Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)
Кафедра Штучного інтелекту
(повна назва)

# КВАЛІФІКАЦІЙНА РОБОТА
# Пояснювальна записка

рівень вищої освіти другий (магістерський)

Генерація адаптивних клінічних сценаріїв з використанням зважених графів знань
(тема)

Виконав:
здобувач другого року навчання,
групи СШІм-24-2
Ігор СІКОРСЬКИЙ
(власне ім'я, прізвище)
Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)
Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)
Освітня програма Системи штучного інтелекту
(повна назва освітньої програми)
Керівник проф. Лариса ЧАЛА
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ШІ _____ Лариса ЧАЛА
(підпис) (власне ім'я, прізвище)

2026 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Штучного інтелекту

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системи штучного інтелекту
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____» _____ 20 ___ р.

# ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Сікорському Ігорю Богдановичу
(прізвище, ім'я, по батькові)

1. Тема роботи Генерація адаптивних клінічних сценаріїв з використанням зважених графів знань

затверджена наказом університету від 6 квітня 20 26 р. № 269Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 22 травня 20 26 р.

3. Вихідні дані до роботи інтернет-джерела та існуючі публікації за тематикою медичних симуляцій та зважених графів знань; датасет DDXPlus як джерело медичних знань для побудови графа; передтренована біомедична мовна модель BioBERT; велика мовна модель Gemma 3 12B для генерації природномовних відповідей віртуального пацієнта; математичний апарат зважених орієнтованих графів, керованих Марківських процесів та ентропії Шеннона.

4. Перелік питань, що потрібно опрацювати в роботі

1) Аналіз предметної галузі та існуючих підходів до побудови клінічних симуляторів

2) Методи та моделі формування адаптивних клінічних сценаріїв

3) Практична реалізація

**КАЛЕНДАРНИЙ ПЛАН**

| № | Назва етапів роботи | Строк / термін виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Отримання завдання на кваліфікаційну роботу | 06.04.2026 | виконано |
| 2 | Дослідження предметної галузі | 10.04.2026 | виконано |
| 3 | Дослідження аналогів і методів реалізації | 12.04.2026 | виконано |
| | медичних симуляцій | | |
| 4 | Вибір мовної моделі та набору даних | 15.04.2026 | виконано |
| 5 | Розробка адаптивної системи генерації | 29.04.2026 | виконано |
| | клінічних сценаріїв | | |
| 6 | Розробка інтерфейсу системи | 02.05.2026 | виконано |
| 7 | Тестування розробленої моделі | 05.05.2026 | виконано |
| 8 | Оформлення пояснювальної записки | 13.05.2026 | виконано |
| 9 | Захист кваліфікаційної роботи | 22.05.2026 | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

Дата видачі завдання 6 квітня 2026 р.

Здобувач _____
(підпис)

Керівник роботи _____ проф. Лариса ЧАЛА
(підпис) (посада, власне ім’я, прізвище)

# РЕФЕРАТ

Пояснювальна записка: 108 с., 42 рисунок, 1 дод., 25 джерел.

АДАПТИВНІ КЛІНІЧНІ СЦЕНАРІЇ, ВЕЛИКІ МОВНІ МОДЕЛІ, ЗВАЖЕНІ ГРАФИ ЗНАНЬ, МАРКІВСЬКІ ПРОЦЕСИ, МОДЕЛЬ BIOBERT, СЕМАНТИЧНА ВЕКТОРИЗАЦІЯ, ШТУЧНИЙ ІНТЕЛЕКТ.

Об'єктом дослідження є генерація та управління динамікою інтерактивних клінічних сценаріїв.

Предметом – моделі зважених графів знань, семантичних векторів та ймовірнісних процесів для створення адаптивної поведінки віртуального пацієнта.

Мета роботи – розробка методу формування нелінійних сценаріїв із семантичною гнучкістю взаємодії студента з системою.

Проаналізовано недоліки систем Full Code Medical Simulation та Ada Health, зокрема залежність від заздалегідь підготовлених сценаріїв та відсутність механізмів оцінювання когнітивної діяльності. Розроблено модель пацієнта у вигляді динамічного зваженого графа знань із TF-IDF масштабуванням матриці суміжності. Запропоновано метод векторизації медичних сутностей через BioBERT для обробки вільного тексту користувача. Динаміку стану пацієнта формалізовано керованими Марківськими процесами. Для оцінювання компетенцій застосовано ентропію Шеннона та показник інформаційного прибутку.

Система може бути використана в медичній освіті для безпечного відпрацювання навичок діагностики та об'єктивного оцінювання компетенцій студентів. Перспективним напрямком подальших досліджень є розширення системи на нові клінічні дисципліни та верифікація результатів у реальному навчальному процесі.

## ABSTRACT

Master’s thesis contains: 108 pp., 42 fig., 1 ann., 25 references.

ADAPTIVE CLINICAL SCENARIOS, ARTIFICIAL INTELLIGENCE, BIOBERT, LARGE LANGUAGE MODELS, MARKOV PROCESSES, SEMANTIC VECTORIZATION, WEIGHTED KNOWLEDGE GRAPHS.

The object of the study is the generation and management of the dynamics of interactive clinical scenarios.

The subject is models of weighted knowledge graphs, semantic vectors, and probabilistic processes for creating adaptive behavior of a virtual patient.

The goal of the work is to develop a method for forming nonlinear scenarios with semantic flexibility in student interaction with the system.

The shortcomings of the Full Code Medical Simulation and Ada Health systems were analyzed, particularly their dependence on pre-written scenarios and the lack of mechanisms for evaluating cognitive activity.

A patient model was developed in the form of a dynamic weighted knowledge graph with TF-IDF scaling of the adjacency matrix. A method for vectorizing medical entities using BioBERT was proposed for processing free-form user input. The dynamics of the patient's state were formalized using controlled Markov processes. Shannon entropy and information gain were applied to assess competencies.

The system can be used in medical education for safe practice of diagnostic skills and objective assessment of student competencies. A promising direction for further research is the extension of the system to new clinical disciplines and the verification of results in a real educational process.

**ЗМІСТ**

Перелік умовних позначень, символів, одиниць, скорочень і термінів ..... 8
Вступ ..... 9
1 Аналіз предметної галузі та існуючих підходів до побудови клінічних симуляторів ..... 10
1.1 Роль інтерактивних сценаріїв у сучасній медичній освіті ..... 10
1.1.1 Віртуальні пацієнти та діалогові тренажери ..... 10
1.1.2 Симуляційні тренінги (екшн-симуляції) ..... 11
1.1.3 Рольові ігри ..... 12
1.2 Аналіз систем-аналогів та існуючих технологічних рішень ..... 13
1.2.1 Full Code Medical Simulation: симуляція медичних рішень ..... 13
1.2.2 Ada Health: графові структури медичних знань ..... 15
1.3 Порівняльний аналіз методів представлення медичних знань ..... 17
1.3.1 Логіко-алгоритмічні методи ..... 17
1.3.2 Графові та ймовірнісні моделі ..... 18
1.3.3 Семантичні векторні представлення ..... 19
1.4 Обґрунтування необхідності розробки та постановка задачі ..... 20
2 Методи та моделі формування адаптивних клінічних сценаріїв ..... 22
2.1 Формалізація моделі пацієнта на основі зважених графів знань ..... 22
2.2 Семантична векторизація медичних сутностей ..... 25
2.3 Моделювання динаміки сценарію (Марківський процес) ..... 26
2.4 Математичний апарат оцінювання респондента ..... 28
3 Практична реалізація ..... 31
3.1 Підготовка даних ..... 33
3.1.1 Обрання датасету ..... 33
3.1.2 Обробка даних ..... 34
3.2 BioBERT: налаштування ..... 40
3.2.1 Ініціалізація моделі та середовища виконання ..... 41
3.2.2 Створення семантичного простору медичних сутностей ..... 42

3.2.3 Інфраструктура векторного кодування ..... 44
3.3 Модель пацієнта ..... 45
3.4 Віртуальний пацієнт ..... 59
3.4.1 Базовий віртуальний пацієнт ..... 59
3.4.2 Інтеграція LLM за допомогою Ollama ..... 67
3.4.3 Розумний віртуальний пацієнт ..... 70
3.5 Система оцінювання ..... 81
3.6 Адаптивний селектор питань ..... 89
3.7 Інтерфейс ..... 96
Висновки ..... 103
Перелік джерел посилання ..... 105
Додаток А Відомість кваліфікаційної роботи ..... 108

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ШІ – штучний інтелект;

AI – Artificial Intelligence – штучний інтелект;

LLM – Large Language Models – велика мовна модель;

Open-source – відкрите програмне забезпечення з відкритим вихідним кодом;

Payload – корисне навантаження – фактичні дані для передачі у повідомленні або запиті;

REST API – Representational State Transfer Application Programming Interface – архітектурний стиль взаємодії між програмами, що використовує стандартні методи протоколу HTTP для обміну даними;

TF-IDF – Term Frequency-Inverse Document Frequency – частота термінів – обернена частота документа;

VRAM – Video Random Access Memory – внутрішня оперативна пам'ять графічного контролера.

# ВСТУП

У сучасному світі стрімкий розвиток технологій штучного інтелекту, зокрема великих мовних моделей (LLM), створює нові можливості не тільки в бізнесі, а й у трансформації освітніх процесів стратегічно важливих професій, зокрема медицині. Підготовка кваліфікованого лікаря – тривалий процес, що може зайняти цілі десятиліття і вимагає не лише глибоких теоретичних знань, а й підготовчої практики під керівництвом вже досвідченого лікаря в інтернатурі.

Проте, стала модель медичної освіти має низку викликів, які стоять перед нею роками: обмеженість доступу студентів до різноманітних клінічних випадків на ранніх етапах свого навчання; високі ризики, пов’язані з помилками при залученості реальних пацієнтів тощо. Впровадження інтелектуальних систем, що здатні симулювати клінічний прийом пацієнта, дозволяє частково вирішити цю проблему.

Актуальність розробки подібної системи зумовлена необхідністю створення інструментів для практики студентів на клінічних сценаріях для закріплення їхніх теоретичних знань без ризику для пацієнтів.

Використання в такій системі векторизації медичних даних та графових структур знань дозволить не обмежуватися жорстко заданими алгоритмами, а застосувати динамічні моделі, що базуються на реальній клінічній статистиці.

Подібна система, що генерує персоналізовані сценарії, дасть студентам можливість відпрацювати навички діагностики у безпечному середовищі. Це не тільки покращить та прискорить процес навчання і може скоротити термін інтернатури, а й дозволить більш об’єктивно оцінювати підходи майбутніх лікарів.

Таким чином, система має потенціал стати сполучною ланкою між теорією, отриманою в університеті, та реальною практикою і дозволить гнучко реагувати на рівень освіти студента в найсприятливіший момент.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ІСНУЮЧИХ ПІДХОДІВ ДО ПОБУДОВИ КЛІНІЧНИХ СИМУЛЯТОРІВ

Перший розділ присвячено аналізу предметної галузі та дослідженню сучасних підходів до побудови клінічних симуляторів. Розглянуто основні аспекти використання інтерактивних сценаріїв у медичній освіті, їхня роль у формуванні практичних навичок та вплив на засвоюваність теоретичних знань.

Особлива увага приділялася порівняльному аналізу підходів до моделювання клінічних сценаріїв, їхніх недоліків та переваг. На основі цього аналізу представлена актуальність та необхідність створення адаптивної системи генерації клінічних сценаріїв, що буде враховувати індивідуальні особливості практиканта та/або студента у забезпеченні більш ефективного процесу навчання.

## 1.1 Роль інтерактивних сценаріїв у сучасній медичній освіті

Інтерактивні сценарії в медичній освіті – це метод навчання, який базується на моделюванні реальних клінічних сценаріїв і ситуацій, що дозволяє студентам та інтернам відпрацьовувати навички в безпечному середовищі.

Існує кілька форматів таких сценаріїв, від новітніх на основі сучасних технологій до більш старих у вигляді рольових ігор із медичними сценаріями.

### 1.1.1 Віртуальні пацієнти та діалогові тренажери

Віртуальні пацієнти – новітній метод в практиці інтерактивних сценаріїв, який став можливим нещодавно, із стрімким розвитком сучасних технологій, зокрема комп’ютерних наук та галузі штучного інтелекту.

В подібних симуляціях роль пацієнта відіграється не людиною, а математичною моделлю або інтелектуальним агентом. Ціль практиканта – під час взаємодії із системою зібрати анамнез пацієнта, призначити діагностування та, можливо, певні методи обстеження, назначити лікування.

Переваги віртуальних пацієнтів у швидкому зворотному зв’язку з оцінюванням дій студента; навчанню бачити чіткіший зв’язок між симптомами та захворюваннями; безпеці, де не задіяні жодні реальні пацієнти і їм не загрожують наслідки помилок практиканта.

Недоліки також значні, оскільки такі системи дороговартісні у розробці та часто не є гнучкими у своїх симуляціях, обмежені заданим деревом рішень, в якому просто неможливо врахувати всі варіанти дій студентів.

Діалогові тренажери не сильно відрізняються від віртуальних пацієнтів, але призначені для розвитку не клінічних та діагностичних, а комунікаційних навичок. Відповідно до цієї подібності, переваги та недоліки залишаються спільними в обох випадках.

### 1.1.2 Симуляційні тренінги (екшн-симуляції)

Симуляційні тренінги (екшн-симуляції) – новітній метод у практиці практичних навичок у безпосередньому фізичному втручанні у організм пацієнта в стресових умовах, як операції [2]. Вони повинні бути максимально наближеними до реальності і часто проводяться в віртуальній або доповненій реальності або спеціальних симуляційних центрах для більш складних симуляцій, де відтворюють інтер'єр операційної, карети швидкої допомоги або палати інтенсивної терапії.

В останніх, симуляційних центрах, можуть використовуватися роботи-симулятори високої реалістичності, які емулюють дихання, мають

пульс та зіниці, що реагують на світло, а стан змінюється залежно від введених ліків.

Також можуть бути так звані стандартизовані пацієнти, що є навченими акторами, які імітують симптоми та емоційні реакції.

Переваги екшн-симуляцій у можливості відпрацювати мануальні навички, що ледь не єдиний спосіб «набити руку» в маніпуляціях (встановлення катетеру, ін’єкції, дефібриляція) без ризику для пацієнта; розвинути певний рівень стресостійкості в умовах шуму апаратури, обмеженого часу та фізичної втоми; набути навичок командної взаємодії під час симульованих операцій та реанімацій. В симуляційних центрах проводять дебрифінги після тренувань, де розбираються відеозапис дій команд практикантів досвідченими лікарями.

Недоліками є висока вартість подібних методів. Так, розробка високоточних симуляторів для віртуальної реальності – тривалий та кошторисний процес, для використання результатів якого треба дороге VR-обладнання. Манекени для центрів симуляцій також коштують десятки тисяч доларів США за одиницю, а їхнє обслуговування може перевершити початкову ціну покупки за кілька років.

Також, для центрів, потрібна спеціальна інфраструктура, як обладнанні приміщення та підготовлений персонал із інструкторів, акторів тощо. А також обов’язкова присутність самих практикантів, оскільки в таких випадках практика тільки офф-лайн.

### 1.1.3 Рольові ігри

Рольові ігри у медичній освіті – це метод навчання, де учасники розігрують конкретні клінічні або етичні ситуації, беручи на себе ролі лікаря, пацієнта, родича або колеги. На відміну від екшн-симуляцій з манекенами, тут головним інструментом є живе спілкування та акторська гра.

Найчастіше рольові ігри використовують для відпрацювання «складних розмов»: повідомлення про смерть, збір анамнезу у конфліктного пацієнта або обговорення помилки з керівництвом.

Перевагами подібного підходу є гнучкість та імпровізація, де живі сторони можуть реагувати непередбачувано, змушуючи студента адаптувати свій тон, слова та поведінку в реальному часі; низька вартість, оскільки це вимагає лише пару годин вільного часу в учасників; безпечне середовище, бо немає залученості реальних пацієнтів.

Недоліки можна побачити у перевагах. Так, оцінка студента є суб'єктивною і залежить лише від того, хто грав роль пацієнта. Присутній ризик зриву подібної гри через надмірну поведінку присутніх, які не сприймають гру серйозно. Головним недоліком може бути недостовірність гри через недостатність акторських навиків у гравців.

## 1.2 Аналіз систем-аналогів та існуючих технологічних рішень

Концепція створення інтелектуальних систем для симуляції медичної практики розвивається протягом останнього десятиліття, втілюючись у таких відомих рішеннях, як Full Code Medical Simulation та Ada Health.

Перша модель, Full Code Medical Simulation, є системою віртуального пацієнта і виступає прикладом, наближеним до того результату, який направлена досягти ця дослідницька робота. Ada Health є системою підтримки пацієнта, а не його симуляції для практики, але вона реалізована за подібними до тих технологіями, шо розглядаються в даному дослідженні.

### 1.2.1 Full Code Medical Simulation: симуляція медичних рішень

Full Code Medical Simulation спеціалізується на розробці медичних симуляційних рішень, які допомагають удосконалити клінічні навички через інтерактивну роботу з віртуальними пацієнтами.

Вони використовують штучний інтелект для імітації поведінки пацієнтів та AI-тутора для порад і рекомендацій практикуючим. Їхня платформа охоплює понад 250 клінічних випадків у 31 медичній спеціальності, що дозволяє користувачам практикуватися у різноманітних медичних сценаріях, включно з операційними процедурами. Особливу увагу в системі приділено використанню штучного інтелекту, який забезпечує реалістичні діалоги з пацієнтами, роблячи взаємодію максимально наближеною до реальних клінічних ситуацій. Система симуляцій налічує 23 різні моделі пацієнтів, що дозволяє відтворювати широкий спектр клінічних випадків [8].

Сама система є мультиплатформеною, на рисунку 1.1 представлено її мобільний інтерфейс, який, як видно з прикладу, має чітку структуру [14].

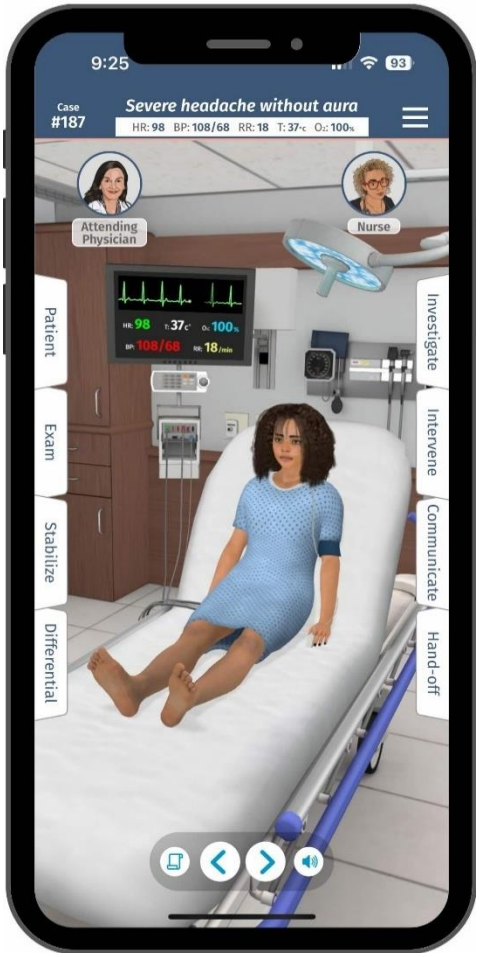



Рисунок 1.1 – Інтерфейс Full Code Medical Simulation

Це дозволяє користувачу легко орієнтуватися у процесі симуляції. На екрані відображається віртуальний пацієнт у лікарняному ліжку з усіма необхідними медичними показниками, а також присутні віртуальні медичні працівники – лікар і медсестра.

### 1.2.2 Ada Health: графові структури медичних знань

Системи Ada Health представляють категорію інтелектуальних агентів, що базуються на імовірнісних методах обробки медичних знань [13]. На відміну від освітніх симуляторів, ці платформи розроблялися як системи підтримки прийняття клінічних рішень (CDSS) для широкого загалу, що і стало причиною вибору їхньої технології.

Основою системи є медичні графи знань (Medical Knowledge Graphs). Це структуровані мережі, вузлами яких виступають хвороби, симптоми, фактори ризику та анатомічні структури, а ребра відображають клінічні зв’язки між ними [13].

В межах цього підходу, Ada Health використовує власну базу знань, в якій є тисячі медичних станів, а кожна пара «симптом-діагноз» має вагу ймовірності, що була визначена на основі оцінок експертів та аналізу великих масиві даних [13].

Ada Health тривалий час використовувала імовірнісні графічні моделі (Probabilistic Graphical Models), які дозволяють системі розраховувати найбільш вірогідний діагноз за допомогою теореми Баєса, враховуючи наявні та відсутні симптоми.

Використання графової логіки дає цим система значну перевагу над класичними і більш звичними деревами рішень. По-перше, такий підхід дає змогу уникнути лінійності. Користувач може не зобов’язаний слідувати чіткому скрипту із обмеженим вибором варіантів дій і може, оскільки це система підтримки пацієнтів, вводити симптоми у довільному порядку, а система перерахує ймовірності всього графа після кожного нового факту.

По-друге, граф дозволяє системі робити уточнення у пацієнта, які найбільш ефективно звужують коло потенційних діагнозів.

Для ілюстрації, на рисунку 1.2 показано інтерфейс Ada Health, де користувач бачить висновок системи за наданими ним симптомами, що демонструє зручність і простоту взаємодії з системою [13].

