефіцієнт детермінації R^2 (3.4) характеризує, яку частку варіації цільової змінної пояснює модель у порівнянні з тривіальним передбаченням середнього значення. Формула має вигляд:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - ÿ)^2}, \quad (3.4)$$

де ÿ- середнє значення фактичного часу виконання на тестовій вибірці.

Значення R^2, близькі до 1, свідчать про хорошу узгодженість моделі з даними; від'ємні значення означають, що модель працює гірше, ніж просте прогнозування середнього.

Точність (accuracy) (3.5) визначається як частка правильно класифікованих прикладів:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}, \quad (3.5)$$

де TP, TN, FP, FN - кількість істинно-позитивних, істинно-негативних, хибно-позитивних та хибно-негативних відповідностей.

Повнота (recall) (3.6) для одного класу відображає, яку частку об'єктів цього класу модель змогла правильно «знайти»:

$$Recall = \frac{TP}{TP+FN} \quad (3.6)$$

Точність (precision) (3.7) показує, яка частка об’єктів, віднесених моделлю до певного класу, справді належить до нього:

$$Precision = \frac{TP}{TP+FP} \tag{3.7}$$

F1-міра є гармонійним середнім між точністю та повнотою:

$$F1 = 2 \times \frac{Precision \times Recall}{Precision+Recall} \tag{3.8}$$

3.2.3 Проведення експерименту та результати

Експеримент було реалізовано у вигляді окремого Python-скрипта, який автоматично виконує всі етапи:

- завантаження даних;
- розбиття на тренувальну та тестову вибірки;
- навчання кожного з чотирьох варіантів моделі;
- прогнозування для тестових прикладів;
- обчислення всіх описаних вище метрик.

Для запобігання випадковим впливам було зафіксовано випадкове зерно генератора псевдовипадкових чисел, тож результати можна відтворити.

Для кожного варіанта методики конвеєр залишався однаковим:

- попередня обробка тексту;
- TF-IDF векторизація;
- кластеризація K-means (номер кластера додавався до ознак);
- навчання Random Forest Regressor.

Відмінність стосувалася лише попередньої обробки. Результат роботи скрипта наведено на рисунку 3.12.

```
C:\OSPanel\home\todo.list\predict\experiment>python test.py
                          model       MAE        MSE      RMSE        R2  Accuracy  Precision_macro  Recall_macro  F1_macro
0  Базовий метод (без леми/стемінгу)  2.771136  15.202493  3.899037 -0.309899  0.698925         0.348148      0.346622  0.303937
1              Метод з лематизацією  2.802297  14.350787  3.788243 -0.236513  0.688172         0.365378      0.341454  0.299337
2                Метод зі стемінгом  2.849411  14.828652  3.850799 -0.277688  0.677419         0.307596      0.336286  0.296296
3         Метод з лемою + стемінгом  2.751104  14.301363  3.781714 -0.232254  0.693548         0.400000      0.349391  0.313511

C:\OSPanel\home\todo.list\predict\experiment>
```

Рис. 3.12 Результат роботи скрипту для обчислення метрик

3.2.4 Аналіз результатів експерименту

Якщо подивитись значення метрик між чотирма варіантами методики, то виявилося, що різниця між ними не є великою, але певні незначні різниці можна побачити. Якщо дивитися лише на середню абсолютну помилку (рис 3.13), найменше значення показав комбінований підхід з лематизацією та стемінгом (MAE ≈ 2.75), хоча базовий метод без обробки за цією метрикою майже не відстає (MAE ≈ 2.77). Тобто за середнім модулем помилки всі варіанти дуже близькі.

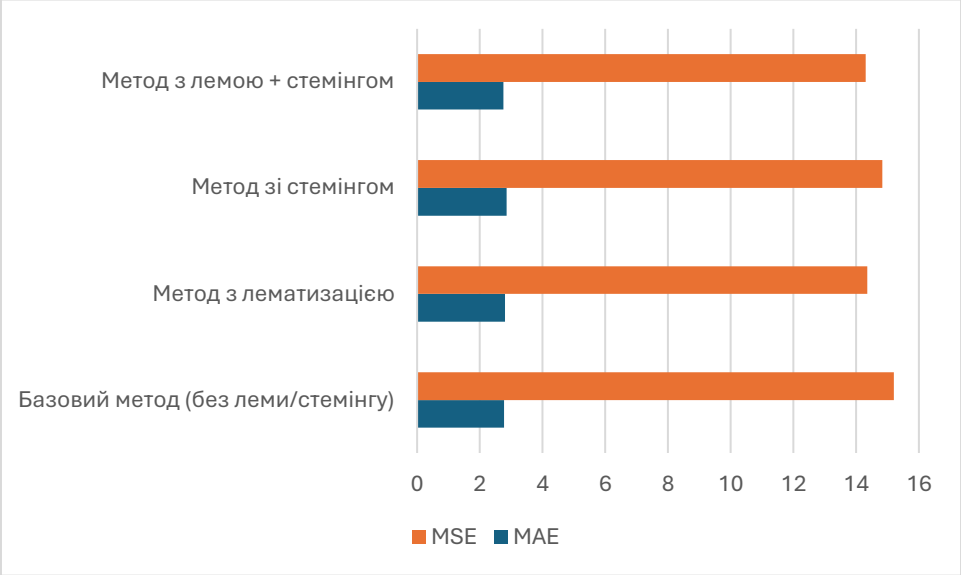


Рис. 3.13 Порівняльна діаграма метрик MSE та MAE

Більш цікавими є результати за RMSE та R^2 можна побачити на рисунку 3.14. Саме ці показники чутливіші до великих відхилень прогнозу від істини.

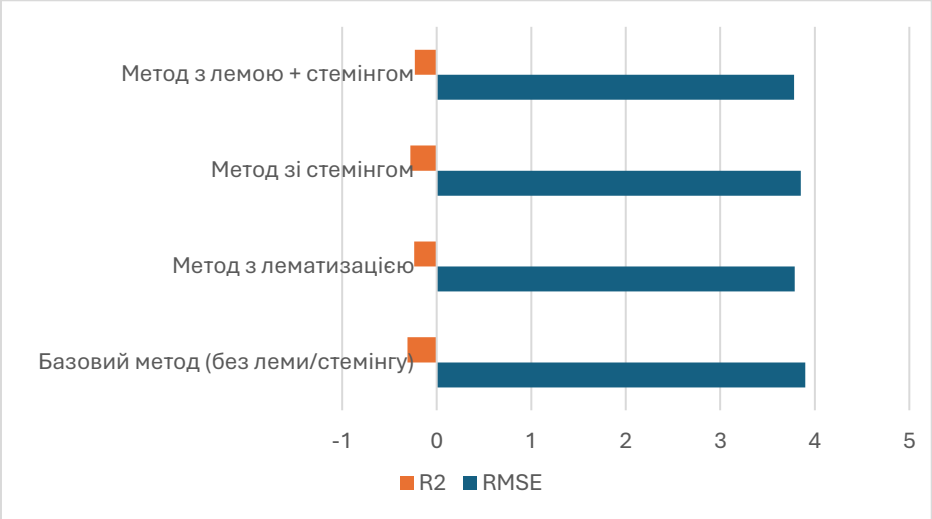


Рис. 3.14 Порівняльна діаграма метрик R^2 та RMSE

Тут комбінований метод із лематизацією та стемінгом демонструє найнижче значення RMSE та одне з найкращих значень коефіцієнта детермінації серед усіх варіантів. Це означає, що він трішки краще справляється з «важкими» випадками, де помилка може бути суттєвою. Базовий метод, навпаки, має вищий RMSE та більш негативне значення R^2, що вказує на менш стабільну поведінку.

