НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці

ДО ЗАХИСТУ ДОПУЩЕНО
В.о. завідувача кафедри
_____ Олександр КОВАЛЬ
«___»_____2026 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем в енергетиці»
спеціальності 121 Інженерія програмного забезпечення

на тему: «Програмне забезпечення для динамічних мімічних реакцій 3D-аватара на основі машинного навчання»

Виконав:
студент IV курсу, групи ТВ-21
Лазарчук Артем Сергійович _____ _____
(прізвище, ім'я, по батькові) (підпис)

Керівник:
Завідувач кафедри, д.т.н., професор, Коваль О.В. _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент:
завідувач кафедри ІТЕ КНЕУ, к.т.н., доцент, Тишковський Б.О. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень із праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2026

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем в енергетиці»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Олександр КОВАЛЬ
«___»_____2026р.

ЗАВДАННЯ

на дипломну роботу студенту

Лазарчуку Артему Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи: Програмне забезпечення для динамічних мімічних реакцій 3D-аватара на основі машинного навчання
керівник роботи Коваль Олександр Васильович, доктор технічних наук, професор
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «28» травня 2026 р. № 1992

2. Строк подання студентом роботи «10» червня 2026 р.

3. Вихідні дані до роботи: мова програмування Python, бібліотека scikit-learn, бібліотека NumPy, вебфреймворк FastAPI, нереляційна база даних MongoDB, мовна модель Together API, бібліотека trafilatura, середовище розробки Visual Studio Code.

4. Зміст (дипломної роботи) пояснювальної записки (перелік завдань, які потрібно розробити): Дослідити концепції емоційного штучного інтелекту та створити модуль відтворення анімацій аватара. Розробити гібридний метод аналізу тексту та математичну модель розпізнавання емоцій користувача. Реалізувати асинхронне серверне програмне забезпечення з інтеграцією бази знань кафедри.

5. Перелік ілюстративного матеріалу: актуальність теми, постановка завдання, аналіз існуючих систем, архітектура модуля анімацій, проєктування гібридного методу аналізу тексту, технічна реалізація серверної частини, оцінка ефективності моделі, висновки.

6. Дата видачі завдання «31» жовтня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Строки виконання етапів роботи	Примітка
1	Отримання завдання	31.10.2025	
2	Дослідження предметної області	01.11.2025 – 31.11.2025	
3	Дослідження існуючих рішень	01.12.2025 – 31.12.2025	
4	Постановка вимог до проєктування системи	01.01.2026 – 31.01.2026	
5	Розробка програмного продукту	01.02.2026 – 03.05.2026	
6	Тестування	04.05.2026 – 10.05.2026	
7	Захист програмного продукту	11.05.2026 – 15.05.2026	
8	Оформлення дипломної роботи	18.05.2026 – 31.05.2026	
9	Передзахист	01.06.2026 – 05.06.2026	
10	Захист	15.06.2026 – 19.06.2026	

Студент _____ Артем ЛАЗАРЧУК
(підпис) (ім’я, ПРІЗВИЩЕ)

Керівник роботи _____ Олександр КОВАЛЬ
(підпис) (ім’я, ПРІЗВИЩЕ)

РЕФЕРАТ

Структура та обсяг дипломної роботи. Робота містить 54 сторінок, 9 рисунків, 3 додатки та 13 посилань.

Метою роботи є проєктування, реалізація та аналіз ефективності віртуального 3D-асистента для сайту кафедри. Програмне забезпечення автоматично аналізує текст користувача, розпізнає його емоції, знаходить відповіді в базі знань за допомогою RAG-архітектури та динамічно керує мімікою аватара в реальному часі.

Практичне значення одержаних результатів полягає у створенні оптимізованого асинхронного програмного комплексу для автоматизації інформаційної підтримки користувачів. Завдяки використанню рушія відеофрагментів (Video Engine) у форматі WebM, система забезпечує миттєву зміну міміки аватара в реальному часі без залучення дорогих серверних відеокарт (GPU) та без навантаження на мобільні пристрої студентів.

Апробація результатів дипломної роботи. Основні положення роботи обговорювалися на конференції:

1. Арсенюк Д.В., Лазарчук А.С., Поліщук М.М., Сарибога Г.В. Програмне забезпечення інтелектуальної селекції мімічних реакцій 3D-аватара на основі гібридного аналізу природної мови. *Матеріали VII Міжнародної науково-практичної конференції молодих вчених, аспірантів і студентів "Сучасні інформаційні технології та системи в управлінні"*. Київ, 28–29 квітня 2026 р. С. 461-462.

URL:https://drive.google.com/file/d/19dd47KHOOOWwFi6b6zNMEw8rFlFWF6LG/edit

Ключові слова: 3D-АВАТАР, МАШИННЕ НАВЧАННЯ, FastAPI, MongoDB, ТЕКСТОВИЙ КОНВЕЄР, АФЕКТИВНІ ОБЧИСЛЕННЯ, WebM, ГІБРИДНИЙ АНАЛІЗ, РОЗПІЗНАВАННЯ ЕМОЦІЙ, RAG.

ABSTRACT

Structure and scope of the thesis. The work contains 54 pages, 9 figure, 3 appendices, and 13 references.

The purpose of the work is the design, implementation, and performance analysis of a virtual 3D assistant for the department's website. The software automatically analyzes user text, recognizes emotions, retrieves answers from the knowledge base using RAG architecture, and dynamically controls the avatar's facial expressions in real time.

The practical value is to create an optimized asynchronous software package for automating user information support. Thanks to the use of a video engine in WebM format, the system ensures instantaneous changes in the avatar's facial expressions in real time without the need for expensive server-side graphics cards (GPUs) and without putting a computational load on students' mobile devices.

Approbation of the results of the thesis. The main provisions and results of the work were discussed at the international scientific and practical conference:

1. Arseniuk, D. V., Lazarchuk, A. S., Polishchuk, M. M., and Saryboha, H. V. (2026). Software for intellectual selection of 3D avatar facial reactions based on hybrid natural language analysis. *Proceedings of the VII International Scientific and Practical Conference of Young Scientists, Postgraduates, and Students "Modern Information Technologies and Systems in Management"*. Kyiv, April 28–29, 2026, pp. 461-462.

URL:https://drive.google.com/file/d/19dd47KHOOOWwFi6b6zNMEw8rFlFWF6LG/edit

Keywords: 3D AVATAR, MACHINE LEARNING, FastAPI, MongoDB, TEXT PIPELINE, AFFECTIVE COMPUTING, WebM, HYBRID ANALYSIS, EMOTION RECOGNITION, RAG.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ШІ(AI)	Штучний інтелект
МН(ML)	Машинне навчання
API	Інтерфейс прикладного програмування (Application Programming Interface)
NLP	Обробка природної мови (Natural Language Processing)
RAG	Генерація з доповненим пошуком (Retrieval-Augmented Generation)
TF-IDF	Статистична міра для оцінки важливості слова в тексті (Term Frequency – Inverse Document Frequency)
Softmax	Математична функція, що перетворює сирі бали у вектор ймовірностей
WebM	Відкритий і безкоштовний медіаформат відеофайлів, розроблений компанією Google
VP9	Сучасний і ефективний кодек стиснення відео даних
FastAPI	Високопродуктивний асинхронний вебфреймворк на мові Python
UML	Уніфікована мова моделювання для графічного опису структури та процесів ПЗ
CPU	Центральний процесор (Central Processing Unit)
GPU	Графічний процесор / відеокарта (Graphics Processing Unit)
SSD	Твердотільний накопичувач даних (Solid-State Drive)
JSON	Текстовий формат обміну даними, що використовується для збереження документів у MongoDB
SSE	Технологія потокової передачі даних від сервера до клієнта в реальному часі (Server-Sent Events)

ВСТУП

Актуальність теми. Сучасний етап розвитку інформаційних технологій характеризується стрімким переходом від статичних людино-машинних інтерфейсів до інтелектуальних людино-орієнтованих систем. Особливої актуальності в межах людино-машинної взаємодії (Human-Machine Interaction) набуває концепція афективних обчислень (Affective Computing), яка передбачає створення програмного забезпечення, здатного розпізнавати, інтерпретувати та імітувати людські емоції[1].

Традиційні вебсервіси, зокрема інформаційні ресурси кафедр, зазвичай обмежуються суто текстовим обміном інформацією. Це створює ефект «інформаційної сухості» та підвищує когнітивне навантаження на користувача. Впровадження візуальних ШІ-агентів, здатних до динамічних мімічних реакцій у реальному часі, дозволяє гуманізувати інтерфейси та забезпечити психологічний комфорт під час діалогу. Створення архітектурних рішень, що поєднують аналіз природної мови (NLP) з ефективним управлінням медіапотоками високої чіткості, є важливим інженерним завданням для підвищення якості дистанційної комунікації в освітньому середовищі[2].

Об’єктом дослідження є архітектура серверної частини інтелектуального асистента та математичні методи обробки емоційної текстової інформації.

Предметом дослідження виступають методи машинного навчання для багатокласової класифікації емоцій, гібридні конвеєри аналізу природної мови, стохастичні алгоритми селекції анімаційних фрагментів та асинхронна серверна логіка (API) для синхронізації станів 3D-аватара з контекстом розмови.

Метою роботи є розробка та програмна реалізація інтелектуального ядра системи (assistant_core), яке забезпечує автоматизований аналіз текстових повідомлень користувача, формування релевантних відповідей на основі бази знань та динамічне управління візуальними станами 3D-аватара у режимі реального часу.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

1. Опрацювати науково-техничну літературу щодо концепцій емоційного штучного інтелекту та принципів афективної людино-машинної взаємодії.

2. Спроєктувати архітектуру модуля відтворення WebM-анімацій (Video Engine) з підтримкою альфа-каналу та механізму асинхронної буферизації.

3. Розробити та натренувати ML-модуль на основі алгоритмів машинного навчання для семантичного аналізу тексту й визначення емоційного забарвлення.

4. Спроєктувати та реалізувати асинхронну серверну частину системи з інтеграцією архітектури Retrieval-Augmented Generation (RAG) та потокової передачі даних.

5. Провести комплексне тестування швидкодії розробленого програмного продукту та оцінити якість роботи математичних моделей.

Методи дослідження. Для розв'язання поставлених завдань використано методи математичної статистики та аналізу природної мови (зокрема, статистичне зважування TF-IDF), алгоритми машинного навчання (багатокласова логістична регресія), ймовірнісне моделювання (функція Softmax та марківські принципи матриць переходів), а також принципи асинхронного та багатопоточного програмування для побудови серверних систем реального часу.

Практичне значення отриманих результатів полягає у створенні готового до розгортання програмного комплексу інтелектуального асистента, який може бути інтегрований у вебресурси для автоматизації інформаційної підтримки користувачів, забезпечуючи високу швидкість відгуку та природну візуальну взаємодію без надлишкового навантаження на клієнтське обладнання.

1 ПОСТАНОВКА ЗАВДАННЯ РОЗРОБКИ СИСТЕМИ ДИНАМІЧНИХ РЕАКЦІЙ 3D-АВАТАРА

Сьогодні звичайні текстові чати на сайтах уже застаріли, тому розробники намагаються створювати візуальних цифрових помічників. Якщо додати на сайт тривимірного аватара, користувачам буде набагато цікавіше й приємніше спілкуватися. Проте зробити так, щоб цей аватар не просто відповідав текстом, а ще й миттєво показував правильні емоції на обличчі під час розмови — це складне завдання. У цьому розділі ми чітко розберемо, яку мету має цей проєкт, з яких етапів складається його розробка та з якими головними технічними труднощами довелося зіткнутися під час створення такої програми.

1.1 Мета

Головною метою бакалаврської роботи є проєктування, програмна реалізація та дослідження ефективності системи віртуального асистента у вигляді тривимірного аватара для сайту кафедри інженерії програмного забезпечення в енергетиці (ІПЗЕ). Розроблюване програмне забезпечення має повністю автоматично аналізувати повідомлення користувача, розпізнавати його поточний емоційний настрій, знаходити офіційні відповіді у базі знань кафедри та динамічно керувати виразом обличчя цифрового персонажа безпосередньо під час діалогу для забезпечення максимального комфорту користувачів.

Щоб повністю досягти поставленої мети, система повинна вирішувати три основні завдання (підзадачі):

- Розумний аналіз вхідного тексту: реалізація алгоритмів штучного інтелекту для миттєвого розпізнавання психоемоційного стану людини (наприклад, радість, сум, здивування, гнів, огида, задумливість або нейтральний спокій) виключно на основі надісланого тексту.

- Пошук інформації та генерація відповідей: розробка модуля, який забезпечує швидкий пошук актуальних даних у базі знань кафедри (про розклад, предмети, правила вступу, заселення тощо) та формування точної відповіді за допомогою сучасної мовної моделі без вигадування неіснуючих фактів.
- Оживлення персонажа без затримок: побудова логіки миттєвого підбору рухів обличчя для 3D-моделі на основі розрахованої емоції та передача цих команд на екран сайту синхронно з появою тексту відповіді.

1.2 Етапи реалізації

Процес створення програмного забезпечення для динамічних мімічних реакцій 3D-аватара на основі машинного навчання планується розділити на п'ять послідовних та взаємопов'язаних етапів:

1. Аналітичний етап, збір даних та дослідження афективних обчислень. Необхідно провести глибоке вивчення концепцій афективних обчислень (Affective Computing) та існуючих теорій розпізнавання людських емоцій комп'ютерними системами. На цьому етапі планується зібрати й структурувати масив текстових повідомлень і емоційно-маркованих лексем українською мовою для навчання класифікатора. Також потрібно підготувати та впорядкувати офіційні матеріали кафедри ІПЗЕ для формування надійної бази знань.

2. Проєктування математичного забезпечення та алгоритмів аналізу природної мови. Цей етап має стати теоретичною основою для семантичного аналізу тексту. Необхідно спроєктувати трирівневий гібридний конвеєр обробки тексту, описати математичні процеси перетворення слів у цифрові вектори, обчислення ймовірності емоцій за допомогою логістичної регресії та функції Softmax, а також побудувати матриці переходів для управління пулами відеофайлів аватара.

3. Розробка асинхронної серверної архітектури. Для забезпечення роботи системи в реальному часі необхідно спроєктувати та реалізувати

внутрішню (серверну) частину на мові Python із використанням асинхронного підходу. Потрібно налаштувати базу даних MongoDB для збереження контексту розмов та інтегрувати API сучасної мовної моделі DeepSeek-V3 для динамічної генерації відповідей[3].

4. Зв'язок із вебплатформою та реалізація модуля візуалізації. На цьому етапі необхідно запрограмувати механізм потокової передачі даних від сервера до сайту для мінімізації часу очікування. Головне завдання — розробити спеціалізований візуальний рушій (Video Engine), який прийматиме команди від сервера та відтворюватиме полегшені WebM-відеофайли з прозорим альфа-каналом прямо на сторінці браузера.

5. Тестування, оцінка якості штучного інтелекту та оптимізація. Фінальний етап передбачає оцінку працездатності системи за допомогою стандартних метрик якості класифікації (Precision, Recall, F1-score) з бібліотеки Scikit-learn.

1.3 Проблеми в реалізації

Під час проєктування та створення системи динамічних мімічних реакцій 3D-аватара очікується виникнення низки складних інженерних та архітектурних викликів, які необхідно вирішити в ході розробки:

1. Проблема затримок в інтернет-мережі: Послідовне виконання операцій (аналіз тексту, пошук у базі даних, генерація тексту мовною моделлю, запуск відео) може призвести до затримок у кілька секунд, що зруйнує відчуття живого діалогу. Шлях вирішення: планується налаштувати сервер на асинхронну роботу та впровадити технологію Server-Sent Events (SSE) для передачі тексту на сайт літера за літерою, щоб аватар починав реагувати ще до завершення генерації повної відповіді[4].

2. Проблема монотонності та неприродності рухів персонажа: Якщо цифровий помічник реагуватиме на однакові емоційні тригери абсолютно

ідентично, його поведінка здаватиметься занадто циклічною, штучною та механістичною, що негативно вплине на користувацький досвід. Шлях вирішення: необхідно розробити алгоритм, який розраховуватиме виразність емоції за трьома рівнями (слабкий, помірний, виражений) та звертатиметься до відповідних пулів анімаційних файлів із варіативними мікрорухами голови й очей для усунення одноманітності.

3. Графічні дефекти та блимання відео при зміні настрою: Пряма заміна відеофайлів у браузері супроводжується миттєвим зникненням або блиманням картинки в момент завантаження нового файлу. Шлях вирішення: візуальний рушій (Video Engine) має бути побудований на принципі подвійної буферизації з паралельним використанням двох шарів відтворення. Наступне відео має наперед асинхронно підвантажуватися у фоні.

4. Обмеженість обчислювальних потужностей: Використання важких нейромереж для розпізнавання мови потребує дорогого обладнання з GPU, що є технічно й економічно недоцільним для звичайних сайтів навчальних кафедр. Шлях вирішення: основним викликом є створення оптимізованого й «легкого» математичного ядра класифікатора.

Висновки до розділу 1

У першому розділі виконано постановку інженерного завдання з розробки програмного забезпечення 3D-аватара для сайту кафедри. Визначено мету проєкту, що полягає у створенні віртуального помічника, здатного вести діалог і візуалізувати емоції в реальному часі. Сформовано п'ять етапів реалізації роботи та проаналізовано ключові технічні проблеми: інтернет-затримки, монотонність рухів, блимання відеоплеєра й обмеженість серверних ресурсів. Для кожного виклику обґрунтовано концептуальні шляхи вирішення, які формують технічні вимоги до системи.

2 АНАЛІЗ ІСНУЮЧИХ ТЕХНОЛОГІЙ ЕМОЦІЙНОГО ШТУЧНОГО ІНТЕЛЕКТУ ТА ВІЗУАЛІЗАЦІЇ

Перш ніж переходити до написання коду та математичних розрахунків, детально дослідив існуючі технології, підходи та світові стандарти у сфері цифрової візуалізації та розпізнавання емоцій. Розробка віртуального помічника вимагала аналізу трьох важливих складових: теоретичних основ взаємодії людини з комп'ютером, архітектурних рішень для створення штучних агентів та сучасних інструментів для якісного відтворення відео у браузері. Цей розділ присвячено огляду концепцій емоційного штучного інтелекту, порівнянню існуючих методів побудови цифрових асистентів та обґрунтуванню вибору найкращих технологій для проєкту.

2.1 Концепції Affective Computing та моделі людино-машинної взаємодії

Концепція афективних обчислень (Affective Computing), яка досліджує методи розпізнавання, інтерпретації та імітації людських емоцій комп'ютерними системами, є базовим фундаментом для проектування сучасних людино-машинних інтерфейсів, більшість традиційних чат-систем, які використовуються на освітніх ресурсах, є емоційно «сліпими», вони обмежуються суто сухим обміном текстом, через що користувач часто стикається з ефектом «інформаційної сухості»[5]. Людина не відчуває зворотного зв'язку про те, як система сприймає її запит, що суттєво підвищує психологічне та когнітивне навантаження під час діалогу.

Впровадження динамічних мімічних реакцій через тривимірного аватара дозволяє ефективно вирішити цю проблему. Візуалізація внутрішніх станів

штучного інтелекту — наприклад, відображення «задумливості» під час пошуку інформації або «нейтральної уважності» під час введення тексту — створює відчуття присутності реального співрозмовника.

Для того щоб візуальний аватар міг природно реагувати на репліки, система повинна вміти перетворювати текст користувача на чіткі емоційні команди. У межах цього проєкту розпізнавання емоцій базується на класифікації повідомлень за сімома базовими емоційними категоріями: нейтральний стан, радість, сум, здивування, задумливість, гнів та огида. Головна інженерна складність на етапі аналізу полягає в тому, що люди виражають свої думки дуже різноманітно, і емоції в тексті рідко бувають очевидними на сто відсотків.

Провівши аналіз існуючих світових технологій обробки природної мови, можна виділити два основні підходи, кожен з яких має свої особливості:

1. Методи на основі чітких лінгвістичних правил (Rule-based підхід): ці системи працюють за принципом прямого пошуку емоційних слів-маркерів, специфічних знаків пунктуації чи смайликів за допомогою готових словників і регулярних виразів[6]. Їхньою головною перевагою є надзвичайно висока швидкість роботи та точність при обробці коротких фраз, очевидних вигуків або специфічних текстових сигналів (наприклад, використання верхнього регістру CAPS LOCK чи штучного подовження слів). Проте суттєвим недоліком є повна відсутність гнучкості: якщо користувач побудує складне, розлоге за структурою речення або використає слова, яких немає у словнику, система не зможе зрозуміти загальний контекст і припуститься помилки.

2. Статистичні методи машинного навчання (Machine Learning підхід): в основі цього підходу лежить аналіз частоти вживання слів та прихованих статистичних зв'язків між ними у всьому тексті, що дозволяє виявляти тонкі емоційні підтексти без прямого використання лексичних словників[7]. Головна перевага полягає у високій стійкості до варіативності мовлення та здатності успішно знаходити прихований зміст у великих і складних реченнях. Водночас класичні моделі машинного навчання мають і свої мінуси: вони часто

припускаються помилок на занадто коротких фразах, де просто недостатньо слів для збору статистичних даних, а також можуть повністю ігнорувати важливі невербальні знаки, такі як емодзі чи знаки питання.

Проведений порівняльний аналіз показує, що використання лише одного з цих підходів не дозволить досягти необхідної точності системи в реальному часі. Наприклад, якщо користувач напише коротке «Огоооо!», статистична модель може розгубитися через брак слів, тоді як модуль правил миттєво визначить «здивування» за подовженням слова та знаком оклику. І навпаки, у розлогих офіційних запитах правила виявляться безсилими, а машинне навчання легко впорається із завданням.