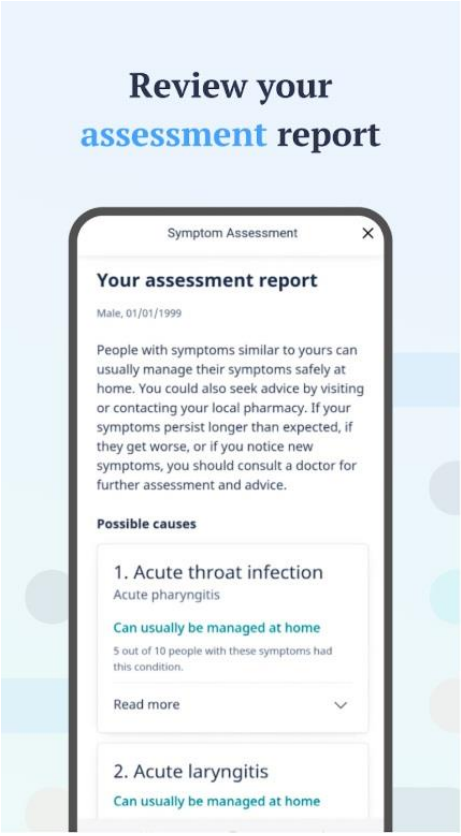



Рисунок 1.2 – Приклад інтерфейсу Ada Health

Такий підхід дозволяє збирати необхідні дані для подальшого аналізу та формування діагнозу.

Звісно, такі системи також мають ряд недоліків. Так, попри високу діагностичну точність, ці обмеження роблять мало застосовуваними для моделювання адаптивних навчальних сценаріїв.

Вузли графів в обох системах є статичними сутностями, які не мають внутрішньої логіки «прихованого стану», який важливий у випадку розробки системи з адаптивною генерацією станів віртуального пацієнта.

Також, попри використання методів обробки природної мови (NLP) для розпізнавання вводу, вони часто потребують вибору симптому з чітко заданого словника, що унеможливлює адаптивність сценаріїв у випадку симуляції клінічних сценаріїв. Використання, наприклад, модифікованої до медичних цілей моделі трансформера Bert, BioBert, дозволить системі самостійно інтерпретувати вільні текстові описи дій студента через векторні представлення.

Попри те, що Ada Health демонструє потужність графового підходу у медицині, його адаптація для освітнього середовища потребуватиме механізмів динамічного оновлення станів та нових методів оцінювання когнітивної діяльності.

## 1.3 Порівняльний аналіз методів представлення медичних знань

Аналіз існуючих освітніх симуляторів та діагностичних систем, проведений в попередніх розділах, демонструє, що ефективність віртуального клінічного сценарію безпосередньо залежить від обраного підходу до представлення знань, що головним фактором у визначенні меж гнучкості, адаптивності та наповненості всієї системи.

Існує кілька методів до представлення знань в медичній сфері, кожен із яких має власні як переваги, так і недоліки та обмеження. В цьому дослідженні розглянуто кілька з них, що були представлені в аналізі систем-аналогів.

### 1.3.1 Логіко-алгоритмічні методи

Логіко-алгоритмічні методи, що базуються на структурах дерев рішень, є одними з найперших і найпоширеніших методів моделювання клінічних сценаріїв [1]. В їх основі лежать представлення знань у вигляді жорсткої ієрархічної структури послідовностей правил [24]. Кожен вузол

подібних дерев відповідає певному стану пацієнта або результату обстеження, а ребра – діям студента, що є каталізаторами зміни цього стану.

Головною перевагою дерев рішень є їхня повна прозорість та детермінованість. Викладач або методист може чітко контролювати траєкторію навчання, закладаючи в систему конкретні клінічні протоколи. Такі моделі легко верифікуються на відповідність медичним стандартам, оскільки кожен можливий шлях прописаний експертом заздалегідь [1].

Проте, зважаючи на те, що кожен із сценарії треба прописувати в ручну, виникає доволі кричуща проблема – комбінаторний вибух [24]. Для моделювання реального клінічного випадку, де можливі сотні комбінацій симптомів та аналізів, дерево стає надзвичайно громіздким. Будь-яка спроба додати нову гілку варіантів вимагає перегляду всієї логіки сценарію.

З того ж джерела виникає і інша проблема – відсутні гнучкості семантики. Системи, що базуються на деревах рішень, не здатні розуміти синоніми або близькі за сенсом дії. Якщо студент вводить запит, який не збігається з ключовим словом у вузлі дерева, система трактує це як діагностичну помилку.

Лінійність такого досвіду також часто стає перешкодою, де студент просто йде по заданих алгоритмах, не надто замислюючись над процесом і просто обирає оптимальний варіант, заздалегідь визначений кимось, що ставить під сумнів академічну доброчесність. Це також нівелює можливість вільної діагностичної імпровізації, оскільки система не здатна динамічно перераховувати стан пацієнта поза межами заданої структури.

### 1.3.2 Графові та ймовірнісні моделі

Новим, більш розвиненим підходом до представлення знань стали графові та ймовірнісні моделі, які відходять від лінійної логіки на користь мережевої структури медичних знань. У таких моделях клінічна інформація представляється у вигляді медичного графа знань (Medical Knowledge

Graph), де вузли – це хвороби та симптоми, а ребра – імовірнісні зв’язки між ними [19].

На відміну від дерев рішень, імовірнісні моделі, зокрема баєсівські мережі, дозволяють системі розраховувати ймовірність того чи іншого діагнозу при кожній новій дії користувача [16]. Це забезпечує нелінійність діагностики, де студент, користувач, може вводити дані в довільному порядку, а система динамічно перераховує стан усього графа. Дозволяє обробляти суперечливі дані, бо модель здатна працювати з частковою інформацією або симптомами, що мають різний рівень специфічності для конкретної хвороби [16].

Попри значну гнучкість, класичні графові моделі, реалізовані в системах на кшталт Ada Health, мають суттєві недоліки для освітніх цілей. Так, ребра графа зазвичай відображають фіксовані статистичні ймовірності, що ускладнює моделювання динамічної зміни стану пацієнта в часі, а вузли графа часто сприймаються як бінарні змінні, що не дозволяє моделювати поступовий розвиток клінічної картини [3].

Також, як і в логіко-алгоритмічних методах, семантика дуже обмежена. Більшість ймовірнісних графів вимагають суворої відповідності вхідного тексту до словнику [17].

Використання імовірнісних графів є значним кроком вперед у порівнянні з деревами рішень [3], проте для створення справді адаптивного сценарію необхідно поєднати структурну цілісність графа із гнучкістю векторних представлень, що розглянуто у наступному підрозділі.

### 1.3.3 Семантичні векторні представлення

Сучасним етапом розвитку методів представлення медичних знань є використання семантичних векторних представлень (Embeddings). Цей підхід базується на відображенні медичних сутностей (симптомів, діагнозів,

дій) у багатовимірному метричному просторі $R^d$, де близькість векторів відповідає семантичній близькості понять [16].

Найбільш ефективним інструментом для створення таких представлень у медицині на даний момент є модель BioBERT – спеціалізована версія архітектури BERT, попередньо навчена на величезних масивах біомедичних текстів (PubMed, PMC) [4]. Використання механізму самоуваги (Self-Attention) дозволяє моделі враховувати контекст кожної дії студента, що критично важливо для точного моделювання клінічного сценарію [4].

Така архітектура дозволяє вирішити спільну проблему попередніх двох методів – семантичну. Система здатна визначити дію студента як відповідну, навіть якщо використана термінологія відрізняється від еталонної, а векторний підхід дозволяє студенту вводити запити у довільній текстовій формі, що максимально наближає симуляцію до реального діалогу з пацієнтом.

І саме цей підхід, у межах даного дослідження, виступить як сполучена ланка, що дозволить активувати вузли динамічного графа пацієнта. Кожна дія студента трансформується у вектор, який ініціює зміну станів у графовій моделі через розрахунок ваг активації.

Таким чином, використання семантичних векторних представлень на основі BioBERT дозволяє подолати обмеження детермінованих алгоритмів та статичних графів, створюючи теоретичний фундамент для розробки справді адаптивних клінічних сценаріїв.

## 1.4 Обґрунтування необхідності розробки та постановка задачі

Проведений аналіз існуючих підходів дозволяє констатувати наявність суттєвого технологічного розриву в галузі медичних симуляторів. З одного боку, існують високоефективні, але статичні освітні системи, обмежені детермінованою логікою дерев рішень. З іншого – потужні

діагностичні графи знань (Ada Health), які, попри свою гнучкість, не адаптовані для динамічного моделювання навчального процесу та не мають об’єктивного інструментарію для оцінювання когнітивної діяльності респондента.

Необхідність цієї розробки зумовлена кількома факторами, а саме проблемою з масштабуванням сценаріїв, ручне написання яких веде до комбінаторного вибуху та високої вартості розробки; відсутністю семантичної гнучкості у сучасних підходах.

На основі виявлених недоліків існуючих підходів, метою даної роботи є розробка методу моделювання адаптивних клінічних сценаріїв, який би базувався на векторних представленнях графових структур знань пацієнта.

Для досягнення цієї мети необхідно виконати наступні задачі:

– розробити математичну модель клінічного сценарію як динамічного зваженого графа, що ітераційно оновлює свій стан залежно від дій респондента;

– сформувати метод семантичного кодування медичних сутностей з використанням навченої моделі BioBERT для забезпечення гнучкості взаємодії в систем;

– розробити алгоритм управління динамікою сценарію на основі апарату Марківських процесів прийняття рішень для реалізації механізму «розкриття» латентних ознак захворювання;

– запропонувати систему оцінювання когнітивних стратегій;

– здійснити програмну реалізацію та експериментальну перевірку запропонованого методу.

# 2 МЕТОДИ ТА МОДЕЛІ ФОРМУВАННЯ АДАПТИВНИХ КЛІНІЧНИХ СЦЕНАРІЇВ

Даний розділ присвячено методам та математичній моделі, що є основою формування адаптивних клінічних сценаріїв у системах медичного навчання.

Підхід, який розглядається в даній роботі, базується на поєднанні графових моделей представлення знань і семантичних векторних просторів, що розглядалися вище, та ймовірнісних методів моделювання. Таке злиття різних підходів дозволяє врахувати як структурні зв’язки між медичними сутностями, так і їхню семантичну близькість. А це критично важливий елемент в обробці природної мови та побудові реалістичних клінічних сценаріїв.

Структура знань про пацієнта базується на концепції зваженого орієнтованого графа. У цій архітектурі окремі вузли виступають цифровими аналогами клінічних понять, тоді як ребра системи візуалізують та кількісно описують логічні взаємозв'язки між ними.

Щоб звести різнорідну (гетерогенну) вхідну інформацію до спільного знаменника, застосовано методику семантичної векторизації. Це дозволяє трансформувати неструктуровані медичні факти у гомогенний векторний простір, придатний для подальшого математичного аналізу та обчислень.

## 2.1 Формалізація моделі пацієнта на основі зважених графів знань

Фундаментальним етапом розробки адаптивної архітектури для генерації клінічних сценаріїв є побудова математичної моделі пацієнта. У даній роботі ця модель реалізована через структуру зваженого орієнтованого графа знань, що дозволяє експліцитно відобразити нелінійні причинно-наслідкові взаємозв'язки між різними медичними категоріями.

Оскільки первинний масив даних (анамнестичні відомості, результати лабораторних досліджень, демографічний профіль та скарги) відзначається високою гетерогенністю, для досягнення обчислювальної уніфікації впроваджено механізм семантичної векторизації (embeddings). У межах цього підходу кожен дискретний медичний факт або симптом конвертується у вектор $v_i$ у багатовимірному метричному просторі $R^d$:

$$v_i = Encoder(f_i, \theta), \tag{2.1}$$

де Encoder – виконує роль функції відображення вхідних даних у векторний простір і є моделлю нейронної мережі;

$\theta$ – множина параметрів Encoder, визначена шляхом претренінгу на масивних галузевих корпусах.

На основі отриманих векторних представлень формується графова топологія $G$, яка математично описується як:

$$G = (V, E, W), \tag{2.2}$$

де $V = \{v_1, v_2, \dots, v_n\}$ – множина векторизованих клінічних сутностей;

$E$ – множина орієнтованих ребр, що визначають наявність зв'язку між сутностями.

Ступінь взаємовпливу між вузлами регулюється матрицею суміжності $W = [w_{ij}]$, елементи якої визначають ступінь впливу вузла $i$ на вузол $j$. Розрахунок вагових коефіцієнтів здійснюється через оцінку семантичної близькості компонентів:

$$w_{ij} = \frac{v_i * v_j}{\|v_i\| * \|v_j\|} * \alpha_{ij}, \tag{2.3}$$

де $v_i * v_j$ – скалярний добуток векторів, що визначає ступінь їхньої семантичної подібності.

Нормалізація результату здійснюється за рахунок використання евклідових норм векторів , що дозволяє усунути вплив різної довжини векторів та зосередитися на їх напрямку у просторі ознак.

Таким чином, отримана величина є косинусною мірою подібності, що набуває значень у діапазоні [-1; 1] та відображає ступінь кореляції між відповідними клінічними сутностями.

Коефіцієнт клінічної релевантності – $\alpha_{ij}$. Додатково вводиться для врахування статистичної залежності між сутностями на основі даних навчальної вибірки:

$$\alpha_{ij} = \frac{Count(v_i, v_j)}{Count(v_i)}, \tag{2.4}$$

де чисельник відображає частоту коокуренції діагнозу та симптому в датасеті, а знаменник – загальну поширеність даної патології.

Синтез семантичної подібності та статистичної значущості дозволяє сформувати релевантну вагу зв'язку, що робить модель чутливою до реальних клінічних закономірностей.

З урахуванням цього коефіцієнта формується узагальнена вага зв’язку між вузлами графа, що дозволяє одночасно враховувати як семантичну близькість понять, так і їх клінічну значущість.

Запропонований граф знань не лише забезпечує структуровану інтерпретацію медичної інформації, а й слугує базисом для симуляції динаміки стану пацієнта, де активація вузлів змінюється в реальному часі під впливом діагностичних маневрів користувача.

## 2.2 Семантична векторизація медичних сутностей

Центральним механізмом запропонованого методу є семантична векторизація, яка виконує роль уніфікуючого інтерфейсу між сирими клінічними даними та математичним апаратом моделі. Цей етап забезпечує трансляцію гетерогенної медичної інформації у гомогенний числовий простір, що критично важливо для подальших обчислювальних операцій. У межах даного підходу всі вхідні вектори, що репрезентують симптоматику, встановлені діагнози, результати об'єктивних обстежень та операційні дії користувача, відображаються у вигляді векторів фіксованої розмірності $R^d$.

Процес перетворення вхідного масиву даних, що охоплює як статичний профіль пацієнта, так і динамічні маніпуляції студента, реалізується за допомогою спеціалізованої функції Encoder згідно з виразом (2.1). Як технологічний базис для імплементації цієї функції обрано архітектуру BioBERT – модифіковану нейромережеву модель BERT, що пройшла етап донавчання (fine-tuning) на специфічних біомедичних корпусах, таких як PubMed. Використання такої моделі дозволяє ефективно нівелювати семантичні розбіжності між різними формами представлення даних (текстові нотатки, лаконічні назви симптомів, числові показники аналізів), зводячи їх до єдиного цифрового формату, придатного для прямої інтеграції в алгоритми моделювання.

Сформовані векторні представлення стають фундаментальними компонентами графової структури знань. Зокрема, вони є вхідними параметрами для обчислення ваг інцидентності між вузлами системи відповідно до формули (2.3), що дає змогу кількісно оцінити концептуальну близькість між клінічними сутностями.

Окрему увагу приділено векторизації інтерактивних дій користувача $A_t$. Незалежно від модальності дії, вона трансформується у векторний простір за допомогою ідентичного алгоритму Encoder. Такий підхід гарантує метричну узгодженість між статичним описом клінічного

випадку та динамічними елементами взаємодії студента з системою. Отриманий вектор дії виступає керуючим параметром у моделі динаміки стану пацієнта (формула (2.5)), безпосередньо модулюючи рівні активації відповідних вузлів графа у часі.

Таким чином, семантична векторизація постає як інтеграційний механізм, що трансформує розрізнені медичні факти у цілісну математичну систему. Це дозволяє здійснити перехід від формалізованого опису «статичного» пацієнта до створення живої адаптивної моделі, спроможної до релевантних реакцій на вільні дії користувача в режимі реального часу.

## 2.3 Моделювання динаміки сценарію (Марківський процес)

Після встановлення статичної структури графа знань та впровадження механізмів векторизації, критичним завданням стає математичне описання еволюції клінічного випадку в часі. Для формалізації цього процесу в роботі застосовано апарат керованих Марківських процесів, що дозволяє представити динаміку сценарію як послідовність переходів, де майбутній стан системи детермінується виключно її актуальними параметрами та вхідним керуючим сигналом (дією користувача). Такий підхід релевантний логіці клінічного мислення, де кожне наступне рішення лікаря базується на поточному статусі пацієнта.

Поточний клінічний статус у довільний момент часу $t$ репрезентується вектором активації вузлів графа $S_t$, компоненти якого відображають інтенсивність прояву відповідних медичних сутностей. Механізм трансформації станів описується наступним ітераційним співвідношенням:

$$S_{t+1} = \sigma(W_t * S_t + \beta * Embed(A_t)), \quad (2.5)$$

де складники моделі визначаються таким чином:

– $\sigma$ – нелінійна функція активації, що моделює фізіологічну реакцію пацієнта на зміну стану системи;

– $W_t$ – матриця суміжності графа на момент часу $t$, що визначає силу та напрямок впливу між клінічними сутностями;

– $S_t$ – вектор стану системи, який відображає поточний розподіл активації вузлів графа;

– $A_t$ – дія користувача у момент часу $t$, що відповідає клінічному рішенню (збір анамнезу, призначення обстежень, лікування тощо);

– $Embed(A_t)$ – операція перетворення дії користувача у векторне представлення за допомогою функції Encoder, визначеної у (2.1);

– $\beta$ – коефіцієнт масштабування впливу дії користувача, значення якого визначається емпірично у процесі калібрування моделі на навчальній вибірці.

Отже, оновлення стану пацієнта постає як синтез внутрішньої інерції графової структури та зовнішнього збурення, внесеного користувачем. Це закладає фундамент для адаптивної поведінки симулятора, де кожна операція студента здатна радикально змінити подальшу траєкторію сценарію.

Використання Марківської парадигми забезпечує високу варіативність навчального процесу: система не слідує жорстко заданому алгоритму, а генерує унікальні шляхи розвитку подій залежно від стратегії, обраної студентом.

На відміну від детермінованих скриптових систем, така модель враховує контекстуальну залежність – ідентична дія може призвести до різних клінічних наслідків залежно від передісторії та поточного стану активації графа. Це суттєво підвищує автентичність симуляції та сприяє розвитку навичок прийняття рішень у складних умовах діагностичної невизначеності.

## 2.4 Математичний апарат оцінювання респондента

Важливою функціональною частиною адаптивної системи є алгоритм верифікації дій користувача, який дозволяє не лише відстежувати хід діагностичного процесу, а й об’єктивно диференціювати рівень компетенцій респондента. В основі механізму оцінювання лежить аналіз поточного вектора стану $S_t$, що відображає динаміку ймовірнісного розподілу між конкуруючими діагностичними гіпотезами.

Для кількісного вимірювання ступеня невизначеності клінічної ситуації застосовано ентропійну міру Шеннона [7]:

$$H(S_t) = -\sum_{i=1}^{m} p_i(t) * \log_2(p_i(t)), \tag{2.6}$$

де $p_i(t)$ позначає ймовірність істинності $i$-го діагнозу. Цей показник слугує індикатором «розмитості» гіпотез: висока ентропія свідчить про низьку визначеність, тоді як її поступове зниження в ході сценарію маркує ефективне звуження диференціального ряду та наближення студента до верифікації патології.

Розрахунок ймовірності $p_i(t)$ здійснюється через нормалізуючу функцію Softmax:

$$p_i(t) = \frac{\exp\left(s_i \,/\, \tau\right)}{\sum_{j=1}^{m} \exp\left(s_j \,/\, \tau\right)}, \tag{2.7}$$

де $s_i$ – поточне значення активації вузла і-го діагнозу у векторі стану $S_t$.

$\exp\left(s_i\right)$ – використовується для того, щоб перетворити будь-які значення активації (навіть від'ємні) у додатні числа та підсилити різницю між найбільш імовірними та малоймовірними діагнозами;

$\sum_{j=1}^{m} \exp(s_j)$ – сума експонент усіх потенційних діагнозів у сценарії. Ділення на цю суму гарантує, що $\sum p_i = 1$ (умова цілісності ймовірності), що необхідно для коректного розрахунку ентропії Шеннона;

τ – гіперпараметр, який регулює «впевненість» моделі.

Якість кожної окремої маніпуляції $A_t$ та вектором еталонної дії оцінюється через показник інформаційного прибутку (Information Gain) – різницю ентропій між послідовними кроками:

$$\Delta H = H(S_t) - H(S_{t+1}) > 0, \tag{2.8}$$

Додатково система аналізує семантичну коректність дій шляхом обчислення косинусної близькості між вектором рішення студента $\mathrm{Embed}(A_t)$ та вектором еталонного медичного протоколу $v_{ideal}$:

$$R = \eta * \frac{\mathrm{Embed}(A_t) * v_{ideal}}{\|\mathrm{Embed}(A_t)\| * \|v_{ideal}\|} - Penalty(A_t) \tag{2.9}$$

де вагові коефіцієнти визначають відносний внесок кожного з критеріїв, а $\eta$ – встановлює ціну помилки. Для життєво важливих призначень (наприклад, антикоагулянти при інфаркті) має високе значення, що робить вплив цієї дії на підсумковий бал визначальним.

Крім того, у моделі може бути передбачено штрафні коефіцієнти для дій, що суперечать клінічній логіці або призводять до деградації стану системи:

$$Penalty(A_t) = \sum w_{side} * (\mathrm{Embed}(A_t) * v_{side_effect}), \tag{2.10}$$

де $v_{side_effect}$ – векторний представник k-го побічного ефекту або небажаного стану отриманий через той самий Encoder.