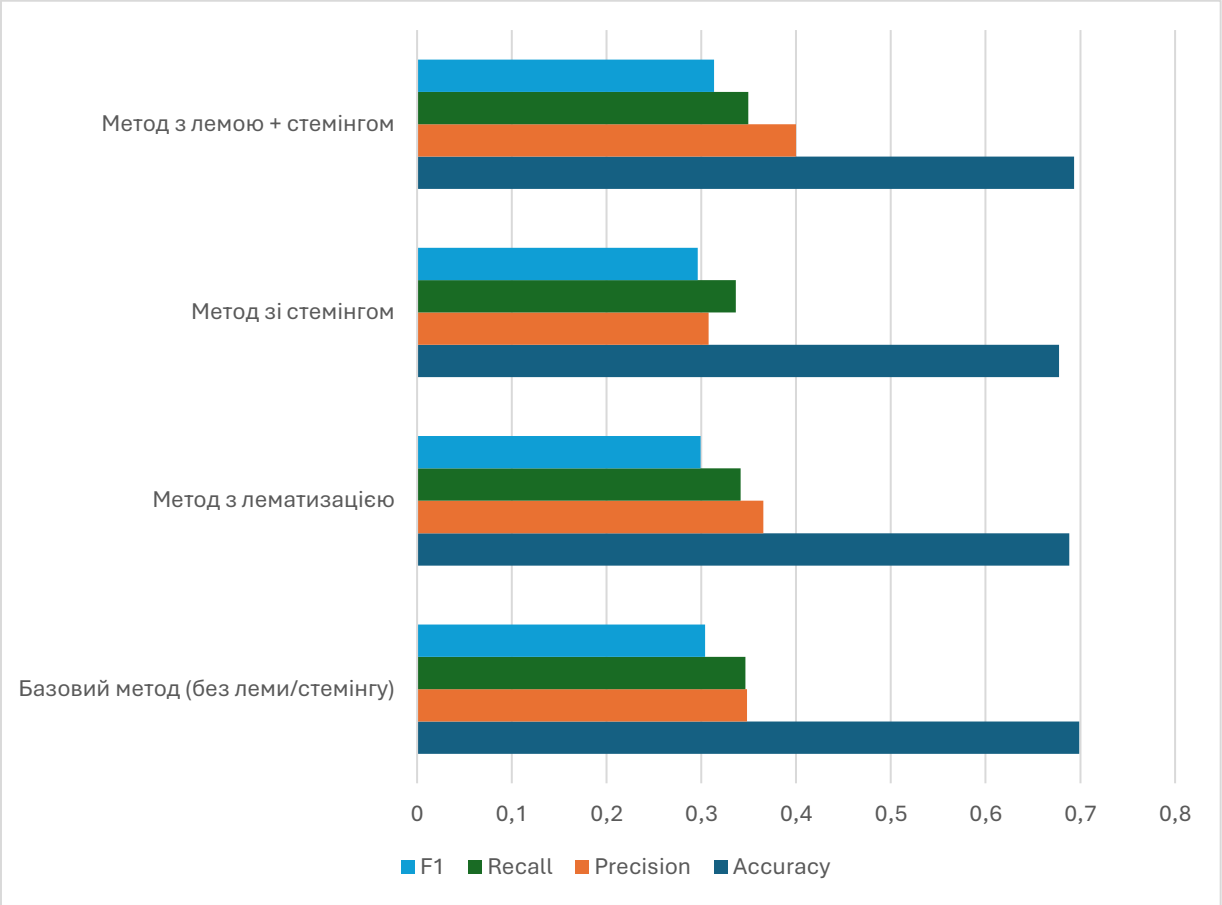


Рис. 3.15 Порівняльна діаграма класифікаційних метрик

Якщо подивитися на класифікаційні метрики після дискретизації часу в категорії на рисунку 3.15, ситуація теж виглядає на користь покращеного підходу. Комбінований метод показує найвищу точність та повноту, а також найвище значення F1-міри. Це говорить про те, що модель з лематизацією та стемінгом краще розрізняє різні класи тривалості завдань, не прогнозуючи у якийсь один домінуючий інтервал.

Розглянемо узагальнену таблицю 3.2 з показниками метрик та підведемо підсуми що і наскільки покращилося в результаті експериментального порівняння різних варіантів методу.

Таблиця 3.2

Узагальнена таблиця з показниками метрик

Модель	MAE	MSE	RMSE	R^2	Accuracy	Precision	Recall	F1
Базовий метод (без леми/стемінгу)	2,77	15,2	3,899	−0,31	0,69892	0,34814	0,3466	0,30393
Метод з лематизацією	2,8	14,4	3,79	−0,24	0,68817	0,36537	0,3414	0,29933
Метод зі стемінгом	2,85	14,8	3,85	−0,28	0,67741	0,30759	0,3362	0,29629
Метод з лемою + стемінгом	2,75	14,3	3,78	−0,23	0,69354	0,4	0,3493	0,31351

Порівняно з базовим варіантом, комбінований підхід (лематизація + стемінг) дав найкращі результати за більшістю показників:

- MAE зменшився на ~0,02 (≈ −0,7%);
- MSE зменшився на ~0,9 (≈ −5,9%);
- RMSE зменшився на ~0,12 (≈ −3,0%);
- R^2 покращився на +0,08;
- Accuracy зменшився на −0,0054 (≈ +-0,77%);
- Precision збільшився на +0,052 (≈ +14,9%);
- Recall збільшився на +0,0028 (≈ +0,8%);
- F1 збільшився на +0,0096 (≈ +3,15%).

Варіанти лише з лематизацією або лише зі стемінгом не перевершили комбінований підхід і місцями поступаються базовому за MAE/RMSE - це підтверджує, що ефект дає саме поєднання нормалізації форм слів (лема) із агресивним зведенням до основ (стемінг).

З чотирьох конфігурацій найкращим виявився метод TF-IDF + K-means + Random Forest із комбінованою попередньою обробкою (лематизація + стемінг). Він забезпечує невелике, але стабільне покращення регресійних метрик та зростання F1/Precision у порівнянні з базовим підходом без лінгвістичних покращень.

ВИСНОВКИ

У даній роботі було досліджено проблему автоматизації прогнозування часу виконання завдань і їх пріоритизації в системах управління проєктами та запропоновано комплексний метод адаптивного прогнозування, що поєднує векторизацію тексту TF-IDF, кластеризацію K-Means як індикатор рівня складності та ансамблеву регресію Random Forest Regressor. На основі критичного аналізу існуючих підходів сформульовано їхні ключові обмеження - суб'єктивність і низька відтворюваність, вимогливість до однорідних історичних даних і невизначеність відповідей - та спроєктовано конвеєр обробки даних, який ці обмеження пом'якшує.

Розроблено методику попередньої обробки тексту (нормалізація, токенізація, видалення стоп-слів, лематизація; окремо досліджено стемінг і комбінацію лемитізації зі стемінгом), описано векторизацію TF-IDF і постановку кластеризації K-Means, а також побудовано регресійну модель Random Forest для прогнозу тривалості у днях.

Проведено експериментальне дослідження для оцінки якості. Було порівняно чотири варіанти реалізації комплексного методу:

- базовий без морфологічної обробки;
- варіант із лематизацією;
- варіант зі стемінгом;
- комбінований підхід із поєднанням лематизації і стемінгу.

Для кожної моделі обчислено MAE, MSE, RMSE, коефіцієнт детермінації R^2, а також класифікаційні метрики (точність, повнота, F1-міра).

Порівняно з базовим варіантом, комбінований підхід (лематизація + стемінг) дав найкращі результати за більшістю показників:

- MAE зменшився на ~0,02 (≈ −0,7%);
- MSE зменшився на ~0,9 (≈ −5,9%);
- RMSE зменшився на ~0,12 (≈ −3,0%);

- R^2 покращився на +0,08;
- Accuracy зменшився на −0,0054 (≈ +-0,77%);
- Precision збільшився на +0,052 (≈ +14,9%);
- Recall збільшився на +0,0028 (≈ +0,8%);
- F1 збільшився на +0,0096 (≈ +3,15%).

Варіанти лише з лематизацією або лише зі стемінгом не перевершили комбінований підхід і місцями поступаються базовому за MAE/RMSE - це підтверджує, що ефект дає саме поєднання нормалізації форм слів (лема) із агресивним зведенням до основ (стемінг).