З огляду на це, найбільш оптимальним архітектурним рішенням для проєкту є проєктування гібридного методу аналізу природної мови. Пропонується створити послідовний конвеєр (пайплайн) обробки, який об'єднає три ієрархічні рівні:

- Лексичний рівень для швидкого пошуку очевидних маркерів та емодзі за словником;
- Структурний рівень для аналізу розділових знаків, патернів та регістру тексту;
- Статистичний рівень машинного навчання (на базі алгоритмів логістичної регресії) для фінального розрахунку ймовірнісного розподілу емоцій за функцією Softmax.

Така гібридна структура дозволить системі взаємно компенсувати недоліки окремих методів, забезпечить високу стійкість до будь-яких помилок чи «шумів» у тексті та гарантуватиме, що візуальний аватар точно й безпомилково підбиратиме міміку під настрій розмови.

2.2 Архітектурні підходи до побудови візуальних ШІ-агентів

При розробці віртуальних асистентів ключовим завданням є вибір системної архітектури, яка зазвичай буває локальною або гібридною хмароорієнтованою. Локальний підхід передбачає розгортання всіх модулів штучного інтелекту на власному сервері, що вимагає купівлі дорогого обладнання з потужними графічними картами (GPU). Для звичайних сайтів навчальних кафедр це економічно та технічно недоцільно.

Саме тому для проєкту обрано гібридний хмароорієнтований підхід. За такої архітектури сервер кафедри виконує лише роль легкого координатора, а генерація відповідей переноситься на готові зовнішні хмарні платформи через інструменти API. Використання сучасної мовної моделі DeepSeek-V3 через інженерні шлюзи Together API дозволяє отримувати точні тексти майже миттєво, не перевантажуючи локальний сервер[8].

Головним недоліком використання великих мовних моделей є ефект «галюцинацій», коли штучний інтелект починає вигадувати неіснуючі факти. Щоби повністю вирішити цю проблему, в архітектуру системи впроваджується захисний шаблон RAG (генерація з доповненим пошуком). Ця технологія працює як розумний фільтр: коли користувач надсилає запит, система спочатку робить миттєвий пошук у локальній базі даних MongoDB, де зберігаються офіційні документи та розклад кафедри ІПЗЕ. Знайдені точні факти автоматично прикріплюються до запиту як обов'язковий контекст, і лише після цього сформований пакет даних надсилається до мовної моделі DeepSeek-V3.

Така гібридна архітектура з інтеграцією RAG-модуля гарантує, що віртуальний аватар формуватиме відповіді виключно на основі офіційних першоджерел, повністю ліквідує ризик вигадування інформації й дозволить системі стабільно працювати на звичайних процесорних потужностях (CPU).

2.3 Порівняльний аналіз існуючих рішень та аналогічних проєктів

Під час розробки віртуального помічника важливо подивитися, які схожі проєкти вже існують у світі, які технології вони використовують та з якими проблемами стикаються. Сьогодні на ринку цифрових аватарів можна виділити три основні інженерні підходи:

1. Комерційні генератори відео (наприклад, Synthesia): ці платформи створюють дуже реалістичних цифрових людей, які красиво говорять і рухаються[9]. Проте вони створені суто для генерації готових статичних відеороликів (наприклад, для презентацій чи реклами). Вони взагалі не вміють вести діалог із користувачем, а сама система є повністю закритою та вимагає дорогої щомісячної підписки, що не підходить для безкоштовного сайту кафедри.

2. Важка ігрова 3D-графіка (наприклад, Unreal Engine MetaHuman): це рішення дає неймовірну фотоякість і реалістичність рухів обличчя. Але для роботи такого аватара потрібен постійний складний прорахунок графіки на потужних ігрових серверах із відеокартами (GPU), звідки картинка транслюється на екран користувача як постійний стрім. Оренда таких серверів для цілодобової роботи сайту кафедри ІПЗЕ є фінансово неможливою.

3. Браузерна 3D-графіка через скрипти (WebGL-асистенти): цей підхід часто використовують у банках чи інтернет-магазинах. 3D-модель завантажується прямо на сайт і рухається за допомогою коду. Головний мінус цього підходу — колосальне навантаження на телефон чи комп'ютер користувача. Якщо студент відкриє такий сайт зі звичайного або слабкого смартфона, аватар буде сильно зависати, а сам телефон почне перегріватися через постійну роботу процесора.

Щоб уникнути фінансових витрат на дорогі сервери з GPU та повністю захистити смартфони користувачів від перегріву й зависань, у цьому проєкті обрано зовсім інший, хитрий інженерний шлях — рушій попередньо записаних

відеофрагментів (Video Engine), який працює в покроковому (асинхронному) режимі «запит-відповідь».

Замість того, щоб змушувати комп'ютер користувача прораховувати мільйони полігонів 3D-моделі в реальному часі, аватар заздалегідь записаний у високій якості у вигляді коротких роликів для кожної емоції та кожного рівня її виразності. Коли користувач надсилає питання, система аналізує текст, знаходить інформацію, визначає емоцію відповіді й підвантажує потрібний готовий відеофайл.

Для наочності всі три існуючі підходи та рішення, що розробляється в роботі, порівняно на рисунку 2.1 за чотирма головними критеріями.

Назва підходу / Критерій	Формат взаємодії з користувачем	Навантаження на сервер (вимоги до GPU)	Навантаження на телефон/ПК користувача	Вартість реалізації та підтримки
1. Комерційні генератори (Synthesia)	Статичне відео (без діалогу)	Відсутнє	Відсутнє	Дуже висока (платна підписка)
2. Важка 3D-графіка (MetaHuman)	Потоковий стрім (у реальному часі)	Надзвичайно високе	Низьке	Величезна (оренда GPU-серверів)
3. Браузерні скрипти (WebGL)	Інтерактивний діалог (у реальному часі)	Відсутнє	Надзвичайно високе	Середня (потрібна складна 3D-оптимізація)
4. Розроблюване рішення (Video Engine)	**Інтерактивний покроковий діалог**	**Відсутнє (працює на звичайному CPU)**	**Низьке (як перегляд звичайного відео)**	**Низька (повністю безкоштовні open-source інструменти)**

Рисунок 2.1 - Порівняльний аналіз архітектурних підходів до візуалізації аватарів.

Як чітко видно з таблиці, розроблюваний підхід є найбільш збалансованим і практичним рішенням для навчальної кафедри. Відмова від складних технологій реального часу дозволила повністю зняти навантаження як із сервера кафедри ІПЗЕ, так і з мобільних телефонів студентів. Використання безкоштовного формату WebM від Google із підтримкою прозорого фону забезпечує ідеальну якість міміки аватара, яка працює стабільно, безкоштовно і запускається навіть на найстаріших мобільних пристроях.

Висновки до розділу 2

У другому розділі було проведено детальний аналіз існуючих технологічних рішень у сфері емоційного штучного інтелекту, архітектури ШІ-агентів та методів вебвізуалізації тривимірних персонажів.

Огляд концепцій афективних обчислень (Affective Computing) та методів обробки природної мови показав, що для точного визначення емоційного настрою користувача найбільш ефективним є проєктування гібридного підходу. Поєднання лексичного пошуку очевидних текстових маркерів зі статистичним машинним навчанням дозволить системі розпізнавати не лише категорію емоції, а й три рівні її виразності (слабкий, помірний, виражений).

Порівняльний аналіз архітектурних підходів довів доцільність використання гібридної хмароорієнтованої моделі. Інтеграція сучасної мовної моделі DeepSeek-V3 через інженерні шлюзи Together API у зв'язці з захисним RAG-алгоритмом на базі бази даних MongoDB дозволяє повністю ліквідувати проблему «інформаційних галюцинацій» штучного інтелекту. Система формуватиме відповіді виключно на основі офіційних документів кафедри ІПЗЕ, не перевантажуючи при цьому серверне обладнання.

На основі проведеного порівняльного аналізу аналогічних світових проєктів та комерційних платформ було обґрунтовано вибір технології візуалізації. Замість складного й витратного рендерингу в реальному часі було обрано альтернативний підхід — рушій попередньо записаних відеофрагментів (Video Engine), який працює в покроковому асинхронному режимі. Використання безкоштовного медіакодека VP9 у форматі WebM із підтримкою прозорого альфа-каналу та впровадження принципу подвійної буферизації забезпечує високу якість міміки аватара. Це дозволяє повністю зняти обчислювальне навантаження з мобільних пристроїв студентів та запустити систему навіть на застарілих смартфонах.

Усі обрані в ході аналізу технологічні інструменти (Python, FastAPI, Scikit-learn, MongoDB, WebM) формують надійну інженерну базу, детальний математичний та програмний опис якої буде представлено у наступних розділах пояснювальної записки.

3 МЕТОДИ РЕАЛІЗАЦІЇ СИСТЕМИ РЕАКЦІЙ ТА АНАЛІЗУ ПРИРОДНОЇ МОВИ

На основі проведеного технологічного аналізу, наступним важливим кроком є побудова чіткої математичної та логічної структури майбутнього програмного забезпечення. Щоб віртуальний помічник працював стабільно, правильно розпізнавав людські почуття у тексті та миттєво реагував візуально, його роботу необхідно описати мовою алгоритмів, математичних моделей та логічних правил взаємодії.

Цей розділ присвячено детальному опису внутрішніх механізмів системи. У ньому розглядається побудова трирівневого гібридного конвеєра для аналізу природної мови, наводиться математичний апарат класифікації текстових повідомлень за допомогою ймовірнісних розподілів, а також описується логіка інтелектуального вибору та безшовного перемикання відеофрагментів міміки 3D-аватара під час діалогу з користувачем.

3.1 Логіка побудови та етапи наскрізного конвеєра керування станами аватара

Щоб віртуальний помічник працював чітко і без збоїв, весь шлях від моменту, коли користувач написав повідомлення, до моменту, коли аватар змінив міміку на екрані, організовано у вигляді наскрізного конвеєра даних (data pipeline). Головна мета такого підходу — автоматизувати процес, розбити його на зрозумілі кроки та зробити поведінку програми повністю передбачуваною. Конвеєр допомагає перетворити звичайний людський текст на точні цифрові команди для графічного рушія, що дозволяє уникнути зависань інтерфейсу чи помилок у розпізнаванні емоцій.

Робота цього конвеєра нагадує звичайний заводський конвеєр: кожен окремий модуль системи приймає дані від попереднього етапу, виконує свою

частину роботи й передає оновлений результат далі за ланцюжком. Загалом весь процес складається з 12 послідовних кроків (етапів), які для зручності поділено на три великі інженерні блоки: аналіз тексту, математична обробка результатів та керування відеороликами аватара.

3.1.1 Блок гібридного аналізу природної мови

Перший великий блок наскрізного конвеєра повністю відповідає за роботу з вхідним текстом користувача, його деконструкцію та вилучення первинних емоційних ознак. Оскільки людська мова є надзвичайно різноманітною, гнучкою та часто містить граматичні чи стилістичні особливості, використання лише одного методу класифікації неминуче призводило б до системних помилок. Для досягнення балансу між швидкістю роботи сервера, стійкістю до помилок та точністю результату, у цьому блоці реалізовано гібридний підхід, який поєднує три незалежні інструменти обробки.

Робота цього лінгвістичного блоку складається з чотирьох послідовних етапів:

1. Первинна фіксація тексту користувача. Вхідною точкою всього конвеєра є необроблений текстовий рядок українською мовою, що надходить від клієнтської частини вебінтерфейсу в момент, коли користувач відправляє повідомлення у чат. Даний рядок може мати довільну довжину і є неоднорідним за своєю структурою: він може містити повнозначні слова, графічні символи емодзі, знаки пунктуації, абревіатури, скорочення чи специфічні емоційні патерни. Головне завдання цього етапу — зафіксувати вхідний сигнал у первинному вигляді, очистити його від випадкових подвійних пробілів чи технічних символів перенесення рядка та без втрат передати у лінгвістичне ядро для подальшого розбору.

2. Лексиконний аналіз. На цьому етапі реалізовано підхід аналізу на основі правил, який виконує швидкий пошук прямих лексичних маркерів у тексті.

У пам'яті модуля зберігається спеціалізований емоційний словник та окрема база графічних юнікод-символів емодзі. Кожному об'єкту в цій базі жорстко присвоєно цільову базову емоцію та числову вагу її інтенсивності (наприклад: слово «чудово» однозначно зв'язується з класом happy, слово «сумно» — з класом sad, а слово «злюсь» — з класом angry). Під час роботи модуля вхідний текст розбивається на окремі слова (токени), після чого програма звіряє їх із базою відповідностей. Система підтримує пошук різних словоформ, базовий stem-пошук (визначення кореня слова) та пряму інтерпретацію графічних емодзі. Результатом етапу є накопичення початкових емоційних балів. Якщо користувач ввів коротку репліку, що складається суто з очевидних маркерів, лексичний рівень фіксує результат, що дозволяє заощадити ресурси сервера.

3. Структурний патерн-аналіз. Якщо на попередньому етапі словники не знайшли явних збігів, текстовий масив передається модулю структурного аналізу. Цей модуль використовує гнучку систему регулярних виразів для сканування синтаксичної та стилістичної архітектури речення. Його головне завдання — знайти та зафіксувати невербальні натяки, які люди підсвідомо закладають у структуру тексту під час комунікації:

- Аналіз регістру текстових символів: написання повідомлення або його окремих фрагментів великими літерами (режим CAPS LOCK) аналізується як підвищення тону мовлення, крик або вираження гніву.
- Пунктуаційні патерни: використання кількох знаків оклику поспіль (наприклад, «!!!») інтерпретується як маркер сильного підсилення поточної емоції (радості або здивування). Повторення знаків питання (наприклад, «????») сигналізує про стан глибокої розгубленості або переводить аватара в стан роздумів (thinking). Наявність багатокрапок вказує на невпевненість, а повтори голосних літер (наприклад, «аааа») розпізнаються як прояв здивування (surprise). Також модуль ідентифікує класичні текстові символьні смайли.

Патерн-аналіз дозволяє успішно визначити емоційне забарвлення фраз навіть за повної відсутності знайомих слів у словниках системи.

4. Статистична класифікація. Четвертий, найбільш інтелектуальний рівень лінгвістичного аналізу, призначений для обробки розгорнутих, синтаксично складних або нейтральних за формою інформаційних запитів користувачів (наприклад, офіційні питання студентів про розклад, умови навчання чи правила кафедри). Робота модуля відбувається у два кроки:

- TF-IDF-векторизація: вхідний текст, попередньо очищений від лінгвістичного шуму (стоп-слів, таких як прийменники та сполучники) та приведений до початкової канонічної форми (лематизований), перетворюється у числовий вектор. Алгоритм розраховує математичну важливість кожного слова, враховуючи частоту його зустрічальності в конкретному повідомленні та в усьому навчальному корпусі даних.
- Логістична регресія (Logistic Regression): навчена модель машинного навчання приймає сформований числовий вектор і виконує класифікацію контексту. Завдяки використанню методу predict_proba() система отримує не один жорсткий варіант, а повний набір математичних ймовірностей для кожної з 7 базових емоцій. Це дозволяє моделі гнучко враховувати прихований контекст і точно обробляти складні конструкції.

Після завершення роботи всіх чотирьох етапів лінгвістичного блоку, отримані сирі дані (бали лексикону, коефіцієнти синтаксичних патернів та ймовірнісний вектор від моделі машинного навчання) передаються далі за ланцюжком конвеєра для подальшого математичного об'єднання.

3.1.2 Блок математичної агрегації, корекції та контекстуалізації емоційного вектора

Другий великий блок конвеєра потрібен для того, щоб зібрати разом усі результати текстового аналізу та правильно їх обчислити. Оскільки на перших етапах дані надходили з абсолютно різних джерел — як звичайні бали зі словника, коефіцієнти від розділових знаків та відсотки від моделі машинного навчання —

їх не можна просто додати один до одного. Система має звести всі ці цифри до єдиного знаменника. Крім того, саме на цьому етапі програма враховує важливі деталі людської мови (наприклад, підсилювальні слова «дуже» або заперечення «не») та стежить за тим, щоб настрій персонажа змінювався плавно і природно.

Цей обчислювальний блок складається з чотирьох послідовних етапів:

1. Об'єднання та вирівнювання оцінок. Головна задача цього кроку — зібрати докупи всі «сирі» результати аналізу тексту. Модуль бере бали, які нарахував словник, оцінки за знаки пунктуації та відсотки від моделі машинного навчання. Оскільки якимось методам система довіряє більше, а якимось менше, кожна оцінка множиться на свій коефіцієнт важливості. Після цього всі цифри додаються і передаються спеціальній математичній функції Softmax. Вона перетворює будь-які змішані числа у зрозумілі відсотки (ймовірності від 0 до 1), причому якщо скласти відсотки для всіх 7 емоцій разом, у сумі ми завжди отримаємо рівно 100%. На виході система має чітку стартову картину настрою користувача.

2. Налаштування сили емоції. Цей модуль визначає, наскільки сильною є емоція людини. Програма шукає у реченні слова, якими ми зазвичай підсилюємо або послаблюємо свої репліки (наприклад: «дуже», «неймовірно», «надзвичайно», «трохи», «ледь»). Якщо користувач пише «дуже сумно», програма автоматично збільшує відсоток для емоції sad (смуток). Це потрібно для того, щоб аватар на екрані не просто злегка зажурився, а показав максимально виразну міміку розпачу. Якщо ж фраза звучить як «трохи злий», рівень злості (angry) знижується до базового, спокійного рівня.

3. Обробка заперечень. Цей модуль рятує систему від дурних помилок, які часто трапляються через особливості нашої мови. Програма шукає у тексті слова заперечення: «не», «ніколи», «жодного» чи «не дуже». Без цього модуля фраза «я не радий» могла б здатися програмі позитивною, бо вона просто побачить слово «радий» (happy). Negation Processor вчасно помічає частку «не»,

швидко зменшує бали радості й переносить вагу на нейтральний або сумний стан. Завдяки цьому аватар реагує інтелектуально та правильно розуміє зміст слів.

4. Емоційна пам'ять чату. Цей крок працює як пам'ять для нашого аватара. У реальному житті настрій людини не стрибає щосекунди від люті до щастя — емоціям потрібен час, щоб змінитися. Модуль запам'ятовує, які почуття були у користувача в попередніх повідомленнях під час поточної розмови. Коли приходить нове репліка, система враховує це минуле: старі емоції плавно згасають, але все ще впливають на теперішній момент. Наприклад, якщо студент кілька хвилин скаржився на важкий іспит (був сумним), а потім написав коротке нейтральне слово, аватар не стане миттєво веселим. Його настрій зміниться плавно, що робить спілкування набагато природнішим.

Після того як усі ці чотири кроки математичної перевірки завершено, система отримує точний, згладжений та вивірений емоційний стан користувача. Ці дані повністю готові для передачі у фінальний блок, який вибере потрібний відеофайл для аватара.

3.1.3 Блок логічного керування та селекції графічних медіапотоків

Фінальний великий блок наскрізного конвеєра відповідає за прийняття остаточного рішення та перетворення розрахованих відсотків емоцій у реальну графіку на екрані користувача. Після того як текст пройшов лінгвістичний аналіз та математичне згладжування, система вже точно знає настрій користувача у цифрах. Тепер головна інженерна задача — змусити 3D-аватара зреагувати на ці цифри миттєво, без різких графічних стрибків і з правильною силою міміки.

Цей логічний блок складається з чотирьох послідовних етапів:

1. Перевірка природності змін. Цей модуль стежить за тим, щоб міміка аватара змінювалася красиво та плавно, без різких «посмикувань» обличчя. У програмі налаштована спеціальна матриця переходів, де для кожної пари емоцій прописані свої правила. Наприклад, перейти зі спокійного стану (neutral) до

радості (happy) — це природно, тому система робить це легко. А от миттєво перестрибнути від бурхливих веселощів до сильної люті (angry) людина фізично не може. Модуль блокує такі неприродні графічні стрибки. Якщо система фіксує подібну різку зміну, вона примусово змушує аватар спочатку пройти через короткий проміжний стан (наприклад, здивування чи нейтральне обличчя), що робить анімацію плавною.

2. Алгоритм вибору поведінки. На цьому кроці обчислювальне ядро приймає остаточне рішення, як саме поводитиметься аватар. Алгоритм одночасно аналізує кілька важливих факторів: яка емоція зараз є головною, наскільки модель у ній впевнена (показник confidence score), чи дозволяє матриця переходів змінити міміку, який загальний контекст розмови та яка сила (інтенсивність) цієї емоції. Зібравши всі ці параметри докупи, алгоритм формує чітку фінальну команду для контролера анімацій.

3. Контролер анімацій. Цей модуль відповідає за підбір конкретного відеофайлу, який потрібно показати користувачу. У системі немає важкої 3D-графіки, яка б рендерилася в реальному часі й гальмувала телефони студентів. Замість цього для кожної з 7 базових емоцій заздалегідь відрендерено цілі пули готових роликів. Вони розділені за трьома рівнями сили: сильна емоція, середня та базова (слабка). Якщо система має високу впевненість у розрахованому результаті, контролер обирає файл із яскравою, активною мімікою. Якщо впевненість низька — береться файл зі спокійною мікромімікою. Результатом роботи модуля є точна назва потрібного медіафайлу.

4. Фінальне відображення. Це завершальний етап роботи всього конвеєра системи. Клієнтський вебрушій у браузері користувача приймає назву файлу від контролера і за допомогою технології Video Engine миттєво запускає його відтворення. Завдяки використанню сучасного кодека VP9 у форматі WebM з підтримкою прозорого фону (альфа-каналу), емоції аватара накладаються прямо на інтерфейс сайту кафедри ІПЗЕ. Відео запускається асинхронно, працює

стабільно та абсолютно не навантажує процесори смартфонів чи комп'ютерів студентів.