$w_{side}$ – ваговий коефіцієнт критичності побічного ефекту. Визначає ступінь зниження загального бала. Його значення встановлюється на основі клінічної класифікації небажаних явищ (Adverse Events) та ризиків, що закладені в структуру датасету.

Це дозволяє враховувати не лише ефективність, але й безпечність прийнятих рішень.

Фінальний результат навчальної сесії (Final Score) формується як інтеграл успішності діагностичного пошуку та точності терапевтичного впливу протягом усього часу взаємодії:

$$FinalScore = \sum_{t=1}^{T} (w_1 * \Delta H + w_2 * R_t), \quad (2.11)$$

де константи $w_1$ та $w_2$ дозволяють адаптувати пріоритети оцінювання під конкретну навчальну мету (наприклад, акцент на диференціальній діагностиці або на невідкладній допомозі). Такий комплексний підхід гарантує глибокий аналіз когнітивної траєкторії респондента та безпечності його професійних рішень.

# 3 ПРАКТИЧНА РЕАЛІЗАЦІЯ

Цей розділ присвячено технічній реалізації інтелектуального агенту для моделювання адаптивних клінічних сценаріїв. Система базується на гібридній архітектурі, яка поєднує методи глибокого навчання (BioBERT) із класичними математичними моделями прийняття рішень.

Ключовий компонент реалізації – PatientModel, динамічний граф, що використовує механізм активації вузлів через векторні представлення BioBERT.

На відміну від статичних систем, обраний підхід дозволяє оновлювати ймовірності станів пацієнта в реальному часі за допомогою спеціальних параметрів, як затухання (decay) та вагові коефіцієнти клінічної релевантності.

Особливістю реалізації є впровадження SmartVirtualPatient – модуля на базі LLM (Gemma 3:12B), який забезпечує природномовну взаємодію, зберігаючи при цьому строгу відповідність медичним фактам, закладеним у графі знань.

На рисунку 3.1 представлена проста схема підготовки даних, яка буде описана в подальшому.

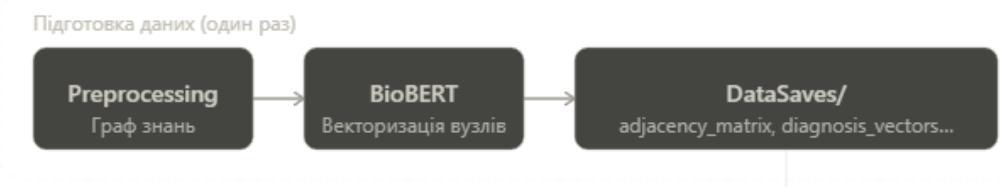



Рисунок 3.1 – Схема підготовки даних

Цей етап включає в себе модулі Preprocessing та BioBERT [10] які трансформують сирі дані датасету DDXPlus [12] у структурований граф знань та набір семантичних векторів і зберігає у відповідну директорію.

Після підготовки даних і збереження артефактів, наступає момент початку сесії. Так, на рисунку 3.2 представлено два рівні: рівень ініціалізації сесії, рівень діагностичного циклу.




Рисунок 3.2 – Логіка процесу сесії

На першому рівні запускається веб-інтерфейс або CLI-тести, де активуються ключові компоненти: PatientModel (математичний стан пацієнта), VirtualPatient (інтерфейс пацієнта) та AdaptiveQuestionSelector (інтелектуальний помічник).

На другому рівні, що представлений, відображається ітераційний процес, у якому дії студента обробляються через семантичний енкодер.

Вони призводять до оновлення ймовірностей у моделі пацієнта, розрахунку інформаційного приросту $\Delta H$ та формування фінального звіту із висновками по діях користувача.

## 3.1 Підготовка даних

Ефективність функціонування інтелектуального агента в медичній області напряму залежить від якості та структурованості даних, на яких базується його логіка прийняття рішень.

Оскільки запропонований метод передбачає динамічне моделювання клінічного сценарію, етап підготовки даних – це фундаментальний процес, який включає не тільки підбір релевантних даних, а й структуризацію масивів інформації у формат, придатний для подальшого використання.

### 3.1.1 Обрання датасету

Реалізація проєкту почалася із вибору відповідного набору даних. Для цього було розглянуто кілька варіантів із вже готових датасетів. Основними варіантами були:

– DDXPlus [12] – датасет, згенерований на основі реальних даних, що містить понад 1.2 мільйонами випадків;

– SympGraph [25] – набір даних, побудований на основі відкритих медичних джерел та статистики CDC;

– MIMIC-IV [20] – велика база деідентифікованих даних реальних пацієнтів відділень інтенсивної терапії.

Після ретельного аналізу цих та ще декількох, не таких великих, наборів даних, було зроблено вибір на користь DDXPlus [5]. Основною причиною було те, що він знаходився у повністю відкритому доступі. Наприклад, хоча MIMIC-IV представляє собою надзвичайно цінний набір даних, отримати до нього доступ тривалий і складний процес [20].

Так, щоб отримати право доступу, треба пройти три етапи. По-перше, завершити спеціальний курс з етики досліджень (CITI Program) тривалістю в чотири тижні. Після цього, подати офіційний запит від імені університету на отримання доступу. Запит може розглядатися кілька тижнів. Якщо він буде погоджений, то вже тоді підписати угоду про нерозголошення [20]. Загалом, весь процес займатиме щонайменше півтора місяці, якщо не виникне жодних затримок, а схвалення запиту не є гарантованим.

Перевагами DDXPlus [12] над іншими датасетами також стали очевидні при детальному аналізі. Ого масштабність дозволяє доволі просто видобути стійкі частотні залежності між симптомами та патологіями для побудови вагових коефіцієнтів графа знань, а чітка структура даних є ідеальними для розрахунку матриці суміжності та подальшої реалізації Марківських процесів [5].

Беззаперечною перевагою також стало те, що датасет містить не лише симптоми, а й антецеденти (фактори ризику, анамнез), що дозволяє ретельніше та точніше побудувати граф знань, виокремлюючи ці фактори у власні вузли.

Всі ці переваги в сукупності дозволили реалізувати одну із головних функцій для достовірної симуляції, а саме латентні симптоми. Так, пацієнт матиме лише початкову скаргу і користувачу доведеться, направляючими питаннями, визначити решту з симптомів та діагноз.

### 3.1.2 Обробка даних

Після обрання датасету DDXPlus [12], дані якого хоча і добре структуровані, ще не придатні для використання, треба обробити той відповідним чином. Для цього було реалізовано комплексний алгоритм попередньої обробки за допомогою модуля Preprocessing [11].

Основним викликом на цьому етапі стала необхідність трансформації технічних ідентифікаторів у назви природною мовою та розмежування типів медичних ознак для коректної роботи графової моделі.

Для початку роботи завантажуються сирі дані із датасету, що збережені локально в папці проєкту.

Спочатку відкриває release_conditions.json, який містить опис усіх діагнозів із переліком їхніх симптомів, антецедентів та рівнем тяжкості. Далі зчитує release_evidences.json – словник усіх можливих симптомів і процедур із їхніми кодами та текстовими описами. Нарешті завантажує тренувальний CSV-файл пацієнтів, автоматично визначаючи розширення .csv якщо воно присутнє [11].

Повертає всі три об'єкти для подальшої обробки і одразу виводить у консоль кількість діагнозів, симптомів та пацієнтів для швидкої перевірки коректності завантаження.

Цей підхід забезпечує структуроване та послідовне завантаження даних, що є критично важливим для подальшого аналізу та моделювання. Завдяки чіткому розділенню інформації на діагнози, симптоми та дані пацієнта, можна ефективно організувати роботу з великим обсягом медичної інформації.

Крім того, автоматичне визначення формату файлу спрощує інтеграцію нових даних у систему без необхідності ручного втручання.

Функція очищення списку симптомів, перетворює початковий текстовий опис симптомів, який може бути присутній у датасеті у вигляді запитання, на коротку іменникову фразу, відкидаючи всі запитальні слова та словосполучення [11].

Після цього видаляє знаки питання, двокрапки та підкреслення, робить першу літеру новосформованої фрази великою. Якщо результат містить назву з міткою хвороби із заздалегідь написаного словнику, то такий запис не вважається симптомом і функція повертає None.

Це сигналізує, що подібну строку не треба записувати. Є обмеження на кількість символів, аби уникнути задовгої назви симптомів.

Для побудови словників використовується вже інша функція, яка проходить по всіх записах у отриманому масиву записів та формує два окремих словники.

Перший словник відображає числовий код кожного симптому на його очищену, перетворену на людську мову, назву і зберігає.

Другий зберігає оригінальну питальну форму запису для тих кодів, де текст містить знак питання. Цей словник пізніше буде використаний для функції рекомендованих запитань у порадах для користувачів.

Паралельно, ця функція формує множину, перевіряючи текс кожного запису на наявність медичних ключових слів із заздалегіть сформованого словнику. Якщо словник очищених назв симптомів порожній, оригінальні коди також йдуть до цієї множини кодів. Виводить загальну кількість записів словнику та кількість запитань у консоль [11].

Для подальшого використання даних будується граф знань, що наведено на рисунку 3.3.

```
def build_graph(conditions :{items}, symptom_names :{get}, procedure_codes :{__contains__}):
    G = nx.Graph()

    for condition_name, cond_data in conditions.items():
        G.add_node(condition_name, node_type="diagnosis",
                   severity=cond_data.get("severity", 0))

        for symptom_code in cond_data.get("symptoms", {}).keys():
            label = symptom_names.get(symptom_code, symptom_code)
            ntype = "procedure" if symptom_code in procedure_codes else "symptom"
            G.add_node(label, node_type=ntype, code=symptom_code)
            G.add_edge(condition_name, label, weight=1.0, edge_type=ntype)

        for ant_code in cond_data.get("antecedents", {}).keys():
            label = symptom_names.get(ant_code, ant_code)
            ntype = "procedure" if ant_code in procedure_codes else "antecedent"
            G.add_node(label, node_type=ntype, code=ant_code)
            G.add_edge(condition_name, label, weight=0.5, edge_type=ntype)

    print(f"   Граф побудовано: {G.number_of_nodes()} вузлів, {G.number_of_edges()} ребер")
    return G
```

Рисунок 3.3 – Побудова графу знань

Для кожного діагнозу із словника додається вузол, тип якого «diagnosis» з атрибутом severity.

Далі, для кожного симптому цього діагнозу визначається тип вузла, «procedure» або «symptom», за тим, в якому із словників присутні їхні назви та додає ребро із початковою вагою в 1.0. Для антецедентів аналогічно додаються вузли з типом «antecedent» або «procedure», але з меншою початковою вагою 0.5, що відображає менший безпосередній зв'язок із діагнозом, в порівнянні із симптомами.

Усі ваги на цьому етапі є умовними та будуть перезаписані в наступному кроці. Саме тут застосовується вираз (2.2), який описував структуру графа [11].

Функція на рисунку 3.4 перераховує ваги ребр графа на основі емпіричної частоти зустрічі симптомів і діагнозів у тренувальних даних [11].

```
def update_graph_weights(G : {has_edge, has_node, __getit... , df : {iterrows} , symptom_names : {get} , conditions : {keys} ):
    co_occurrence = defaultdict(lambda: defaultdict(int))
    diagnosis_count = defaultdict(int)

    print("Розрахунок частот...")
    for _, row in df.iterrows():
        diagnosis = row["PATHOLOGY"]
        diagnosis_count[diagnosis] += 1
        try:
            evid_list = ast.literal_eval(row["EVIDENCES"])
        except Exception:
            continue
        seen = set()
        for ev_code in evid_list:
            seen.add(ev_code.split("_@_")[0])
        for base_code in seen:
            ev_name = symptom_names.get(base_code, base_code)
            co_occurrence[diagnosis][ev_name] += 1

    for diagnosis, symp_freq in co_occurrence.items():
        total = diagnosis_count[diagnosis]
        for symptom, count in symp_freq.items():
            alpha_ij = count / total
            if G.has_edge(diagnosis, symptom):
                G[diagnosis][symptom]["weight"] = alpha_ij
            elif G.has_node(diagnosis):
                G.add_edge(diagnosis, symptom, weight=alpha_ij, edge_type="frequency")

    first = list(conditions.keys())[0]
    top5 = sorted(G[first].items(), key=lambda x : x[1]["weight"], reverse=True)[:5]
    print(f"\nТоп-5 симптомів для '{first}':")
    for symp, data in top5:
        print(f"  α={data['weight']:.3f}  →  {symp}")
```

Рисунок 3.4 – Оновлення ваг ребр графу

Для цього він проходить по кожному рядку датасету пацієнтів, розбирає стовпець «EVIDENCES» із рядка у список та витягує базові коди симптомів та і нараховує лічильники в двовимірному словнику co_occurrence [diagnosis] [symptom].

Після повного обходу датасету для кожної пари діагноз-симптом обчислює нормовану частоту і записує її як нову вагу ребра, замінюючи початкове значення. Якщо ребро між діагнозом і симптомом ще не існує в графі але зустрічається в даних – воно додається з типом «frequency». Відповідним виразом для відображення цих змін є вираз (2.4), який демонструє логіку розрахунку ваг ребр.

На графі (рисунок 3.5) зображено підграф із трьох діагнозів – HIV (initial infection), Inguinal hernia та Anaphylaxis – які були обрані алгоритмом як найбільш взаємопов'язана трійка серед перших п'ятнадцяти діагнозів датасету [11].

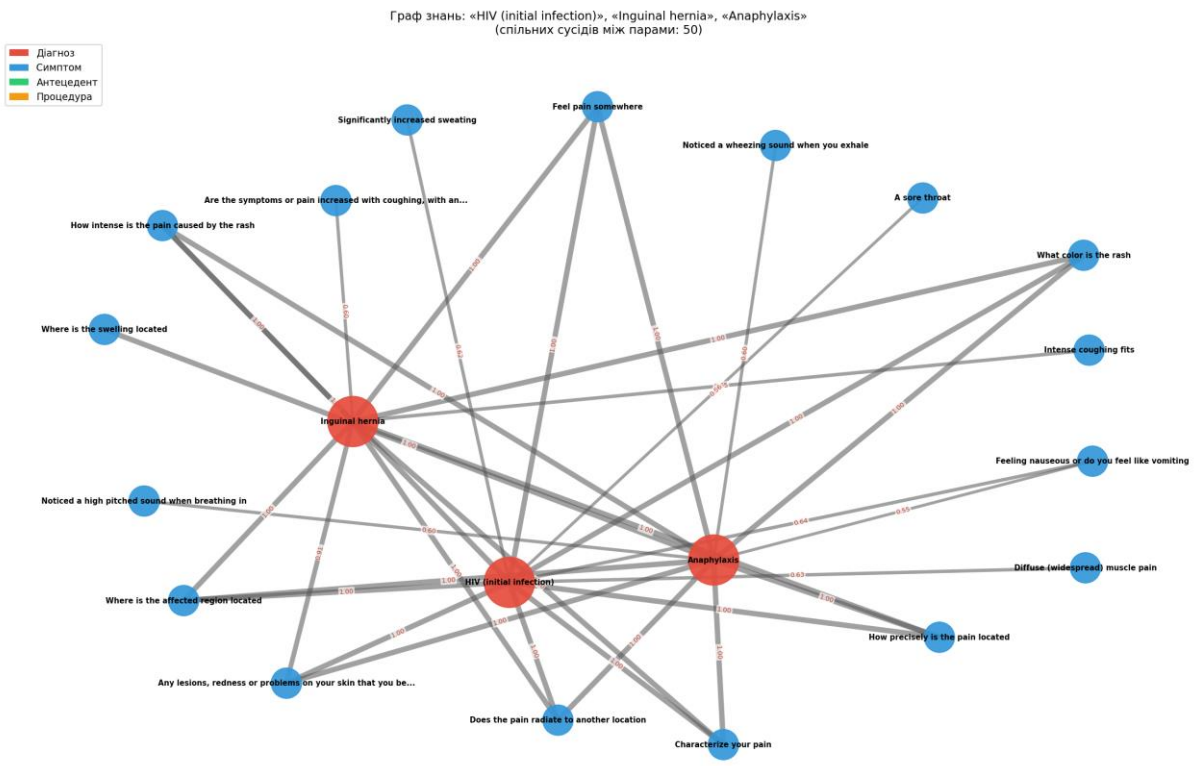



Рисунок 3.5 – Приклад побудованого графу

Більшість ребр мають вагу 1.00, що означає максимальну частоту – ці симптоми зустрічаються у практично кожного пацієнта з відповідним діагнозом. Кілька ребр мають нижчі ваги.

Це свідчить про те, що між цими діагнозами існує тісний зв'язок через спільні симптоми, які часто супроводжують різні захворювання. Наприклад, симптоми, що мають вагу 1.00, є ключовими маркерами для діагностики, оскільки їхня присутність майже гарантована при відповідному діагнозі. Водночас ребра з меншою вагою відображають симптоми, які зустрічаються рідше або мають меншу специфічність, що може бути пов'язано з індивідуальними особливостями пацієнтів або різними стадіями захворювання.

Представлена на рисунку 3.6 функція призначена для побудови матриці суміжності [11].

```
def build_matrices_and_metadata(G : {nodes, has_edge, __getitem_... , conditions):  1 usage  Ihor S
    diagnoses  = [n for n, d in G.nodes(data=True) if d.get("node_type") == "diagnosis"]
    symptoms   = [n for n, d in G.nodes(data=True)
                  if d.get("node_type") in ("symptom", "antecedent", "procedure")]
    procedures = [n for n, d in G.nodes(data=True) if d.get("node_type") == "procedure"]

    adj_matrix = np.zeros((len(diagnoses), len(symptoms)))
    for i, diag in enumerate(diagnoses):
        for j, symp in enumerate(symptoms):
            if G.has_edge(diag, symp):
                adj_matrix[i][j] = G[diag][symp]["weight"]

    diagnosis_tests = {}
    for diag in diagnoses:
        tests = [(nbr, G[diag][nbr]["weight"])
                 for nbr in G.neighbors(diag)
                 if G.nodes[nbr].get("node_type") == "procedure"]
        tests.sort(key=lambda x : {__getitem__} : x[1], reverse=True)
        diagnosis_tests[diag] = [t[0] for t in tests[:5]]

    return diagnoses, symptoms, procedures, adj_matrix, diagnosis_tests
```

Рисунок 3.6 – Побудова матриці суміжності

Вона витягує з графа три вузли і розподіляє їх у відповідні впорядковані списки. На основі цих списків будує щільну матрицю

суміжності, розміром кількості діагнозів на кількість симптомів і заповнює кожну клітинку вагами ребр із графа або нулями при відсутності ребра.

Ця матриця є центральним об’єктом в подальшому. Додатково формує словник, умовних, діагностичних тестів – для кожного діагнозу бере до п’яти процедур із найвищими вагами ребр, які слугуватимуть як релевантні діагностичні тести. Для цієї функції застосований вираз (2.3), але лише частково. Вираз був розділений на частини і його продовження буде представлено в подальшому.

Після цього всі згенеровані артефакти зберігаються у папку DataSaves/ з відповідними форматами на назвами. Також зберігаються метадані графа у форматі JSON. Сам граф зберігається за допомогою pickle у файл knowledge_graph.pkl.

Остання функція будує наочну візуалізацію підграфу для трьох найбільше взаємопов’язаних діагнозів (рисунок 3.5).

## 3.2 BioBERT: налаштування

Після завершення обробки даних, побудови графа знань та їхнього збереження, необхідно створити механізм, який дозволить системі розуміти текстовий вільний ввід користувача. Оскільки традиційні методи пошуку за ключовими словами не здатні сприймати синоніми та складні медичні формулювання, то в проєкті використовується передтренована на медичних даних модель BERT, а саме BioBERT [10].

Саме ця модель була обрана з ряду певного ряду причин. Основною стало те, що оригінальний BERT є перевіреною моделлю, яка зарекомендувала себе в багатьох сферах, а її модифікація, BioBERT [10], спеціалізується на медицині, профільній літературі та наукових текстах [4]. Це забезпечує значно вищу семантичну точність у порівнянні з універсальними мовними моделями.

Загалом, на цьому етапі ми реалізовуємо перетворення текстової інформації графа у багатовимірний векторний простір, де близькість об’єктів визначається їхнім клінічним значенням.

### 3.2.1 Ініціалізація моделі та середовища виконання

Перед створенням і заповненням семантичного простору медичних сутностей, слід ініціалізувати саму модель та її середовище виконання. Для цього потрібно всі вузли раніше побудованого графа перетворити у нормалізовані вектори, що дозволить реалізувати механізм пошуку, де система знаходить найбільш подібні медичні сутності навіть за неточного текстового опису [11].

Застосовується функція завантаження всіх попередньо збережених артефактів.

Функція їх викриває та розпаковує. Окремо вона також зчитує CSV-файл тренувальних пацієнтів. Як і в функціях обробки даних, тут перевіряється наявність розширення .csv, щоб уникнути помилок при завантаженні.

Для ініціалізації моделі використовується спеціальний метод, що має, умовно, три кроки.

По-перше, за допомогою токенізатора (AutoTokenizer) завантажується сама модель. Він же відповідає за перетворення тексту на числові токени, які відповідають словнику моделі.

По-друге, Модель (AutoModel) буде представляти саму нейронну мережу BioBERT, яка спеціально навчена на біомедичних текстах.

По-третє, визначається пристрів обчислення даних моделлю. За наявності ядр CUDA, саме вони будуть застосовані, в іншому випадку – CPU.

Оскільки BioBERT найкраще працює з англомовною термінологією, було впроваджено модуль автоматичного перекладу вводу респондента з української мови перед векторизацією.