Отримані результати показали, що всі розглянуті варіанти демонструють порівнюваний рівень якості, але методика з поєднанням лематизації та стемінгу забезпечує найкращий баланс між помилкою регресії та класифікаційними показниками, хоч і з відносно невеликим, але стабільним покращенням порівняно з базовим підходом. Результати роботи можуть бути використані при розробці та впровадженні модулів прогнозування часу виконання.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей та стаття:

1. І.Ю. Коломієць, І.В. Замрій, Б. С. Калинюк, Ю. П. Бажан, Т. П. Довженко. Сучасні підходи до автоматизації управління завданнями для малих команд з використанням методів машинного навчання // Зв’язок, №3, 2025. С. 66-72.

2. Коломієць І.Ю., Герцюк М.М. Використання машинного навчання для оптимізації управління завданнями в малих командах: підхід та інструменти // ІІ міжнародна науково-практична конференція «Сучасні аспекти діджиталізації та інформатизації в програмній та комп'ютерній інженерії». - Київ: ДУІКТ, 2024. С.245-246.

3. Коломієць І.Ю., Замрій І.В. Метод адаптивного прогнозування часу для автоматизації процесів розподілу завдань. // VIII Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ». - Київ: ДУІКТ, 2025. С.288-291.

ПЕРЕЛІК ПОСИЛАНЬ

1. Мельниченко Д. В., та ін. Проєктування вебзастосунку для керування робочим часом. Вісник Херсонського національного технічного університету. 2024. № 3(90). С. 238-250. URL: https://doi.org/10.35546/kntu2078-4481.2024.3.30 (дата звернення: 25.02.2025).

2. Сотнікова Ю., Лугова В. Автоматизація управлінської діяльності в умовах цифрової економіки. *Економіка та суспільство*. 2024. № 65. URL: https://doi.org/10.32782/2524-0072/2024-65-74 (дата звернення: 25.02.2025).

3. Лошенюк О. Автоматизація управління бізнес-процесами підприємства як основа гнучкості його діяльності: характеристика систем управління. *Економіка та суспільство*. 2022. № 45. URL: https://doi.org/10.32782/2524-0072/2022-45-46 (дата звернення: 25.02.2025).

4. Тонковид, Т.А., Милашенко В.М. Машинне навчання для автоматизації управління базами даних: розробка алгоритмів для автоматичного моніторингу та оптимізації. *Європейський університет. 2024. URL*: https://doi.org/10.36919/978-966-301-266-7/1.2024.125 (дата звернення: 25.02.2025).

5. Krasnokutska N. S., Podoprykhina T. O. Analysis of Project Management Methodologies in IT Industry. *Business Inform*. 2020. Т. 8, № 511. С. 217-222. URL: https://doi.org/10.32983/2222-4459-2020-8-217-222. (дата звернення: 25.02.2025).

6. О. Криворучко та ін. Інформаційно-управляючі системи автоматизації бізнес-процесів підприємства на мобільних платформах. *Наука і техніка сьогодні*. 2024. № 4(32). URL: https://doi.org/10.52058/2786-6025-2024-4(32)-1085-1101 (дата звернення: 25.02.2025).

7. Kravchenko S. N., Grishkun E. O., Vlasenko O. V. Classification methods for machine learning using the scikit-learn library. *Scientific notes of Taurida National V.I. Vernadsky University. Series: Technical Sciences*. 2020. Т. 1, №3. С. 121-125. URL: https://doi.org/10.32838/tnu-2663-5941/2020.3-1/19 (дата звернення: 25.02.2025).

8. Олещенко Л. М., Мельничук О. Г. Застосування асамблевих методів машинного навчання для виявлення неправдивого тексту. *Вісник Херсонського*

національного технічного університету. 2024. № 1(88). С. 258-263. URL: https://doi.org/10.35546/kntu2078-4481.2024.1.36 (дата звернення: 25.02.2025).

9. Chary, Deekshith, Review on Advanced Machine Learning Model: Scikit-Learn (July 4, 2020). P. Deekshith chary, Dr.R.P.Singh, *International Journal of Scientific Research and Engineering Development (IJSRED)* Vol3-Issue4 | 526-529., Режим доступу: https://ssrn.com/abstract=3694350 (дата звернення: 25.02.2025).

10. Krzeszewska U., Poniszewska-Marańda A., Ochelska-Mierzejewska J. Systematic Comparison of Vectorization Methods in Classification Context. *Applied Sciences*. 2022. Т. 12, № 10. С. 5119. URL: https://doi.org/10.3390/app12105119 (дата звернення: 25.02.2025).

11. О. Гарбич-Мошора та ін. Розробка мобільного додатку для трекінгу та контролю роботи над завданнями. *Наука і техніка сьогодні*. 2024. № 4(32). URL: https://doi.org/10.52058/2786-6025-2024-4(32)-882-893 (дата звернення: 25.02.2025).

12. Orkhan Masimov, Yashar Hajiyev O. M. Y. H. integration of artificial intelligence (AI) in task management systems. *PAHTEI-Procedings of Azerbaijan High Technical Educational Institutions*. 2024. Т. 41, № 06. С. 501-509. URL: https://doi.org/10.36962/pahtei41062024-55 (дата звернення: 25.02.2025).

13. Iwendi C., та ін. An Efficient and Unique TF/IDF Algorithmic Model-Based Data Analysis for Handling Applications with Big Data Streaming. *Electronics*. 2019. Т. 8, № 11. С. 1331. URL: https://doi.org/10.3390/electronics8111331 (дата звернення: 25.02.2025).

14. Sheikh M. S., Enam R. N., Qureshi R. I. Machine learning-driven task scheduling with dynamic K-means based clustering algorithm using fuzzy logic in FOG environment. *Frontiers in Computer Science*. 2023. Т. 5. URL: https://doi.org/10.3389/fcomp.2023.1293209 (дата звернення: 25.11.2024).

15. A. O. Sousa та ін. Applying Machine Learning to Estimate the Effort and Duration of Individual Tasks in Software Projects. *IEEE Access*. 2023. С. 1. URL: https://doi.org/10.1109/access.2023.3307310 (дата звернення: 25.11.2024).

16. scikit-learn: machine learning in Python - scikit-learn 1.5.2 documentation [Електронний ресурс] // scikit-learn: machine learning in Python - scikit-learn 0.16.1 documentation. URL: https://scikit-learn.org/stable/ (дата звернення: 25.11.2024).

17. Pham T.-P., Durillo J. J., Fahringer T. Predicting Workflow Task Execution Time in the Cloud Using A Two-Stage Machine Learning Approach. *IEEE Transactions on Cloud Computing*. 2020. Т. 8, № 1. С. 256-268. URL: https://doi.org/10.1109/tcc.2017.2732344 (дата звернення: 25.02.2025).

18. Sebastian R. Liu, Y. Mirjalili, V. Machine Learning with Pytorch and Scikit-Learn: Develop Machine Learning and Deep Learning Models with Python. Packt Publishing, Limited, 2022. 771 с.

19. scikit-learn: machine learning in Python - scikit-learn 1.1.3 documentation. *scikit-learn: machine learning in Python - scikit-learn 0.16.1 documentation*. URL: https://scikit-learn.org/1.1/ (дата звернення: 25.02.2025).

20. Koval I., Holovnia S. Методи лінійної регресії та k-means для прогнозування і кластеризації виробничих показників у Orange Data Mining. *Computer-Integrated Technologies: Education, Science, Production*. 2025. № 57. С. 57-68. URL: https://doi.org/10.36910/6775-2524-0560-2024-57-08 (дата звернення: 25.02.2025).

21. Ткачик О. А., Бойко Н. І. Оцінка Методів Кластеризації Різнотипових Даних. *Automation of technological and business processes*. 2023. Т. 15, № 1. С. 1-12. URL: https://doi.org/10.15673/atbp.v15i1.2508 (дата звернення: 25.02.2025).

22. Євланов М. В., Черепньов І. А. Використання алгоритмів кластеризації для вирішення задачі призначення завдань виконавцям проєкту. *І міжнародна науково-практична конференція таврійського національного університету до 160-річниці від дня народження в. І. Вернадського. Частина 2*. 2023. URL: https://doi.org/10.36059/978-966-397-303-6-49 (дата звернення: 25.02.2025).