Таким чином, розроблений 12-етапний наскрізний конвеєр повністю завершує свій цикл: він приймає звичайне текстове повідомлення від людини, розбирає його на лінгвістичні деталі, прораховує математичну логіку розпізнавання емоцій та виводить на екран якісну, плавну і швидку візуальну реакцію тривимірного помічника.

3.2 Математичні та лінгвістичні моделі аналізу природної мови

Комп'ютер не вміє відчувати емоції чи читати текст так, як це роблять люди. Для нього будь-яке людське повідомлення — це просто набір літер. Тому, щоб перетворити звичайний текст на точні команди для рухів 3D-аватара, в обчислювальному ядрі системи використовуються математичні формули та алгоритми. Математика допомагає програмі переводити результати аналізу у цифри, правильно об'єднувати дані з різних модулів, рахувати відсотки ймовірності для кожної емоції та стежити за тим, щоб міміка персонажа змінювалася плавно і без помилок.

Варто зазначити, що для базових етапів розпізнавання — перетворення тексту на числа (векторизації) та первинної класифікації — у проєкті використовуються готові перевірені інструменти з популярної бібліотеки машинного навчання scikit-learn (а саме методи TfidfVectorizer та LogisticRegression). Проте, щоб змусити ці стандартні інструменти працювати в комплексі з усім конвеєром системи, враховувати заперечення, контекст розмови та логіку рухів аватара, було вручну розроблено та закодовано у модулі emotion_engine.py власну математичну модель, яка складається з п'яти ключових етапів.

Після того як вхідний текст паралельно проаналізували три різні модулі (словник емоційних слів, правила регулярних виразів для знаків пунктуації та модель машинного навчання), система отримує три окремі оцінки. Задача першого кроку — змішати їх разом. Оскільки модель машинного навчання зазвичай є точнішою, їй надається найбільший пріоритет (вага 0.45). Словнику система довіряє трохи менше (вага 0.35), а знакам пунктуації — найменше (вага 0.20).

Підсумковий емоційний бал для кожної окремої емоції розраховується за формулою (3.1):

$$score(e) = 0.35 \times lex(e) + 0.20 \times pat(e) + 0.45 \times ml(e) \times 5.0 \quad (3.1)$$

де score(e) — підсумковий бал для конкретної аналізованої емоції e;

lex(e) — бали емоції, які зміг знайти словниковий модуль;

pat(e) — бали емоції за знаки пунктуації та стиль написання від модуля регулярних виразів;

ml(e) — оцінка ймовірності емоції, яку згенерувала модель машинного навчання.

Якщо повідомлення користувача є занадто коротким (наприклад, лише один розділовий знак або графічний емодзі), модель машинного навчання автоматично не запускається для економії обчислювальних ресурсів сервера, а формула балансується між словником та патернами у пропорції 0.70 на 0.30 відповідно.

Отримані після додавання за формулою (3.1) «сирі» змішані бали можуть бути абсолютно різними за розміром. Щоб перетворити їх на зрозумілі відсотки ймовірності (від 0 до 1), використовується функція Softmax. Проте під час комп'ютерних обчислень великі числа в експонентах можуть викликати серйозний збій — переповнення обчислювальної пам'яті. Щоб цього не сталося, у коді проєкту реалізовано модифіковану формулу, яка перед розрахунком автоматично знаходить найбільший бал і віднімає його від усіх інших значень.

Нормування та розрахунок підсумкових відсотків ймовірності здійснюється за формулою (3.2):

$$P(e_i) = \frac{e^{score_i - max}}{\sum_j e^{score_j - max}} \quad (3.2)$$

де $P(e_i)$ — фінальна нормована ймовірність для і-ї емоції системи;

$score_i$ — сирий обчислений бал для поточної і-ї емоції;

max — максимальний емоційний бал серед усіх наявних класів у поточному наборі;

$score_j$ — сирий обчислений бал для кожного j-го класу, що використовується у знаменнику для загального нормування.

Наступний алгоритм рятує систему від грубих логічних помилок, які часто трапляються через особливості людської граматики. Наприклад, без цього модулю фраза «я не рад» здалася б програмі веселою, бо вона просто побачить корінь «рад». Спеціальний модуль знаходить у реченні заперечну частку «не», бере бал домінуючої емоції, множить його на коефіцієнт згасання і переносить цю вагу на протилежний або нейтральний стан.

Математична інверсія емоційного значення при виявленні заперечень виконується за формулою (3.3):

$$score_{inv} \mathrel{+}= score_{dom} \cdot 0.8 \quad (3.3)$$

де $score_{inv}$ — накопичуваний бал нової, інвертованої емоції;

$score_{dom}$ — високий емоційний бал тієї домінуючої емоції, яка зазнала лінгвістичного заперечення.

У реальному житті настрій людини не може хаотично стрибати щомиті від люті до щастя — емоціям потрібен певний час, щоб змінитися. Тому програма має емоційну пам'ять і враховує історію попередніх повідомлень у поточному чаті.

Фінальна впевненість системи в емоції з урахуванням динамічного контексту розмови вираховується за формулою (3.4):

$$conf_{out} = \alpha \cdot conf_{ctx} + (1 - \alpha) \cdot conf_{cur} \quad (3.4)$$

де $conf_{out}$ — підсумковий згладжений рівень впевненості системи в емоції;

α — коефіцієнт інерції емоційного стану, що жорстко зафіксований у коді на рівні 0.25;

$conf_{ctx}$ — збережений рівень впевненості в емоції з попереднього повідомлення розмови;

$conf_{cur}$ — розрахований рівень впевненості в емоції для щойно написаного повідомлення.

На самому фініші обчислень система має ухвалити остаточне рішення: чи дозволено аватару фізично змінити одну міміку на іншу на екрані пристрою. Для цього у коді створено спеціальну таблицю (матрицю переходів), де кожна пара емоцій має свій коефіцієнт сумісності.

Дозвіл на безшовну заміну графічного файла видається у тому випадку, якщо виконується математична нерівність (3.5):

$$confidence \cdot transition\_score \geq \theta \quad (3.5)$$

де $confidence$ — підсумкова впевненість системи в емоції, отримана за формулою (3.4);

$transition\_score$ — коефіцієнт дозволу на перехід між конкретною парою емоцій із системної матриці;

θ — математичний поріг чутливості, який дорівнює 0.30 для звичайних переходів та знижується до 0.12 у випадку виходу аватара з абсолютно нейтрального стану.

Якщо ліва частина нерівності (3.5) виявляється більшою або рівною за поріг θ, аватар отримує остаточну команду на зміну відеофайлу.

3.3 Визначення функціоналу користувача та логіки інтелектуальної селекції анімацій

У цьому підрозділі розберемо, як саме користувач взаємодіє з розробленою програмою, а також покроково розглянемо внутрішню логіку системи, яка допомагає тривимірному аватару миттєво обирати правильні відеоролики для відображення своїх емоцій.

Оцінка функціональних можливостей користувача системи. Коли проєктували вебінтерфейс для сайту кафедри ІПЗЕ, головною метою було зробити його максимально простим та зручним для будь-якого студента чи абітурієнта. Оскільки інтерфейс створювався інтуїтивним, користувачу не потрібно проходити жодних складних налаштувань — уся взаємодія з віртуальним помічником зводиться до кількох простих кроків:

- Спілкування у вільному стилі. Користувач відкриває звичайне вікно чату і пише туди будь-яке запитання українською мовою. При цьому можна писати абсолютно природно: використовувати знаки оклику чи питання, частку «не», підсилювальні слова (наприклад, «дуже» або «неймовірно») та навіть звичайні графічні смайлики.
- Миттєве отримання відповідей. Після того як повідомлення відправлено, користувачу не доводиться чекати. Текстова відповідь від чат-боту та відповідна мімічна реакція аватара з'являються на екрані одночасно і повністю асинхронно, тобто без жодних гальмувань сторінки сайту.
- Жива присутність персонажа. Користувач постійно бачить перед собою тривимірного помічника на екрані. Навіть якщо людина нічого не пише і просто читає інформацію на сайті, аватар не застигає як статична картинка, а перебуває у природному стані очікування (легко кліпає очима та дихає), що створює ефект реальної розмови.

Логіка інтелектуальної селекції відеороликів міміки. Головна інженерна особливість проєкту полягає в тому, що система не рендерить 3D-графіку на ходу

(це б дуже сильно навантажувало смартфони студентів). Замість цього заздалегідь підготувлений пул готових оптимізованих відеофайлів у форматі WebM для кожної з 7 базових емоцій. Щоб міміка персонажа була реалістичною, розділено ці ролики за трьома рівнями сили:

- Базовий (слабкий) рівень: це ледь помітна мікроміміка, наприклад, легкий рух брів чи куточків губ.
- Середній рівень: стандартна, добре виражена емоція.
- Сильний рівень: дуже яскрава, бурхлива та емоційна реакція обличчя.

Щоб розібратися, який саме ролик увімкнути у відповідь на репліку користувача, спеціальний алгоритм (контролер анімацій) виконує послідовну перевірку. Весь цей процес наочно зображено на блок-схемі алгоритму інтелектуальної селекції мімічних станів на рисунку 3.1.

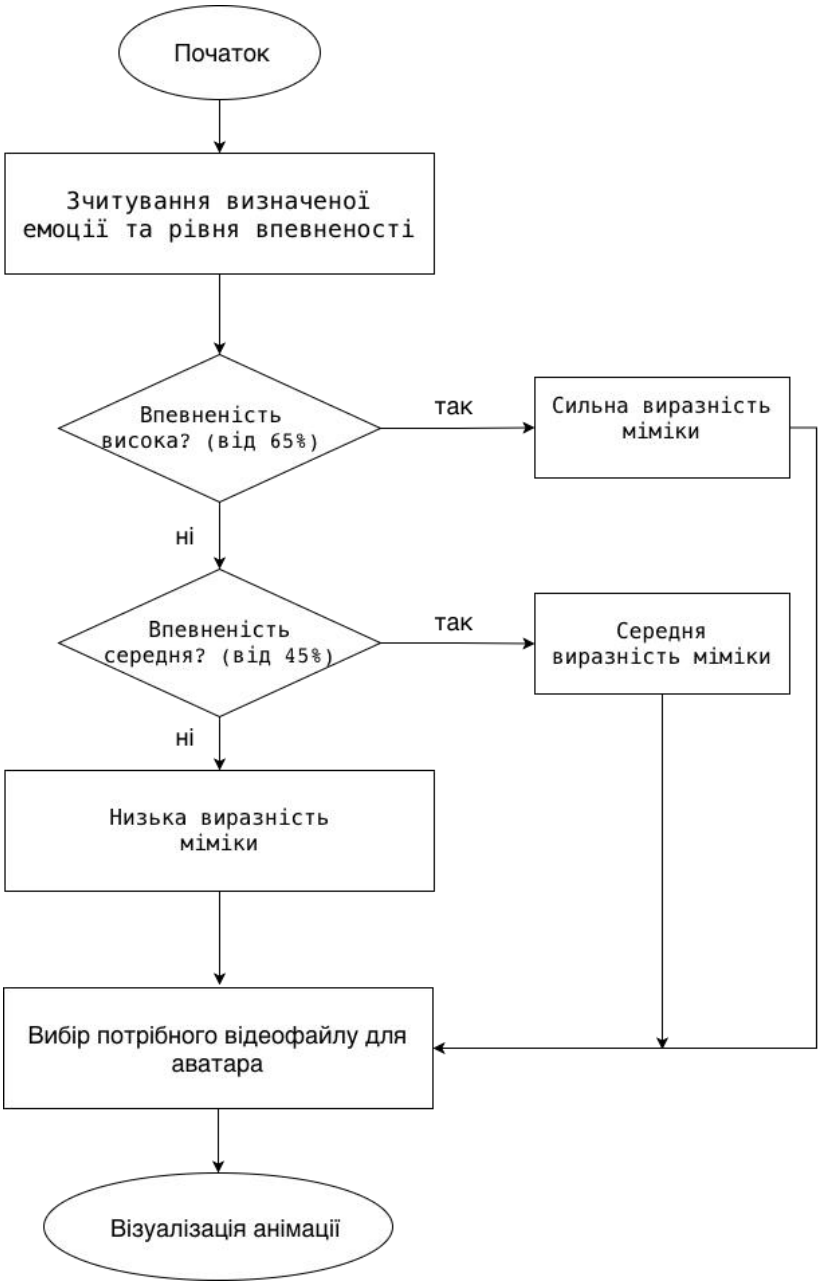


Рисунок 3.1 — Алгоритм інтелектуальної селекції мімічних станів

Якщо подивитися на крок за кроком, як працює цей алгоритм (див. рис. 3.1), то вся логіка зводиться до наступного ланцюжка дій:

1. Зчитування визначеної емоції та рівня впевненості. Спочатку програма забирає результати математичних розрахунків. Вона вже чітко знає, яка емоція є головною і на скільки відсотків комп'ютер у цьому впевнений.

2. Перевірка на високу впевненість. Система дивиться на отримане число. Якщо впевненість дуже висока (65% і більше), алгоритм одразу робить вибір на користь сильної експресії. Проте перед цим обов'язково перевіряється

матриця переходів: якщо різка зміна емоцій є неприродною (наприклад, миттєвий стрибок від люті до радості), то щоб не було графічних посмикувань, система тимчасово залишає поточну міміку без змін.

3. Перевірка на середня впевненість. Якщо впевненість виявилася меншою за 65%, алгоритм спускається до наступного ромба умов і перевіряє, чи є вона хоча б середньою (від 45%). Якщо так — обирається ролик середньої експресії.

4. Активація базової мікроміміки. У випадку, коли відсоток впевненості є зовсім низьким (менше 45%), система не ризикує вмикати яскраві емоції й автоматично обирає спокійну базову мікроміміку.

5. Формування команди на запуск. Коли рівень сили остаточно визначено, контролер формує точну назву потрібного .webm файлу і передає його медіарушію сайту. Відео миттєво запускається на сторінці, і користувач бачить плавну, красиву та логічну відповідь від 3D-аватара.

Висновки до розділу 3

У третьому розділі було виконано повне проектування архітектури інтелектуального ядра системи та розроблено математичне забезпечення для розпізнавання емоційного контексту текстових повідомлень і подальшого керування станами тривимірного аватара помічника.

На основі проведеної роботи було запроектовано наскрізний дванадцятиетапний конвеєр обробки даних, який забезпечує послідовну трансформацію вхідного неструктурованого тексту користувача у детерміновані команди керування візуальним стеком. Розроблена архітектура дозволяє паралельно залучати словникові методи, синтаксичні правила регулярних виразів та ймовірнісні моделі машинного навчання, що суттєво підвищує загальну точність аналізу.

Разом із тим було розроблено та математично обґрунтовано унікальний комплекс лінійних рівнянь та нерівностей, інтегрованих безпосередньо у програмний модуль аналізу. Створена модель зваженої агрегації дозволяє гнучко балансувати пріоритети лінгвістичних сигналів залежно від довжини та специфіки реплік співрозмовника. При цьому реалізація захищеного алгоритму Softmax-нормалізації повністю нівелює ризики аномального переповнення обчислювальної пам'яті сервера під час розрахунку експонент великих чисел.

Модуль обробки заперечень за допомогою динамічного коефіцієнта згасання мінімізує логічні помилки класифікації, які часто виникають через специфіку граматики та використання заперечних часток.

Особливу увагу приділено алгоритму каскадної інтелектуальної селекції анімаційних медіапотоків, який на основі показника впевненості диференціює вихідну реакцію аватара за трьома рівнями експресії міміки.

На завершення було обґрунтовано вибір технічного стеку візуалізації на основі використання медіафайлів у форматі WebM із кодеком VP9 та підтримкою прозорого альфа-каналу. Такий підхід забезпечує асинхронне, безшовне та швидке відображення тривимірного помічника на клієнтській частині сайту кафедри ІПЗЕ без створення надлишкового обчислювального навантаження на процесори мобільних пристроїв чи комп'ютерів студентів.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ КЕРУВАННЯ 3D-АВАТАРОМ

Четвертий розділ присвячено практичним аспектам інженерного втілення запроектованої архітектури та математичних моделей інтелектуального аналізу тексту в діючий програмний комплекс. Головною метою цього етапу є безпосередня розробка обчислювальних модулів, проєктування структури збереження даних, інтеграція графічного стека тривимірної візуалізації на вебресурс, а також проведення комплексного тестування для підтвердження надійності та точності роботи системи.

У межах програмної реалізації особливу увагу приділено створенню асинхронного, швидкодійного середовища, здатного обробляти запити користувачів сайту кафедри ІПЗЕ у реальному часі без створення критичного навантаження на клієнтські пристрої. Нижче наведено обґрунтування обраного технологічного стеку, детальний опис архітектури бази даних, структурну організацію ключових програмних модулів конвеєра, демонстрацію інтерфейсу користувача та результати експериментальної оцінки працездатності розробленого рішення.

4.1 Обґрунтування вибору технологічного стеку та інструментів розробки

Під час практичної реалізації системи керування тривимірним аватаром першочерговим інженерним завданням став вибір оптимального технологічного стеку. Основними критеріями вибору інструментів були висока швидкість обробки текстових запитів у реальному часі, асинхронність обчислювальних процесів, наявність надійних бібліотек для аналізу природної мови та можливість відображення якісної графіки без перевантаження комп’ютерів чи смартфонів користувачів.

Для створення серверної (бекенд) частини програмного комплексу було обрано мову програмування Python. Цей вибір зумовлений тим, що Python є світовим стандартом для розробки систем штучного інтелекту, машинного навчання та лінгвістичного аналізу завдяки величезній екосистемі готових інструментів[10].

Як основний серверний фреймворк у проєкті використано FastAPI. Головними перевагами FastAPI, які стали вирішальними для архітектури емоційного конвеєра, є швидкість роботи, за результатами незалежних тестів FastAPI є одним із найшвидших фреймворків на Python, поступаючись лише рішенням на NodeJS або Go.

Автоматична валідація даних: завдяки інтеграції з бібліотекою Pydantic, фреймворк автоматично перевіряє структуру вхідних текстових повідомлень та вихідних емоційних векторів, що мінімізує виникнення помилок під час передачі даних.

Для реалізації математичного та лінгвістичного ядра системи розпізнавання емоцій було залучено спеціалізовані Python-бібліотеки. Модуль векторизації тексту TF-IDF та базова модель логістичної регресії побудовані на базі бібліотеки scikit-learn, яка забезпечує високу точність та швидкість матричних обчислень. Для швидкої роботи з числовими емоційними векторами та реалізації функцій Softmax-нормалізації використано бібліотеку NumPy. Обробка специфічних мовних зворотів та синтаксичних знаків пунктуації реалізована за допомогою вбудованого модулю регулярних виразів[11].

Клієнтська (фронтенд) частина системи, яка інтегрується безпосередньо на вебсайт кафедри ІПЗЕ, реалізована за допомогою стандартного стеку веброзробки: мови програмування JavaScript та розмітки HTML5/CSS3. Для інтерактивного відтворення емоцій 3D-персонажа було відмовлено від використання важких клієнтських 3D-рушіїв (таких як WebGL або Three.js), оскільки вони вимагають високої потужності графічного процесора пристрою[12]. Замість цього роботу візуального стеку організовано через стандартний HTML5-компонент Video та

програмний модуль Video Engine, який керує безшовним перемиканням заздалегідь відрендерених відеофайлів у форматі WebM з кодеком VP9.

4.2 Архітектура бази даних та структура збереження лінгвістичних ресурсів

Для забезпечення стабільного збереження даних, підтримки контексту користувацьких сесій та швидкого пошуку інформації було спроєктовано архітектуру бази даних на основі NoSQL СУБД MongoDB. Вибір документо-орієнтованої бази даних зумовлений гнучкістю її схем, високою швидкістю виконання операцій запису та зчитування емоційного контексту, а також зручністю збереження слабоструктурованих лінгвістичних ресурсів та логів діалогів у форматі JSON документів[13].

Логічна структура розробленої бази даних складається з шести взаємопов'язаних колекцій. Графічне представлення схем документів та логічних зв'язків між ними наведено на рисунку 4.1.