### 3.2.2 Створення семантичного простору медичних сутностей

Після підготовки середовища виконання та завантаження даних , починається реалізація перетворення всіх вузлів побудованого раніше графа у нормалізовані вектори.

На рисунку 3.7 відображено програмну реалізації виразу (2.1), а саме створення енкодеру на основі моделі BioBERT [11].

```
def make_encoder(tokenizer, model, device):  1 usage  Ihor Sikorskyi
    def encode_text(text: str) -> np.ndarray:  Ihor Sikorskyi
        inputs = tokenizer(text, return_tensors="pt", truncation=True,
                           max_length=128, padding=True).to(device)
        with torch.no_grad():
            outputs = model(**inputs)
        return outputs.last_hidden_state[:, 0, :].cpu().numpy().squeeze()
    return encode_text
```

Рисунок 3.7 – Створення енкодера

Ця функція дозволяє зберегти токенізатор, модель та пристрій у замкнутому контексті за допомогою повернення внутрішньої функції encode_text.

Ця функція здійснює токенізацію вхідного тексту із обрізанням до 128 токенів. Потім виконується прямий прохід через модель в контексті torch.no_grad() для мінімізації споживання пам’яті. Після цього процесу функція повертає CLS-токен з вихідного прихованого стану як вектор розмірності. Це і семантичне представлення вхідного тексту.

Функція normalize виконує L2-нормалізацію [18] матриці векторних представлень по рядках. Для цього, для кожного рядка, обчислюється евклідова норма.

При нульових значеннях цієї норми замінюються одиницями з метою запобіганню ділення на нуль. Нормалізація застосовується, щоб привести всі вектори до одиничної довжини. Це, в свою чергу, дозволить звести обчислення, в подальших застосуваннях матриці, до скалярного добутку та загалом підвищить ефективність подальших операцій порівняння. Саме в цій функції застосовується ще одна частина виразу (2.3). А саме знаменник представленого виразу.

Для того, щоб перетворити структурну інформацію графа у математичну формулу, яка буде придатна для використання у семантичному аналізі, використовується ітераційна обробка кожного елементу. Процес починається із формування єдиного масиву із усіма наявними медичними сутностями (рисунок 3.8) [11].

```
def vectorize_nodes(diagnoses :{__add__} , symptoms, encode_text):
    all_nodes = diagnoses + symptoms
    node_vectors = {}
    print(f"   Векторизація {len(all_nodes)} вузлів...")
    for node_name in tqdm(all_nodes):
        node_vectors[node_name] = encode_text(node_name)
    return node_vectors
```

Рисунок 3.8 – Формування словника вузлів графу

Саме функція vectorize_nodes() формує словник векторних представлень для всіх вузлів графа знань. Списки діагнозів та симптомів об'єднуються в єдину послідовність, для кожного елемента якої викликається функція encode_text та отриманий вектор зберігається у словник із ключем у вигляді назви вузла.

### 3.2.3 Інфраструктура векторного кодування

Для стабільного функціонування семантичного шару системи, було розроблено набір допоміжних функцій та методів, що реалізовані у окремому модулі. Ці інструменти виконують роль проміжного шару, який котує сирий текстовий ввід користувача до математичної обробки моделлю BioBERT [10]. Функція normalize_vec виконує L2-нормалізацію окремого вектора [18].

Обчислює евклідову норму вектора і, якщо вона відмінна від нуля, ділить на неї кожен елемент. У випадку нульового вектора повертає його без змін, що запобігає діленню на нуль. Нормалізований вектор має одиничну довжину, що є необхідною умовою для коректного обчислення косинусної подібності.

Функція, зображена на рисунку 3.9, призначена для приховування складної послідовності низькорівневих операцій, щоб ті могли бути виконані одним викликом замість кількох послідовних [11].

```
def embed_action(action: str) -> np.ndarray:  13+ usages  Ihor Sik
    if action not in _embed_cache:
        eng = translate_to_english(action)
        _embed_cache[action] = normalize_vec(encode_text(eng))
    return _embed_cache[action]
```

Рисунок 3.9 – Отримання нормалізованого вектора

Ця функція реалізує повний процес отримання нормалізованого векторного представлення для дій користувача системи.

Спочатку текст перекладається на англійську через метод, аналогічний до того, що застосовується у функції перекладу для BioBERT, після чого отримується вектор через вищезгадану функцію encode_text (рисунок 3.7) та нормалізується за допомогою normalize_vec.

Далі реалізоване кешування результатів у глобальному словнику, щоб, у випадку повторного звернення з ідентичним текстом, вектор повертався з кешу без звернення до моделі наново. Це суттєво підвищує продуктивність системи в умовах багаторазового виконання одних і тих самих дій.

## 3.3 Модель пацієнта

Головним і основним компонентом розробленої системи є математична модель пацієнта, реалізована у класі PatientModel [11]. Цей модуль виконує роль імітаційного середовища, яке акумулює знання, отримані в ході діагностичного процесу.

Він динамічно змінює свій внутрішній стан у відповідь на дії користувача. Відмінність від статичних моделей полягає в тому, що ця модель базується на ітераційному оновлені характеристик ймовірності станів.

Це оновлення дозволяє відтворити процес поступового зниження невизначеності при постановці діагнозу.

В основі роботи PatientModel лежить поєднання графової структури знань із їхнім векторним представленням. Взаємодія з моделлю відбувається за принципом байєсівського оновлення переконань.

PatientModel забезпечує гнучкість і адаптивність у процесі діагностики, що дозволяє враховувати нові дані та симптоми в режимі реального часу.

Завдяки ітераційному оновленню ймовірностей станів, модель поступово уточнює діагноз, зменшуючи невизначеність і підвищуючи точність результатів.

Сама модель PatientModel (рисунок 3.10) є класом, який реалізує адаптивну мережу діагностики пацієнта [11].

```
class PatientModel:  4+ usages  Ihor Sikorskyi *
    def __init__(self, diagnoses : {__len__} , symptoms : {__len__} , adj_matrix : {__gt__, __mul__} ,
                 diag_vectors : {__truediv__} , symp_vectors : {__truediv__} ,
                 beta: float = 0.4,
                 decay: float = 0.85,
                 top_k: int = 5,
                 eta: float = 0.15,
                 w_clip: float = 3.0,
                 adaptive_adj: bool = True):

        self.diagnoses = diagnoses
        self.symptoms  = symptoms
        self.beta  = beta
        self.decay = decay
        self.top_k = top_k
        self.eta   = eta
        self.w_clip = w_clip
        self.adaptive_adj = adaptive_adj
```

Рисунок 3.10 – Клас PatientModel та його аргументи

Як основні характеристики, він має два вектори станів – для симптомів та діагнозів, а також матрицю зв’язків між ними, яка перебудовується під кожного нового згенерованого пацієнта.

Модель працює за принципом рекурентної асоціативної пам’яті, де кожна наступна дія користувача або підтверджений симптом змінює внутрішній стан. Дані цього стану накопичуються між кроками на впливають на наступні дії пацієнта. Цей віртуальний пацієнт містить всі його дані: виявлені та ще ні симптоми, відповідність діагнозу лікаря тощо.

Щоб модель працювала коректно, вона потребує ряду параметрів і характеристик, які ініціалізуються в середині функції init (рисунок 3.17), яка приймає, загалом, одинадцять аргументів і виконує три підготовчі операції перед тим, як модель буде готова до роботи.

Першим аргументом, який приймається, є список рядків із назвами діагнозів. В цьому списку кожному діагнозу присвоюється свій унікальний індекс, де діагноз під індексом 0 відповідає нульовому рядку матриці суміжності та нульовому елементу вектору станів діагнозів. Список є статичним і не буде змінюватися після ініціалізації. Використовується в

подальшому для спрощеного пошуку діагнозів у методі оновлення стану моделі.

Аналогічним аргументом є список симптомів, який задає порядок стовбців матриці та індексів у векторі станів симптомів. Таким самим чином використовується в подальшому для перетворення текстових назв підтверджених симптомів на числові індекси у методі для оновлення стану моделі.

Також приймається матриця суміжності, що являє собою двовимірний масив, де кожен елемент є значенням початкової сили зв’язку між діагнозом та симптомом.

Також є згаданий вектор діагнозів, розмір якого визначається кількістю діагнозів та розмірністю семантичного простору. Кожен рядок, відповідно, векторне представлення одного діагнозу. Використовується для прямого порівняння вектора дії з діагнозами через косинусну подібність. Конструктор одразу нормалізує цей вектор.

Аналогічно до вектору діагнозів, обробляється і вектор симптомів за тим самим принципом визначення його розмірності. Використовується для пошуку симптомів, що семантично близькі до вектору, що представляє поточну дію користувача.

Аргумент beta – коефіцієнт швидкості реакції на нові спостереження. Цей аргумент відповідає за те, наскільки сильно один новий крок може змінити активації симптомів. За замовчуванням, якщо не задавати це значення, воно буде рівним 0.4. При такому рівні, новий крок додає 40% свого значення впливу до вже накопиченого стану. Але на цей вплив також діють аргументи tanh та decay. Через них реальний вплив також залежить від поточного рівня активації симптомів. Якщо beta буде занадто низьким, то модель буде майже інертною, реагувати надто повільно та постійно потребувати додаткового підтвердження дій. При завеликому значенні одна єдина дія може різко змінити діагностичний процес і розкрити забагато інформації.

Згаданий аргумент decay, який за замовчуванням – 0.85, є коефіцієнтом пам’яті між кроками. На кожному кроці попередній стан активації множиться на це значення і, відповідно, зменшується на 15%. Це забезпечує певний рівень втрати пам’яті для давніх сигналів. Такий підхід дозволяє уникнути зациклювання моделі на одних і тих же, вже розкритих, фактах, які ніколи не з’являлися після першої появи. Значення за замовчуванням достатньо велике, щоб зберігати пам’ять протягом кількох кроків, але достатньо мале, щоб модель не «застрягала» на ранніх сигналах.

Аргумент top_k – максимальна кількість симптомів, що беруться з косонусної подібності на одному кроці. Після обчислення схожості вектору дії з усіма симптомами беруться тільки top_k найближчих. За умови, що подібність більше за 0.7. Це додатковий фільтр, щоб не активовувати всі симптоми, а лише найрелевантніші. Такий підхід зменшує шум і не дає розмитися діагностичному сигналу.

За швидкість адаптації матриці суміжності під поточного пацієнта відповідає аргумент eta. Це довжина кроку, на який зміщується поточний розподіл діагнозів при підтвердженні симптомів. Що менше значення цього аргументу, то довше матриця залишається в межах норм початкової матриці. Що більше значення eta, то швидша ця перебудова. Із зависоким параметром існує ризик перенавчання із скороченням під більш обмежений набір спостережень.

За допомогою w_clip встановлюється верхня межа для значень у матриці суміжності після кожного оновлення. Після збільшення значень елементів матриці у функції _update_adj, що представлена нижче, всі значення вище встановленого обрізаються до максимально можливого. Це захищає від ситуації, коли кілька підряд підтверджень одного симптому призводять до нескінченного зростання ваги і домінування одного зв'язку над усіма іншими.

adaptive_adj – це аргумент, який або вмикає, або вимикає адаптацію матриці для автоматичного режиму тестування системи.

Після ініціалізації основних аргументів відбувається процес нормалізації векторів діагнозів та симптомів моделі (рисунок 3.11) [11].

```
self.diag_vectors = diag_vectors / np.linalg.norm(diag_vectors, axis=1, keepdims=True)
self.symp_vectors = symp_vectors / np.linalg.norm(symp_vectors, axis=1, keepdims=True)

N    = len(diagnoses)
df_  = np.sum(adj_matrix > 0, axis=0)
df_[df_ == 0] = 1
idf  = np.log(N / df_)
adj_tfidf = adj_matrix * idf[np.newaxis, :]

row_norms = np.linalg.norm(adj_tfidf, axis=1, keepdims=True)
row_norms[row_norms == 0] = 1
self._adj_base = adj_tfidf / row_norms
self.adj_matrix = self._adj_base.copy()

self.diag_scores = np.zeros(len(diagnoses))
self.symp_scores = np.zeros(len(symptoms))

self.step = 0
```

Рисунок 3.11 – Налаштування моделі

Для цього кожен вектор ділиться на його евклідову норму, що перетворює ті на одиничні вектори. Скалярний добуток цих векторів дорівнює їхній косинусній подібності, що дозволяє cosine_similarity в наступних функціях коректно та швидко працювати. Налаштування keepdims=True зберігає оригінальну розмірність векторів для коректного застосування.

N – загальна кількість діагнозів, в той час як df_ – кількість діагнозів, у який присутній певний симптом (рисунок 3.11). Параметр df_ захищений від ділення на нуль для симптомів, що не зустрічаються жодного разу.

Вираз (2.4) у реалізації розширюється TD-IDF перетворенням матриці суміжності. Після обчислення частотних ваг елементів матриці за виразом(2.4) виконується додаткове масштабування у параметрі idf (рисунок 3.11).

Після цього відбувається масштабування матриці сумісності покомпонентно в параметрі adj_tfidf (рисунок 3.11). Множення в цій операції йде поелементно по стовбцях. Це забезпечує незалежне масштабування кожного симптому.

У підсумку, специфічні симптоми, що характерні лише для однієї-двох хвороб набуваються значно вищої ваги при діагностуванні порівняно із більш поширеними.

Після цього йде процес L2-нормалізації [18] рядків та збереження еталону матриці.

Кожен рядок adj_tfidf нормалізується до одиничної норми, щоб діагнози з великою кількістю симптомів не домінували лише через кількість. _adj_base зберігає цю нормалізовану популяцію як еталонну і далі буде застосовуватися для регуляції оновлення робочої копії adj_matrix (рисунок 3.11).

На старті всі активації нульові, бо модель ще не знає нічого про пацієнта. Також існує параметр step для підрахунку кроків, який збільшується з кожним оновленням стану.

Ця snapshot створює словник копій чотирьох об’єктів. Ці копії будуть залишатися незмінними, якщо не буде використана функція відновлення для скидання кроку, якщо сталася помилка в роботі пацієнта.

Якби це не були саме копії об’єктів, то словник би зберігав посилання на оригінальні об’єкти і їхнє відновлення було б неможливим.

Функція restore приймає словник із snapshot і відновлює з нього стан моделі.

Також використовує метод копіювання, щоб при відновленні модель і snapshot не ділили спільні масиви пам’яті. Без цього подальше оновлення стану моделі змінювало б і snapshot, що унеможливило б повторне відновлення стану із того ж словника.

Для реалізації динамічного оновлення стану пацієнта використовується спеціальна функція, яка ілюструється на рисунку 3.12. Ця

функція відповідає за логіку переходу між станами, враховуючи поточні активації, параметр step та інші фактори, що впливають на стан пацієнта [11].

```
def _update_adj(self, confirmed_symp_indices: list[int]) -> None:  1 usage  Ihor Sikors
    exp_d = np.exp(self.diag_scores - self.diag_scores.max())
    p_diag = exp_d / (exp_d.sum() + 1e-12)

    if confirmed_symp_indices:
        for j in confirmed_symp_indices:
            self.adj_matrix[:, j] += self.eta * p_diag

        if self.adaptive_adj:
            not_confirmed = np.ones(len(self.symptoms), dtype=bool)
            not_confirmed[confirmed_symp_indices] = False
            self.adj_matrix[:, not_confirmed] = (
                    (1 - self.eta * self.decay) * self.adj_matrix[:, not_confirmed]
                    + self.eta * self.decay * self._adj_base[:, not_confirmed]
            )
```

Рисунок 3.12 – Оновлення матриці зв’язків при підтверджених симптомах

Вона оновлює матрицю зв’язків на основі підтверджених симптомів. Викликається в методі для оновлення стану пацієнту після кожного кроку, де хоч один симптом був підтверджений. Якщо список підтверджених симптомів порожній, то не виконується жодних дій.

До того, як оновлювати значення всередині самої матриці, метод має з’ясувати, які діагнози є найбільш достовірними для моделі на даному кроці.

Для цього поточні активації діагнозів diag_scores перетворюються на ймовірнісний розподіл через softmax [6]. Віднімання максимально можливого значення цього параметру є стандартним підходом для забезпечення чисельної стабільності. Без цього, при великих значеннях активацій np.exp повертає нескінченість (inf), що ламає всі подальші ділення. Після цієї дії з відніманням найбільше значення exp_d, яке є можливим, це одиниця і всі менші знаходяться в проміжку [0; 1].

В розрахунках для p_diag відбувається процес ділення, де у знаменнику до exp_d додається певне значення для запобіганню діленню на нуль у теоретично можливому крайньому випадку, коли всі активації однакові. Результатом цієї операції ділення є масив розміром у кількість діагнозів, де сума всіх елементів дорівнює одиниці, а кожне значення означає впевненість моделі у діагнозі.

Для кожного підтвердженого симптому з індексом j до цілого стовбця додається вектор exp_d * p_diag. Це пряма реалізація правил Хебба [15] про те, що нейрони, які активуються разом, з’єднуються сильніше.

Завдяки цьому правилу оновлення пропорційне поточній впевненості моделі: чим сильніша гіпотеза – тим більше вона підсилюється підтвердженим симптомом.

Це потрібно для автоматизованого тестування системи, яке відбувається без участі студента для підтвердження роботи моделі. І в іншому випадку не застосовується.

В цій функції також відбувається регуляція непідтверджених симптомів (рисунок 3.12).

Це потрібно для того, бо після посилення підтверджених зв’язків виникає ризик спотворення матриці: якщо постійно посилювати певні стовпці, а решту залишати без змін, то матриця поступово припиняє відображати популяційну статистику і стає надто специфічною для поточного пацієнта.

Це не є проблемою у випадку, коли системою користується реальна людина, оскільки матриця створюється під кожного пацієнта окремо, проте може стати на заваді при автоматичному тестуванні.

Щоб цього уникнути, для всіх непідтверджених стовбців застосовується рух значень в бік базовою матриці. Це відбувається за формулою EMA.

Це означає, що, якщо в попередніх кроках якийсь зв’язок різко виріс через збіг обставин, але відповідний симптом більше не з’являється, то

кожен наступний крок буде ближчим до популяційного на значення на певний, вирахуваний, відсоток.

У випадку, коли підтверджених симптомів немає, але увімкнений режим адаптивної матриці, виконується пасивне оновлення (рисунок 3.13) [11].

```
else:
    if self.adaptive_adj:
        soft_signal = self.symp_scores.clip(0)
        soft_signal = soft_signal / (soft_signal.sum() + 1e-12)
        self.adj_matrix += self.eta * 0.05 * np.outer(p_diag, soft_signal)
        self.adj_matrix = (
                (1 - self.eta * 0.02) * self.adj_matrix
                + self.eta * 0.02 * self._adj_base
        )

np.clip(self.adj_matrix, a_min: 0.0, self.w_clip, out=self.adj_matrix)
row_norms = np.linalg.norm(self.adj_matrix, axis=1, keepdims=True)
row_norms = np.maximum(row_norms, x2: 1.0)
self.adj_matrix /= row_norms
```

Рисунок 3.13 – Оновлення матриці при не підтверджених симптомах

На відміну від активного режиму, де матриця різко підсилює підтверджені зв'язки, тут відбувається повільний дрейф на основі поточного стану моделі.

Спочатку з вектору symp_scores відбираються лише позитивні активації. Отриманий вектор нормалізується до суми одиниці, утворюючи розподіл soft_signal, який відображає відносну силу поточних активацій симптомів. У випадку з негативними значеннями, то вони означають пригнічені симптоми і не мають впливу на оновлення.

Далі, через зовнішній добуток, формується матриця розміром кількість діагнозів на кількість симптомів, де кожен елемент є добутком впевненості моделі у діагнозі та сили симптому. Ця матриця додається до

adj_matrix з малим коефіцієнтом eta, який значно нижчий ніж в активному режимі, щоб не спотворювати матрицю без явних підтверджень.

Після цього відбувається м'яке повернення матриці до базових значень. Це запобігає накопиченню похибок від пасивних оновлень протягом тривалого діалогу – якщо модель довго не отримує підтверджень, матриця поступово повертається до популяційної статистики, в еталоні.

Це потрібно у випадку, коли жодні симптоми не підтверджуються тривалий час. Тоді може статися так, що матриця б просто не могла б коректно оновлюватися, втрачаючи адаптацію до контексту. Це оновлення при відсутності підтверджень допомагає цьому запобігти.

Після всіх оновлень, значення матриці можуть бути занадто великими, особливо у випадку, якщо один і той самий симптом був підтверджений кілька разів поспіль.

Для цього всі значення в матриці, що виходять за межі діапазону [0.0; 3.0], обрізаються прямо в пам’яті, що важливо для продуктивності. Нижня межа запобігає від’ємним вагам, бо такий зв’язок немає медичного сенсу. Верхня ж запобігає надмірному привалюванню одного симптому, що переважив би усі інші ознаки. Після цього всі рядки матриці нормалізуються знову до L2-норми [18].

Для забезпечення коректної роботи моделі пацієнта необхідно не лише фіксувати окремі симптоми, а й підтримувати узгоджений внутрішній стан, який змінюється в процесі діалогу. Саме тому в системі передбачено механізм послідовного оновлення параметрів, що відображають актуальний стан пацієнта.

Такий підхід дає змогу враховувати як нові підтвердження симптомів, так і відсутність змін протягом певного часу, зберігаючи стабільність моделі. Крім того, це забезпечує адаптацію до контексту розмови та запобігає накопиченню випадкових похибок під час тривалої взаємодії.