23. Kaplina A. Controlling In The System Of Effective Management. *Ефективна економіка*. 2021. № 2. URL: https://doi.org/10.32702/2307-2105-2021.2.70 (дата звернення: 25.02.2025).

24. Крикун О. О., Медяник Ю. Г. Вибір менеджером електронних інструментів для створення ефективної системи тайм-менеджменту. *Проблеми сучасних трансформацій. Серія: економіка та управління*. 2024. № 11. URL: https://doi.org/10.54929/2786-5738-2024-11-04-04 (дата звернення: 25.02.2025).

25. Л. Приходченко, Н. Піроженко, М. Кернова, І. Синчак. Технології тайм-менеджменту в управлінській діяльності державних службовців [Електронний ресурс]. ІПСУ Національного Університету «Одеська Політехніка». URL: http://www.oridu.odessa.ua/9/buk/05072021.pdf (дата звернення: 25.02.2025).

26. Таранич А., Пелехацький Д. Використання штучного інтелекту в процесах стратегічного управління підприємствами. *Economy of Ukraine*. 2024. Т. 67, № 1(746). С. 54-65. URL: https://doi.org/10.15407/economyukr.2024.01.054 (дата звернення: 25.02.2025).

27. Ірина Хорошилова Автоматизація обліку і управління та перспективи розвитку для малого бізнесу. *Проблеми і перспективи розвитку підприємництва*. 2022. URL: https://doi.org/10.30977/PPB.2226-8820.2022.28.138 (дата звернення: 25.02.2025).

28. Кузьма, К., & Мельник, О. Дослідження Методів векторизації текстів у задачах валідації відповідей, поданих природною мовою. *Таврійський науковий вісник. Серія: Технічні науки. Наукові видання Херсонського державного аграрно-економічного університету*. 2021. URL: https://doi.org/10.32851/tnv-tech.2021.6.5 (дата звернення: 25.02.2025).

29. Джоші А., Павленко В. І. Порівняння методів машинного навчання для визначення фейкових новин. Інфокомунікаційні та комп’ютерні технології. 2024. Т. 1, № 07. С. 46-55. URL: https://doi.org/10.36994/2788-5518-2024-01-07-06 (дата звернення: 25.02.2025).

30. Михайлов Н. О. Методи високоефективного планування проєктів: традиційні підходи та машинне навчання. *Таврійський науковий вісник. Серія: Технічні науки*. 2024. № 4. С. 186-192. URL: https://doi.org/10.32782/tnv-tech.2024.4.18 (дата звернення: 25.02.2025).

31. Що таке Trello та як ним користуватися? URL: https://softlist.com.ua/ua/news/shcho-take-trello-ta-yak-nim-koristuvatisya (дата звернення: 01.11.2024).

32. Trello vs. Asana: який продукт вибрати для керування проєктами URL: https://blog.keycrm.app/trello-vs-asana-kakoj-produkt-vybrat-dlya-upravleniya-proektami/ (дата звернення: 01.11.2024).

33. YouTrack Powerful project management for all your teams URL: https://www.jetbrains.com/youtrack/ (дата звернення: 01.11.2024).

34. Microsoft Project 2010 URL: https://www.microsoft.com/uk-ua/microsoft-365/previous-versions/microsoft-project-2010 (дата звернення: 01.11.2024).

35. 7 способів використання ClickUp Андрій Андрєєв автор ApiX-Drive URL: https://apix-drive.com/ua/blog/useful/7-sposobiv-vikoristannja-clickup (дата звернення: 01.11.2024).

36. Wrike для бізнесу або як організувати роботу в команді. Журнал AntMedia URL: https://www.theantmedia.com/post/wrike-dlya-biznesu-abo-yak-organizuvati-robotu-v-komandi (дата звернення: 01.11.2024).

37. Що таке Jira і для чого вона потрібна URL: https://goit.global/ua/articles/shcho-take-jira-i-dlia-choho-vona-potribna/ (дата звернення: 01.11.2024).

38. Огляд Notion: корисні функції та на що здатна програма URL: https://icoola.ua/blog/programa-notion-na-iphone/ (дата звернення: 01.11.2024).

39. Basecamp by 37signals URL: https://basecamp.com/ (дата звернення: 01.11.2024).

40. HELLIP SAAS CONNECTOR Monday.com URL: https://hellip.com/ua/product/monday.html (дата звернення: 01.11.2024).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Магістерська робота

«Методика автоматизації прогнозування термінів виконання завдань для малих команд з використанням машинного навчання»

Виконав: студент групи ПДМ-63 Ігор КОЛОМІЄЦЬ

Керівник: доктор техн. наук, професор, зав. кафедри ІПЗ Ірина ЗАМРІЙ

Київ - 2025

МЕТА, ОБ'ЄКТА ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: підвищення ефективності прогнозування часу виконання завдань шляхом розробки комплексного методу адаптивного прогнозування часу, який поєднує алгоритми TF-IDF, кластеризація K-means та ансамблевого навчання Random Forest Regressor.

Об'єкт дослідження: процеси прогнозування часу виконання та пріоритизації в управлінні завданнями.

Предмет дослідження: методика автоматизації прогнозування, що поєднує методи векторизації тексту (TF-IDF), кластеризації (K-Means) та ансамблевого навчання (Random Forest Regressor) для аналізу та оцінки завдань.

АКТУАЛЬНІСТЬ РОБОТИ

Існуючі алгоритми вирішення задачі, їх переваги та недоліки

Підхід	Опис	Переваги	Недоліки	Чому недостатньо для нашої мети
Експертна оцінка	Колективна/індивідуальна оцінка у балах, без прямої прив'язки до годин/днів	Швидко; враховує неявний досвід команди	Суб'єктивність, низька відтворюваність; залежить від складу/втоми; важко масштабувати	Не дає стабільної, відтворюваної числової оцінки в днях; не автоматизується
Статистичний метод (лінійна регресія на простих ознаках)	Прогноз з використанням історичних числових полів (тип, пріоритет, довжина опису)	Простий, працює на малих обсягах	Ігнорує зміст тексту; лінійність зв'язків; потребує чистих історичних даних	Недостатня точність для нетипових задач; не розуміє опис завдання
LLM (ChatGPT)	Відповідь за текстом опису в чаті	Розуміє природну мову; швидкий старт	Відсутність проєктного контексту; невідтворюваність; "галюцинації"; іноді дуже повільно; не автоматизований.	Непридатний для продукційного планування; можливий лише як довідковий інструмент

3

Математична модель комплексного методу

TF-IDF

$$TF(t,d) = \frac{f(t,d)}{|d|}$$, де $f(t, d)$ – кількість разів, яку термін t з'являється в документі d;

$|d|$ – загальна кількість термінів у документі d.

$$IDF(\mathrm{t},\mathrm{D}) = log\left(\frac{N}{df(t)}\right)$$, де N – загальна кількість документів у корпусі;

$df(t)$ – кількість документів з корпусу, які містять термін t.

Кінцева вага *TF-IDF* є добутком цих двох величин

$$TF - IDF(t,d,D) = TF(t,d) \times IDF(t,D)$$

K-Means

Розділяє n завдань на k кластерів, так щоб кожне завдання належало до кластера з найближчим до нього середнім значенням.

Random Forest Regressor

$$\hat{y} = \frac{1}{N}\sum_{i=1} \hat{y}_i(X)$$

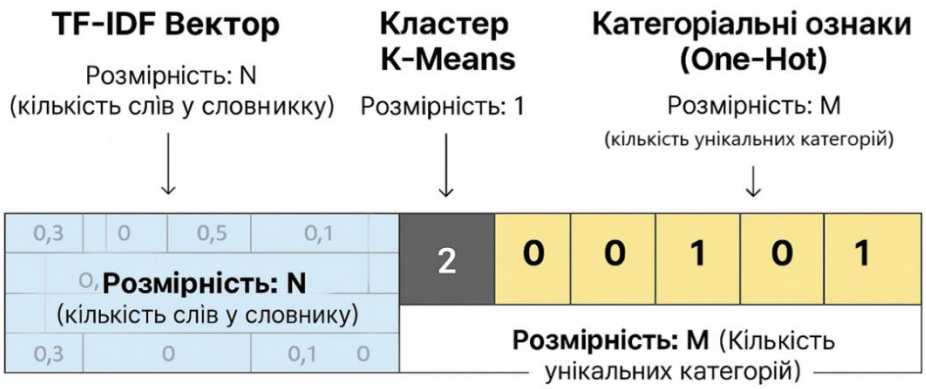
де $\hat{y}$ – фінальний прогноз ансамблю;

N – кількість дерев рішень в ансамблі (параметр n_estimators);

$\hat{y}_i(X)$ – прогноз i-го дерева рішень для вхідного вектора ознак X.