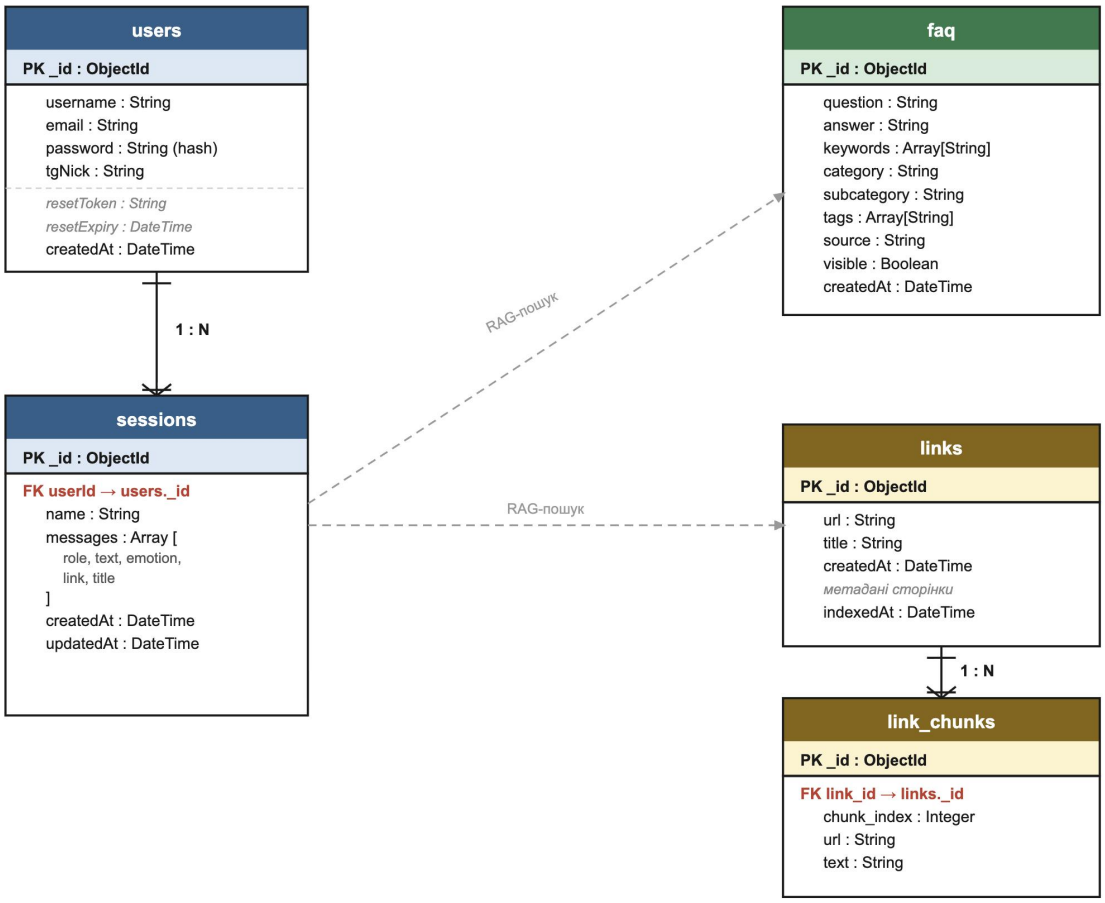


Рисунок 4.1 – Схема зв'язків та структура колекцій бази даних MongoDB

Відповідно до розробленої архітектури, кожна колекція виконує чітко визначену функцію у загальному конвеєрі обробки даних:

- users — зберігає облікові дані користувачів для безпечної автентифікації.
- sessions — підтримує контекст розмови. Через поле user_id пов'язана з користувачем і містить історію повідомлень, послідовність емоційних станів та прапорці активності сесії.
- faq — база знань кафедри ІПЗЕ, що містить готові питання, відповіді, категорії та ключові токени для швидкого реагування на типові запити.
- links — зберігає метадані інформаційних джерел (URL, назву, опис, дату індексації) та навчальних матеріалів.

- link_chunks — містить атомарні фрагменти великих текстів (chunk_text), їхні порядкові номери та векторні представлення для семантичного пошуку. Пов'язана з колекцією links через link_id.

Індексація ключових полів (user_id, session_id, link_id) мінімізує час відгуку СУБД, що критично важливо для миттєвого та безшовного перемикання емоційних WebM-анімацій аватара в режимі реального часу.

4.3 Серверна частина системи

Серверна частина розробленого програмного комплексу виступає головним зв'язуючим вузлом всієї системи, який керує взаємодією між вебінтерфейсом сайту кафедри ІПЗЕ, базою даних MongoDB, інтелектуальним модулем емоцій та сервісом Together API. Завдяки використанню сучасного асинхронного фреймворку FastAPI, сервер здатний миттєво обробляти повідомлення від багатьох користувачів одночасно в режимі реального часу, забезпечуючи високу швидкість відповіді без жодних затримок.

Для чіткого розуміння того, які саме завдання виконує бекенд і які права доступу мають різні користувачі, було розроблено модель прецедентів, яку наочно зображено на UML-діаграмі на рисунку 4.2.

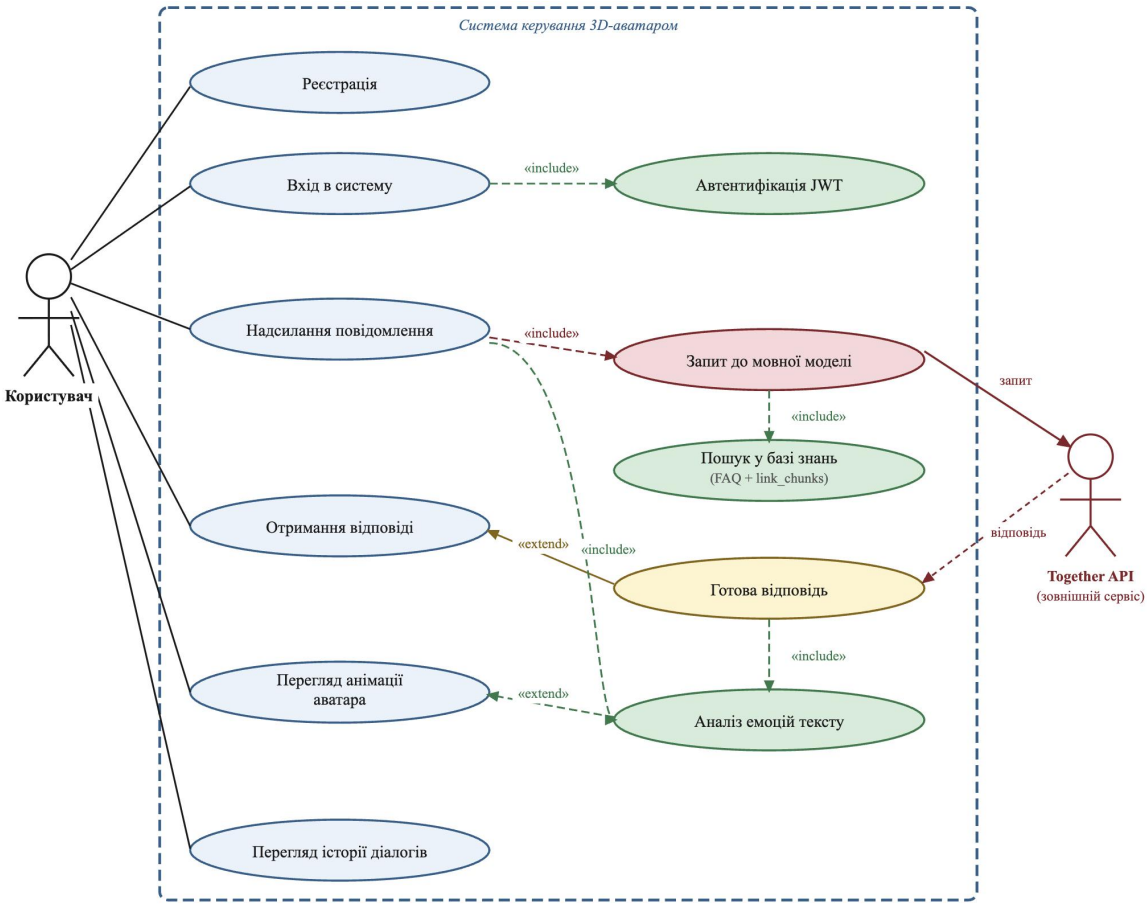


Рисунок 4.2 – Діаграма прецедентів серверної частини системи

Ця схема наочно демонструє розподіл обов'язків між двома основними учасниками системи. Звичайний користувач, тобто студент або абітурієнт, спілкується з сервером через маршрути реєстрації та входу, а під час діалогу надсилає текстові репліки, отримує відповіді з емоційними маркерами, може переглядати або повністю очищати історію поточної сесії. Водночас саме обчислювальне ядро системи працює повністю в автоматичному фоновому режимі — воно перевіряє захищені токени користувачів, зберігає всю історію в MongoDB, взаємодіє з модулем семантичного пошуку для витягування потрібних текстових фрагментів, звертається до штучного інтелекту через Together API, і в результаті запускає конвеєр визначення емоцій та вибору правильного відеофайлу для анімації аватара.

4.4 Програмна структура та логіка роботи модуля обробки емоцій

Цей підрозділ присвячено детальному розбору внутрішнього устрою та програмної реалізації головного обчислювального компонента системи (assistant_core), який відповідає за розпізнавання емоційного забарвлення тексту та вибір правильної сили міміки персонажа. Оскільки весь емоційний конвеєр побудовано на принципах об'єктно-орієнтованого програмування, для наочного відображення статичної структури коду, його внутрішніх змінних та функцій було розроблено UML-діаграму класів, яку наведено на рисунку 4.3.

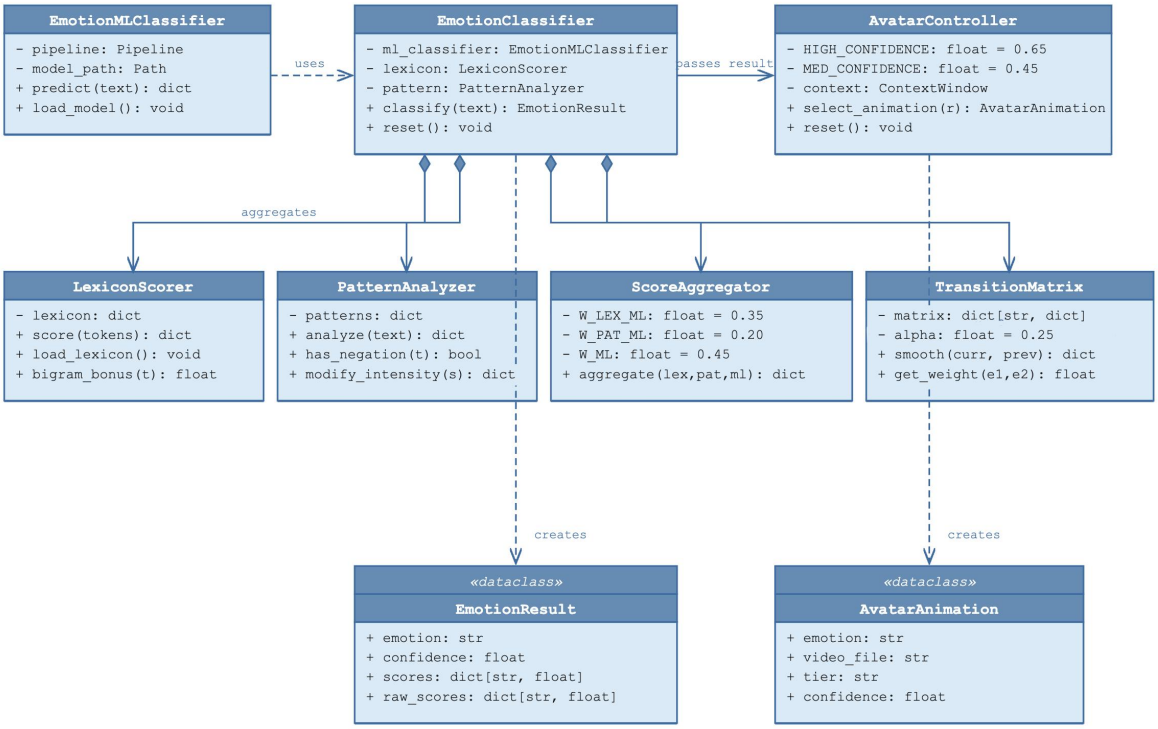


Рисунок 4.3 – Діаграма класів модуля обробки емоцій assistant_core

Якщо поглянути на розроблену схему, роботу модуля можна порівняти з злагодженою командою, де кожен елемент виконує свою роль. Головним керівником є модуль EmotionClassifier. Він приймає текст від користувача, запускає аналіз і в кінці видає готовий результат. Для цього він одночасно звертається до кількох помічників.

Перший помічник, EmotionMLClassifier, — це штучний інтелект (модель машинного навчання), який завантажує свій цифровий досвід через функцію load_model() і намагається вгадати емоцію за допомогою команди predict(). Другий помічник, LexiconScorer, працює як розумний словник: він шукає окремі слова у базі даних і визначає, наскільки вони радісні чи сумні. Третій помічник, PatternAnalyzer, шукає в тексті особливі правила та знаки. Наприклад, за допомогою функції has_negation() він помічає частку «не», яка повністю змінює зміст речення (як у фразі «я не радий»).

Коли всі помічники зібрали свої дані, у роботу включається математичний об'єднувач ScoreAggregator. Він має чітко налаштовані вагові коефіцієнти для кожного джерела інформації (для словника — 35%, для правил — 20%, а для штучного інтелекту — 45%). Цей модуль змішує всі оцінки за спеціальною формулою і створює проміжний звіт EmotionResult, де записано назву емоції та відсоток впевненості комп'ютера в ній.

Отриманий результат передається фінальному контролеру AvatarController, який і вирішує, що робити 3D-аватару. Щоб робота виглядала реалістично, цей контролер радиться із захисним фільтром TransitionMatrix. Цей фільтр стежить за історією розмови і не дозволяє персонажу робити дивні, неприродні для людини рухи (наприклад, миттєво перескакувати від сильного гніву до бурхливої радості). Після такої перевірки контролер оцінює відсоток впевненості: якщо він високий (65% і більше) — обирається яскрава емоція, якщо середній (від 45%) — звичайна, а якщо низький — спокійна і ледь помітна міміка. Фінальним аккордом є команда select_animation(), яка створює чітку інструкцію AvatarAnimation із назвою потрібного відеофайлу у форматі WebM, який фронтенд сайту миттєво запускає на екрані користувача.

Для того щоб детально побачити, як ці компоненти взаємодіють між собою у часі під час реального сеансу спілкування, було розроблено UML-діаграму послідовності. Вона наочно відображає динаміку передачі даних від моменту

введення репліки користувачем до моменту запуску відеофайлу і наведена на рисунку 4.4.

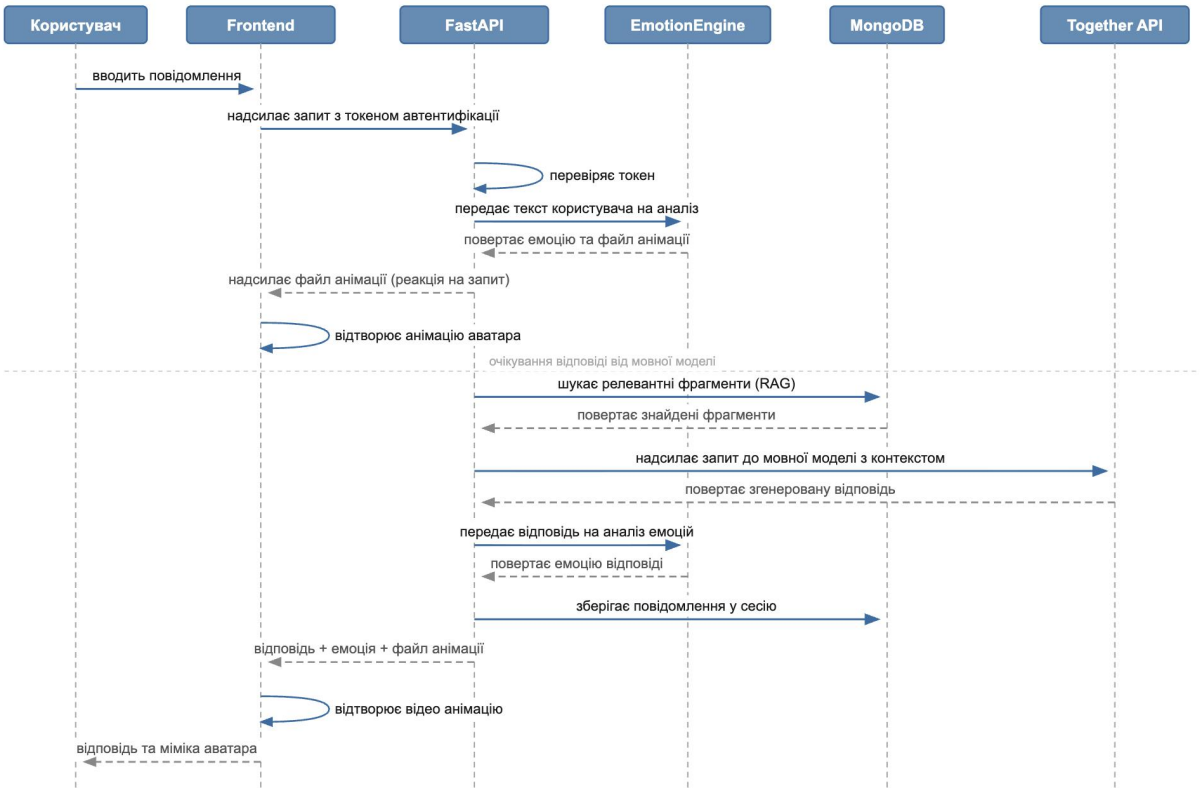


Рисунок 4.4 – Діаграма послідовності обробки запиту та активації міміки

Логіка виконання цього процесу у часі є повністю лінійною та асинхронною. Усе починається з того, що користувач надсилає текстове повідомлення через клієнтський інтерфейс чату, який миттєво перенаправляє цей запит на сервер FastAPI. Сервер передає отриману репліку головному контролеру обробки емоцій. Контролер по черзі викликає внутрішні сервіси: спочатку лексичний сканер витягує з тексту емоційні токени, потім агрегатор обчислює загальні бали за формулами, а модуль валідації перевіряє дозволеність зміни стану за матрицею переходів. Після цього контролер формує команду з точним ім'ям потрібного відеофайлу та повертає її назад на FastAPI. Сервер миттєво відправляє готову відповідь разом із назвою файлу на клієнтську частину системи, де вбудований графічний рушій фронтенду без жодних затримок підміняє джерело відеопотоку. В результаті користувач бачить на екрані плавний запуск відповідної емоційної WebM-анімації тривимірного аватара помічника.

4.5 Тестування програмного комплексу та аналіз результатів

Цей підрозділ присвячено перевірці працездатності, стабільності та коректності роботи всіх розроблених алгоритмів системи керування аватаром. Для підтвердження надійності програмного забезпечення було реалізовано комплексне модульне тестування за допомогою фреймворку pytest. Написаний тестовий набір повністю покриває як окремі лінгвістичні компоненти обчислювального конвеєра, так і наскрізні сценарії взаємодії. Результати успішного запуску та виконання всієї серії перевірок у терміналі розробника наведено на рисинку 4.5.

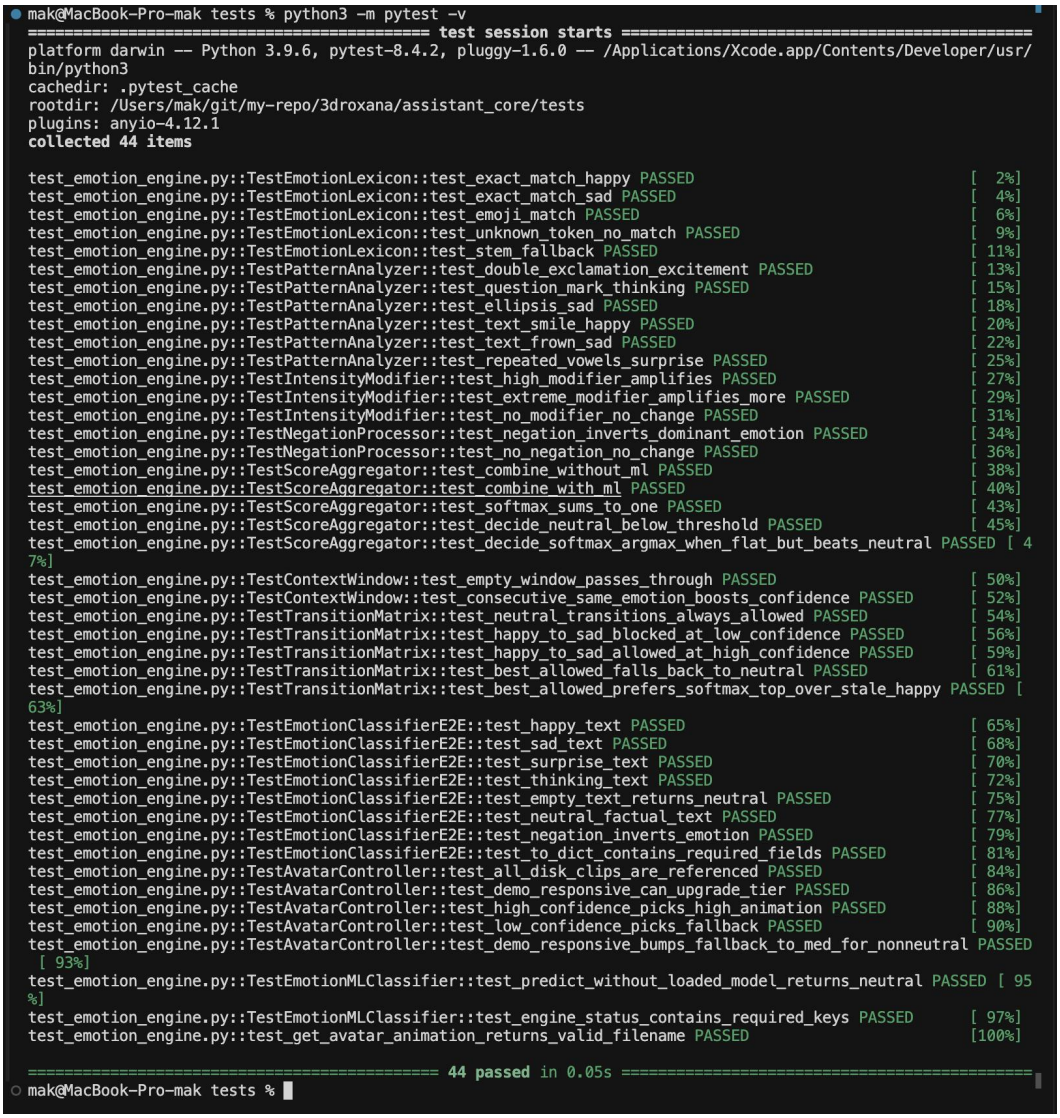


Рисунок 4.4 – Лог успішного виконання 44 модульних тестів

Представлений знімок екрана підтверджує, що розроблені сценарії детально перевірили кожну ланку обчислювального конвеєра. Зокрема, тести класу TestEmotionLexicon підтвердили точність пошуку слів у базі даних, розпізнавання емодзі та механізм аналізу невідомих слів, а класи TestPatternAnalyzer і TestIntensityModifier довели правильну реакцію на знаки пунктуації, символьні смайли й слова-підсилювачі. Обробник заперечень у класі TestNegationProcessor продемонстрував безпомилкову інверсію знаку емоцій при появі частки «не», тоді як математичний агрегатор TestScoreAggregator підтвердив точність змішування сигналів з урахуванням вагових коефіцієнтів та правильність Softmax-нормалізації, яка утримує суму ймовірностей рівною одиниці й захищає пам'ять від переповнення. Перевірки класів TestContextWindow та TestTransitionMatrix доведуть коректність накопичення контексту розмови й надійність захисної матриці переходів, яка успішно блокує різкі й неприродні стрибки міміки персонажа. Наскрізні тести TestEmotionClassifierE2E повністю підтвердили якість класифікації реальних фраз (радість, сум, роздуми), порожніх рядків та нейтральних речень, а контролер TestAvatarController довів точність вибору потрібного файлу анімації залежно від порогів впевненості. Нарешті, тести TestEmotionMLClassifier та інтеграційна перевірка публічного API підтвердили стійкість системи до збоїв за відсутності моделі штучного інтелекту й гарантували, що програма завжди повертає клієнтській частині сайту коректні, існуючі імена відеофайлів у форматі WebM, забезпечуючи стабільну роботу віртуального помічника для студентів кафедри ІПЗЕ за будь-яких умов.