Основний метод для роботи моделі пацієнта, який використовується на кожному кроці взаємодії з пацієнтом: оновлення стану пацієнта. На

рисунку 3.14 представлений його фрагмент, що відповідає за оновлення стану симптомів [11].

```
def update_state(self, action_vector: np.ndarray,  6 usages (6 dynamic)  Ihor Sikorskyi
                 revealed_symptoms: list[str] | None = None) -> np.ndarray:
    self.step += 1

    symp_sim = cosine_similarity(action_vector.reshape(1, -1), self.symp_vectors)[0]

    sparse_symp = np.zeros_like(symp_sim)
    top_k_idx = np.argsort(symp_sim)[::-1][:self.top_k]
    for idx in top_k_idx:
        if symp_sim[idx] > 0.7:
            sparse_symp[idx] = symp_sim[idx]

    confirmed_indices = []
    if revealed_symptoms:
        for name in revealed_symptoms:
            if name in self.symptoms:
                idx = self.symptoms.index(name)
                sparse_symp[idx] = 1.5
                confirmed_indices.append(idx)

    multiplier = 1.0 if confirmed_indices else 0.3

    raw_symp = self.decay * self.symp_scores + (self.beta * sparse_symp * multiplier)
    self.symp_scores = np.tanh(raw_symp)

    self._update_adj(confirmed_indices)
```

Рисунок 3.14 – Оновлення стану пацієнта: симптоми

Починається він з збільшення кількості кроків на один для відстежування тривалості діалогу.

Далі обчислюється косинусна подібність між вектором дії та вектором симптомів. У результаті отримується вектор symp_sim, розмір якого рівний кількості симптомів. В цьому векторі кожне значення означає семантичну близькість поточної дії до відповідного симптому.

Після цього, на основі отриманого вектору, створюється ще один з такою ж розмірністю, але повністю заповнений нулями. Це потрібно, щоб далі його використати його як фільтр і відсіяти симптоми, чий поріг подібності менше 0.7, але перед цим симптоми були відсортовані за спаданням для легшого відбирання побідних. Це усуває слабкі асоціації, які замінюються нулями із створеного нульового вектору.

Як тільки загальна обробка симптомів пройшла, починається обробка лише підтверджених за допомогою умови наявності в списку підтвердженних симптомів revealed_symptoms.

Для кожної назви в цьому списку знаходиться відповідник в списку симптомів, що був переданий моделі при ініціалізації (рисунок 3.17), і відповідна позиція отримує фіксоване значення, яке більше за будь-яке автоматично розраховане, у раніше відфільтрованому векторі симптомів. Це гарантує домінацію підтвердженого симптому над рештою. Індекси додається в масив confirmed_indices для подальшої використання в оновлені матриці зв'язків (рисунок 3.21–3.22).

На основі наявності підтверджень визначається множник впливу, де 1.0 для максимального впливу, якщо є хоча би одне підтвердження та 0.3, якщо немає.

Це пригнічує вплив автоматичного розрахованих сигналів у кроках без підтвердження, що запобігає накопиченню шуму.

Нові значення symp_scores (рисунок 3.18) мали розраховуватися за виразом (2.5), але розрахунок був видозмінений, дещо ускладнений та розділений для симптомів та діагнозів у зв'язку з потребами моделі. Тут буде представлений розрахунок для симптомів:

$$S_{symp}^{(t+1)} = \tanh\left(\lambda * S_{symp}^{t} + \beta * m * Embed(A_t)\right), \qquad (3.1)$$

де $\lambda$ – коефіцієнт затухання;

$\beta$ – коефіцієнт оновлення;

m – множник впливу.

Стискання все функцією tanh в проміжку [-1; 1] запобігає необмеженому зростанню активацій.

Після оновлення симптомних активацій викликається метод оновлення матриці зв'язків, куди передається писок підтверджених індексів поточного кроку.

Далі формується діагностичний сигнал, а точніше два незалежних сигнали (рисунок 3.15) [11].

```
diagnosis_from_symptoms = self.adj_matrix @ self.symp_scores
direct_diagnosis_sim = cosine_similarity(action_vector.reshape(1, -1), self.diag_vectors)[0]

raw_diag = (self.decay * self.diag_scores
            + 0.8 * diagnosis_from_symptoms
            + 0.2 * self.beta * direct_diagnosis_sim)
self.diag_scores = np.tanh(raw_diag)

return np.concatenate([self.diag_scores, self.symp_scores])
```

Рисунок 3.15 – Оновлення стану пацієнта: діагноз

Перший формується від множення поточної матриці зв’язків на вектор активацій симптомів. Цей процес дозволяє врахувати накопичену інформацію про симптоми, які вже були підтверджені або виявлені у пацієнта.

Матриця є адаптивною і оновлюється після кожного нового підтвердженого симптому, що дає змогу моделі динамічно коригувати зв’язки між симптомами та можливими діагнозами. Таким чином, цей сигнал відображає непрямий, але глибокий контекст, який формується на основі історії симптомів пацієнта.

Другий – пряма косинусна подібність між вектором дії та вектором діагнозів, що відображає безпосередній семантичний контекст питання студента без урахування попередньої історії діалогу. Цей сигнал відображає безпосередній семантичний контекст поточного запиту, не враховуючи попередню історію діалогу.

Використання косинусної подібності дозволяє оцінити, наскільки поточне питання семантично близьке до кожного з можливих діагнозів, що допомагає швидко зорієнтуватися у напрямку діагностичного пошуку.

Отримане значення використовується як основа для подальшого коригування ваг діагнозів у моделі.

І вже тут застосовується новий вираз для оновлення діагностичного стану, який є модифікованим виразом (2.5):

$$S_{diag}^{(t+1)} = \tanh\left(\lambda * S_{diag}^{t} + 0.8 * W_t * S_{symp}^{(t+1)} + 0.2 * \beta * cos(Embed(A_t), V_{diag})\right), \quad (3.2)$$

де $W_t$ – адаптивна матриця суміжності, що оновлюється після кожного підтвердженого симптому, що є першим діагностичним сигналом, а $V_{diag}$ це другий, прямий, сигнал.

PatientModel має ряд допоміжних методів, згаданих раніше, які призначенні для виправлення несправності в матриці знань (рисунок 3.16) [11].

```
def get_diagnosis_activations(self) -> np.ndarray:  12 usages (11 dynamic)  Ihor Sik
    return self.diag_scores

def get_matrix_drift(self) -> float:  Ihor Sikorskyi
    return float(np.linalg.norm(self.adj_matrix - self._adj_base,  ord: 'fro')
                 / (self._adj_base.shape[0] * self._adj_base.shape[1]))

def reset_adj(self) -> None:  Ihor Sikorskyi
    self.adj_matrix = self._adj_base.copy()
    self.step = 0
```

Рисунок 3.16 – Допоміжні методи PatientModel

Це ряд незначних методів, які несуть допоміжну функцію. Перший з них, get_diagnosis_activations, повертає поточне значення діагностичних активацій diag_scores (рисунок 3.11).

Використовується зовнішніми компонентами системи для отримання поточного стану гіпотез моделі без необхідності звертатися до внутрішнього стану об'єкта напряму.

Метод get_matrix_drift повертає скалярну міру того, наскільки поточна матриця зв'язків відхилилася від базової.

Саме відхилення рахується за нормою Фробеніуса [23], як різниця між поточної матрицею та еталоном, що нормалізована на кількість елементів матриці. Ця норма враховує всі елементи матриці одночасна, що показує відхилення у вигляді числового значення, що вище яке, то більше відхилення. І останній метод, reset_adj, призначений для скидання стану матриці зв’язків до еталонних значень та обнулінню лічильнику кроків.

## 3.4 Віртуальний пацієнт

Взаємодія – ключовий компонент адаптивного симулятора клінічних сценаріїв, оскільки саме тут відбувається перетворення абстрактних станів графу на природну мову, структуровану відповідь.

Щоб забезпечити зручність та гнучкість навчального процесу під час використання симулятора було розроблено два типи моделей пацієнта: стандартна, яка базується на чітких збігах семантичних векторів, та розумна, що використовує можливості великої мовної моделі для імітації вільного діалогу.

### 3.4.1 Базовий віртуальний пацієнт

Для реалізації базового сценарію роботи системи необхідно визначити основний клас, який відповідає за зберігання ключових параметрів віртуального пацієнта та керування його початковим станом.

Саме цей клас формує основу для подальшої взаємодії з користувачем, оскільки забезпечує ініціалізацію необхідних структур даних і підготовку моделі до обробки діалогових запитів. Крім того, він містить елементи, що використовуються для розпізнавання типових запитань та вибору

відповідних мовних шаблонів, що є важливим для коректної роботи всієї системи.

Базова модель віртуального пацієнту представлена класом VirtualPatient, процес ініціалізації параметрів якого представлений на рисунку 3.17 [11].

```
class VirtualPatient:  4+ usages  Ihor Sikorskyi
    _DEMO_WORDS = {...}

    _TEMPLATES = {...}

    def __init__(self, df : {sample} , diagnoses, symptoms, symptom_names_dict : {get} ,  Ihor Sikorskyi
                 reveal_ratio: float = 0.3,
                 similarity_threshold: float = 0.90,
                 max_reveal_per_question: int = 3,
                 lang: str = 'uk'):
        self.diagnoses = diagnoses
        self.symptoms = symptoms
        self.sim_threshold = similarity_threshold
        self.max_reveal = max_reveal_per_question
        self.lang = lang
        self._T = self._TEMPLATES[lang if lang in self._TEMPLATES else 'uk']

        row = df.sample(1).iloc[0]
        self.true_diagnosis = row["PATHOLOGY"]
        self.age = int(row["AGE"]) if "AGE" in row.index and pd.notna(row["AGE"]) else None
        self.sex = str(row["SEX"]).strip() if "SEX" in row.index and pd.notna(row["SEX"]) else None
```

Рисунок 3.17 – Ініціалізація параметрів моделі

Клас VirtualPatient містить два статичні атрибути: _DEMO_WORDS та _TEMPLATES, що є словниками, що потрібні для коректної роботи моделі. Перший – словник ключових слів для розпізнавання питань про час та вік, а другий – словник мовних шаблонів.

Для розпізнавання того, який тип питань було задано пацієнту та формуванню відповіді в класі є вже згадані словники. Перший з них, це множина ключових слів _DEMO_WORDS.

Цей словник містить терміни, що пов’язані з віком та статтю пацієнта. Слова представлені двома мовами – англійська та українська.

Використовується в одному із методів моделі пацієнта для визначення, чи стосується питання відповідної категорії пацієнта.

Другий словника є шаблоном ключових слів для двох мов. Цей словник містить готові рядки відповідей для кожної із підтримуваних мов.

Шаблони охоплюються більшість можливих сценаріїв відповіді, від підтвердження вже обговорених симптомів, до розкриття нових, заперечення на питання та результати діагностик.

Також, частина рядків є статичним текстом, інша – лямбда-функції з аргументами для підстановки конкретних значень. Конструктор класу приймає таблицю даних пацієнтів, списки діагнозів та симптомів, словник їх назв та чотири параметри для налаштування моделі.

Параметр reveal_ratio визначає те, скільки симптомів будуть розкриті при початковій скарзі пацієнта, а similarity_threshold встановлює мінімальний поріг косинусної подібності для розкриття прихованих симптомів. Інший параметр, max_reveal_per_question, обмежує те, скільки симптомів модель може розкрити користувачу за відповідь на одне питання.

Шаблон мови обирається при ініціалізації моделі, де за замовчуванням встановлено українську, якщо шаблон не задано або не підтримується.

З таблиці даних пацієнтів випадковим чином обирається один рядок, що забезпечує унікальність кожної сесії. Вік та стать беруться з перевіркою на присутність поля та відсутність пропущених значень, якщо поле відсутнє або порожнє, значення встановлюється як None. Після ініціалізації основних параметрів пацієнта виконується формування його картини симптомів на основі поля EVIDENCES з обраного рядка датасету (рисунок 3.18) [11].

З кожного запису симптомів, за допомогою розбиття по роздільнику, береться значення базового коду симптому, а дублікати усуваються. Сам код симптому перетворюється на читабельну назву за допомогою словнику імен симптомів. Якщо код відсутній у словнику, то він буде використаний напряму, замість назви. Кожен симптом зберігається у новоствореному

словнику patient_symptoms, де їм присвоюється значення за замовчуванням False. Це означає, що симптом ще не розкритий.

```
evid_list = ast.literal_eval(row["EVIDENCES"])
base_codes = list({ev.split("_@_")[0] for ev in evid_list})

self.patient_symptoms = {}
for code in base_codes:
    name = symptom_names_dict.get(code, code)
    self.patient_symptoms[name] = False

all_s = list(self.patient_symptoms.keys())
n_reveal = max(2, int(len(all_s) * reveal_ratio))
for s in random.sample(all_s, n_reveal):
    self.patient_symptoms[s] = True

self.revealed_symptoms = [s for s, v in self.patient_symptoms.items() if v]
self.hidden_symptoms = [s for s, v in self.patient_symptoms.items() if not v]
```

Рисунок 3.18 – Формування картини симптомів

Далі визначається початкова кількість симптомів для початкового розкриття у скарзі лікарю. Для розкриття встановлена нижня межа на кількість початкових симптомів і гарантує їхню наявність і вичерпність для початку опитування пацієнта. Обрані симптоми позначаються у patient_symptoms міткою True.

На основі цього словника формуються два окремі списки – revealed_symptoms з вже відомими симптомами та hidden_symptoms з прихованими, що використовуються у подальшій логіці відповідей.

Для обробки демографічних питань було створено два допоміжні методи, що працюють разом (рисунок 3.19) [11]. Перший визначає чи є питання таким, що стосується віку або статі, а другий формує відповідь на нього.

Для визначення, чи є питання демографічним, використовується метод перетину множин. Весь текст питання переводиться в нижній регістр та розбивається на окремі слова. Отримана матриця порівнюється із словником _DEMO_WORDS операцією перетину. Якщо є хоча б одне

співпадіння, то питання стосується віку або статі пацієнта і є демографічним.

```
def _is_demographic(self, question: str) -> bool:  1 usage  Ihor Sikorskyi
    return bool(set(question.lower().split()) & self._DEMO_WORDS)

def _answer_demographic(self, question: str) -> tuple:  1 usage  Ihor Sikorskyi
    q = question.lower()
    T = self._T
    if self.age is not None and any(w in q for w in ("age", "old", "year", "вік", "років",
        return T['age'](self.age), []
    if self.sex is not None and any(
            w in q for w in ("male", "female", "man", "woman", "gender", "sex", "чоловік",
        sex_str = str(self.sex).upper() in ("M", "MALE")
        return (T['sex_m'] if sex_str else T['sex_f']), []
    return T['no_info'], []
```

Рисунок 3.19 – Методи демографічних питань

Другий допоміжний метод формує шаблонну відповідь на таке питання. Текст тут теж переводиться у нижній регістр і перевіряється, чи стосується воно віку, чи статі пацієнта за співпадінням із словами у все тому ж словнику.

За умови, що вік та стать пацієнта відомі, то вони підставляються у відповідні лямбда-функції шаблону після того, як були приведені до верхнього регістру. Якщо жодна з умов не виконалась або відповідне поле має значення None – повертається шаблон no_info.

Клас також містить допоміжний метод для отримання початкової скарги пацієнта. Метод формує початкову скаргу пацієнта для користувача. Якщо список розкритих симптомів не порожній, тоді вони всі об’єднуються у один рядок, розділені комами, та передаються до шаблону complaint.

Якщо список порожній, то повертається вже інший шаблон, checkup, що означає простий профілактичний огляд без чітких скарг. Ці шаблони будуть використані для формування початкової скарги.

Також модель пацієнта містить допоміжний метод для визначення поточного стану сесії. Цей метод повертає словник із повносю статистикою поточного пацієнта. Список містить істинний діагноз, вік та стать пацієнта,

загальну кількість симптомів, скільки з них розкрито, а скільки – ні і список назв розкритих симптомів.

Окрім цих допоміжних методів модель віртуального пацієнта має два основних, кожен відповідає за частину обстеження пацієнта. Перший із цих методів, це метод, що формує відповіді на питання та містить основну логіку обробки цього питання.

Метод реалізує послідовну систему фільтрів, де кожен наступний рівень активується лише якщо попередній не дав результату.

Метод починається з перевірки вхідних даних та визначення типу питання, після чого виконується семантичний пошук серед симптомів пацієнта.

На початку методу виконується перевірка даних. Якщо питання порожнє або містить лише пробіли, то як відповідь повертається шаблон nothing. Якщо питання стосується віку або статі пацієнта, то виконується один із допоміжних методів, який відповідає за цей тип питань (рисунок 3.19).

Якщо питання пройшло перевірку і не підпадає під жодний із їхніх критеріїв, то воно далі оброблюється поточним методом (рисунок 3.20) [11].

Якщо питання не стосується віку чи статі та не є порожнім, то воно перекладається на англійську, щоб спростити подальший семантичний пошуку, та передається, закодованим у вектор, енкодеру для нормалізації.

```
q_vec = normalize_vec(encode_text(translate_to_english(question)))

confirmed = []
for symp in self.revealed_symptoms:
    s_vec = normalize_vec(encode_text(symp))
    sim = float(cosine_similarity(q_vec.reshape(1, -1), s_vec.reshape(1, -1))[0][0])
    if sim >= 0.94:
        confirmed.append(symp)

if confirmed:
    return T['confirmed'](confirmed), []
```

Рисунок 3.20 – Метод відповіді: перевірка розкритих симптомів

Отриманий вектор порівнюється з векторами всіх вже розкритих симптомів. Якщо подібність хоча би одного з них перевищує поріг у 0.94, то симптом вважається вже підтвердженим та обговореним і метод повертає шаблон confirmed з породнім списком нових розкритих симптомів.

Такий високий поріг встановлений навмисно, щоб запобігти повторному підтвердженню симптому або, якщо поріг буде занизький, то не підтвердженню нового розкритого симптому.

Якщо питання не стосується вже відомих симптомів, метод переходить до пошуку серед прихованих (рисунок 3.21) [11].

```
candidates = []
for symp in list(self.hidden_symptoms):
    s_vec = normalize_vec(encode_text(symp))
    sim = float(cosine_similarity(q_vec.reshape(1, -1), s_vec.reshape(1, -1))[0][0])
    if sim >= self.sim_threshold:
        candidates.append((symp, sim))

candidates.sort(key=lambda x :{__getitem__} : x[1], reverse=True)
newly_revealed = []
```

Рисунок 3.21 – Метод відповіді: обробка прихованих симптомів

Для кожного симптому зі списку hidden_symptoms обчислюється косинусна подібність з вектором питання. Симптоми, що подолали поріг sim_threshold, додаються до списку кандидатів разом зі значенням подібності.

Після перебору всіх прихованих симптомів кандидати сортуються за спаданням подібності – це гарантує що при наявності кількох відповідних симптомів студент отримає найбільш релевантні. Важливо, що ітерація виконується по копії списку через list (self.hidden_symptoms) – це запобігає помилкам при можливій модифікації списку під час ітерації.

Варто зазначити, що поріг sim_threshold зі значенням за замовчування у 0.9 є свідомо таким високим. Це принципове архітектурне рішення – система має розкривати симптом лише тоді, коли питання семантично дуже

близьке до нього, а не при будь-якій віддаленій асоціації. Такий підхід змушує користувача формулювати конкретні та точні питання, що відповідає реальній клінічній практиці де лікар має чітко називати те, про що запитує.

Занижений поріг перетворив би систему на надто поблажливу – користувач міг би отримувати підтвердження симптомів через розмиті або випадково близькі формулювання, що знижувало б навчальну цінність взаємодії.

Відібрані кандидати розкриваються і повертаються студенту, при необхідності з перекладом на українську (рисунок 3.22) [11].

З відсортованого списку кандидатів беруться не більше, ніж визначена кількість симптомів за одним із заданих аргументів.

```
for symp, sim in candidates[:self.max_reveal]:
    self.patient_symptoms[symp] = True
    self.revealed_symptoms.append(symp)
    self.hidden_symptoms.remove(symp)
    newly_revealed.append((symp, sim))

if newly_revealed:
    if self.lang == 'uk':
        try:
            from deep_translator import GoogleTranslator
            translated = []
            for s, sim in newly_revealed:
                tr = GoogleTranslator(source='en', target='uk').translate(s)
                translated.append((tr if tr and tr.strip() else s, sim))
            newly_revealed = translated
        except Exception:
            pass
    return T['revealed']([s for s, _ in newly_revealed]), newly_revealed

return T['denied'], []
```

Рисунок 3.22 – Метод відповіді: розкриття симптомів та формування відповіді

Для кожного з них значення відкритості у patient_symptoms встановлюється як True, а симптом видаляється з hidden_symptoms і додається в revealed_symptoms та до newly_revealed разом з оцінкою подібності.

Якщо мова інтерфейсу українська – назви симптомів перекладаються через try/except, де у разі помилки перекладу залишається оригінальна англійська назва.

Якщо список newly_revealed не порожній, то повертається шаблон revealed зі списком розкритих симптомів та сам список з оцінками подібності для подальшого використання в PatientModel. Якщо жоден симптом не подолав поріг подібності, тоді повертається шаблон denied з порожнім списком, що означає відсутність відповідного симптому у пацієнта.

### 3.4.2 Інтеграція LLM за допомогою Ollama

Для того, щоб реалізувати одну із моделей пацієнта, а саме розумну модель, використовується LLM Gemma 3 12B [9], яка інтегрована за допомогою спеціального інструменту Ollama [21].

Ollama – це безкоштовний інструмент, що розробляється за принципом open-source і призначений для локального запуску великих мовних моделей, як Gemma, Llama [22] тощо за простим у реалізації підході REST API. Хоча Ollama має свій графічний інтерфейс, основна взаємодія відбувається через консоль.

Такий інструмент ідеально підходить у даному випадку, оскільки звернення до хмарних сервісів OpenAPI чи Anthropic було недоцільним у межах цього проєкту, а також більш складним у реалізації. Також, подібний підхід дозволяє зберегти кондиційність інформації, бо жодні дані не будуть передані до зовнішніх сервісів.