4

Структура фінального вектора ознак



Фінальний вектор ознак X

(Розмірність: N + 1 + M)

X = [TF-IDF_vector | K-Means_feature | Categorical_features]

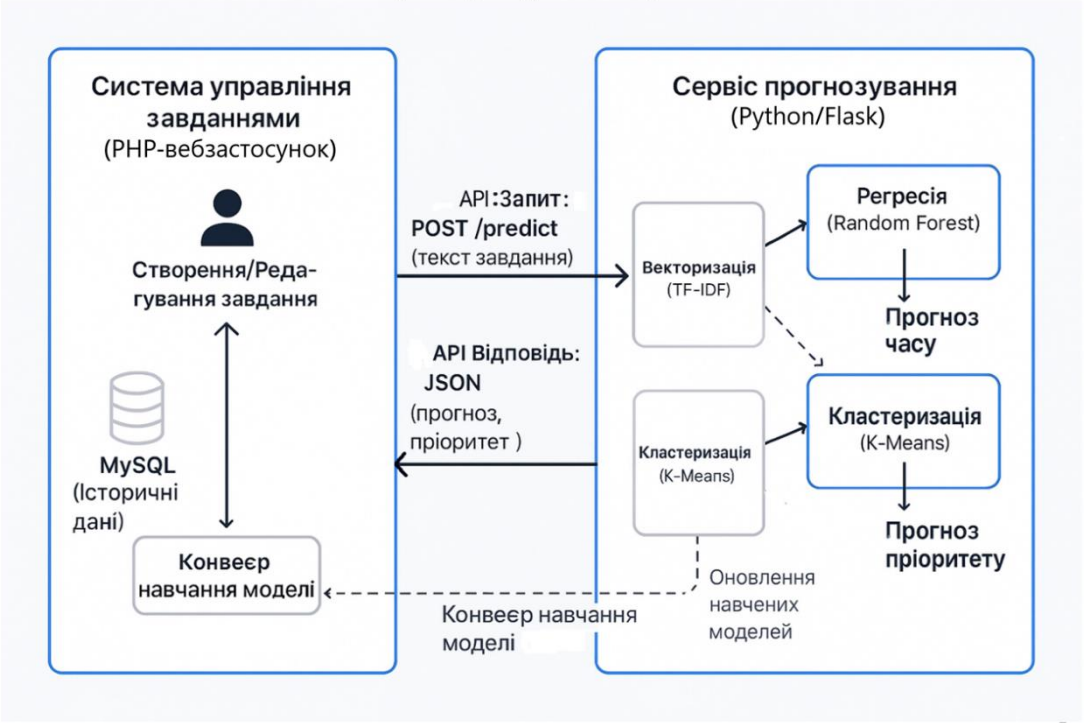
де TF-IDF_vector – це розріджений вектор, отриманий після векторизації текстового опису завдання;
K-Means_feature – це одна числова ознака, що відповідає номеру кластера, до якого було віднесено завдання;
Categorical_features – це набір бінарних ознак, отриманих після One-Hot Encoding формальних атрибутів завдання (наприклад, "Тип: Bug", "Пріоритет: High").