Висновки до розділу 4

У четвертому розділі було успішно реалізовано практичний етап розробки програмного комплексу системи керування тривимірним аватаром віртуального помічника. На основі інженерних вимог щодо швидкодії та асинхронності обробки даних у реальному часі було обґрунтовано вибір технологічного стеку, де

серверну частину побудовано на базі мови Python та фреймворку FastAPI, а клієнтську частину інтегровано на вебресурс за допомогою стандартних інструментів фронтенд-розробки та оптимізованого графічного рушія відтворення WebM-відеопотоків. Для надійного збереження даних, ведення бази знань кафедри ІПЗЕ та підтримки довготривалого контексту сесій користувачів було спроєктовано логічну структуру з шести колекцій документо-орієнтованої бази даних MongoDB з індексацією ключових полів для мінімізації часу відгуку системи.

У межах програмного втілення було детально описано об'єктно-орієнтовану структуру обчислювального модуля та динаміку взаємодії його компонентів у часі, що дозволило наочно продемонструвати наскрізний шлях обробки запиту від моменту введення тексту студентом до моменту безшовної підміни анімації на екрані. Фінальним етапом роботи стало комплексне автоматизоване тестування розробленого програмного забезпечення за допомогою фреймворку pytest. Успішне виконання сорока чотирьох модульних та інтеграційних тестів повністю підтвердило технічну стабільність обчислювального конвеєра, його стійкість до математичного переповнення пам'яті сервера, правильність інверсії емоцій при появі заперечень, а також надійність роботи захисної матриці переходів. Отримані результати практичної реалізації та тестування довели високу точність і надійність створеного рішення, яке повністю готове до стабільної експлуатації в режимі реального часу.

5 РОБОТА КОРИСТУВАЧА З РОЗРОБЛЕНОЮ ПРОГРАМНОЮ СИСТЕМОЮ

У даному розділі наведено опис практичного функціонування розробленого програмного забезпечення, визначено ключові вимоги до середовища розгортання та запуску, а також продемонстровано реальні сценарії взаємодії користувача з користувацьким вебінтерфейсом інтелектуального 3D-асистента.

5.1 Визначення системних й апаратних вимог

Завдяки оптимізації обчислювального конвеєра та відмові від важкого тривимірного рендерингу в реальному часі, розроблена система не потребує потужних графічних відеокарт і має низькі апаратні вимоги.

Вимоги до серверного обладнання: Для стабільної роботи розгорнутого програмного комплексу, забезпечення асинхронної обробки запитів та функціонування бази даних достатньо звичайного двоядерного процесора з тактовою частотою від 2.0 ГГц. Мінімальний обсяг оперативної пам'яті сервера становить 4 ГБ. Для збереження інформації, логів та всього пулу готових відеофайлів із мімікою аватара потрібно від 2 ГБ вільного місця на швидкому твердотільному накопичувачі (SSD). Швидкість мережевого підключення сервера має бути не меншою за 10 Мбіт/с для забезпечення миттєвої передачі медіафайлів. Рекомендованою операційною системою для розгортання є Linux, зокрема дистрибутиви Ubuntu Server або Rocky Linux.

Вимоги до пристрою користувача: З боку користувача система є максимально невимогливою, оскільки вся логіка відтворення працює безпосередньо у вікні браузера. Запустити інтерактивного асистента можна на будь-кому сучасному смартфоні, планшеті або персональному комп'ютері, що має від 2 ГБ оперативної пам'яті. Головною вимогою є наявність сучасного веббраузера (наприклад, Google Chrome, Apple Safari, Mozilla Firefox або Microsoft

Edge) із підтримкою стандарту HTML5, технології Video Engine для плавного програвання легких медіафайлів та можливістю обробки асинхронних запитів.

5.2 Взаємодія користувача з вебінтерфейсом інтелектуального асистента

Практична експлуатація розробленого програмного комплексу здійснюється через графічний користувацький вебінтерфейс. Взаємодія з системою побудована за інтуїтивно зрозумілим сценарієм, який дозволяє користувачеві керувати сесією спілкування, ініціювати аналіз реплік та візуалізувати відповідні мімічні реакції персонажа у реальному часі.

Навігація між основними модулями програми реалізована за допомогою контекстного меню користувача, елементи якого представлено на рисунку 5.1.

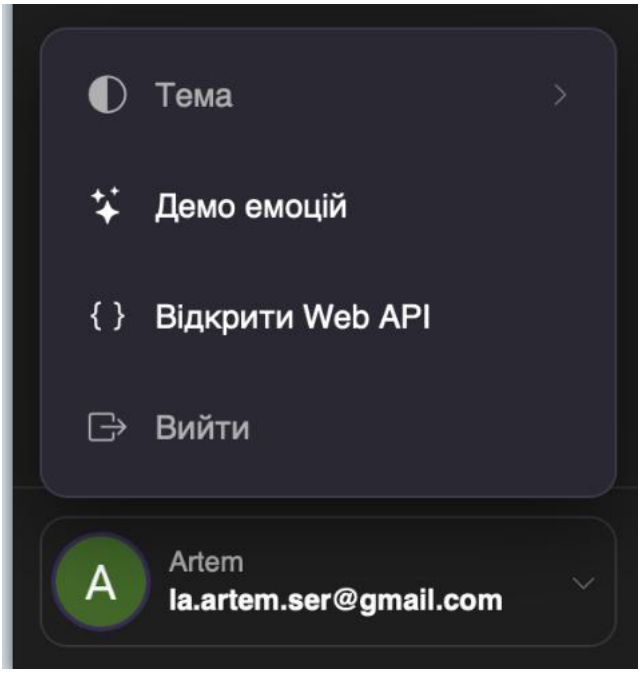


Рисунок 5.1 — Елементи керування та навігаційне меню користувача

Після успішної авторизації в системі користувачеві стає доступна персоналізована панель, яка відображає поточний профіль та надає наступні функціональні можливості:

- Тема — перемикання кольорової схеми інтерфейсу для забезпечення ергономічності сприйняття інформації;
- Демо емоцій — перехід до основного робочого простору інтерактивної демонстрації алгоритмів аналізу мови та селекції графічних потоків;
- Відкрити Web API — швидкий доступ до інтерактивної технічної документації серверних ендпоінтів для розробників;
- Вийти — коректне завершення поточної сесії користувача з очищенням токенів доступу.

Основний робочий простір демонстраційного режиму, куди користувач потрапляє після вибору відповідного пункту меню, наведено на рисунку 5.2.

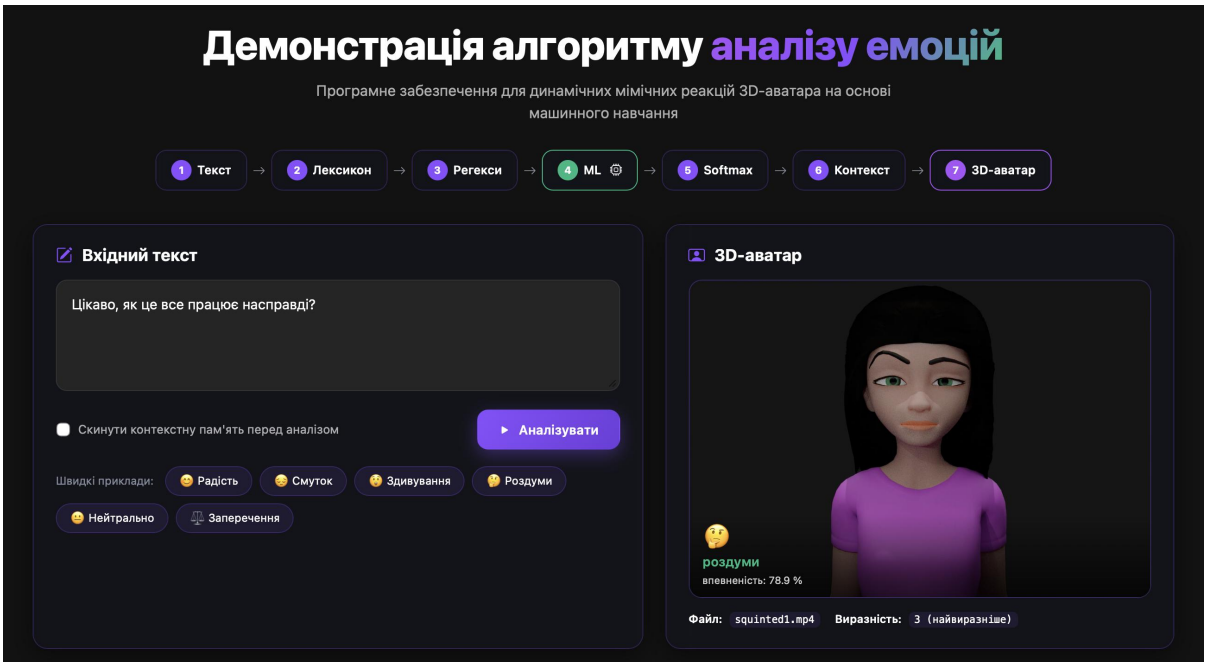


Рисунок 5.2 — Інтерфейс демонстраційної сторінки алгоритму аналізу емоцій

Екран інтерактивної взаємодії візуально розділено на дві основні робочі зони:

- Область введення (ліворуч): містить поле «Вхідний текст», де користувач може надрукувати довільне текстове повідомлення. Нижче розташовано чекбокс «Скинути контекстну пам'ять перед аналізом», який дозволяє за потреби примусово очистити історію попередніх реплік поточної сесії.

Для зручності та швидкого тестування системи передбачено блок «Швидкі приклади» із попередньо налаштованими емоційними пресетами («Радість», «Смуток», «Здивування», «Роздуми», «Нейтрально», «Заперечення»). Обробка запиту ініціюється натисканням кнопки «Аналізувати».

- Область візуалізації 3D-аватара (праворуч): призначена для відображення зворотного зв'язку від системи. У верхній частині блока розміщено інтерактивне вікно програвача мімічних реакцій. Після обробки тексту внизу вікна миттєво з'являється графічний індикатор із назвою розпізнаної домінуючої емоції та точним відсотком впевненості обчислювального конвеєра (наприклад, стан «роздуми» з впевненістю 78,9%). Додатково для технічного моніторингу виводиться назва відтворюваного медіафайлу у форматі WebM та розрахований рівень виразності міміки.

У верхній частині сторінки інтегровано візуальну схему наскрізного конвеєра, яка покроково підсвічує поточний етап обробки даних (від приймання тексту, через лексикони, регулярні вирази, ML-модель, Softmax та контекст до фінального виклику 3D-аватара). Це забезпечує наочність роботи алгоритмів під час проведення презентацій та демонстрацій програмного продукту.

Висновки до розділу 5

У п’ятому розділі було успішно проаналізовано умови практичної експлуатації створеного програмного продукту. Завдяки відмові від важкого тривимірного рендерингу на користь селекції легких відеофайлів у форматі WebM, розроблене програмне забезпечення отримало низькі вимоги до обладнання, забезпечуючи стабільну роботу на стандартних серверах під керуванням Linux та будь-яких сучасних пристроях користувачів. Створений адаптивний графічний вебінтерфейс із контекстним меню навігації та двозонною демонстраційною сторінкою довів свою ергономічність. Система забезпечує зручне введення реплік, асинхронний прорахунок конвеєра, наочне відображення метрик впевненості.

ВИСНОВКИ

У дипломній роботі повністю досягнуто поставлену мету та виконано всі завдання щодо розробки програмного забезпечення для динамічних мімічних реакцій 3D-аватара на основі машинного навчання. На основі дослідження концепцій афективних обчислень (Affective Computing) було спроєктовано та реалізовано наскрізний 12-етапний обчислювальний конвеєр, який забезпечує автоматизований аналіз тексту та миттєве керування візуальним стеком персонажа в реальному часі.

Для балансу швидкості та точності розроблено блок гібридного аналізу природної мови, що поєднує словникові методи пошуку емодзі, синтаксичні правила регулярних виразів та статистичну ML-класифікацію на базі логістичної регресії та векторизації TF-IDF за допомогою бібліотеки scikit-learn. Створено та математично обґрунтовано унікальну модель зваженої агрегації сигналів із модифікованою Softmax-нормалізацією, яка захищає пам'ять сервера від переповнення. Інтегровані модулі обробки заперечений та емоційної пам'яті чату забезпечили плавне згасання настрою аватара без хаотичних графічних стрибків.

Практична реалізація серверної частини виконана на мові Python із використанням асинхронного фреймворку FastAPI . Впровадження захисного шаблону RAG у зв'язці з хмароорієнтованим Together API дозволило повністю ліквідувати проблему «інформаційних галюцинацій» штучного інтелекту.

Завдяки відмові від важкого 3D-рендерингу на користь розробленого візуального рушія (Video Engine) створена система отримала низькі вимоги до обладнання й стабільно працює на стандартних процесорах (CPU) та будь-яких смартфонах студентів. Комплексне модульне тестування за допомогою фреймворку pytest підтвердило надійність архітектури (успішно пройдено 44 тести), а навчена модель показала достатню точність класифікації (Accuracy — 71,2%).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Human-Machine Interaction — an overview. URL: https://www.sciencedirect.com/topics/engineering/human-machine-interaction (дата звернення: 04.06.2026).
2. Єрмоленко С. В. Машинне навчання: загальний огляд та застосування для розпізнавання текстових і візуальних об’єктів. URL: https://www.researchgate.net/publication/395072317_MASINNE_NAVCANNA_ZAGALNIJ_OGLAD_TA_ZASTOSUVANNA_DLA_ROZPIZNAVANNA_TEKSTOVIH_I_VIZUALNIH_OB'EKTIV (дата звернення: 04.06.2026)
3. DeepSeek-V3 Technical Report. URL: https://arxiv.org/html/2412.19437v1 (дата звернення: 04.06.2026).
4. Server-Sent Events // Сучасний підручник з JavaScript. URL: https://uk.javascript.info/server-sent-events (дата звернення: 04.06.2026).
5. Picard R. W. Affective Computing. Cambridge : MIT Press, 2000. 304 p.
6. Bing Liu. Sentiment Analysis: Mining Opinions, Sentiments, and Emotions. Cambridge : Cambridge University Press, 2020. 404 p.
7. Christopher M. Bishop. Pattern Recognition and Machine Learning. New York : Springer, 2006. 738 p.
8. Together AI Documentation: Introduction. URL: https://docs.together.ai/intro (дата звернення: 04.06.2026).
9. Synthesia Research: AI Video Generation and Virtual Avatars. URL: https://www.synthesia.io/research (дата звернення: 04.06.2026).
10. Документація по Python: Підручник Python. URL: https://docs.python.org/uk/3/tutorial/index.html (дата звернення: 04.06.2026).
11. NumPy Documentation: NumPy User Guide. URL: https://numpy.org/doc/stable/ (дата звернення: 04.06.2026).
12. Самовчитель HTML та CSS. URL: https://html-css.co.ua/ (дата звернення: 04.06.2026).