Хоча останній пункт не є принциповим у випадку синтетичних даних, але в більш великих проєктах, де застосовуються реальні медичні дані, з реальними прикладами та випадками, конфіденційність – критичний критерій.

LLM для інтеграції в проєкт, на початку, була не визначена і розглядалося два основних варіанти, вже згадані Gemma 3 [9] та Llama 3 [22].

На початку реалізації було обрано модель від компанії Meta – Llama [22] і вона непогано справлялася із імпровізацією діалогів в межах заданих параметрів, але виявила критичний недолік для цього проєкту в пізніших етапах розробки.

Ця мовна модель була майже нездатна на взаємодію будь-якою мовою, окрім англійської. Цей недолік і став причиною пізнішої заміни цієї моделі на Gemma 3 [9] від компанії Google. Ця модель значно краще розуміє природну мову і надає реалістичні текстові відповіді за заданими параметрами.

Щоб під’єднати обрану модель було створено клас VirtualPatientEngine (рисунок 3.23), який містив базові налаштування з’єднання із обраною LLM на локальному хості [11].

Клас ініціалізується з назвою моделі та адресою серверу Ollama, де за замовчуванням використовується одна із варіацій Gemma 3 [9], а саме Gemma 3 12B.

Перевага конкретно цієї моделі в тому, що вона споживає не надто багато ресурсів VRAM і придатна для локального розгортання без спеціального обладнання, але залишається достатньо потужною для того, щоб розуміти медичні терміни та контекст розмов.

За наявності ресурсів і потреби, вибір конкретної моделі може бути змінений при ініціалізації, що дозволяє адаптувати систему під різні умови.

Клас VirtualPatientEngine, окрім ініціалізації, має один метод – generate_response. Це метод приймає основний запит та опціональне

системне повідомлення. З них він формує payload для запиту на завдання згенерувати текст-відповідь.

```
class VirtualPatientEngine:  4+ usages  Ihor Sikorskyi
    def __init__(self, model_name : str ="gemma3:12b", host : str ="http://localhost:11434"):
        self.model_name = model_name
        self.url = f"{host}/api/generate"

    def generate_response(self, prompt, system_prompt=None):  2 usages (2 dynamic)  Ihor Sikorsky
        payload = {
            "model": self.model_name,
            "prompt": prompt,
            "stream": False,
            "options": {
                "temperature": 0.7,
                "num_predict": 150
            }
        }
        if system_prompt:
            payload["system"] = system_prompt

        try:
            response = requests.post(self.url, json=payload, timeout=30)
            response.raise_for_status()
            return response.json().get("response", "").strip()
        except Exception as e:
            return f"Помилка зв'язку з Gemma: {e}"
```

Рисунок 3.23 – Встановлення з’єднання з LLM

Системне повідомлення передаєтьс я окремим полем у payload, якщо воно було надане. Ці опціональні повідомлення дозволяють задати певні зміни в поведінку пацієнта з конкретним віком, статтю.

Параметр temperature забезпечує варіативність відповідей. Залежно від величини значення, відповіді можуть бути як майже одноманітними, так і непередбачуваними, що могло б призвести до відповідей, що не відповідають заданій ролі.

Параметр num_predict – це обмеження кількості токенів на відповідь моделі. Встановлене за замовчуванням значення дає змогу LLM відповісти у межах 1-2 речень, що запобігає генерації надмірної кількості тексту, що міг би не відповідати ролі, або містити забагато інформації, непотрібні пояснення.

Параметр stream, що за замовчуванням False вказує Ollama [21] повернути згенеровану відповідь одним повідомлення замість потокової передаці токенів. Це спрощує обробку результату. Далі згенерований payload очищується від зайвих пробілів.

У разі будь-якої помилки, як от відсутність доступу до мережі, перевищення часу очікування генерації більше, ніж 30 секунд або некоректної відповіді сервера – метод перехоплює виключення і повертає текстове повідомлення про помилку, що запобігає падінню системи при тимчасовій недоступності моделі.

### 3.4.3 Розумний віртуальний пацієнт

Після забезпечення стабільного підключення до обраної LLM, її інтеграція відбувається у моделі розумного віртуального пацієнта. Перед самим створенням класу, створюється три глобальні словники, що містять ключові слова, що вказують на питання не про симптоми.

Перший словник представляє набір слів, що вказують на питання про час. Наприклад, про те, як давно триває симптом або біль, коли почалося тощо.

Другий словник містить набір ключових слів для визначення питань про те, що пацієнт може вважати причиною своєї хвороби або травми. Містяться базові слова, як про місце, де вперше це проявилося, так і про те, що людина могла вживати.

Третій словник містить ключові слова для питань про самопочуття пацієнта, як те, наскільки сильний біль, як його можна оцінити за певною шкалою, чи посилюється він, чи навпаки, послаблюється час від часу.

Усі три словники разом формують повноцінне охоплення діагностичного діалогу: від збору анамнезу та можливих причин до суб'єктивної оцінки стану пацієнта. Такий підхід дозволяє системі

класифікувати довільне питання студента та обрати відповідну стратегію відповіді залежно від його типу.

Далі відбувається створення моделі SmartVirtualPatient (рисунок 3.24) головна особливість якої – інтеграція LLM для генерації природних відповідей [11].

```
class SmartVirtualPatient:  4+ usages  Ihor Sikorskyi
    def __init__(self, patient_row : {__getitem__, get} , G : {has_edge} , symptom_names : {get} , llm_connector, lang: str = 'en'):
        self.llm = llm_connector
        self.G = G
        self.symptom_names = symptom_names
        self.lang = lang

        self.true_diagnosis = patient_row["PATHOLOGY"]
        self.age = patient_row.get("AGE", "unknown")
        self.sex = "male" if str(patient_row.get("SEX")).upper() == "M" else "female"

        evid_list = ast.literal_eval(patient_row["EVIDENCES"])
        self.actual_symptoms: dict[str, dict] = {}
        for ev in evid_list:
            code = ev.split("_@_")[0]
            name = symptom_names.get(code, code)
            weight = (
                G[self.true_diagnosis][name]["weight"]
                if G.has_edge(self.true_diagnosis, name)
                else 0.1
            )
            self.actual_symptoms[name] = {"weight": weight, "revealed": False}

        self.revealed_history: list[str] = []
```

Рисунок 3.24 – Ініціалізація моделі SmartVirtualPatient

Конструктор класу приймає шість аргументів. Першим є рядок датасету patient_row, з якого витягуються істинний діагноз, вік та стать пацієнта.

На відміну від VirtualPatient, де вік перевіряється через pd.notna, тут використовується простіший метод get з запасним значенням «unknown».

Це пов'язано з тим, що вік у SmartVirtualPatient використовується лише для формування системного повідомлення до мовної моделі, а не для числових розрахунків. Стать нормалізується до рядків «male» або «female» для підстановки у персону пацієнта.

Другим аргументом є граф G зі зваженими зв'язками між діагнозами та симптомами. Третім – словник symptom_names для перетворення кодів

симптомів на читабельні назви, а четвертим є llm_connector, тобто об'єкт VirtualPatientEngine для зв'язку з мовною моделлю. П’ятим аргументом виступає обрана користувачем мова.

Далі формується список симптомів на основі поля EVIDENCES через ast.literal_eval. Кожен запис розбивається по спеціальному роздільнику для отримання базового коду симптому, який перетворюється на назву через словник.

Для кожного симптому з графу витягується вага ребра між істинним діагнозом та симптомом – вона відображає силу клінічного зв'язку і використовується при формуванні контексту запитів. Якщо ребро у графі відсутнє, вага встановлюється у мінімальне значення 0.1.

Кожен симптом зберігається у словнику actual_symptoms з вагою та позначкою revealed: False, подібно до базової моделі VirtualPatient.

Окремий список revealed_history зберігає порядок розкриття симптомів і використовується для формування контексту запитів на наступних кроках діалогу.

Для коректної обробки питань студента та генерації відповідей клас використовує допоміжні методи.

Один з цих методів, _classify_question визначає тип питання на основі ключових слів з глобальних словників. Питання переводиться у нижній регістр і послідовно перевіряється на належність до кожної категорії – темпоральної, інтенсивності та контекстної.

Перевірки виконуються саме в такому порядку, оскільки категорії є взаємовиключними і перша відповідність одразу повертає результат без подальших перевірок. Такий підхід є обчислювально ефективним – замість семантичного кодування через BERT, що потребує значних обчислювальних ресурсів, класифікація виконується через просте порівняння рядків.

Це виправдано тому, що темпоральні, контекстні та питання про інтенсивність мають достатньо чіткий і передбачуваний словниковий склад.

Якщо жодне ключове слово не знайдено – питання вважається симптомним, що є найпоширенішим типом у діагностичному діалозі і обробляється через семантичний пошук.

Для того, щоб знайти симптоми із actual_symptoms, які є найбільш семантично подібним до вектору питання студента, створено допоміжний метод, що представлено на рисунку 3.25 [11].

```
def _best_symptom_match(self, q_vec :{reshape} ) -> tuple[str, float]:  1 usage
    best_sim, best_name = -1.0, ""
    for s_name in self.actual_symptoms:
        s_vec = normalize_vec(encode_text(s_name))
        sim = float(
            cosine_similarity(q_vec.reshape(1, -1), s_vec.reshape(1, -1))[0][0]
        )
        if sim > best_sim:
            best_sim, best_name = sim, s_name
    return best_name, best_sim
```

Рисунок 3.25 – Метод перебору симптомів

Метод перевіяє всі симптоми пацієнта по черзі, кодує кожен через BERT та обчислює косинусну подібність з переданим вектором питання. Початкове значення best_sim = -1.0 гарантує, що будь-який реальний результат подібності його перевищить, навіть якщо всі симптоми семантично далекі від питання. Після перебору повертається пара із назви найближчого симптому та значення подібності.

Результат цього методу використовується в answer_question для двох цілей: визначення чи є питання симптомним через порівняння best_sim з порогом SYMPTOM_THRESH, та ідентифікації конкретного симптому для підтвердження або заперечення у промпті до мовної моделі.

Варто зазначити що на відміну від VirtualPatient, який шукає серед прихованих і розкритих симптомів окремо, цей метод завжди шукає серед усіх симптомів actual_symptoms – це дозволяє мовній моделі коректно

відповідати на повторні питання про вже розкриті симптоми, підтверджуючи їх з природною варіативністю формулювань замість шаблонного повідомлення про повторне запитання.

SYMPTOM_THRESH є глобальною константою, що визначає мінімальну косинусну подібність для того щоб питання вважалося симптомним. Значення встановлено у 0.8 і є нижчим за подібне у базові варіації віртуального пацієнта, оскільки тут ним визначається, чи передавати симптом у запит як цільовий факт для мовної моделі – тому він м'якший. Сама мовна модель потім вирішує як відповісти, спираючись на контекст запиту, а не на жорстку порогову логіку.

Наступний допоміжний метод формує системне повідомлення для мовної моделі, що встановлює базову персону пацієнта. Повідомлення містить вік та стать пацієнта, мовне обмеження та вимогу відповідати коротко в 1-2 речення і при цьому залишаючись у ролі пацієнта. Мовне обмеження формулюється жорстко – при українському інтерфейсі модель отримує пряму заборону використовувати англійські слова, що є важливим оскільки мовні моделі схильні переходити на англійську при обробці медичної термінології.

Допоміжний метод мовних правил повертає коротке мовне правило для підстановки безпосередньо в тіло промпту.

Дублювання мовної вимоги з попереднього методу є навмисним, оскільки було перевірено, що мовні моделі надійніше дотримуються мовних обмежень тоді, коли вони присутні як у системному повідомленні, так і в самому промпті одночасно.

Також, подібно до базового класу віртуального пацієнту, присутній допоміжний метод, який відповідає за переклад назв симптомів на українську мову, якщо така було обрана. Якщо переклад видав помилку, то назва симптома повертається оригінальною мовою.

Зображений на рисунку 3.26 метод є центральним для роботи моделі віртуального пацієнта і реалізує багаторівневу логіку обробки питань студента з генерацією природніх відповідей через мовну модель [11].

```
def answer_question(self, question: str) -> tuple[str, list]:  4 usages (4 dynamic)  Ihor Sikorskyi
    from sklearn.metrics.pairwise import cosine_similarity

    q_en = translate_to_english(question)
    q_vec = normalize_vec(encode_text(q_en))

    target_symptom, best_sim = self._best_symptom_match(q_vec)
    q_type = self._classify_question(q_en)

    is_symptom_q = (best_sim >= SYMPTOM_THRESH) and (q_type == "symptom")

    system_msg = self._system_msg()

    # — ФІЛЬТРАЦІЯ КОНТЕКСТУ —
    all_known = [s for s in self.actual_symptoms]
    filtered_symps = []

    q_vec_reshaped = q_vec.reshape(1, -1)
    for s_name in all_known:
        s_vec = normalize_vec(encode_text(s_name)).reshape(1, -1)
        sim = float(cosine_similarity(q_vec_reshaped, s_vec)[0][0])

        if sim > 0.45:
            filtered_symps.append(s_name)

    context_symps = filtered_symps if filtered_symps else all_known[:2]
    symps_str = ", ".join(context_symps)
    revealed_str = ", ".join(self.revealed_history) if self.revealed_history else "none yet"
```

Рисунок 3.26 – Початок методу відповіді: підготовка контексту для запиту

На початку використовується один із методі класу BioBERT, а саме переклад запиту на англійську, що потрібно для уніфікації подальшої обробки незалежно від мови інтерфейсу, після чого кодується у вектор та нормалізується все тим же енкодером.

Паралельно виконуються два незалежних аналізи – пошук найближчого симптому через згаданий вище допоміжний метод перебору симптомів (рисунок 3.25) та класифікація типу питання через допоміжний

метод класифікації. На основі їх результатів формується булева змінна is_symptom_q, що визначає чи є питання симптомним одночасно за двома критеріями: подібність має перевищувати SYMPTOM_THRESH і тип питання має бути «symptom».

Така подвійна умова запобігає хибному розпізнаванню, як, наприклад, темпоральне питання про тривалість симптому може мати високу подібність до самого симптому, але не є прямим запитом про його наявність.

Після цього виконується фільтрація контексту для промпту. З усіх симптомів пацієнта відбираються лише ті, чия косинусна подібність з вектором питання перевищує поріг 0.45, що значно нижчий за SYMPTOM_THRESH, що дозволяє включати в контекст семантично пов'язані симптоми навіть якщо вони не є прямою відповіддю на питання.

Це збагачує контекст запиту і дозволяє мовній моделі генерувати більш природні та клінічно узгоджені відповіді. Якщо жоден симптом не подолав поріг фільтрації, тоді до контексту передаються перші два симптоми зі списку як мінімальний запасний контекст. Додатково формується рядок revealed_str з історією вже розкритих симптомів, який дозволяє мовній моделі уникати суперечностей з попередніми відповідями протягом діалогу.

Центральною частиною методу є система з п'яти гілок формування запиту, що визначається типом питання та результатами семантичного пошуку.

Гілка 1 обробляє темпоральні питання про тривалість та початок симптомів. Запит явно забороняє моделі переходити до інших симптомів і вимагає відповіді виключно про час, що є принципово важливо, оскільки без такого обмеження модель схильна розширювати контекст відповіді до загального опису самопочуття. До запиту передається як відфільтрований контекст релевантних симптомів, так і історія вже розкритих, щоб модель не суперечила попереднім відповідям про той самий симптом.

Гілка 2 обробляє питання про інтенсивність та прогресію симптомів. Запит є більш лаконічним порівняно з іншими гілками, оскільки лише релевантний контекст та вимога відповідати про силу або динаміку. Це виправдано тим, що питання про інтенсивність зазвичай конкретні і не потребують додаткових обмежень для утримання моделі в ролі.

Гілка 3 обробляє контекстні питання про анамнез, спосіб життя та супутні фактори. На відміну від інших гілок, тут виконується повторна фільтрація симптомів з тим самим порогом 0.45, який необхідний для контекстних питань, бо вони можуть стосуватися зовсім іншого аспекту ніж симптоми, і релевантний контекст може відрізнятися від загально відфільтрованого на початку методу. Якщо фільтрація не дала результатів, використовується revealed_history як контекст – студент вже знає ці симптоми, тому їх згадування не розкриває нової інформації.

До запиту передається істинний діагноз для формування реалістичного анамнезу, але з явною забороною згадувати непов'язані стани, бо без цього модель схильна додавати типові для діагнозу супутні умови, які можуть не бути присутніми у конкретного пацієнта.

Гілка 4 обробляє прямі питання про наявність симптому, що є найважливішою гілкою з точки зору медичної точності. Одна з умов перевіряє чи симптом вже був розкритий раніше або чи подібність перевищує поріг, а потім, в обох випадках, факт наявності симптому передається до запиту як абсолютна істина. Це гарантує що мовна модель не може «вигадати» відсутність симптому через особливості генерації. Аналогічно при відсутності симптому. Тоді модель отримує жорсткий факт заперечення.

Гілка 5 є запасною і спрацьовує тоді, коли питання не підпадає під жодну з попередніх категорій. Тобто, коли подібність до найближчого симптому нижча за SYMPTOM_THRESH або питання не класифіковане. Запит передає лише загальний симптомний контекст без додаткових

обмежень, дозволяючи моделі відповідати природно на нестандартні питання.

Завершальна частина методу answer_question відповідає за оновлення внутрішнього стану моделі після кожної взаємодії з користувачем та формування результату, що повертається зовнішнім компонентам системи. Відповідний фрагмент представлено на рисунку 3.27 [11].

```
newly_revealed: list[tuple[str, float]] = []
if is_symptom_q and best_sim >= SYMPTOM_THRESH:
    self.actual_symptoms[target_symptom]["revealed"] = True
    if target_symptom not in self.revealed_history:
        self.revealed_history.append(target_symptom)
        translated = self._translate_symptom(target_symptom)
        newly_revealed.append((translated, best_sim))

return response, newly_revealed
```

Рисунок 3.27 – Оновлення стану виявлених симптомів

На початку цього блоку ініціалізується порожній типізований список. Кожен елемент цього списку є парою, що містить локалізовану назву симптому та відповідний коефіцієнт косинусної подібності. Цей список виступає носієм інформації між внутрішньою логікою моделі та зовнішнім інтерфейсом системи, дозволяючи відстежувати, які саме симптоми були встановлені в ході поточного діалогового кроку.

Блок оновлення активується лише за одночасного виконання двох умов: питання повинно бути класифіковане як симптомне і рівень семантичної подібності запиту до найближчого симптому має перевищувати встановлений поріг SYMPTOM_THRESH. Така подвійна перевірка запобігає хибному розкриттю симптомів у разі темпоральних, контекстних або оціночних запитань, навіть якщо їхній текст семантично близький до певного медичного терміну.

У разі виконання умови, у словнику actual_symptoms для відповідного цільового симптому встановлюється «revealed»: True, що фіксує факт його виявлення у поточній сесії.

Далі виконується перевірка на відсутність симптому в списку revealed_history, що унеможливлює дублювання записів за умови повторного звернення до вже встановленого симптому.

Якщо симптом виявлено вперше, його назва передається до допоміжного методу із перекладу назв симптомів. Перекладена назва разом із значенням подібності додається до списку newly_revealed.

Сам метод answer_question повертає два значення: текстову відповідь пацієнта response, сформовану мовною моделлю, та список newly_revealed. У разі відсутності нових виявлень цей список є порожнім, що дозволяє зовнішнім компонентам розрізняти інформативні та нейтральні кроки діалогу без додаткових перевірок.

Також, подібно до базової моделі віртуального пацієнту, окрім методу відповіді на питання користувача, розумна варіація містить метод, що відповідає за тестування пацієнта.

Метод perform_test реалізує логіку імітації діагностичних процедур і є спільним для обох реалізацій віртуального пацієнта. Його сигнатура приймає три аргументи: текстовий запит користувача test_query, список доступних процедур procedures_list та словник diagnosis_tests, що відображає відповідність між діагнозами та переліком характерних для них обстежень. Усередині методу запит користувача векторизується через BioBERT та порівнюється з векторними представленнями доступних процедур за допомогою косинусної подібності.

Це дозволяє системі коректно інтерпретувати довільно сформульований запит студента та зіставити його з переліком підтримуваних діагностичних процедур.

Якщо знайдено процедуру з достатнім рівнем подібності, метод перевіряє її наявність у словнику diagnosis_tests для поточного діагнозу

пацієнта. За результатом перевірки формується текстова відповідь, що імітує реакцію пацієнта на проведену процедуру – або із зазначенням результату обстеження, або із повідомленням про його відсутність у переліку призначених процедур.

Саме цей методі і зображений на рисунку 3.28 [11].

```
def perform_test(self, test_query: str, procedures_list: list, diagnosis_tests: dict) -> str:
    if not test_query:
        return "Please specify which test you want to perform."

    q_vec = normalize_vec(encode_text(translate_to_english(test_query)))
    best_proc, max_sim = None, -1.0

    for proc in procedures_list:
        p_vec = normalize_vec(encode_text(proc))
        sim = float(
            cosine_similarity(q_vec.reshape(1, -1), p_vec.reshape(1, -1))[0][0]
        )
        if sim > max_sim:
            max_sim, best_proc = sim, proc

    if max_sim < 0.80:
        return f"I'm not sure what '{test_query}' refers to."

    relevant_tests = diagnosis_tests.get(self.true_diagnosis, default: [])
    if best_proc in relevant_tests:
        return (
            f"Results for '{best_proc}': "
            f"Abnormal findings consistent with {self.true_diagnosis}."
        )
    return f"Results for '{best_proc}': Findings are within normal range."
```

Рисунок 3.28 – Метод проведення тестування пацієнта

На початку виконується перевірка на порожній запит. У разі відсутності тексту метод негайно повертає рядок із повідомленням про необхідність уточнення процедури, не виконуючи жодних подальших обчислень.