5

Алгоритм роботи комплексного методу

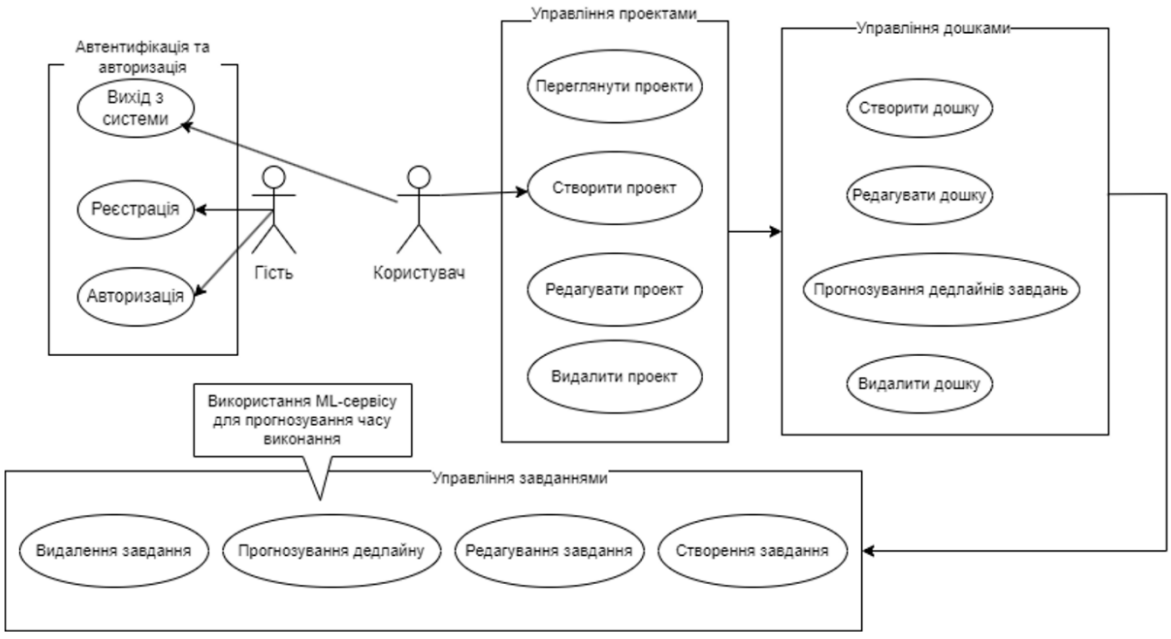
6

Архітектура взаємодії системи управління завданнями та сервісу прогнозування



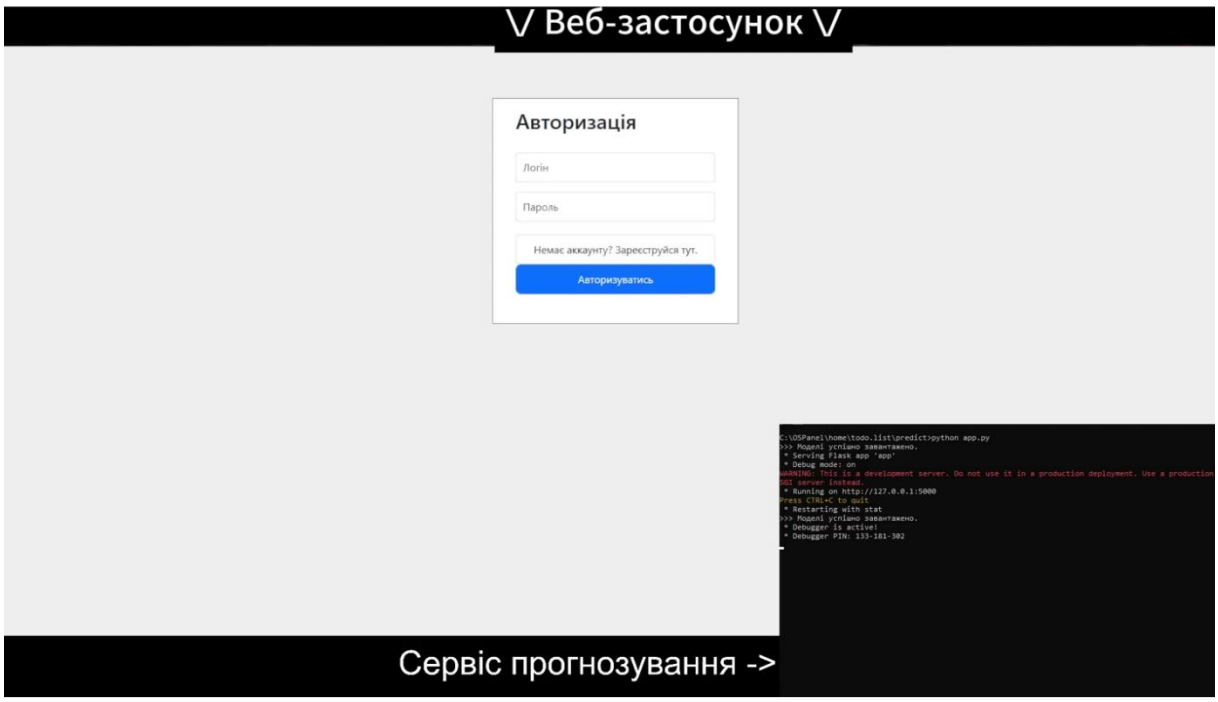
7

Діаграма Use-Case веб-застосунку системи управління завданнями



8

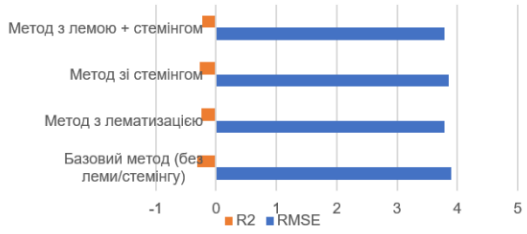
Практичний результат

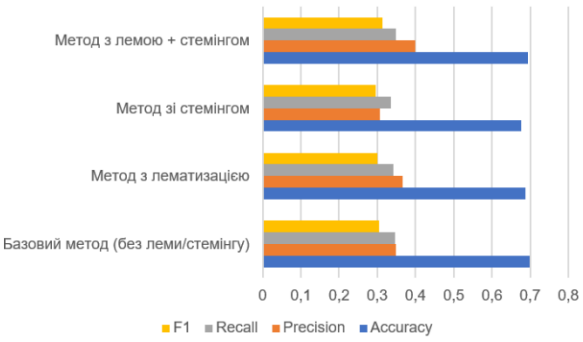


9

Порівняльний аналіз ефективності розробленої методики

Модель	MAE	MSE	RMSE	R^2	Accuracy	Precision	Recall	F1
Базовий метод	2,77	15,2	3,899	−0,31	0,69892	0,34814	0,3466	0,30393
З лематизацією	2,8	14,4	3,79	−0,24	0,68817	0,36537	0,3414	0,29933
З стемінгом	2,85	14,8	3,85	−0,28	0,67741	0,30759	0,3362	0,29629
Лематизація + стемінг	2,75	14,3	3,78	−0,23	0,69354	0,4	0,3493	0,31351





10

ВИСНОВКИ

1. Досліджено проблему прогнозування часу виконання завдань і їх пріоритизації, проаналізовано існуючі підходи та виявлено їх ключові обмеження.
2. Запропоновано комплексний метод, що поєднує TF-IDF, K-Means та Random Forest Regressor для автоматизованого прогнозування тривалості завдань.
3. Розроблено та впроваджено методику попередньої обробки тексту (нормалізація, токенізація, стоп-слова, лематизація, стемінг).
4. Проведено експериментальне порівняння кількох варіантів методу за метриками MAE, MSE, RMSE, R^2, точністю, повнотою та F1-мірою. Покращений метод в порівнянні з базовим:

– MAE зменшилась на ~0,02 (≈ −0,7%);
– MSE зменшилась на ~0,9 (≈ −5,9%);
– RMSE зменшилась на ~0,12 (≈ −3%);
– R^2 покращився на +0,08 (≈ 25,8%);

5. Методика може бути використана для інтеграції модулів прогнозування часу виконання завдань у системи управління проєктами.

ПУБЛІКАЦІЇ ТА АПРОБАЦІЯ РОБОТИ

Статті:

1. І.Ю. Коломієць, І.В. Замрій, Б. С. Калинюк, Ю. П. Бажан, Т. П. Довженко. Сучасні підходи до автоматизації управління завданнями для малих команд з використанням методів машинного навчання // Зв'язок, №3, 2025. С. 66-72.

Тези доповідей:

1. Коломієць І.Ю., Герцюк М.М. Використання машинного навчання для оптимізації управління завданнями в малих командах: підхід та інструменти // II міжнародна науково-практична конференція «Сучасні аспекти діджиталізації та інформатизації в програмній та комп'ютерній інженерії», 19-21 грудня 2024 р., Київ: ДУІКТ, 2024. С.245-246.
2. Коломієць І.Ю., Замрій І.В. Метод адаптивного прогнозування часу для автоматизації процесів розподілу завдань. // VIII Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ», 24 квітня 2025 р., Київ: ДУІКТ, 2025. С.288-291.

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
# app.py
from flask import Flask, request, jsonify
import pandas as pd
import joblib
import json
import os
from preprocessor import preprocess_text

app = Flask(__name__)

# --- Завантаження навчених моделей та конфігурації пріоритету при старті сервісу ---
try:
  vectorizer = joblib.load('models/tfidf_vectorizer.pkl')
  kmeans_model = joblib.load('models/kmeans_model.pkl')
  rf_model = joblib.load('models/random_forest_model.pkl')

  # Завантаження конфігурації пріоритету
  priority_config_path = 'models/priority_config.json'
  if os.path.exists(priority_config_path):
    with open(priority_config_path, 'r') as f:
      priority_config = json.load(f)
      priority_threshold_low = priority_config.get('threshold_low', 3.0)
      priority_threshold_high = priority_config.get('threshold_high', 7.0)
    print(f">>> Моделі успішно завантажено.")
  else:
    priority_threshold_low = 3.0
    priority_threshold_high = 7.0
    print(">>> Моделі завантажено, але конфігурація пріоритету не знайдена. Використовуються значення за замовчуванням.")
except FileNotFoundError:
  vectorizer, kmeans_model, rf_model = None, None, None
  priority_threshold_low, priority_threshold_high = 3.0, 7.0
  print("!!! ПОПЕРЕДЖЕННЯ: Файли моделей не знайдено. Запустіть training.py. !!!")

def calculate_priority(predicted_time_days):

  if predicted_time_days < priority_threshold_low:
    return 1 # Низький пріоритет
  elif predicted_time_days < priority_threshold_high:
    return 2 # Середній пріоритет
  else:
    return 3 # Високий пріоритет

# --- Ендпоінт для одного завдання ---
@app.route('/predict', methods=['POST'])
def predict_single():
  print("\n" + "="*80)
  print("Початок прогнозування для одного завдання")
  print("="*80)

  if not all([vectorizer, kmeans_model, rf_model]):
    print("ERROR: Моделі не завантажено!")
    return jsonify({"error": "Сервіс не готовий. Моделі не навчено."}), 503

  data = request.get_json()
  print(f"Отримано запит: {data}")

  if data and 'task' in data:
    # Старий формат - тільки опис
    task_text = data['task']
    print(f"Використано формат (тільки опис)")
  elif data and 'task_name' in data and 'task_description' in data:
    # Новий формат - назва + опис
    task_name = data.get('task_name', '')
    task_description = data.get('task_description', '')
    task_text = f"{task_name} {task_description}".strip()
    print(f"Використано формат (назва + опис)")
    print(f"  Назва: '{task_name}'")
    print(f"  Опис: '{task_description}'")
  else:
    print("ERROR: Неправильний формат запиту")
    return jsonify({"error": "Неправильний формат запиту. Потрібні поля 'task_name' та 'task_description' або 'task'."}), 400

  if not task_text:
    print("ERROR: Текст завдання порожній")
    return jsonify({"error": "Текст завдання не може бути порожнім."}), 400

  print(f"Об'єднаний текст завдання: '{task_text}'")
  print(f"  Довжина тексту: {len(task_text)} символів")