13. MongoDB Documentation. URL: https://www.mongodb.com/docs/ (дата звернення: 04.06.2026).

ДОДАТОК А Лістинг розробленої системи

Модуль аналізу емоцій:

```
from __future__ import annotations

import math
import re
import time
from collections import deque
from dataclasses import dataclass, field
from typing import Deque, Dict, FrozenSet, List, Optional, Tuple

from assistant_core.ml_classifier import (
    EmotionMLClassifier,
    get_ml_classifier,
    is_ml_available,
)

EMOTIONS: FrozenSet[Emotion] = frozenset(
    {"neutral", "happy", "sad", "surprise", "thinking", "angry", "disgust"}
)
EMOTION_LIST: List[Emotion] = [
    "neutral", "happy", "sad", "surprise", "thinking","angry", "disgust",
]

# Фактичний набір .mp4 у avatar/animations/ — усі використовуються в ANIMATION_MAP (high/med/fallback).
ANIMATION_CLIP_FILES: Tuple[str, ...] = (
    "angry.mp4",  "confused.mp4", "disgust.mp4", "excited.mp4", "fear.mp4", "happy.mp4", "muse.mp4",
"sad.mp4","speak_blink.mp4","speak.mp4","squinted1.mp4", "surprize1.mp4",
)

# Мінімальна впевненість для зміни стану аватара (нижче — лишаємо neutral)
CONFIDENCE_THRESHOLD = 0.30

# На переході neutral → не-neutral: окремий нижній поріг product(confidence, score),
# бо після softmax на 7 класах argmax часто ~0.14–0.18 (агрегатор уже відсіяв «ніякого сигналу»).
NEUTRAL_EXIT_CONFIDENCE_FACTOR = 0.12

# Якщо softmax-лідер ≠ поточний стан аватара: дозволити перехід до лідера при слабшому product,
# інакше «застряє» self-loop (happy→happy з p≈0.12), хоча діаграма показує thinking/sad тощо.
SOFTMAX_TOP_TRANSITION_FACTOR = 0.15

# Розмір вікна контексту (кількість попередніх результатів, що впливають на поточний)
CONTEXT_WINDOW_SIZE = 5

# Вага контексту у фінальному рішенні (0.0 = без згладжування, 1.0 = тільки контекст)
CONTEXT_SMOOTHING_ALPHA = 0.25

# Мінімальна кількість балів для виходу з нейтральної зони
NEUTRAL_FLOOR_SCORE = 1.5

@dataclass
class EmotionResult:
    emotion: Emotion
    confidence: float
    scores: Dict[Emotion, float]
    raw_scores: Dict[Emotion, float]
```

```
    method: str
    context_smoothed: bool = False
    tokens_matched: List[str] = field(default_factory=list)
    component_scores: Dict[str, Dict[Emotion, float]] = field(default_factory=dict)
    ml_used: bool = False

    def to_dict(self) -> Dict:
        return {
            "emotion": self.emotion,
            "confidence": round(self.confidence, 4),
            "scores": {k: round(v, 4) for k, v in self.scores.items()},
            "raw_scores": {k: round(v, 4) for k, v in self.raw_scores.items()},
            "method": self.method,
            "context_smoothed": self.context_smoothed,
            "tokens_matched": self.tokens_matched,
            "component_scores": {
                comp: {k: round(v, 4) for k, v in vec.items()}
                for comp, vec in self.component_scores.items()
            },
            "ml_used": self.ml_used,
        }


@dataclass
class AvatarAnimation:
    filename: str
    emotion: Emotion
    priority: int
    loop: bool = True

class EmotionLexicon:

    LEXICON: Dict[str, Dict[Emotion, float]] = {
        "дякую":       {"happy": 2.5},
        "дяку":        {"happy": 1.5},
        "вдячний":     {"happy": 3.0},
        "вдячна":      {"happy": 3.0},
        "вдячність":   {"happy": 3.0},
        "радий":       {"happy": 3.5},
        "рада":        {"happy": 3.5},
        "радість":     {"happy": 3.5},
        …..
        # emoji sad
        "    ": {"sad": 4.0},
        "    ": {"sad": 4.5},
        "    ": {"sad": 3.5},
        "    ": {"sad": 3.0},
        "    ": {"sad": 4.0},
        "    ": {"sad": 3.5},
        "    ": {"sad": 3.0},
        "    ": {"sad": 3.0},
        "    ": {"sad": 3.5},
        "    ": {"sad": 3.5},
        "    ": {"sad": 3.0},
        "    ": {"sad": 3.0},
        "ого":          {"surprise": 4.0}
```

```
    }

    def __init__(self) -> None:
        # Нормалізований словник для O(1) пошуку
        self._lookup: Dict[str, Dict[Emotion, float]] = {
            k.lower(): v for k, v in self.LEXICON.items()
        }

    def score(self, tokens: List[str]) -> Tuple[Dict[Emotion, float], List[str]]:

        scores: Dict[Emotion, float] = {e: 0.0 for e in EMOTION_LIST}
        matched: List[str] = []

        for tok in tokens:
            # Пошук точного збігу
            if tok in self._lookup:
                for emotion, weight in self._lookup[tok].items():
                    scores[emotion] += weight
                matched.append(tok)
                continue
            if len(tok) >= 5:
                stem = tok[:5]
                for key in self._lookup:
                    if key.startswith(stem) and abs(len(key) - len(tok)) <= 3:
                        for emotion, weight in self._lookup[key].items():
                            # Стем дає 70% від повної ваги
                            scores[emotion] += weight * 0.7
                        matched.append(f"~{tok}")
                        break

        return scores, matched

class PatternAnalyzer:
    # (regex_pattern, {emotion: score_добавка})
    PATTERNS: List[Tuple[str, Dict[Emotion, float]]] = [
        # Подвійний/потрійний оклик
        (r"!{2,}", {"happy": 1.2, "surprise": 1.0, "angry": 0.8}),
        # Чотири й більше окликів — частіше агресія / напруження
        (r"!{4,}", {"angry": 2.5, "surprise": 1.0}),
        # Питальний знак
        (r"\?{1,}", {"thinking": 1.5}),
        # Питання + оклик
        (r"[?!]{2,}", {"surprise": 2.0, "thinking": 0.5}),
        # Питальні слова на початку рядка
        (r"^(чому|навіщо|як|де|коли|хто|що|скільки)\b", {"thinking": 1.5}),
        # Caps lock (≥4 символів) → злість / акцент
        (r"\b[А-ЯЁЇІЄA-Z]{4,}\b", {"angry": 2.5, "surprise": 0.8}),
        # Повтор голосних (ааааа, оооо)
        (r"([аеиіоуяєї])\1{2,}", {"surprise": 2.0}),
        # Три і більше крапки
        (r"\.{3,}", {"sad": 1.5, "thinking": 1.0}),
        # Текстові смайлики — радість
        (r"[:;8XB]-?[\)D\]3]", {"happy": 2.5}),
        # ^_^, >^_^<, очі ^
        (r"(?:[\^◠][_.\s]?[\^◠])+", {"happy": 2.2}),
        # Текстові смайлики — сум (двокрапка або крапка з комою)
        (r"[:;≈≤]-?[\(\[<]", {"sad": 2.5}),
        # «=», «o» або цифра як очі + рот
        (r"[0oOoO_=][:_.''\-][(\[\{cc]", {"sad": 2.5}),
        (r"T[._'-]?T|(?<!\w)u_u(?!\w)", {"sad": 2.8}),
```

```
        # Одна «(» / повна ширина «（» у кінці — часто смайл суму без ':'
        (r"(?:\(|\uff08)\s*$", {"sad": 3.5}),
        # Рот «cc» або латинське C після ':' (:( :c)
        (r"[:;8][\-~^]?[ccCCc]\b", {"sad": 2.8}),
        # XD, xd
        (r"\bx[dD]\b", {"happy": 2.5}),
        # лол, ахаха, хехе
        (r"\b(лол|лол+|ахах+|хехе|хіхі|haha|hehe)\b", {"happy": 2.5}),
        # «як жити далі…» тригерить сум у навчанні ML; коли поруч є явно позитив → радість/здивування
        (r"жити\s+далі[^\.\?!]{0,80}хорош\w*", {"happy": 2.5, "surprise": 2.0}),
        # Студентські провали (ML часто дає хибну радість)
        (r"\bне\s+отрима\w+\s+залік\w*", {"sad": 3.5}),
        (r"\bне\s+(?:склав|здав)\w*\s+(?:залік\w*|іспит\w*|екзамен\w*|сесі\w*)", {"sad": 3.5}),
        (r"\bпровалив\w*\s+(?:залік|іспит|сесі)\w*", {"sad": 3.2}),
        # Вигуки
        (r"^(ого|ой|ах|ааа|оо+)\b", {"surprise": 2.5}),
    ]

    # Патерни, які мусять бути case-sensitive (CAPS-детектор).
    _CASE_SENSITIVE_PATTERNS: FrozenSet[str] = frozenset(
        {r"\b[А-ЯЁЇІЄA-Z]{4,}\b"}
    )

    def __init__(self) -> None:
        self._compiled = []
        for p, scores in self.PATTERNS:
            flags = re.UNICODE
            if p not in self._CASE_SENSITIVE_PATTERNS:
                flags |= re.IGNORECASE
            self._compiled.append((re.compile(p, flags), scores))

    def score(self, text: str) -> Dict[Emotion, float]:
        scores: Dict[Emotion, float] = {e: 0.0 for e in EMOTION_LIST}
        for compiled, contribution in self._compiled:
            matches = compiled.findall(text)
            if matches:
                count = min(len(matches), 3)  # обмеження: не більше 3 співпадінь на патерн
                for emotion, weight in contribution.items():
                    scores[emotion] += weight * count
        return scores

class IntensityModifier:

    INTENSITY_MAP: Dict[str, float] = {
        # Extreme
        "надзвичайно": 2.0,
        "неймовірно":  2.0,
        "неймовірний": 2.0,
        …
        # High
        "дуже":       1.7,
        "надто":      1.7,
        …
        # Medium
        "досить":     1.4,
        "досить":     1.4,
       ….
        # Low
        "трохи":      0.9,
        "ледь":       0.8,
```

```
        "злегка":      0.8,
        "трішки":      0.8,
        …
    }

    def __init__(self) -> None:
        self._lookup = {k.lower(): v for k, v in self.INTENSITY_MAP.items()}

    def apply(
        self,
        tokens: List[str],
        scores: Dict[Emotion, float],
    ) -> Dict[Emotion, float]:

        modified = dict(scores)
        max_multiplier = 1.0

        for tok in tokens:
            if tok in self._lookup:
                max_multiplier = max(max_multiplier, self._lookup[tok])

        if max_multiplier != 1.0:
            for emotion in EMOTION_LIST:
                if emotion != "neutral":
                    modified[emotion] *= max_multiplier

        return modified

class NegationProcessor:
    NEGATION_TOKENS: FrozenSet[str] = frozenset({
        "не", "ні", "ніяк", "ніколи", "нічого", "ніхто",
        "ніде", "нікуди", "нізащо", "жодного", "жодна",
        "жоден", "без", "проти", "навпаки",
    })

    NEGATION_WINDOW = 4  # кількість токенів після маркера

    # Таблиця інверсії емоцій при запереченні
    INVERSION_TABLE: Dict[Emotion, Emotion] = {
        "happy":    "sad",
        "sad":      "happy",
        "surprise": "thinking",
        "thinking": "neutral",
        "angry":    "neutral",
        "disgust":  "neutral",
        "neutral":  "neutral",
    }

    def apply(
        self,
        tokens: List[str],
        scores: Dict[Emotion, float],
    ) -> Dict[Emotion, float]:

        has_negation = any(tok in self.NEGATION_TOKENS for tok in tokens)
        if not has_negation:
            return scores

        # Знаходимо домінуючу не-neutral емоцію
        non_neutral = {e: v for e, v in scores.items() if e != "neutral" and v > 0}
```

```
        if not non_neutral:
            return scores

        dominant = max(non_neutral, key=lambda e: non_neutral[e])
        if dominant not in ("happy", "surprise"):
            return scores

        inverted_emotion = self.INVERSION_TABLE.get(dominant, "neutral")

        # Переносимо бали домінуючої емоції на інвертовану
        modified = dict(scores)
        dominant_score = modified.pop(dominant, 0.0)
        modified[dominant] = 0.0
        modified[inverted_emotion] = modified.get(inverted_emotion, 0.0) + dominant_score * 0.8

        return modified

class ScoreAggregator:
    # Ваги при наявності ML-моделі
    W_LEX_ML = 0.35
    W_PAT_ML = 0.20
    W_ML     = 0.45

    # Ваги без ML
    W_LEX_NOML = 0.70
    W_PAT_NOML = 0.30

    # Множник для ML-ймовірностей (приводимо [0..1] до шкали 0..~5)
    ML_SIGNAL_SCALE = 5.0

    def combine(
        self,
        lexicon_scores: Dict[Emotion, float],
        pattern_scores: Dict[Emotion, float],
        ml_scores: Optional[Dict[Emotion, float]] = None,
    ) -> Dict[Emotion, float]:
        """
        Зважена комбінація балів з 2-х або 3-х джерел.

        Параметри:
            lexicon_scores – сирі бали з лексикона
            pattern_scores – сирі бали з регексів
            ml_scores      – probability-розподіл [0..1] від ML-моделі (опціонально)
        """
        combined: Dict[Emotion, float] = {}
        if ml_scores is not None:
            for e in EMOTION_LIST:
                combined[e] = (
                    self.W_LEX_ML * lexicon_scores.get(e, 0.0)
                    + self.W_PAT_ML * pattern_scores.get(e, 0.0)
                    + self.W_ML * ml_scores.get(e, 0.0) * self.ML_SIGNAL_SCALE
                )
        else:
            for e in EMOTION_LIST:
                combined[e] = (
                    self.W_LEX_NOML * lexicon_scores.get(e, 0.0)
                    + self.W_PAT_NOML * pattern_scores.get(e, 0.0)
                )
        return combined
```

```
    @staticmethod
    def softmax(scores: Dict[Emotion, float]) -> Dict[Emotion, float]:
        vals = {e: scores.get(e, 0.0) for e in EMOTION_LIST}
        max_val = max(vals.values()) if vals else 0.0
        exp_vals = {e: math.exp(v - max_val) for e, v in vals.items()}
        total = sum(exp_vals.values()) or 1.0
        return {e: exp_vals[e] / total for e in EMOTION_LIST}

    def decide(
        self,
        raw_scores: Dict[Emotion, float],
        normalized: Dict[Emotion, float],
    ) -> Tuple[Emotion, float]:
        total_raw = sum(raw_scores.get(e, 0.0) for e in EMOTION_LIST if e != "neutral")

        if total_raw < NEUTRAL_FLOOR_SCORE:
            return "neutral", normalized.get("neutral", 0.5)

        best_emotion = max(normalized, key=lambda e: normalized[e])
        confidence = normalized[best_emotion]
        neut = normalized.get("neutral", 0.0)

        if confidence < CONFIDENCE_THRESHOLD:
            if best_emotion == "neutral" or confidence <= neut + 1e-12:
                return "neutral", neut
            return best_emotion, confidence

        return best_emotion, confidence

@dataclass
class _ContextEntry:
    emotion: Emotion
    confidence: float
    timestamp: float

class ContextWindow:

    def __init__(self, maxlen: int = CONTEXT_WINDOW_SIZE) -> None:
        self._buffer: Deque[_ContextEntry] = deque(maxlen=maxlen)

    def add(self, emotion: Emotion, confidence: float) -> None:
        """Додає результат до буферу."""
        self._buffer.append(
            _ContextEntry(emotion=emotion, confidence=confidence, timestamp=time.time())
        )

    def smooth(
        self,
        current_emotion: Emotion,
        current_confidence: float,
    ) -> Tuple[Emotion, float, bool]:

        if not self._buffer:
            return current_emotion, current_confidence, False

        # Частота кожної емоції у вікні (зважена за часом)
        now = time.time()
        freq: Dict[Emotion, float] = {e: 0.0 for e in EMOTION_LIST}
        total_weight = 0.0
```

```
        for entry in self._buffer:
            # Експоненційне затухання: нещодавні записи важливіші
            age = now - entry.timestamp
            weight = math.exp(-age / 60.0)  # half-life ≈ 60 секунд
            freq[entry.emotion] += weight * entry.confidence
            total_weight += weight

        if total_weight > 0:
            freq = {e: v / total_weight for e, v in freq.items()}

        context_emotion = max(freq, key=lambda e: freq[e])
        context_confidence = freq[context_emotion]

        # Якщо емоції збігаються — підсилюємо впевненість
        if current_emotion == context_emotion:
            boosted = min(1.0, current_confidence + 0.05 * context_confidence)
            return current_emotion, boosted, False

        # Якщо емоції різняться — частково зберігаємо поточну
        if context_confidence > 0.5 and current_confidence < 0.5:
            # Контекст сильніший — повертаємо контекстну емоцію
            return context_emotion, context_confidence * CONTEXT_SMOOTHING_ALPHA + current_confidence * (1 -
CONTEXT_SMOOTHING_ALPHA), True

        # Поточна емоція впевненіша — приймаємо, але зменшуємо стрибок
        damped_confidence = current_confidence * (1.0 - CONTEXT_SMOOTHING_ALPHA * context_confidence)
        return current_emotion, max(damped_confidence, CONFIDENCE_THRESHOLD), False

class TransitionMatrix:
      MATRIX: Dict[Emotion, Dict[Emotion, float]] = {
        "neutral": {
            **{e: 1.0 for e in EMOTION_LIST},
        },
        "happy": {
            "neutral": 1.0,
            "happy": 1.0,
            "sad": 0.3,
            "surprise": 0.8,
            "thinking": 0.7,
            "angry": 0.35,
            "disgust": 0.35,
        },
        "sad": {
            "neutral": 1.0,
            "happy": 0.3,
            "sad": 1.0,
            "surprise": 0.6,
            "thinking": 0.8,
            "angry": 0.55,
            "disgust": 0.5,
        },
        …
    }

    def allowed(
        self,
        from_emotion: Emotion,
        to_emotion: Emotion,
        confidence: float,
        *,
```

```
        softmax_argmax: Optional[Emotion] = None,
    ) -> bool:
        if from_emotion == to_emotion:
            return confidence >= NEUTRAL_EXIT_CONFIDENCE_FACTOR
        transition_score = self.MATRIX.get(from_emotion, {}).get(to_emotion, 1.0)
        product = confidence * transition_score
        need = CONFIDENCE_THRESHOLD
        if (
            from_emotion == "neutral"
            and to_emotion != "neutral"
            and transition_score >= 0.99
        ):
            need = NEUTRAL_EXIT_CONFIDENCE_FACTOR
        if product >= need:
            return True
        if (
            softmax_argmax is not None
            and to_emotion == softmax_argmax
            and from_emotion != to_emotion
        ):
            return product >= SOFTMAX_TOP_TRANSITION_FACTOR
        return False

    def best_allowed(
        self,
        from_emotion: Emotion,
        scores: Dict[Emotion, float],
    ) -> Tuple[Emotion, float]:
        softmax_top = max(scores, key=lambda e: scores[e])
        candidates = sorted(scores.items(), key=lambda x: -x[1])
        for emotion, confidence in candidates:
            if emotion == from_emotion and softmax_top != from_emotion:
                continue
            if self.allowed(
                from_emotion, emotion, confidence, softmax_argmax=softmax_top
            ):
                return emotion, confidence
        return "neutral", scores.get("neutral", 0.5)

class EmotionClassifier:
    def __init__(self, ml_classifier: Optional[EmotionMLClassifier] = None) -> None:
        self._lexicon = EmotionLexicon()
        self._patterns = PatternAnalyzer()
        self._intensity = IntensityModifier()
        self._negation = NegationProcessor()
        self._aggregator = ScoreAggregator()
        self._context = ContextWindow()
        self._transitions = TransitionMatrix()
        self._current_emotion: Emotion = "neutral"
        # ML-класифікатор: lazy-load singleton (працює і без натренованої моделі)
        self._ml: EmotionMLClassifier = ml_classifier or get_ml_classifier()

    @staticmethod
    def _preprocess(text: str) -> List[str]:
        if not text:
            return []
        # Нормалізація пробілів
        text = re.sub(r"\s+", " ", text.strip())
        # Токенізація: слова + emoji + числа + пунктуація
        tokens = re.findall(
```

```
        r"[\U0001F600-\U0001F64F\U0001F300-\U0001F5FF"
        r"\U0001F680-\U0001F6FF\U0001F700-\U0001F77F"
        r"\U0001F780-\U0001F7FF\U0001F800-\U0001F8FF"
        r"\U0001F900-\U0001F9FF\U0001FA00-\U0001FA6F"
        r"\U0001FA70-\U0001FAFF\U00002702-\U000027B0"
        r"\U000024C2-\U0001F251]+"
        r"|[a-яёїієa-z`']+[-a-яёїієa-z`']*"
        r"|\d+",
        text.lower(),
        re.UNICODE,
    )
    return [t for t in tokens if t]

def analyze(self, text: str) -> EmotionResult:
    if not text or not text.strip():
        return EmotionResult(
            emotion="neutral",
            confidence=1.0,
            scores={e: (1.0 if e == "neutral" else 0.0) for e in EMOTION_LIST},
            raw_scores={e: 0.0 for e in EMOTION_LIST},
            method="empty",
        )

    tokens = self._preprocess(text)

    # Крок 1: Лексичне оцінювання
    lex_scores, matched_tokens = self._lexicon.score(tokens)

    # Крок 2: Паттерне оцінювання (на оригінальному тексті)
    pat_scores = self._patterns.score(text)

    # Крок 3: ML-передбачення (predict_proba). Якщо модель не завантажена —
    # повертає neutral=1.0, тому ансамбль не страждає.
    ml_used = self._ml.is_loaded
    ml_scores: Optional[Dict[Emotion, float]] = None
    if ml_used:
        ml_scores = self._ml.predict(text)
        # Підтримка повного словника (на випадок якщо модель навчена не на всіх класах)
        ml_scores = {e: ml_scores.get(e, 0.0) for e in EMOTION_LIST}

    # Крок 4: Зважена комбінація сигналів (lex + pat [+ ml])
    combined = self._aggregator.combine(lex_scores, pat_scores, ml_scores)

    # Крок 5: Модифікатори інтенсивності
    intensified = self._intensity.apply(tokens, combined)

    # Крок 6: Обробка заперечень
    adjusted = self._negation.apply(tokens, intensified)

    # Зберігаємо сирі бали для звіту
    raw_scores = dict(adjusted)

    # Крок 7: Softmax нормалізація
    normalized = self._aggregator.softmax(adjusted)

    # Крок 8: Вибір класу
    emotion, confidence = self._aggregator.decide(raw_scores, normalized)

    # Крок 9: Контекстне згладжування
    emotion, confidence, was_smoothed = self._context.smooth(emotion, confidence)
```

```
        # Крок 10: Перевірка матриці переходів
        emotion, confidence = self._transitions.best_allowed(
            self._current_emotion, {emotion: confidence, **{e: normalized[e] for e in EMOTION_LIST}}
        )

        # Оновлення поточного стану
        self._context.add(emotion, confidence)
        self._current_emotion = emotion

        # Визначення методу
        active = []
        if sum(lex_scores.values()) > 0:
            active.append("lexicon")
        if sum(pat_scores.values()) > 0:
            active.append("pattern")
        if ml_used and ml_scores and any(v > 0.05 for v in ml_scores.values()):
            active.append("ml")
        if not active:
            method = "fallback"
        elif len(active) == 1:
            method = active[0]
        else:
            method = "hybrid"

        component_scores: Dict[str, Dict[Emotion, float]] = {
            "lexicon": dict(lex_scores),
            "pattern": dict(pat_scores),
        }
        if ml_scores is not None:
            component_scores["ml"] = dict(ml_scores)

        return EmotionResult(
            emotion=emotion,
            confidence=confidence,
            scores=normalized,
            raw_scores=raw_scores,
            method=method,
            context_smoothed=was_smoothed,
            tokens_matched=matched_tokens,
            component_scores=component_scores,
            ml_used=ml_used,
        )

    def reset_context(self) -> None:
        """Скидає контекстне вікно (для нової сесії чату)."""
        self._context = ContextWindow()
        self._current_emotion = "neutral"

class AvatarController:
      HIGH_CONFIDENCE = 0.65
    MED_CONFIDENCE = 0.45
    # Для POST /api/emotion/analyze? демо-сторінки: зсув порогів униз → частіший вибір
    # «strong» і «medium» кліпа при середній впевненості (чат і прод не змінюються).
    DEMO_HIGH_DELTA = 0.10
    # Для демо med з ~0.30: типовий softmax-p(класу)≈0.31 уже дає тематичний кліп, не «розмовний» blink.
    DEMO_MED_DELTA = 0.15

    # Базовий кліп, якщо клас невідомий або немає запису в мапі
    DEFAULT_CLIP = "muse.mp4"
```

```
    # emotion → (high_conf_file, med_conf_file, fallback_file)
    ANIMATION_MAP: Dict[Emotion, Tuple[str, str, str]] = {
        "neutral": ("muse.mp4", "speak_blink.mp4", "speak.mp4"),
        # fallback = м'який, але все ще клас «радість» (не нейтральний blink)
        "happy": ("excited.mp4", "happy.mp4", "happy.mp4"),
        "sad": ("sad.mp4", "speak.mp4", "muse.mp4"),
        "surprise": ("surprize1.mp4", "fear.mp4", "confused.mp4"),
        "thinking": ("squinted1.mp4", "confused.mp4", "speak_blink.mp4"),
        "angry": ("angry.mp4", "speak.mp4", "fear.mp4"),
        "disgust": ("disgust.mp4", "squinted1.mp4", "muse.mp4"),
    }

    def select_animation(
        self,
        result: EmotionResult,
        *,
        demo_responsive: bool = False,
    ) -> AvatarAnimation:
        d = self.DEFAULT_CLIP
        high, med, fallback = self.ANIMATION_MAP.get(
            result.emotion,
            (d, d, d),
        )

        high_t = self.HIGH_CONFIDENCE - (
            self.DEMO_HIGH_DELTA if demo_responsive else 0.0
        )
        med_t = self.MED_CONFIDENCE - (
            self.DEMO_MED_DELTA if demo_responsive else 0.0
        )

        if result.confidence >= high_t:
            filename = high
            priority = 3
        elif result.confidence >= med_t:
            filename = med
            priority = 2
        else:
            filename = fallback
            priority = 1
        if (
            demo_responsive
            and result.emotion != "neutral"
            and priority == 1
        ):
            filename = med
            priority = 2

        return AvatarAnimation(
            filename=filename,
            emotion=result.emotion,
            priority=priority,
        )
_classifier: Optional[EmotionClassifier] = None
_demo_classifier: Optional[EmotionClassifier] = None
_avatar_controller: Optional[AvatarController] = None


def get_classifier() -> EmotionClassifier:
```

```
    """Повертає або створює singleton EmotionClassifier для чату."""
    global _classifier
    if _classifier is None:
        _classifier = EmotionClassifier()
    return _classifier

def get_demo_classifier() -> EmotionClassifier:
    """Окремий singleton для демо-сторінки з власним ContextWindow, ізольованим від чату."""
    global _demo_classifier
    if _demo_classifier is None:
        _demo_classifier = EmotionClassifier()
    return _demo_classifier

def get_avatar_controller() -> AvatarController:
    """Повертає або створює singleton AvatarController."""
    global _avatar_controller
    if _avatar_controller is None:
        _avatar_controller = AvatarController()
    return _avatar_controller

def analyze_emotion(text: str) -> EmotionResult:
    return get_classifier().analyze(text)


def select_animation_for_emotion_label(
    emotion: str,
    confidence: float = 0.72,
) -> AvatarAnimation:
    em: Emotion = emotion if emotion in EMOTIONS else "neutral"
    one_hot = {e: (1.0 if e == em else 0.0) for e in EMOTION_LIST}
    result = EmotionResult(
        emotion=em,
        confidence=confidence,
        scores=dict(one_hot),
        raw_scores=dict(one_hot),
        method="emotion_label",
    )
    return get_avatar_controller().select_animation(result)


def get_avatar_animation(text: str) -> AvatarAnimation:
    result = analyze_emotion(text)
    return get_avatar_controller().select_animation(result)


def reset_session_context() -> None:
    get_classifier().reset_context()


def get_engine_status() -> Dict:
    aggregator = ScoreAggregator()
    return {
        "ml_available": is_ml_available(),
        "ml_loaded": get_ml_classifier().is_loaded,
        "ml_model_path": str(get_ml_classifier().model_path),
        "emotions": EMOTION_LIST,
        "confidence_threshold": CONFIDENCE_THRESHOLD,
        "neutral_floor_score": NEUTRAL_FLOOR_SCORE,
        "context_window_size": CONTEXT_WINDOW_SIZE,
        "weights_with_ml": {
```

```
            "lexicon": aggregator.W_LEX_ML,
            "pattern": aggregator.W_PAT_ML,
            "ml": aggregator.W_ML,
        },
        "weights_no_ml": {
            "lexicon": aggregator.W_LEX_NOML,
            "pattern": aggregator.W_PAT_NOML,
        },
    }
```