Якщо запит присутній, він передається через конвеєр семантичного кодування: спочатку перекладається на англійську мову, після чого векторизується за допомогою encode_text та нормалізується функцією

normalize_vec, все за допомогою енкодеру. Отриманий вектор представляє семантичний зміст запиту у просторі BioBERT-представлень.

Далі виконується пошук найближчої процедури з переліку procedures_list. Для кожної процедури обчислюється її векторне представлення та косинусна подібність із вектором запиту. За результатами перебору фіксується процедура best_proc із максимальним значенням подібності max_sim. Пошук є вичерпним, тобто розглядаються всі доступні процедури без попередньої фільтрації.

Після завершення пошуку значення порівнюється із фіксованим порогом. Якщо жодна процедура не набрала достатньої семантичної близькості, метод повертає повідомлення про неможливість інтерпретації запиту. Це захищає систему від хибних результатів у разі введення нерелевантного або надто розмитого тексту.

Якщо процедуру ідентифіковано успішно, виконується звірення з еталонним словником diagnosis_tests. З порожнім списком за замовчуванням отримується перелік процедур, що є діагностично значущими для істинного діагнозу поточного пацієнта. Якщо best_proc входить до цього переліку, метод повертає рядок із повідомленням про патологічні зміни, характерні для відповідного діагнозу. В протилежному випадку результат визначається як норма.

Така логіка дозволяє системі імітувати клінічно достовірну поведінку: специфічні для захворювання обстеження повертають аномальні результати, тоді як нерелевантні процедури – нормальні показники, що відтворює реальну практику диференційної діагностики.

## 3.5 Система оцінювання

Невід'ємною частиною адаптивного тренажеру є система оцінювання, що забезпечує об'єктивну верифікацію дій студента протягом діалогу. На відміну від традиційних підходів де оцінка виставляється лише за фінальний

діагноз, реалізована система аналізує кожен крок взаємодії одразу за трьома незалежними критеріями: інформаційною цінністю питання, відповідністю клінічним протоколам та комунікативною якістю звернення до пацієнта.

Це дозволяє отримати багатовимірний профіль компетенцій студента, що відображає не лише правильність кінцевого рішення, але й якість діагностичного мислення на кожному етапі обстеження.

На рисунку 3.29 представлені три функції для базового математичного оцінювання [11].

```
def compute_probabilities(activations: np.ndarray, tau: float = 1.0) -> np.ndarray:
    return softmax(activations / tau)

def shannon_entropy(probs: np.ndarray) -> float:  16+ usages  Ihor Sikorskyi
    probs = np.clip(probs, a_min: 1e-10, a_max: 1.0)
    return float(-np.sum(probs * np.log2(probs)))

def get_tau(step: int, tau_start: float = 1.2,  12+ usages  Ihor Sikorskyi
            tau_end: float = 0.35, n_steps: int = 10) -> float:
    alpha = (tau_start - tau_end) / n_steps
    return max(tau_end, tau_start - alpha * step)
```

Рисунок 3.29 – Базова логіка оцінювання

Перша з них перетворює вектор діагностичних активацій на ймовірнісний розподіл через softmax [6] з температурним параметром tau. Ділення активацій на tau перед подачею до softmax [6] регулює «впевненість» розподілу – при малих значеннях tau розподіл загострюється і домінуючий діагноз отримує значно більшу ймовірність відносно інших, при великих – розподіл розмивається і всі гіпотези отримують близькі ймовірності незалежно від різниці в активаціях. Це безпосередня реалізація виразу (2.7).

Функція використовує реалізацію softmax з бібліотеки scipy.special, що є чисельно стабільнішою за наївну реалізацію через вбудоване віднімання максимуму перед експонентою.

Другий метод, shannon_entropy, обчислює ентропію Шеннона [7] переданого ймовірнісного розподілу відповідно до формули (2.6). Попереднє обмеження значень з нижньою межею 1e-10 та верхньою 1.0 є обов'язковим захисним кроком – логарифм нуля є математично невизначеним і призвів би до нескінченності у результаті, а значення вище одиниці є некоректними для ймовірнісного розподілу.

Результат повертається як скалярне значення типу float, що спрощує подальше використання в арифметичних операціях при розрахунку інформаційного прибутку як різниці двох послідовних ентропій. Чим більш рівномірній розподіл – тим вища ентропія, чим більше модель впевнена в одному діагнозі – тим нижча. Саме це робить ентропію природною мірою діагностичної невизначеності.

Останній із цих трьох методів реалізує лінійне спадання температурного параметру протягом сесії. Швидкість спадання обчислюється як відношення різниці між початковим і кінцевим значеннями до кількості кроків, після чого на кожному кроці від початкового значення віднімається добуток alpha на номер кроку.

На початку діалогу tau_start дорівнює 1.2, що робить розподіл достатньо розмитим, щоб відображати реальну початкову невизначеність і невеликі зміни активацій дають помірний інформаційний прибуток.

З кожним кроком tau лінійно зменшується до мінімального значення tau_end, що загострює розподіл і підвищуючи чутливість ентропії до змін у діагностичних активаціях, а це, в свою чергу, означає що пізні підтвердження симптомів дають більший вимірюваний інформаційний прибуток ніж ранні, що відповідає реальній клінічній логіці де кожне додаткове підтвердження на фоні вже сформованої гіпотези є цінніщим. Функція max гарантує, що tau ніколи не опуститься нижче мінімального порогу навіть при дуже тривалому діалозі.

Цей підхід дозволяє ефективно моделювати зміну невизначеності протягом сесії, що є критично важливим для адекватної оцінки

інформаційного внеску кожного нового спостереження. Завдяки цьому механізму система адаптується до поступового накопичення знань і відповідно коригує вагу нових даних.

Описаний математичний апарат базових функцій оцінювання є основою для повноцінного рушія оцінювання, реалізованого у класі EvaluationEngine (рисунок 3.30) [11].

```
class EvaluationEngine:  7+ usages  Ihor Sikorskyi
    DEFAULT_PROTOCOLS = {...}

    def __init__(self, protocols: dict = None,  Ihor Sikorskyi
                 w1: float = 0.6, w2: float = 0.4,
                 w_side: float = 1.5, penalty_threshold: float = 0.82):
        self.w1 = w1
        self.w2 = w2
        self.w_side = w_side
        self.penalty_threshold = penalty_threshold

        raw_protocols = protocols if protocols is not None else self.DEFAULT_PROTOCOLS
        self.eval_data = self._vectorize_protocols(raw_protocols)
        print(f"EvaluationEngine: векторизовано протоколи для {len(self.eval_data)} діагнозів")
```

Рисунок 3.30 – Ініціалізація EvaluationEngine

Клас містить статичний атрибут DEFAULT_PROTOCOLS – словник еталонних протоколів для двох діагнозів, що використовується за замовчуванням якщо зовнішні протоколи не передані.

Конструктор приймає чотири параметри. w1 та w2, які є ваговими коефіцієнтами інформаційного прибутку та семантичної відповідності протоколу у формулі фінального балу, за замовчуванням 0.6 та 0.4 відповідно, бо така пропорція відображає пріоритет діагностичного мислення над точністю окремих дій.

Параметр w_side – ваговий коефіцієнт критичності порушення протоколу зі значенням 1.5, що свідомо перевищує суму w1 та w2, таким чином гарантуючи, що одна критична помилка може повністю нівелювати

позитивний бал кроку. Це є важливим, бо помилки часто можуть стати причиною погіршення стану пацієнта.

Параметр penalty_threshold встановлює мінімальну косинусну подібність до red flag для нарахування штрафу, а значення за замовчуванням в 0.82 відсікає випадкові семантичні збіги, реагуючи лише на дії що явно відповідають небезпечним втручанням.

Після збереження параметрів протоколи передаються до _vectorize_protocols, що кодує всі еталонні дії та red flags через BERT і зберігає у eval_data для подальшого використання при оцінюванні кожного кроку.

Згаданий red flag є одним із розроблених списків у клінічних протоколах еталонної поведінки користувача стосовно пацієнта та діагнозів.

Для кожного діагнозу протокол містить два списки. gold_standard – перелік правильних терапевтичних дій, семантична близькість до яких формує позитивний бал кроку та red_flags, що є переліком небезпечних або протипоказаних втручань, близькість до яких тягне штраф.

Клас також містить три статичні списки регулярних виразів для оцінювання комунікативної якості звернень студента до пацієнта.

Перший з них, _POLITE_PATTERNS, містить шаблони для розпізнавання ввічливих форм звернення, як от прохальні конструкції, подяки, вибачення та вирази розуміння.

Наявність таких шаблонів у тексті підвищує складову ввічливості у комунікативному балі.

Наступний, _RUDE_PATTERNS, містить шаблони для виявлення грубих або наказових форм звернення: образливі слова, наказовий тон, надмірна наполегливість.

Окрім шаблонів фраз та слів список включає структурні ознаки грубості – два або більше знаки оклику поспіль та послідовності з п'яти або більше великих літер, що є типовими індикаторами агресивного тону в письмовому тексті.

Спрацювання цих шаблонів знижує комунікативний бал та фіксується у списку rude_flags для відображення у фінальному звіті.

Останній список, _EMPATHY_PATTERNS, містить шаблони для розпізнавання емпатійних висловлювань, як, наприклад, вирази підтримки, готовності допомогти та активного слухання.

Такі форми звернення є важливою складовою клінічної комунікації і підвищують відповідну складову балу поряд з ввічливістю.

Центральним методом класу є calculate_step_metrics, що реалізує вирази (2.9) та (2.10) і представлений на рисунку 3.31 [11].

```
def calculate_step_metrics(self, action_text: str,  7 usages (3 dynamic)  Ihor Sikorskyi
                           true_diagnosis: str,
                           delta_h: float) -> dict:
    v_at = encode_text(action_text)
    protocol = self.eval_data.get(true_diagnosis,  default: {"gold_standard": [], "red_flags": []})

    r_score = 0.0
    if protocol["gold_standard"]:
        r_score = max(
            cosine_similarity(v_at.reshape(1, -1), v.reshape(1, -1))[0][0]
            for v in protocol["gold_standard"]
        )

    penalty = 0.0
    if protocol["red_flags"]:
        max_p = max(
            cosine_similarity(v_at.reshape(1, -1), v.reshape(1, -1))[0][0]
            for v in protocol["red_flags"]
        )
        if max_p > self.penalty_threshold:
            penalty = self.w_side * max_p

    comm = self.score_communication(action_text)
    comm_bonus = 0.05 * comm["comm_score"] - 0.10 * comm["rudeness"]
    step_score = self.w1 * delta_h + self.w2 * r_score - penalty + comm_bonus
```

Рисунок 3.31 – Розрахунок метрик оцінювання

Слід зазначити, що вирази (2.9) та (2.10) зазнали певних змін під час реалізації запланованого проєкту і були ускладнені відповідно до нових критеріїв оцінювання.

Перепрацьована версія виразу (2.9) виглядає наступним чином:

$$Score_t = w_1 * \Delta H + w_2 * R_t - Penalty(A_t) + CommBonus(A_t) \quad (3.2)$$

де $Score_t$ – підсумковий бал поточного кроку t, що відображає загальну якість дії студента за всіма критеріями одночасно;

$w_1$ – ваговий коефіцієнт інформаційного прибутку. Він є домінуючим у формулі, оскільки діагностичне мислення вважається пріоритетнішим за точність окремих дій;

$\Delta H$ – інформаційний прибуток кроку, розрахований як різниця ентропії до і після дії студента. Відображає наскільки питання або дія звузили діагностичну невизначеність;

$w_2$ – ваговий коефіцієнт семантичної відповідності протоколу;

$R_t$ – максимальна косинусна подібність між вектором дії студента та векторами еталонних дій gold_standard для істинного діагнозу. Відображає наскільки дія студента відповідає правильному клінічному протоколу;

$Penalty(A_t)$ – штраф за наближення до небезпечних втручань згідно з оновленим виразом (2.10), що буде приведений наступним. Нараховується лише якщо подібність до будь-якого red flag перевищила встановлений поріг, і масштабується за коефіцієнтом w_side.

$CommBonus(A_t)$ – комунікативний бонус, що розраховується як невеликий позитивний внесок за ввічливість та емпатію з подвійним штрафом за грубість відносно позитивної складової.

Для розрахунку штрафу планувалося використати вираз (2.10), але в ході програмної реалізації підхід до його розрахунку змінився, а сам він став пороговим, а не безперервним і нараховується лише якщо подібність до будь-якого red flag перевищує поріг penalty_threshold, а не безперервно як у оригінальній ідеї.

Оновлена версія виразу має наступний вигляд:

$$Penalty(A_t) = \begin{cases} w_{side} * \ max_k \left(\cos\left(Embed(A_t), v_{red_{flagsk}}\right)\right), \\ 0, \end{cases} \quad (3.3)$$

де $w_{side}$ – ваговий коефіцієнт критичності небезпечного втручання, а вираз $max_k \left(\cos\left(Embed(A_t), v_{red_{flagsk}}\right)\right)$ є представленням максимальної косинусної подібності між вектором дії студента та векторами всіх red flags протоколу для істинного діагнозу.

Значення $v_{red_{flagsk}}$ – це вектор k-го небезпечного втручання з протоколу, закодований через BERT при ініціалізації у _vectorize_protocols.

Логіка оцінювання комунікації реалізована у методі, що представлений на рисунку 3.32 [11].

```
def score_communication(self, text: str) -> dict:  1 usage  Ihor Sikorskyi
    low = text.lower()
    pol_hits   = sum(1 for p in self._POLITE_PATTERNS  if re.search(p, low))
    emp_hits   = sum(1 for p in self._EMPATHY_PATTERNS if re.search(p, low))
    rude_flags = [p for p in self._RUDE_PATTERNS       if re.search(p, low)]
    rude_hits  = len(rude_flags)
    words      = max(len(text.split()), 1)
    politeness = min(pol_hits / max(words / 8, 1), 1.0)
    empathy    = min(emp_hits / max(words / 8, 1), 1.0)
    rudeness   = min(rude_hits / 2, 1.0)
    raw        = (politeness + empathy) / 2
    comm_score = max(0.0, raw * (1 - rudeness) - rudeness * 0.15)
    if pol_hits == 0 and emp_hits == 0 and rude_hits == 0:
        comm_score = 0.5
    return {
        "politeness": round(politeness,  3),
        "empathy":    round(empathy,     3),
        "rudeness":   round(rudeness,    3),
        "comm_score": round(comm_score,  3),
        "flags":      rude_flags,
    }
```

Рисунок 3.32 – Метод оцінки комунікації

Метод починається з підрахунку збігів з кожним із трьох списків шаблонів у тексті зверненого до нижнього регістру. Для ввічливості та

емпатії підраховується кількість шаблонів що знайшли збіг, для грубості – зберігається сам список спрацьованих шаблонів для подальшого відображення у звіті.

Отримані лічильники нормалізуються відносно довжини тексту через вираз max (words / 8, 1), який забезпечує те, що короткі речення не отримують несправедливо високих балів через одне влучне слово, а довгі не штрафуються за відносно меншу щільність ввічливих виразів.

Значення 8 є емпіричним і відповідає приблизній кількості слів на одне очікуване ввічливе або емпатійне вираження у природній клінічній репліці. politeness та empathy обмежуються зверху одиницею через min (..., 1.0), що гарантує що жодна складова не вийде за межі нормованого діапазону. rudeness нормалізується діленням на 2. Цим передбачається, що два грубих патерни вже є максимальним рівнем грубості незалежно від їх фактичної кількості.

Проміжне значення raw є середнім між ввічливістю та емпатією і відображає позитивну комунікативну складову без урахування грубості. Фінальний комунікативний бал формується у два етапи – спочатку позитивна складова пригнічується пропорційно рівню грубості, після чого додатково віднімається прямий адитивний штраф що забезпечує зниження балу навіть при повній відсутності позитивних сигналів. Невід'ємність результату гарантується обмеженням знизу нулем.

Окремо обробляється нейтральний випадок – якщо жоден патерн жодного типу не спрацював, комунікативний бал встановлюється у значення 0.5, що відповідає формально коректному але беземоційному зверненню яке не заслуговує ні нагороди ні покарання.

## 3.6 Адаптивний селектор питань

Клас AdaptiveQuestionSelector реалізує логіку адаптивного підбору наступного питання або діагностичної процедури, що задається

віртуальному пацієнту в процесі симуляції клінічного опитування. Його роль у системі – виступати інтелектуальним радником, який на кожному кроці сесії визначає, яке питання чи тест принесе найбільше діагностичної користі.

Клас містить статичний атрибут _FALLBACK_QUESTIONS – список із десяти базових клінічних питань, що охоплюють найпоширеніші симптоми, зокрема біль у грудях, задишку, кашель та лихоманку. Цей список використовується як резервний банк питань у випадку, якщо зовнішній набір кандидатів не був переданий при ініціалізації.

Сигнатуру та повну реалізацію методу ініціалізації наведено на рисунку 3.33 [11].

```
def __init__(self, patient_model, tau_fn, procedures=None, candidate_questions=None,
             cache_dir: str = _CACHE_DIR):
    self.model = patient_model
    self.tau_fn = tau_fn
    self.asked = set()
    self.procedures = procedures if procedures else []

    if candidate_questions:
        self.CANDIDATE_QUESTIONS = list(candidate_questions)
    else:
        self.CANDIDATE_QUESTIONS = self._FALLBACK_QUESTIONS
        print("[AdaptiveQuestionSelector] УВАГА: використовуються fallback-питання. "
              "Передайте candidate_questions з symptom_questions.json")

    self._cache_dir = cache_dir
    self._cache_path = os.path.join(cache_dir, "question_embeddings_cache.npz")
    self._embed_cache: dict[str, np.ndarray] = {}
    self._load_disk_cache()
```

Рисунок 3.33 – Ініціалізація моделі

Конструктор приймає п'ять параметрів, де patient_model – це екземпляр класу PatientModel, що підтримує векторний стан діагнозів і симптомів, а також механізм знімків через методи snapshot та restore, які селектор використовує для симуляції гіпотетичних кроків без незворотної

зміни стану. tau_fn є функцією температурного параметру, що регулює гостроту розподілу ймовірностей діагнозів при обчисленні ентропії Шеннона на кожному кроці сесії.

Параметр procedures приймає необов'язковий список діагностичних процедур, що відповідають тестам, які може виконати SmartVirtualPatient або базовий VirtualPatient через метод perform_test, за замовчуванням підставляється порожній список.

Параметр candidate_questions приймає зовнішній перелік питань-кандидатів; за його відсутності активується резервний список _FALLBACK_QUESTIONS із виведенням попередження у консолі, що сигналізує про неповну конфігурацію системи.

Параметр cache_dir визначає директорію для збереження дискового кешу векторних представлень зі значенням за замовчуванням DataSaves/, що дозволяє уникнути повторної BERT-векторизації питань між сесіями та суттєво пришвидшує запуск системи.

Після збереження всіх параметрів ініціалізується порожня множина asked, що відстежує вже задані питання впродовж поточної сесії та унеможливлює їх повторне задання віртуальному пацієнту. Далі формується повний шлях до файлу кешу та викликається метод _load_disk_cache, що завантажує раніше збережені векторні представлення із файлової системи.

Клас також містить чотири допоміжні методи, що забезпечують роботу з дисковим кешем векторних представлень та їх обчислення.

Метод _load_disk_cache завантажує збережені векторні представлення питань із файлової системи при ініціалізації об'єкта. Якщо файл кешу за вказаним шляхом відсутній, метод завершується без дій.

У разі успішного зчитування всі вектори завантажуються до словника _embed_cache у форматі {текст_питання: вектор}, а до консолі виводиться повідомлення з кількістю завантажених векторів. Якщо файл пошкоджений

або недоступний, виключення перехоплюється та виводиться у консоль без переривання роботи системи.

Метод _save_disk_cache зберігає поточний стан словника _embed_cache на диск у форматі .npz. Перед збереженням перевіряється, чи кеш не є порожнім, а також створюється директорія cache_dir якщо вона ще не існує.

Метод викликається автоматично щоразу після обчислення нового векторного представлення, що гарантує актуальність дискового кешу після кожного нового питання.

Метод _get_emb є центральним інтерфейсом доступу до векторних представлень. Він перевіряє наявність вектора для переданого рядка у словнику _embed_cache, і лише за його відсутності виконує повний ланцюжок обчислень: переклад тексту на англійську через translate_to_english, кодування через BERT за допомогою encode_text та нормалізацію результату через normalize_vec.

Після обчислення вектор зберігається у кеші та одразу записується на диск через _save_disk_cache.Таким чином, кожне питання векторизується лише один раз за весь час роботи системи.

Метод precompute_all призначений для попередньої векторизації всіх питань-кандидатів та процедур до початку сесії.

Метод об'єднує списки CANDIDATE_QUESTIONS та procedures в єдиний пул, підраховує кількість елементів, що ще не мають кешованого вектора, та послідовно викликає _get_emb для кожного з них.

Якщо всі вектори вже присутні у кеші, обчислення пропускається з відповідним повідомленням у консолі.

Використання цього методу перед стартом сесії дозволяє уникнути затримок під час реального діалогу з віртуальним пацієнтом, перенісши обчислювальне навантаження на етап підготовки.

На рисунку 3.34 показано фрагмент коду методу вибору найкращої дії, який ілюструє основні етапи: обчислення температурного параметра,

отримання поточних активацій, розрахунок ентропії, формування пулу кандидатів та перевірку наявності доступних питань і процедур.