  # ЕТАП 1: Попередня обробка тексту
  print(f"\nПопередня обробка тексту (preprocess_text)...")
  processed_desc = preprocess_text(task_text)
  print(f"  Результат обробки: '{processed_desc}'")
  print(f"  Довжина після обробки: {len(processed_desc)} символів")

  # ЕТАП 2: Векторизація TF-IDF
  print(f"\nВекторизація тексту (TF-IDF)...")
  text_vector = vectorizer.transform([processed_desc])
  print(f"  Розмір вектора: {text_vector.shape}")
  print(f"  Кількість ненульових елементів: {text_vector.nnz}")

  # ЕТАП 3: Кластеризація K-Means
  print(f"\nВизначення кластера (K-Means)...")
  cluster = kmeans_model.predict(text_vector)[0]
  print(f"  Знайдено кластер: {cluster}")

  # ЕТАП 4: Формування фінальних ознак
  print(f"\nФормування фінальних ознак...")
  features_df = pd.DataFrame(text_vector.toarray(), columns=vectorizer.get_feature_names_out())
  features_df['cluster'] = cluster
  print(f"  Розмір матриці ознак: {features_df.shape}")
  print(f"  Кількість ознак: {len(features_df.columns)}")

  # ЕТАП 5: Прогнозування часу (Random Forest)
  print(f"\nПрогнозування часу виконання (Random Forest)...")
  predicted_time = rf_model.predict(features_df)[0]
  print(f"  Прогноз: {predicted_time} днів")

  # Визначення пріоритету на основі прогнозованого часу
  predicted_time_days = float(round(predicted_time, 2))
  priority = calculate_priority(predicted_time_days)

  print(f"\nФормування відповіді...")
  print(f"  Прогнозований час: {predicted_time_days} днів")
  print(f"  Пріоритет: {priority} ({'Низький' if priority == 1 else 'Середній' if priority == 2 else 'Високий'})")

  response = {
    "predicted_time_days": predicted_time_days,
    "predicted_priority": priority
```

```
  }

  print(f"Відповідь: {response}")
  print("="*80 + "\n")

  return jsonify(response)


# --- Ендпоінт для пакетної обробки завдань ---
@app.route('/predict-board', methods=['POST'])
def predict_board():
  print("\n" + "="*80)
  print("Початок пакетного прогнозування для дошки")
  print("="*80)

  if not all([vectorizer, kmeans_model, rf_model]):
    print("ERROR: Моделі не завантажено!")
    return jsonify({"error": "Сервіс не готовий. Моделі не навчено."}), 503

  data = request.get_json()
  print(f"Отримано запит з {len(data.get('tasks', [])) if data else 0} завданнями")

  # Очікуємо JSON формату {"tasks": [{"task_name": "...", "task_description": "..."}, ...]}
  if not data or 'tasks' not in data or not isinstance(data['tasks'], list):
    print("ERROR: Неправильний формат запиту")
    return jsonify({"error": "Неправильний формат. Очікується JSON з ключем 'tasks' та масивом об'єктів."}), 400

  tasks = data['tasks']
  if not tasks:
    print("WARNING: Отримано пустий список завдань")
    return jsonify([])

  print(f"Обробка {len(tasks)} завдань...")

  # Обробляємо всі тексти: об'єднуємо назву + опис
  task_texts = []
  for idx, task in enumerate(tasks):
    if isinstance(task, dict):
      task_name = task.get('task_name', '')
      task_description = task.get('task_description', '')
      task_text = f"{task_name} {task_description}".strip()
      print(f"  Завдання {idx + 1}: '{task_name}' + '{task_description[:50]}...'")
    elif isinstance(task, str):
      task_text = task
      print(f"  Завдання {idx + 1}: '{task_text[:50]}...'")
    else:
      task_text = ''
    task_texts.append(task_text)

  # ЕТАП 1: Попередня обробка тексту
  print(f"\nПопередня обробка всіх текстів (preprocess_text)...")
  processed_descs = [preprocess_text(text) for text in task_texts]
  for idx, (original, processed) in enumerate(zip(task_texts, processed_descs)):
    print(f"  Завдання {idx + 1}: '{original[:30]}...' -> '{processed[:30]}...'")

  # ЕТАП 2: Векторизація TF-IDF
  print(f"\nВекторизація всіх текстів (TF-IDF)...")
  text_vectors = vectorizer.transform(processed_descs)
  print(f"  Розмір матриці векторів: {text_vectors.shape}")
  print(f"  Загальна кількість ненульових елементів: {text_vectors.nnz}")

  # ЕТАП 3: Кластеризація K-Means
  print(f"\nВизначення кластерів для всіх завдань (K-Means)...")
  clusters = kmeans_model.predict(text_vectors)
  print(f"  Знайдені кластери: {clusters}")

  # ЕТАП 4: Формування фінальних ознак
  print(f"\nФормування фінальної матриці ознак...")
  features_df = pd.DataFrame(text_vectors.toarray(), columns=vectorizer.get_feature_names_out())
  features_df['cluster'] = clusters
  print(f"  Розмір матриці ознак: {features_df.shape}")
  print(f"  Кількість ознак: {len(features_df.columns)}")

  # ЕТАП 5: Прогнозування часу (Random Forest)
  print(f"\nПрогнозування часу виконання для всіх завдань (Random Forest)...")
  predicted_times = rf_model.predict(features_df)
  print(f"  Прогнози: {predicted_times}")

  # Формуємо масив результатів
  print(f"\nФормування результатів...")
  results = []
  priority_distribution = {1: 0, 2: 0, 3: 0} # Для статистики
  for i in range(len(task_texts)):
    # Перетворюємо numpy типи на Python типи для JSON серіалізації
    predicted_time_days = float(round(predicted_times[i], 2))
    # Визначення пріоритету на основі прогнозованого часу
    priority = calculate_priority(predicted_time_days)
    priority_distribution[priority] += 1
    priority_label = 'Низький' if priority == 1 else 'Середній' if priority == 2 else 'Високий'
    print(f"  Завдання {i + 1}: {predicted_time_days} днів, пріоритет {priority} ({priority_label})")
    results.append({
      "predicted_time_days": predicted_time_days,
      "predicted_priority": priority
    })


  print(f"Повертаємо {len(results)} результатів")
  print("="*80 + "\n")

  return jsonify(results)

# --- Запуск веб-сервера ---
if __name__ == '__main__':
  app.run(debug=True, port=5000)
import re
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import nltk
from nltk.tokenize import word_tokenize

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import (
  mean_absolute_error,
  mean_squared_error,
  r2_score,
```

```
  accuracy_score,
  precision_recall_fscore_support
)

import pymorphy3


try:
  nltk.data.find('tokenizers/punkt')
except LookupError:
  nltk.download('punkt')

morph = pymorphy3.MorphAnalyzer(lang='uk')

UKRAINIAN_STOP_WORDS = {
  'і', 'в', 'та', 'а', 'у', 'на', 'до', 'з', 'за', 'по', 'для', 'від', 'під',
  'над', 'про', 'чи', 'це', 'цей', 'ця', 'ці', 'що', 'як', 'він', 'вона',
  'воно', 'вони', 'ми', 'ви', 'ти', 'я', 'або', 'але', 'щоб', 'якщо',
  'був', 'була', 'було', 'були', 'є', 'буде'
}


# Датасет

data = [
  [1, "Налаштування репозиторію Git",
   "Створення віддаленого репозиторію додавання гілок налаштування гілки main та захисту від прямого пушу", 0.5],
  [2, "Ініціалізація проєкту PHP",
   "Створення структури проєкту налаштування autoload підключення базових залежностей через composer", 1],
  [3, "Реалізація форми логіну",
   "Розробка HTML форми логіну бекенд перевірка користувача по базі та обробка помилок авторизації", 1.5],
  …
]

df = pd.DataFrame(data, columns=["id", "title", "description", "actual_time_days"])