Модуль машинного навчання для класифікації емоцій:

```
from __future__ import annotations

import logging
import os
from dataclasses import dataclass
from pathlib import Path
from typing import Dict, List, Optional, Tuple

logger = logging.getLogger(__name__)

# Шлях за замовчуванням, куди зберігається натренована модель
DEFAULT_MODEL_PATH = Path(__file__).resolve().parent / "models" / "emotion_model.joblib"

try:
    import joblib  # type: ignore
    from sklearn.feature_extraction.text import TfidfVectorizer  # type: ignore
    from sklearn.linear_model import LogisticRegression  # type: ignore
    from sklearn.metrics import (  # type: ignore
        accuracy_score,
        classification_report,
        confusion_matrix,
        f1_score,
        precision_score,
        recall_score,
    )
    from sklearn.model_selection import train_test_split  # type: ignore
    from sklearn.pipeline import FeatureUnion, Pipeline  # type: ignore

    _HAS_SKLEARN = True
except ImportError:  # pragma: no cover
    _HAS_SKLEARN = False
    logger.warning(
        "scikit-learn / joblib не встановлено — ML-класифікатор недоступний. "
        "Виконайте: pip install scikit-learn joblib"
    )
EMOTION_CLASSES: List[str] = [
    "neutral",
    "happy",
    "sad",
    "surprise",
    "thinking",
    "angry",
    "disgust",
]


@dataclass
```

```
class TrainingMetrics:
    accuracy: float
    precision_macro: float
    recall_macro: float
    f1_macro: float
    classification_report: str
    confusion_matrix: List[List[int]]
    classes: List[str]
    n_train: int
    n_test: int

    def to_dict(self) -> Dict:
        return {
            "accuracy": round(self.accuracy, 4),
            "precision_macro": round(self.precision_macro, 4),
            "recall_macro": round(self.recall_macro, 4),
            "f1_macro": round(self.f1_macro, 4),
            "classification_report": self.classification_report,
            "confusion_matrix": self.confusion_matrix,
            "classes": self.classes,
            "n_train": self.n_train,
            "n_test": self.n_test,
        }
class EmotionMLClassifier:
    def __init__(self, model_path: Optional[Path] = None) -> None:
        self._model_path = Path(model_path) if model_path else DEFAULT_MODEL_PATH
        self._pipeline: Optional[object] = None  # sklearn Pipeline | None
        self._classes: List[str] = list(EMOTION_CLASSES)

    @property
    def is_loaded(self) -> bool:
        """Чи готова модель до інференсу."""
        return self._pipeline is not None

    @property
    def model_path(self) -> Path:
        return self._model_path

    @property
    def classes(self) -> List[str]:
        return list(self._classes)

    @staticmethod
    def _build_pipeline():
        if not _HAS_SKLEARN:
            raise RuntimeError("scikit-learn не встановлено. Запустіть: pip install scikit-learn joblib")

        word_vec = TfidfVectorizer(
            analyzer="word",  # ознаки з токенів-слів (пробіл як межа)
            ngram_range=(1, 2),  # уніграми + біграми слів
            min_df=1,  # слово/н-грама входить у словник, якщо є хоча б у 1 документі
            max_df=0.95,  # відсікати дуже «загальні» терміни (є >95% документів)
            sublinear_tf=True,  # TF як log(1+count), щоб довгі тексти не домінували
            lowercase=True,  # привести текст до нижнього регістру перед лічильником
        )
        char_vec = TfidfVectorizer(
            analyzer="char_wb",  # символьні n-грами в межах слова (краще для флексії)
            ngram_range=(3, 5),  # шматки з 3–5 символів
            min_df=1,  # той самий поріг рідкості, що й для word
            max_df=0.95,  # той самий відсікання загальних n-грам
```

```
            sublinear_tf=True,  # той самий згладжений TF
            lowercase=True,  # нижній регістр перед підрахунком символів
        )
        features = FeatureUnion(
            [
                ("word_tfidf", word_vec),  # гілка: словесні TF-IDF ознаки
                ("char_tfidf", char_vec),  # гілка: символьні TF-IDF ознаки
            ]
        )
         clf = LogisticRegression(
            solver="liblinear",  # алгоритм оптимізації ваг (стійкий для цього пайплайна)
            max_iter=2000,  # максимум ітерацій навчання; збільшено на випадок повільної збіжності
            C=1.0,  # сила L2-регуляризації: менше C → сильніший штраф на великі ваги
            class_weight="balanced",  # зважити класи зворотно до їх частоти в train
        )
        pipeline = Pipeline(
            [
                ("features", features),  # крок 1: текст → вектор ознак
                ("clf", clf),  # крок 2: вектор → ймовірності класів (LogReg)
            ]
        )
        return pipeline

    def train(
        self,
        texts: List[str],
        labels: List[str],
        test_size: float = 0.2,
        random_state: int = 42,
    ) -> TrainingMetrics:
        if not _HAS_SKLEARN:
            raise RuntimeError("scikit-learn не встановлено.")

        if len(set(labels)) < 2:
            raise ValueError("Для навчання потрібно щонайменше 2 класи.")

        X_train, X_test, y_train, y_test = train_test_split(
            texts, labels,
            test_size=test_size,
            random_state=random_state,
            stratify=labels,
        )

        pipeline = self._build_pipeline()
        pipeline.fit(X_train, y_train)
        self._pipeline = pipeline

        y_pred = pipeline.predict(X_test)
        labels_sorted = sorted(set(labels))

        metrics = TrainingMetrics(
            accuracy=accuracy_score(y_test, y_pred),
            precision_macro=precision_score(y_test, y_pred, average="macro", zero_division=0),
            recall_macro=recall_score(y_test, y_pred, average="macro", zero_division=0),
            f1_macro=f1_score(y_test, y_pred, average="macro", zero_division=0),
            classification_report=classification_report(
                y_test, y_pred, labels=labels_sorted, zero_division=0
            ),
            confusion_matrix=confusion_matrix(y_test, y_pred, labels=labels_sorted).tolist(),
            classes=labels_sorted,
```

```
            n_train=len(X_train),
            n_test=len(X_test),
        )
        return metrics

    def save(self, path: Optional[Path] = None) -> Path:
        """Зберігає натреновану модель у файл (joblib)."""
        if not _HAS_SKLEARN:
            raise RuntimeError("scikit-learn не встановлено.")
        if self._pipeline is None:
            raise RuntimeError("Модель ще не натренована. Викличте .train() спочатку.")
        target = Path(path) if path else self._model_path
        target.parent.mkdir(parents=True, exist_ok=True)
        joblib.dump(self._pipeline, target)
        logger.info("Emotion ML model saved → %s", target)
        return target

    def load(self, path: Optional[Path] = None) -> bool:
        if not _HAS_SKLEARN:
            return False
        target = Path(path) if path else self._model_path
        if not target.exists():
            logger.info("ML модель не знайдена за шляхом %s — працюємо без ML", target)
            return False
        try:
            self._pipeline = joblib.load(target)
            logger.info("Emotion ML model loaded ← %s", target)
            return True
        except Exception as exc:  # noqa: BLE001
            logger.exception("Не вдалося завантажити ML-модель: %s", exc)
            self._pipeline = None
            return False

    def predict(self, text: str) -> Dict[str, float]:
        if not text or not text.strip():
            return self._empty_distribution()

        if self._pipeline is None:
            return self._empty_distribution()

        try:
            probas = self._pipeline.predict_proba([text])[0]
        except Exception as exc:  # noqa: BLE001
            logger.exception("ML.predict() помилка: %s", exc)
            return self._empty_distribution()

        try:
            classes = list(self._pipeline.named_steps["clf"].classes_)
        except Exception:  # noqa: BLE001
            classes = list(self._classes)

        result = {cls: 0.0 for cls in EMOTION_CLASSES}
        for cls, p in zip(classes, probas):
            if cls in result:
                result[cls] = float(p)
        return result

    def predict_label(self, text: str) -> Tuple[str, float]:
        """Зручний шорткат: повертає (label, confidence)."""
        dist = self.predict(text)
```

```
        if not dist:
            return "neutral", 1.0
        label = max(dist, key=lambda k: dist[k])
        return label, dist[label]

    @staticmethod
    def _empty_distribution() -> Dict[str, float]:
        """Розподіл «модель неактивна» — увесь вес на neutral."""
        return {cls: (1.0 if cls == "neutral" else 0.0) for cls in EMOTION_CLASSES}
_ml_classifier: Optional[EmotionMLClassifier] = None

def get_ml_classifier() -> EmotionMLClassifier:
    global _ml_classifier
    if _ml_classifier is None:
        clf = EmotionMLClassifier()
        clf.load()
        _ml_classifier = clf
    return _ml_classifier

def reload_ml_classifier() -> bool:
    clf = get_ml_classifier()
    return clf.load()

def is_ml_available() -> bool:
    return _HAS_SKLEARN
```

Модуль чат-асистента / Сервіс діалогу з LLM:

```
import asyncio
import json
import os
import re
import threading
import traceback
from datetime import datetime
from typing import Any, AsyncIterator, Dict, FrozenSet, List, Optional, Union

from bson import ObjectId
from dotenv import load_dotenv
from together import Together

from assistant_core.link_indexing import build_rag_context
from assistant_core.emotion_engine import (
    analyze_emotion,
    get_avatar_controller,
)


load_dotenv()

TOGETHER_API_KEY = os.environ.get("TOGETHER_API_KEY")
if not TOGETHER_API_KEY:
    raise RuntimeError(
        "TOGETHER_API_KEY is not set. Configure it via environment variables or .env file."
    )

client = Together(api_key=TOGETHER_API_KEY)
ALLOWED_EMOTIONS: FrozenSet[str] = frozenset(
    {"neutral", "happy", "sad", "surprise", "thinking", "angry", "disgust"}
```

```
)
_SESSION_NAME_DEFAULTS: FrozenSet[str] = frozenset({"Новий чат", "Без назви", ""})
_SESSION_NAME_MAX_LEN = 100

# Спочатку «основний текст» — щоб при streaming користувач бачив відповідь раніше за службові поля.
SYSTEM_PROMPT = (
    "Ти — чат-асистент для студентів КПІ: спокійний, чемний, по суті. "
    "Тон дружній, але без зайвої панібратської розмовності. "
    "Емодзі та смайлики в полях «основний текст» і «текст чату» не використовуй; якщо дуже доречно — не більше одного на все повідомлення. "
    "Не додавай рядки на кшталт «вау!», «ого!» лише заради ефекту. "
    "Якщо користувач ставить офіційне запитання щодо навчання чи КПІ, дай чітку, інформативну відповідь. "
    "Для таких відповідей використовуй лише ту інформацію, яка є в контексті (FAQ, посилання з бази, уривки зі сторінок). "
    "Посилання ти можеш використовувати тільки з контексту. "
    "Не вигадуй нові посилання."
    "Якщо користувач просто хоче поспілкуватися, підтримай розмову стримано й доброзичливо; якщо це не факти з контексту — вкажи, що це твоя особиста думка. "
    "ВАЖЛИВО: ти ЗАВЖДИ надаєш відповідь тільки в одному строго визначеному форматі. "
    "Не додаєш жодного слова, жодного речення, жодного пояснення за межами шаблону. Не ігноруєш структуру. "
    "Порядок полів ЗАВЖДИ такий (спочатку основний текст — для зручності читання):\n"
    "основний текст: {розгорнута відповідь у Markdown (GFM, як на GitHub): **жирний**, списки, абзаци. "
    "Для порівнянь, критеріїв, правил, стипендій тощо **за замовчуванням використовуй GFM-таблиці** (рядки з |), а не лише списки. "
    "Перед кожною таблицею — 1–2 речення, що вона показує; за потреби після таблиці — короткий висновок. Не залишай «голу» таблицю без пояснення. "
    "Математику й формули оформлюй у **LaTeX**: у рядку через \\( ... \\), блочно через $$ ... $$ або **зворотний сліш** + дужки \\[ ... \\]. "
    "Блочне $$ ... $$ пиши в один рядок без порожніх рядків усередині; змінні після формули краще як \\(I\\), \\(q\\), а не ( I ). "
    "Ніколи не обгортай формулу лише «голими» квадратними дужками [ ] на окремих рядках без \\ — так воно не відрендериться. "
    "Уникай «сирих» символів замість формули, якщо доречніше LaTeX. Якщо поруч таблиця й формула — одне коротке речення, що їх зв’язує. "
    "Таблиці не огороджуй у ``` — інакше вони стануть кодом, а не таблицею. Блоки ``` — лише для справжнього коду. "
    "Без сирого URL у тексті (URL лише в полі «посилання»). Без емодзі, якщо не виняток вище.}\n"
    "текст чату: {короткий нейтральний заголовок теми, як назва розділу. "
    "НЕ питання. Наприклад: «Оцінювання в КПІ», «Стипендії та бал», «Важливо для першокурсників».}\n"
    "емоція: {СТРОГО одне англійське слово: neutral, happy, sad, surprise, thinking, angry, disgust. "
    "neutral — спокійно; happy — радісно; sad — шкода; surprise — здивування; thinking — роздуми; "
    "angry — роздратування; disgust — огида. Лише слово.}\n"
    "посилання: {повний URL з контексту, якщо доречно; інакше рівно: немає}\n"
    "Не додавай нічого за межами цього шаблону. НЕ змінюй назви полів і порядок полів."
)


def _chat_messages(
    user_message: str,
    context: str,
    attachment_block: Optional[str] = None,
) -> List[Dict[str, str]]:
    msgs: List[Dict[str, str]] = [
        {"role": "system", "content": SYSTEM_PROMPT},
        {"role": "user", "content": f"Контекст:\n{context}"},
    ]
    if attachment_block:
        msgs.append(
            {
                "role": "user",
```

```
                "content": (
                    "Нижче — вміст або опис файлу, який надіслав користувач. "
                    "Відповідай на його запит, спираючись на цей вміст разом із контекстом FAQ/сайтів; "
                    "не вигадуй факти про файл, яких там немає.\n\n"
                    + attachment_block
                ),
            }
        )
    msgs.append({"role": "user", "content": user_message})
    return msgs


def _sse_event(obj: Dict[str, Any]) -> bytes:
    return f"data: {json.dumps(obj, ensure_ascii=False)}\n\n".encode("utf-8")

async def generate_model_answer(
    message: str,
    context: str,
    attachment_block: Optional[str] = None,
) -> str:
    try:
        response = await asyncio.to_thread(
            lambda: client.chat.completions.create(
                model="meta-llama/Llama-3.3-70B-Instruct-Turbo",
                messages=_chat_messages(message, context, attachment_block),
                max_tokens=1024,
            )
        )
    except Exception as exc:  # noqa: BLE001
        traceback.print_exc()
        raise ChatServiceError(f"Помилка при зверненні до LLM: {exc}") from exc

    return response.choices[0].message.content

def _normalize_link(raw: str) -> str:
    if not raw:
        return ""
    s = raw.strip()
    if s.lower() in ("немає", "none", "n/a", "-", "null"):
        return ""
    return s


def parse_answer(answer: str) -> Dict[str, str]:
    """Парсить відповідь моделі у структурований вигляд."""

    main_text_match = re.search(
        r"(?i)основний\s*текст\s*:\s*(.+?)(?=\n\s*(текст\s*чату|емоція|посилання)\s*:|\Z)",
        answer,
        re.DOTALL,
    )
    link_match = re.search(r"(?i)посилання\s*:\s*([^\n]+)", answer)
    emotion_match = re.search(r"(?i)емоція\s*:\s*([^\n]+)", answer)
    title_match = re.search(
        r"(?i)текст\s*чату\s*:\s*(.+?)(?=\n\s*(основний\s*текст|емоція|посилання)\s*:|\Z)",
        answer,
        re.DOTALL,
    )

    main_text = (
```

```
        main_text_match.group(1).strip()
        if main_text_match
        else "Вибач, не вдалося отримати основний текст відповіді."
    )
    link_raw = link_match.group(1).strip() if link_match else ""
    link = _normalize_link(link_raw)
    emotion_raw = emotion_match.group(1).strip() if emotion_match else "neutral"
    emotion = _normalize_emotion(emotion_raw)
    chat_title = title_match.group(1).strip() if title_match else "Без назви"

    return {
        "answer_raw": answer,
        "response": main_text,
        "link": link,
        "emotion": emotion,
        "title": chat_title,
    }


async def append_user_message(db, session_id: str, user_text: str) -> None:
    """Додає лише повідомлення користувача (для streaming перед генерацією)."""

    user_msg: Dict[str, Any] = {
        "role": "user",
        "text": user_text,
        "timestamp": datetime.utcnow(),
    }
    update_result = await db["sessions"].update_one(
        {"_id": ObjectId(session_id)},
        {"$push": {"messages": user_msg}, "$set": {"updatedAt": datetime.utcnow()}},
    )
    if update_result.modified_count == 0:
        raise ChatSessionNotFound("Сесія не знайдена")


async def append_assistant_message(
    db,
    session_id: str,
    assistant_text: str,
    assistant_emotion: str,
    assistant_link: str,
    assistant_title: str,
) -> None:
    """Додає лише повідомлення асистента."""

    assistant_msg: Dict[str, Any] = {
        "role": "assistant",
        "text": assistant_text,
        "emotion": assistant_emotion,
        "link": assistant_link or "",
        "title": assistant_title,
        "timestamp": datetime.utcnow(),
    }
    update_result = await db["sessions"].update_one(
        {"_id": ObjectId(session_id)},
        {"$push": {"messages": assistant_msg}, "$set": {"updatedAt": datetime.utcnow()}},
    )
    if update_result.modified_count == 0:
        raise ChatSessionNotFound("Сесія не знайдена")
```

```
    topic = (assistant_title or "").strip()
    if topic and topic not in _SESSION_NAME_DEFAULTS:
        short = topic[:_SESSION_NAME_MAX_LEN].rstrip()
        if short:
            await db["sessions"].update_one(
                {
                    "_id": ObjectId(session_id),
                    "$or": [
                        {"name": {"$in": list(_SESSION_NAME_DEFAULTS)}},
                        {"name": {"$exists": False}},
                    ],
                },
                {"$set": {"name": short}},
            )


async def append_messages_to_session(
    db,
    session_id: str,
    user_text: str,
    assistant_text: str,
    assistant_emotion: str,
    assistant_link: str,
    assistant_title: str,
) -> None:
    """Оновлює сесію в MongoDB: користувач + асистент (не streaming)."""

    await append_user_message(db, session_id, user_text)
    await append_assistant_message(
        db, session_id, assistant_text, assistant_emotion, assistant_link, assistant_title
    )


def _stream_worker(
    loop: asyncio.AbstractEventLoop,
    queue: asyncio.Queue,
    user_message: str,
    context: str,
    attachment_block: Optional[str] = None,
) -> None:
    """Блокуючий збір токенів Together (stream=True) у фоновому потоці."""

    def put(kind: str, payload: Union[str, None] = None) -> None:
        fut = asyncio.run_coroutine_threadsafe(queue.put((kind, payload)), loop)
        fut.result(timeout=120)

    try:
        stream = client.chat.completions.create(
            model="meta-llama/Llama-3.3-70B-Instruct-Turbo",
            messages=_chat_messages(user_message, context, attachment_block),
            stream=True,
            max_tokens=1024,
        )
        parts: list[str] = []
        for chunk in stream:
            if not chunk.choices:
                continue
            delta = chunk.choices[0].delta
            if delta is None:
                continue
```

```
            piece = getattr(delta, "content", None) or ""
            if piece:
                parts.append(piece)
                put("delta", piece)
        put("done", "".join(parts))
    except Exception as exc:  # noqa: BLE001
        traceback.print_exc()
        put("error", str(exc))

async def stream_chat_events(
    db,
    session_id: str,
    user_message: str,
    attachment_block: Optional[str] = None,
    attachment_filename: Optional[str] = None,
) -> AsyncIterator[bytes]:
    """
    SSE-потік: status (думає) → delta (фрагменти сирої відповіді) → done (розпарсені поля) або error.
    Користувача записує в БД одразу; асистента — після повної відповіді.
    """

    cleaned = (user_message or "").strip() or "Проаналізуй вкладений файл."
    display_user = cleaned
    if attachment_filename:
        display_user = f"{cleaned}\n\n📎 {attachment_filename}"
    user_emotion_result = analyze_emotion(cleaned)
    user_emotion_data = {
        "emotion":    user_emotion_result.emotion,
        "confidence": round(user_emotion_result.confidence, 4),
        "scores":     {k: round(v, 4) for k, v in user_emotion_result.scores.items()},
        "method":     user_emotion_result.method,
        "tokens":     user_emotion_result.tokens_matched[:10],  # топ-10 для дебагу
        "avatar_filename": get_avatar_controller().select_animation(
            user_emotion_result, demo_responsive=True
        ).filename,
    }

    await append_user_message(db, session_id, display_user)
    yield _sse_event({"type": "status", "phase": "thinking"})
    # Одразу повідомляємо фронтенд про емоцію користувача (до відповіді LLM)
    yield _sse_event({"type": "user_emotion", **user_emotion_data})

    context = await build_rag_context(db, cleaned)
    loop = asyncio.get_running_loop()
    queue: asyncio.Queue = asyncio.Queue()
    thread = threading.Thread(
        target=_stream_worker,
        args=(loop, queue, cleaned, context, attachment_block),
        daemon=True,
    )
    thread.start()

    while True:
        kind, payload = await queue.get()
        if kind == "delta" and payload:
            yield _sse_event({"type": "delta", "content": payload})
        elif kind == "done":
            raw = payload or ""
            parsed = parse_answer(raw)
            await append_assistant_message(
```

```
                db,
                session_id,
                parsed["response"],
                parsed["emotion"],
                parsed["link"],
                parsed["title"],
            )
            assistant_emotion_result = analyze_emotion(parsed["response"])
            assistant_emotion_data = {
                "emotion":    assistant_emotion_result.emotion,
                "confidence": round(assistant_emotion_result.confidence, 4),
                "scores":     {k: round(v, 4) for k, v in assistant_emotion_result.scores.items()},
                "method":     assistant_emotion_result.method,
                "avatar_filename": get_avatar_controller().select_animation(
                    assistant_emotion_result, demo_responsive=True
                ).filename,
            }
            yield _sse_event(
                {
                    "type": "done",
                    "response": parsed["response"],
                    "link": parsed["link"],
                    "emotion": parsed["emotion"],
                    "title": parsed["title"],
                    "user_emotion": user_emotion_data,
                    "assistant_emotion": assistant_emotion_data,
                }
            )
            return
        elif kind == "error":
            yield _sse_event(
                {"type": "error", "detail": payload or "Помилка генерації"}
            )
            return


async def process_chat(
    db,
    message: str,
    session_id: str,
    attachment_block: Optional[str] = None,
    attachment_filename: Optional[str] = None,
) -> Dict[str, str]:

    cleaned_message = (message or "").strip() or "Проаналізуй вкладений файл."
    display_user = cleaned_message
    if attachment_filename:
        display_user = f"{cleaned_message}\n\n     {attachment_filename}"

    # NLP-аналіз емоції користувача (не-streaming шлях)
    user_emotion_result = analyze_emotion(cleaned_message)

    context = await build_rag_context(db, cleaned_message)
    answer = await generate_model_answer(
        cleaned_message, context, attachment_block
    )
    parsed = parse_answer(answer)

    await append_messages_to_session(
        db=db,
```

```
    session_id=session_id,
    user_text=display_user,
    assistant_text=parsed["response"],
    assistant_emotion=parsed["emotion"],
    assistant_link=parsed["link"],
    assistant_title=parsed["title"],
)

parsed["user_emotion"] = {
    "emotion":    user_emotion_result.emotion,
    "confidence": round(user_emotion_result.confidence, 4),
    "scores":     {k: round(v, 4) for k, v in user_emotion_result.scores.items()},
    "method":     user_emotion_result.method,
    "avatar_filename": get_avatar_controller().select_animation(
        user_emotion_result, demo_responsive=True
    ).filename,
}

# Аналізуємо емоцію ВІДПОВІДІ асистента через EmotionEngine
assistant_emotion_result = analyze_emotion(parsed["response"])
parsed["assistant_emotion"] = {
    "emotion":    assistant_emotion_result.emotion,
    "confidence": round(assistant_emotion_result.confidence, 4),
    "scores":     {k: round(v, 4) for k, v in assistant_emotion_result.scores.items()},
    "method":     assistant_emotion_result.method,
    "avatar_filename": get_avatar_controller().select_animation(
        assistant_emotion_result, demo_responsive=True
    ).filename,
}

return parsed
```

ДОДАТОК Б Апробація

Арсенюк Д.В., Лазарчук А.С., Поліщук М.М., Сарибога Г.В. Програмне забезпечення інтелектуальної селекції мімічних реакцій 3D-аватара на основі гібридного аналізу природної мови. Матеріали VII Міжнародної науково-практичної конференції молодих вчених, аспірантів і студентів “Сучасні інформаційні технології та системи в управлінні”. Київ, 28–29 квітня 2026 р. С. 461-462.

URL:https://drive.google.com/file/d/19dd47KHOOOWwFi6b6zNMEw8rFlFWF6LG/edit

Арсенюк Д.В., студентка
Національний технічний університет України
"Київський політехнічний інститут імені Ігоря Сікорського"
arseniukdaryna@gmail.com

Лазарчук А.С., студент
Національний технічний університет України
"Київський політехнічний інститут імені Ігоря Сікорського"
la.artem.ser@gmail.com

Поліщук М.М., студентка
Національний технічний університет України
"Київський політехнічний інститут імені Ігоря Сікорського"
polishchuk.mariyatv21@gmail.com

Сарибога Г.В., старший викладач
Національний технічний університет України
"Київський політехнічний інститут імені Ігоря Сікорського"
sarigana-eds@lll.kpi.ua

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ З ІНТЕГРОВАНИМ 3D-АВАТАРОМ ДЛЯ ЕМОЦІЙНОГО СУПРОВОДУ ВІДВІДУВАЧІВ ВЕБРЕСУРСУ

Анотація. У роботі досліджено підходи до підвищення ефективності взаємодії користувачів із освітніми вебресурсами шляхом інтеграції 3D-аватарів. Представлено програмне забезпечення, реалізоване на базі Vue.js 3, що забезпечує поєднання інформаційної підтримки, навігації та емоційного супроводу взаємодії. Запропоновано оптимізований підхід до реалізації аватара на основі відео з альфа-каналом.

Вступ. Розвиток цифрових освітніх середовищ зумовлює необхідність створення інтерфейсів нового покоління, орієнтованих на підвищення рівня взаємодії користувача із системою. Емоційні характеристики інтерфейсу суттєво впливають на ефективність сприйняття інформації [3], що обґрунтовує доцільність використання 3D-аватарів у вебзастосунках освітнього призначення.

Мета роботи полягає у розробці програмного забезпечення з інтегрованим 3D-аватаром для підвищення ефективності інформаційної та емоційної взаємодії користувачів із вебресурсом кафедри.

Основна частина. Програмний продукт реалізовано на основі фреймворку Vue.js 3, який забезпечує реактивність інтерфейсу та ефективне керування станом застосунку [2]. Архітектура системи побудована за компонентно-орієнтованим підходом, що забезпечує її модульність та масштабованість. Навігацію реалізовано за допомогою Vue Router, а клієнт-серверну взаємодію – відповідно до принципів REST-архітектури [4] з використанням формату JSON і Fetch API [5]. Взаємодію користувача з системою формалізовано у вигляді блок-схеми:

ДОДАТОК В Презентація

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут атомної та теплової енергетики

Кафедра інженерії програмного забезпечення в енергетиці

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ДИНАМІЧНИХ МІМІЧНИХ РЕАКЦІЙ 3D-АВАТАРА НА ОСНОВІ МАШИННОГО НАВЧАННЯ

виконав: студент групи ТВ-21 Лазарчук Артем Сергійович
керівник: професор, д.т.н., проф. Коваль Олександр Васильович

2026

Актуальність теми

Чат-боти застаріли. Сьогодні звичайні чат-боти не здатні забезпечити природне та захопливе спілкування.

3D-аватари — майбутнє. Розумні 3D-аватари є дорогими й закритими комерційними розробками, недоступними для більшості проєктів.

Інженерний виклик. Головна проблема аналогів – механічна міміка або високі вимоги до серверів, що обмежує застосування та масштабування.

3D-модель аватара

Постановка задачі

Метою роботи є розробка програмного забезпечення для динамічних мімічних реакцій 3D-аватара на основі машинного навчання.

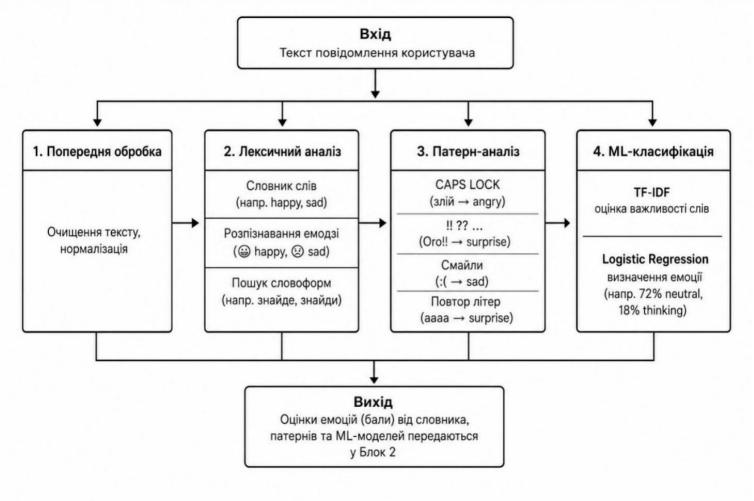
Для досягнення поставленої мети необхідно вирішити такі завдання:

- Реалізувати наскрізний обчислювальний конвеєр для максимально швидкої обробки запитів користувача;
- Розробити та інтегрувати блок гібридного аналізу природної мови;
- Створити модулі математичної агрегації, корекції та контекстуалізації емоційного вектора;
- Реалізувати блок логічного керування та селекції графічних медіапотоків;
- Провести комплексне тестування функціональності програмного забезпечення та оцінку точності навченої моделі;
- Розгорнути готову систему на віддаленому сервері для демонстрації її стабільної роботи в реальному часі.

Аналіз предметної області

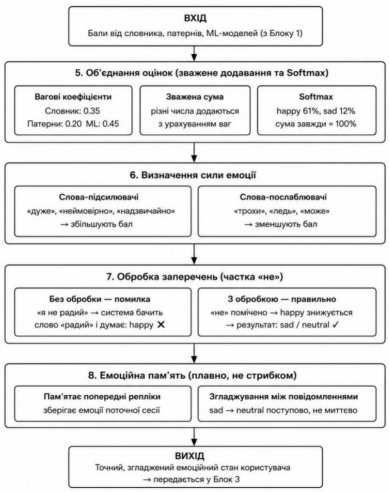
Підхід	Масштабованість обчислень	Швидкість відгуку	Оптимізація трафіку	Автономність від GPU	Фінансова вигода	Відкритість
Synthesia (комерційний SaaS)	✗ ліміти API	✗ хвилини на генерацію	✓ хмарна оптимізація	✓ GPU на стороні Synthesia	✗ дорога підписка	✗ закрита платформа
MetaHuman (важка 3D-графіка)	✗ 1 GPU = 1 сесія	✓ реальний час	✗ важкий стрімінг	✗ вимагає GPU	✗ оренда серверів	✗ закрита ліцензія
WebGL (браузерне 3D)	✓ без серверів	✓ локальний рендер	✗ важка 3D-модель	✗ навантажує GPU	✓ безкоштовний	✓ відкритий стандарт
Розроблений проєкт (Video Engine + WebM)	✓ асинхронна роздача файлів	✓ миттєве відтворення	✓ легкі відеофайли	✓ працює без відеокарти	✓ 0 грн витрат	✓ код належить кафедрі

Пайплайн функціонування системи



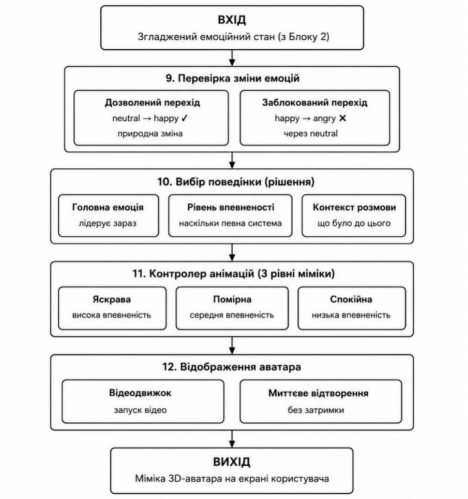
Гібридний аналіз природньої мови

Пайплайн функціонування системи(продовження)



Математична агрегація, корецію та контекстуалізація

Пайплайн функціонування системи(продовження)

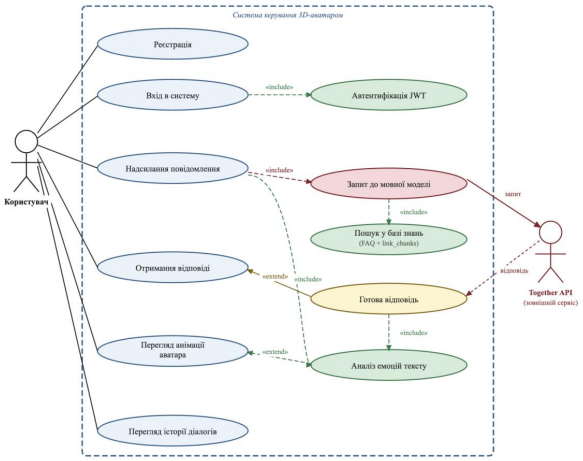


Логічне курування та селекція графічних медіапотоків

Проєктування архітектури системи

Діаграма послідовності взаємодії компонентів системи

Проєктування процесів та алгоритмів функціонування ПЗ



Діаграма прецедентів використання системи

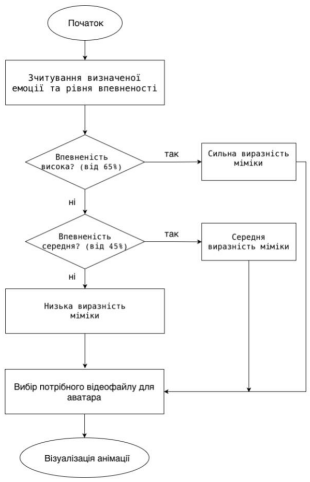
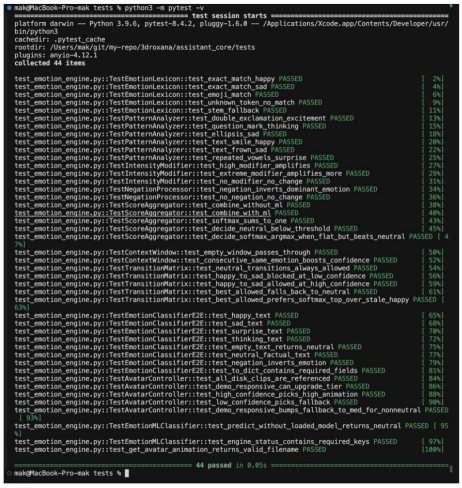


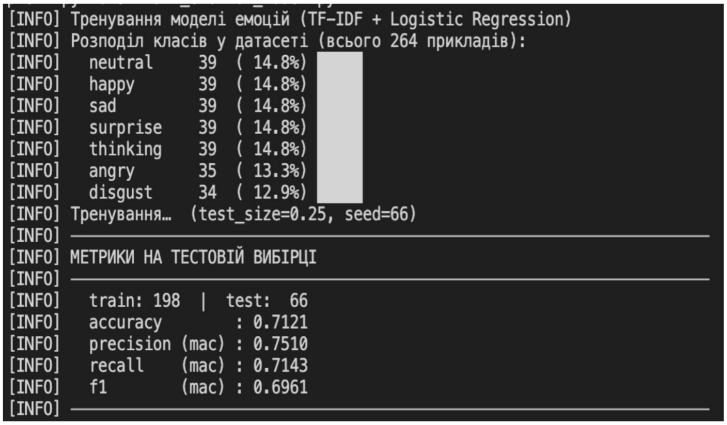
Схема алгоритму інтелектуальної селекції та визначення інтенсивності міміки

Засоби розробки серверної частини

Тестування системи

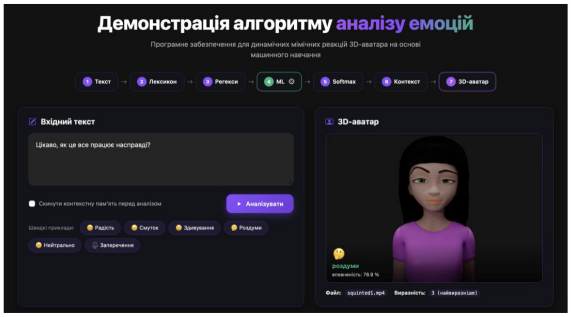


Результати модульного тестування компонентів конвеєра

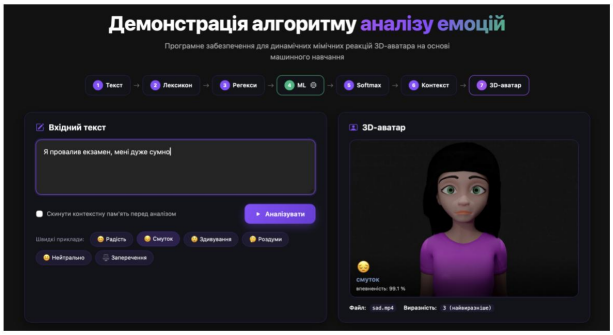


Результати навчання ML-моделі класифікації емоцій

Результат тестування



Візуалізація емоції «роздуми»



Візуалізація емоції «смуток»

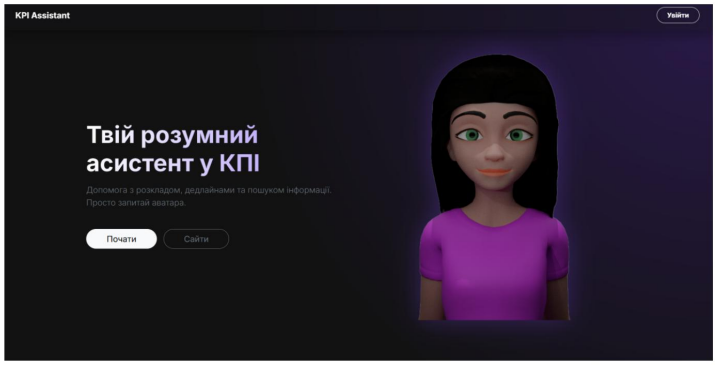
Впровадження та супровід

Програмний комплекс успішно розгорнуто на віддаленому сервері кафедри за веб-адресою:
http://77.47.192.6:6060

Для керування інфраструктурою та конфігурації сервера використовується SSH-клієнт Termius.

Забезпечено ізоляцію середовища, налаштування зворотного проксі-сервера (Reverse Proxy) та маршрутизацію трафіку через фіксований мережевий порт 6060.

ПЗ в процесі оформлення авторських прав.



Впровадження та супровід

Арсенюк Д.В., Лазарчук А.С., Поліщук М.М., Сарибога Г.В. Програмне забезпечення інтелектуальної селекції мімічних реакцій 3D-аватара на основі гібридного аналізу природної мови. *Матеріали VII Міжнародної науково-практичної конференції молодих вчених, аспірантів і студентів "Сучасні інформаційні технології та системи в управлінні"*. Київ, 28–29 квітня 2026 р. С. 461-462.


Міністерство освіти і науки України
Київський національний економічний університет імені Вадима Гетьмана
Інститут інновацій та конкурентної Макса Планка (Німеччина, Мюнхен)
Національний університет Ірландії в Мейнуті (Ірландія, Мейнуут)
Інститут кібернетики ім. В.М. Глушкова НАН України
Інститут проблем моделювання в енергетиці ім. Г.Є. Пухова НАН України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Одеський національний економічний університет
СУЧАСНІ ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ТА СИСТЕМИ В УПРАВЛІННІ
Збірник матеріалів
VII Міжнародної науково-практичної конференції молодих вчених, аспірантів і студентів
28–29 квітня 2026 р.

Висновки

У результаті виконання роботи повністю досягнуто поставлену мету та виконано такі завдання:

- Реалізовано наскрізний обчислювальний конвеєр для швидкої обробки запитів;
- Розроблено та інтегровано блок гібридного аналізу природної мови;
- Створено модулі математичної агрегації, корекції та контекстуалізації емоційного вектора;
- Реалізовано блок логічного керування та селекції графічних медіапотоків;
- Проведено комплексне тестування ПЗ та оцінено точність навченої моделі;
- Розгорнуто готову систему на віддаленому сервері для роботи в реальному часі.