# Препроцесори тексту


def preprocess_base(text: str) -> str:
  return text

def preprocess_lemma(text: str) -> str:
  """Лематизація з pymorphy3."""
  text = text.lower()
  text = re.sub(r'[^а-яіїєґ\s]', ' ', text)
  tokens = word_tokenize(text)
  lemmas = []
  for t in tokens:
    if t in UKRAINIAN_STOP_WORDS or len(t) <= 1:
      continue
    lemmas.append(morph.parse(t)[0].normal_form)
  return " ".join(lemmas)

def simple_uk_stem(word: str) -> str:
  suffixes = ['ями', 'ями', 'ями', 'ові', 'еві', 'ами', 'ями',
        'ами', 'ями', 'ями', 'ями', 'ями', 'ями',
        'ими', 'ій', 'ий', 'им', 'их', 'ою', 'ему', 'ому',
        'ами', 'ями', 'ів', 'ів', 'ев', 'ем', 'ам', 'ах',
        'ою', 'ею', 'ю', 'я', 'у', 'і', 'а', 'е', 'о']
  for suf in sorted(suffixes, key=len, reverse=True):
    if word.endswith(suf) and len(word) > len(suf) + 1:
      return word[:-len(suf)]
  return word

def preprocess_stem(text: str) -> str:
  text = text.lower()
  text = re.sub(r'[^а-яіїєґ\s]', ' ', text)
  tokens = word_tokenize(text)
  stems = []
  for t in tokens:
    if t in UKRAINIAN_STOP_WORDS or len(t) <= 1:
      continue
    stems.append(simple_uk_stem(t))
  return " ".join(stems)

def preprocess_lemma_stem(text: str) -> str:
  text = text.lower()
  text = re.sub(r'[^а-яіїєґ\s]', ' ', text)
  tokens = word_tokenize(text)
  out = []
  for t in tokens:
    if t in UKRAINIAN_STOP_WORDS or len(t) <= 1:
      continue
    lemma = morph.parse(t)[0].normal_form
    stem = simple_uk_stem(lemma)
    out.append(stem)
  return " ".join(out)


# Перетворення регресії у класи


def time_to_classes(y):
  y = np.array(y)
  classes = np.zeros_like(y, dtype=int)
  classes[(y > 1.5) & (y <= 3)] = 1
  classes[y > 3] = 2
  return classes

def compute_metrics(y_true, y_pred):
  """
  MAE, MSE, RMSE, R2 + Accuracy, Precision_macro, Recall_macro, F1_macro.
  """
  mae = mean_absolute_error(y_true, y_pred)
  mse = mean_squared_error(y_true, y_pred)
  rmse = np.sqrt(mse)
  r2 = r2_score(y_true, y_pred)

  y_true_cls = time_to_classes(y_true)
  y_pred_cls = time_to_classes(y_pred)

  acc = accuracy_score(y_true_cls, y_pred_cls)
  precision, recall, f1, _ = precision_recall_fscore_support(
    y_true_cls, y_pred_cls, average='macro', zero_division=0
  )

  return {
    "MAE": mae,
    "MSE": mse,
    "RMSE": rmse,
    "R2": r2,
    "Accuracy": acc,
    "Precision_macro": precision,
    "Recall_macro": recall,
    "F1_macro": f1
  }


# Побудова та оцінка моделі
```

```
def build_and_evaluate_model(
  model_name: str,
  preprocess_func,
  X_train_text,
  X_test_text,
  y_train,
  y_test,
  random_state: int = 42
):
  """
  1) Препроцесинг тексту
  2) TF-IDF
  3) KMeans (cluster як ознака)
  4) RandomForestRegressor
  5) Повертає y_test, y_pred, dict метрик
  """
  # 1. Препроцесинг
  X_train_proc = [preprocess_func(t) for t in X_train_text]
  X_test_proc = [preprocess_func(t) for t in X_test_text]

  # 2. TF-IDF
  vectorizer = TfidfVectorizer(max_features=100)
  X_train_vec = vectorizer.fit_transform(X_train_proc)
  X_test_vec = vectorizer.transform(X_test_proc)

  # 3. KMeans кластеризація
  kmeans = KMeans(n_clusters=3, random_state=random_state,
n_init=10)
  cluster_train = kmeans.fit_predict(X_train_vec)
  cluster_test = kmeans.predict(X_test_vec)

  # 4. Формуємо фінальні ознаки
  X_train_df = pd.DataFrame(X_train_vec.toarray(),
columns=vectorizer.get_feature_names_out())
  X_test_df = pd.DataFrame(X_test_vec.toarray(),
columns=vectorizer.get_feature_names_out())

  X_train_df["cluster"] = cluster_train
  X_test_df["cluster"] = cluster_test

  # 5. Random Forest
  rf = RandomForestRegressor(
    n_estimators=100,
    random_state=random_state
  )
  rf.fit(X_train_df, y_train)
  y_pred = rf.predict(X_test_df)

  # Метрики
  metrics = compute_metrics(y_test, y_pred)
  metrics["model"] = model_name

  return y_test, y_pred, metrics



if __name__ == "__main__":
  # Формуємо текст та ціль
  X_text = df["description"].values
  y = df["actual_time_days"].values

  # Один і той самий train/test split для всіх методів
  X_train_text, X_test_text, y_train, y_test = train_test_split(
    X_text, y, test_size=0.3, random_state=42
  )

  results = []
  preds_store = {}

  # 1) Базовий метод
  y_test_base, y_pred_base, m_base =
build_and_evaluate_model(
    "Базовий метод (без леми/стемінгу)",
    preprocess_base,
    X_train_text, X_test_text,
    y_train, y_test
  )
  results.append(m_base)
  preds_store["base"] = y_pred_base

  # 2) Лематизація
  _, y_pred_lemma, m_lemma = build_and_evaluate_model(
    "Метод з лематизацією",
    preprocess_lemma,
    X_train_text, X_test_text,
    y_train, y_test
  )
  results.append(m_lemma)
  preds_store["lemma"] = y_pred_lemma

  # 3) Стемінг
  _, y_pred_stem, m_stem = build_and_evaluate_model(
    "Метод зі стемінгом",
    preprocess_stem,
    X_train_text, X_test_text,
    y_train, y_test
  )
  results.append(m_stem)
  preds_store["stem"] = y_pred_stem

  # 4) Лема + стемінг
  _, y_pred_lemma_stem, m_lemma_stem =
build_and_evaluate_model(
    "Метод з лемою + стемінгом",
    preprocess_lemma_stem,
    X_train_text, X_test_text,
    y_train, y_test
  )
  results.append(m_lemma_stem)
  preds_store["lemma_stem"] = y_pred_lemma_stem

  # Підсумкова таблиця метрик


  df_metrics = pd.DataFrame(results)[[
    "model", "MAE", "MSE", "RMSE", "R2",
    "Accuracy", "Precision_macro", "Recall_macro",
"F1_macro"
  ]]

  print(df_metrics)


  os.makedirs("reports", exist_ok=True)
  df_metrics.to_csv("reports/metrics_table_tf_idf_variants.csv",
index=False, encoding="utf-8-sig")
# preprocessor.py
import re
import nltk
from nltk.tokenize import word_tokenize
import pymorphy3

try:
  nltk.data.find('tokenizers/punkt')
except LookupError:
  nltk.download('punkt')

morph = pymorphy3.MorphAnalyzer(lang='uk')
```

```
UKRAINIAN_STOP_WORDS = [
  'і', 'в', 'та', 'а', 'у', 'на', 'до', 'з', 'за', 'по', 'для', 'від', 'під', 'над',
  'про', 'чи', 'це', 'цей', 'ця', 'ці', 'що', 'як', 'він', 'вона', 'воно',
'вони',
  'ми', 'ви', 'ти', 'я', 'або', 'але', 'щоб', 'якщо', 'був', 'була', 'було',
  'були', 'є', 'буде'
]

def preprocess_text(text: str) -> str:
  """Реалізує конвеєр попередньої обробки тексту з
лематизацією для української мови."""
  text = text.lower()
  text = re.sub(r'[^а-яіїєґ\s]', '', text)
  tokens = word_tokenize(text)

  lemmatized_tokens = [morph.parse(token)[0].normal_form for
token in tokens if token not in UKRAINIAN_STOP_WORDS]

  return " ".join(lemmatized_tokens)
```