Цей код демонструє ключові кроки підготовки до вибору наступного питання у сесії [11].

```
def select_best(self, step: int, initial_complaint_vec=None) -> tuple:  3 usages (3 dynam
    tau = self.tau_fn(step)
    current_activations = self.model.get_diagnosis_activations()
    H_before = shannon_entropy(compute_probabilities(current_activations, tau))

    best_action = None
    best_ig = -1e10
    best_type = "question"

    snap = self.model.snapshot()

    pool = [(q, "question") for q in self.CANDIDATE_QUESTIONS if q not in self.asked]
    pool += [(p, "test") for p in self.procedures if p not in self.asked]

    if not pool:
        return None, 0.0, None
```

Рисунок 3.34 – Метод вибору найкращої дії: підготовка

Метод приймає два параметри: step – це номер поточного кроку сесії, та необов'язковий initial_complaint_vec, що є векторним представленням початкової скарги пацієнта.

На початку метод обчислює температурний параметр tau через виклик tau_fn (step), після чого отримує поточні активації діагнозів із моделі та обчислює початкове значення ентропії Шеннона H_before, яке відображає міру невизначеності діагностичного розподілу до задання нового питання.

Далі ініціалізуються змінні для відстеження найкращої знайденої дії: best_action зі значенням None, best_ig зі значенням -1e10 як від'ємна нескінченність та best_type зі значенням «question» за замовчуванням. Після цього через метод snapshot зберігається поточний стан моделі, що дозволить відновлювати його після кожної симуляції у циклі оцінювання.

Формується пул кандидатів шляхом об'єднання питань із CANDIDATE_QUESTIONS та процедур із procedures, з яких виключаються вже задані елементи через перевірку множини asked. Якщо пул виявляється порожнім, метод негайно повертає кортеж None, 0.0, None, сигналізуючи про відсутність доступних дій для вибору.

Далі на рисунку 3.35 показано продовження методу вибору найкращої дії, де відбувається оцінка інформаційного приросту для кожного кандидата, враховуючи релевантність до початкової скарги, а також оновлення стану моделі і вибір оптимальної дії [11].

```
for action, a_type in pool:
    a_vec = self._get_emb(action) if a_type == "question" else embed_action(action)

    self.model.update_state(a_vec, revealed_symptoms=None)
    H_after = shannon_entropy(compute_probabilities(self.model.get_diagnosis_activations(), tau))
    ig = H_before - H_after

    self.model.restore(snap)

    if a_type == "test":
        ig -= 0.05

    if initial_complaint_vec is not None:
        relevance = float(cosine_similarity(a_vec.reshape(1, -1),
                                            initial_complaint_vec.reshape(1, -1))[0][0])
        ig += (relevance * 0.1)

    if ig > best_ig:
        best_ig = ig
        best_action = action
        best_type = a_type

if best_action is None and pool:
    best_action, best_type = pool[0]
    best_ig = 0.0

if best_action:
    self.asked.add(best_action)

return best_action, best_ig, best_type
```

Рисунок 3.35 – Метод вибору питання: оцінка кандидатів та вибір найкращої дії

Для кожної дії з пулу кандидатів формується її векторне представлення: якщо дія є питанням, використовується метод _get_emb для отримання ембеддингу, інакше – застосовується функція embed_action для процедур.

Після цього модель оновлює свій внутрішній стан із урахуванням цієї дії, що імітує отримання відповіді від пацієнта, при цьому параметр revealed_symptoms залишається None.

Наступним кроком обчислюється нове значення ентропії Шеннона H_after на основі оновлених активацій діагнозів, що дозволяє оцінити зміну невизначеності діагностичного розподілу після застосування дії.

Інформаційний приріст (ig) визначається як різниця між початковою ентропією H_before та новою H_after, що відображає ступінь зменшення невизначеності.

Для процедур інформаційний приріст коригується шляхом віднімання фіксованої величини 0.05, що враховує їхню відносно меншу інформативність порівняно з питаннями.

Якщо вхідний параметр initial_complaint_vec заданий, додатково обчислюється релевантність дії до початкової скарги за допомогою косинусної подібності між векторами, і ця релевантність додається до інформаційного приросту з коефіцієнтом 0.1, що підсилює пріоритет дій, більш релевантних початковій скарзі пацієнта.

Якщо обчислений інформаційний приріст перевищує поточне найкраще значення best_ig, відповідна дія зберігається як найкраща, а також оновлюється тип дії.

У випадку, коли жодна дія не перевищує початкові порогові значення, але пул кандидатів не порожній, метод вибирає першу доступну дію з пулу з інформаційним приростом 0.0.

Наприкінці обрана дія додається до множини вже заданих питань asked, що запобігає її повторному вибору в подальшому. Метод повертає кортеж із найкращою дією, її інформаційним приростом та типом.

## 3.7 Інтерфейс

Взаємодія студента з системою відбувається через веб-інтерфейс, реалізований на основі Flask та Socket.IO. Інтерфейс забезпечує повний цикл навчальної сесії – від вибору режиму роботи до отримання фінального звіту про компетенції – без необхідності перезавантаження сторінки завдяки двосторонньому WebSocket з'єднанню.

Початковий екран системи представлений на рисунку 3.36.

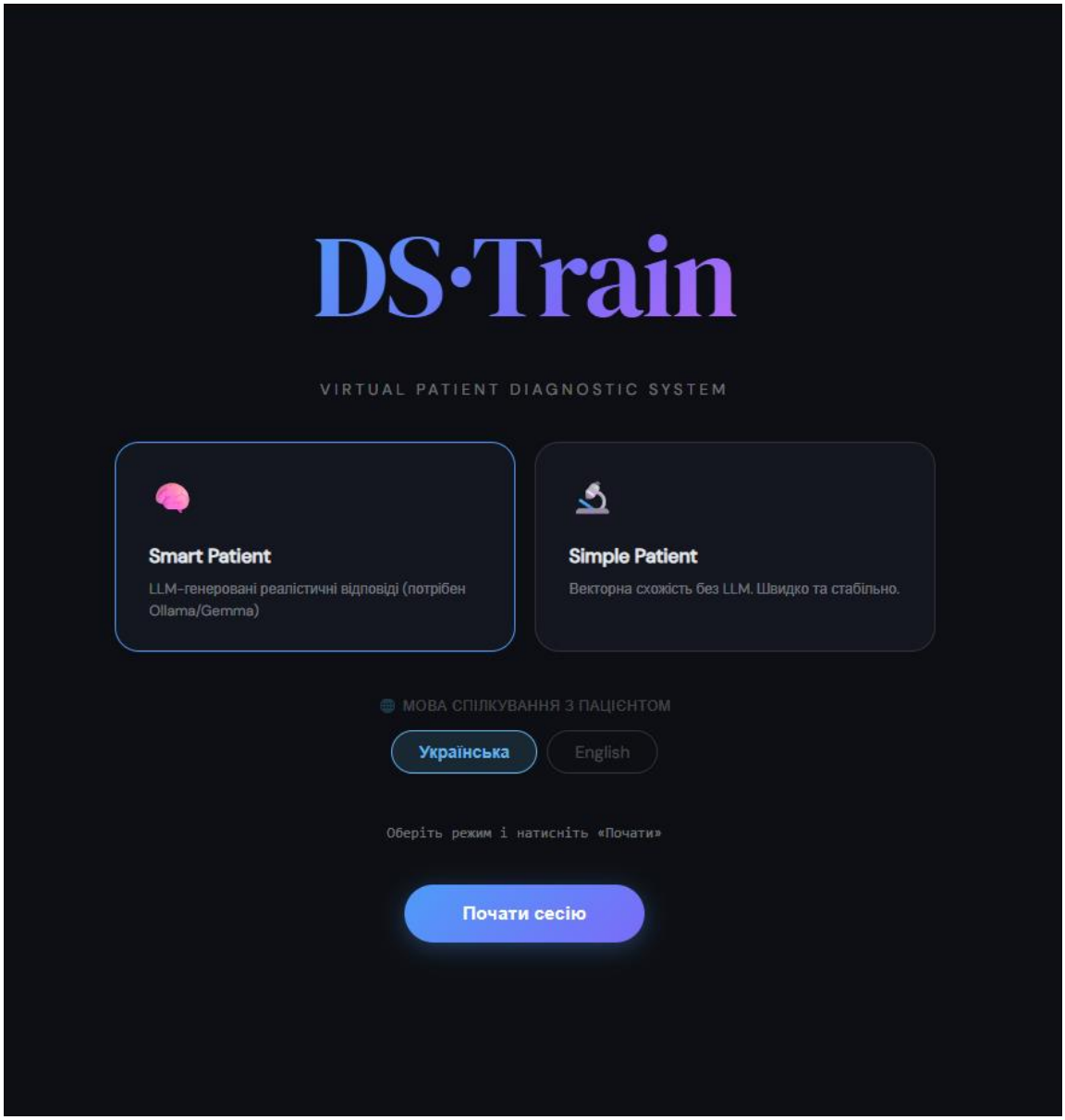



Рисунок 3.36 – Початкова сторінка

Початковий екран пропонує студенту вибір між двома режимами роботи. Режим Smart Patient використовує LLM для генерації природніх відповідей і потребує запущеного локального сервера Ollama з моделлю Gemma. Режим Simple Patient працює на основі векторної подібності без мовної моделі – швидше та стабільніше, але з шаблонними відповідями. Також доступний вибір мови спілкування з пацієнтом – українська або англійська. Після вибору налаштувань студент розпочинає сесію натисканням відповідної кнопки.

Основний екран діагностичної сесії складається з трьох панелей. Ліва панель стану пацієнта представлена на рисунку 3.37.

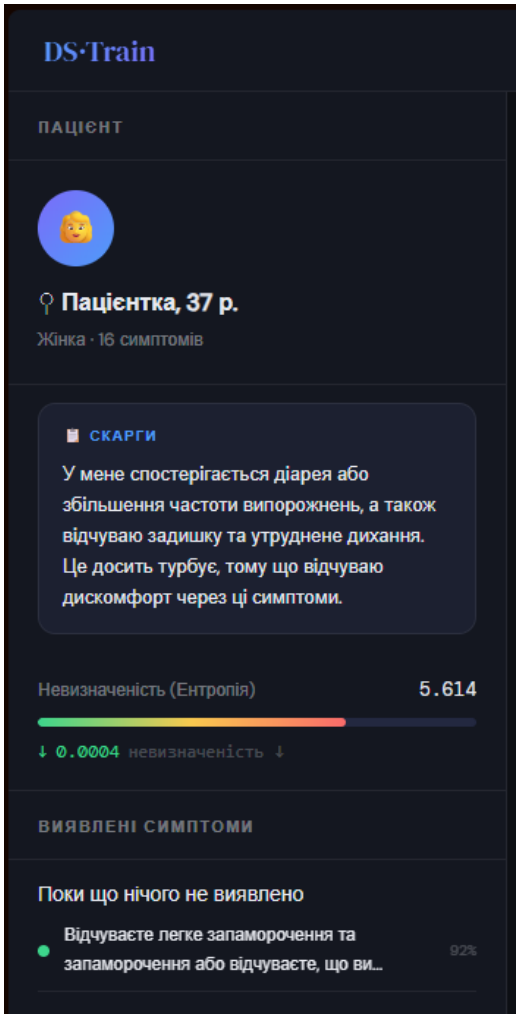



Рисунок 3.37 – Панель стану пацієнта

Ліва панель відображає поточний стан віртуального пацієнта протягом сесії. У верхній частині показана базова інформація – стать, вік та загальна кількість симптомів. Нижче розміщений блок початкової скарги, що формується при ініціалізації через get_initial_complaint. Індикатор ентропії з кольоровою шкалою відображає поточний рівень діагностичної невизначеності та зміну після останнього кроку. У нижній частині панелі відображаються вже виявлені симптоми, а також підказка наступного найбільш інформативного питання з відсотком очікуваного інформаційного прибутку. Центральна зона діалогу представлена на рисунку 3.38.

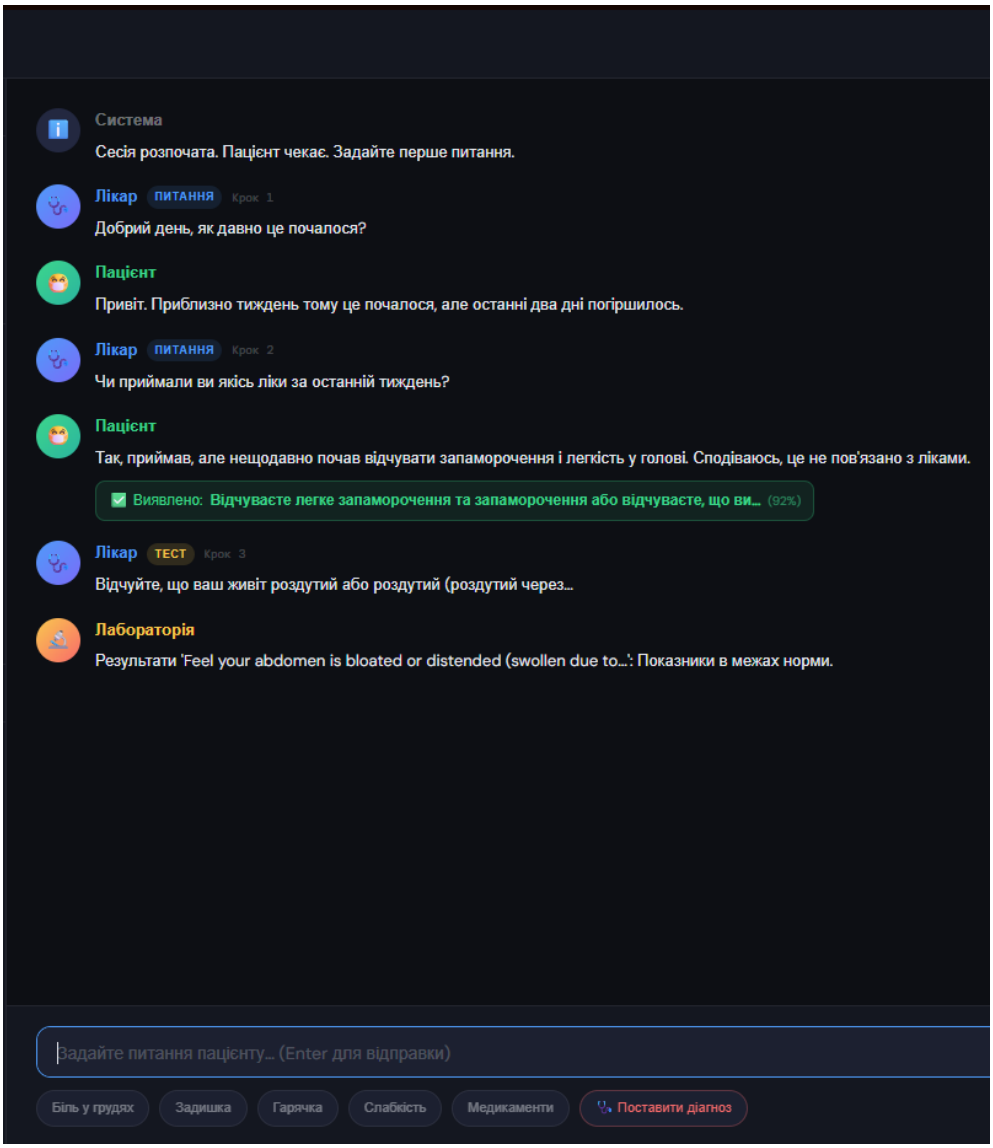



Рисунок 3.38 – Діалогове вікно користувача з пацієнтом

Центральна зона містить повну історію діалогу між студентом та віртуальним пацієнтом. Кожне повідомлення має візуальну мітку ролі – система, лікар або пацієнт – та номер кроку. Питання студента позначаються тегом «ПИТАННЯ», діагностичні процедури – тегом «ТЕСТ», відповіді лабораторії виділяються окремим кольором. При розкритті симптому відображається зелений банер з назвою виявленого симптому та відсотком подібності. У нижній частині розміщене поле введення питання та швидкі кнопки з найпоширенішими симптомами і кнопка постановки діагнозу.

Права панель підказок представлена на рисунку 3.39.

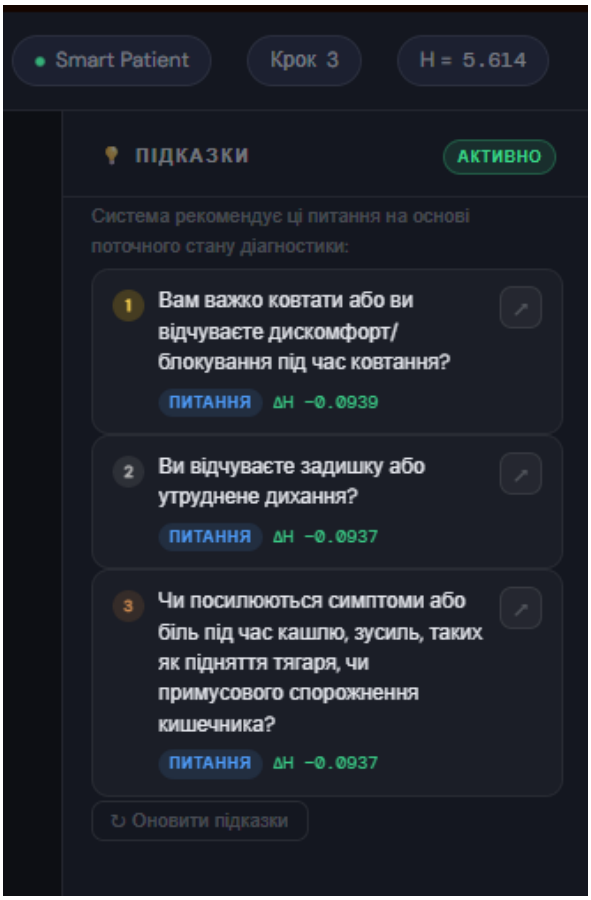



Рисунок 3.39 – Панель підказок до запитань

Права панель відображає рекомендації системи на основі поточного стану діагностики. Підказки не відображаються одразу – вони стають доступними лише після того, як студент поставив певну кількість питань і

система накопичила достатньо інформації про поточний стан діагностичних активацій для формування змістовних рекомендацій.

Для кожної підказки показується текст питання або процедури, її тип та очікуване значення зміни ентропії ΔH. Підказки формуються компонентом AdaptiveQuestionSelector і оновлюються після кожного кроку, відображаючи найбільш інформативні наступні дії з точки зору зниження діагностичної невизначеності. У верхній частині панелі відображається поточний режим роботи, номер кроку та поточне значення ентропії.

Для додаткового обстеження, є можливість провести аналіз/обстеження пацієнту (рисунок 3.40).

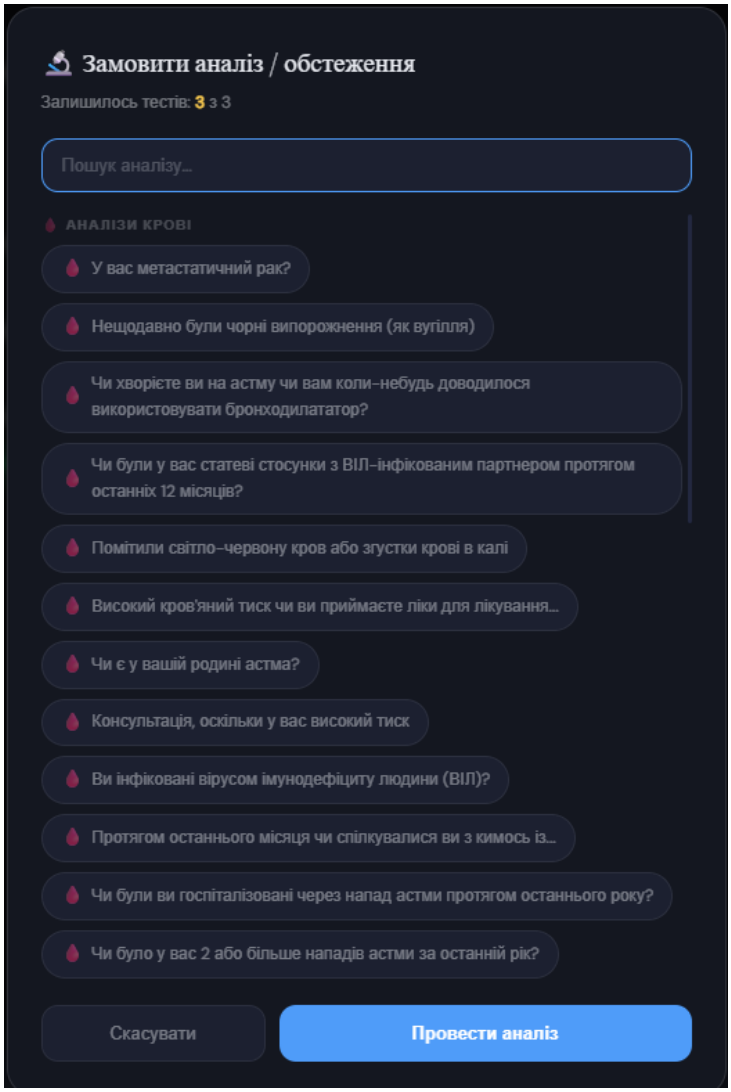



Рисунок 3.40 – Панель тестування пацієнта

Панель відкривається при натисканні кнопки замовлення тесту і відображає повний список доступних діагностичних процедур. У верхній частині показується лічильник залишкових тестів – студент має обмежену кількість діагностичних процедур на сесію, що стимулює обдумане їх використання.

Поле пошуку дозволяє швидко знайти потрібну процедуру серед усього переліку. Процедури згруповані за типом – аналізи крові, інструментальні обстеження тощо – і відображаються з відповідною іконкою категорії. Після вибору процедури студент підтверджує замовлення кнопкою «Провести аналіз», після чого результат відображається у центральній зоні діалогу як відповідь лабораторії.

Після завершення опитування студент виставляє діагноз, на основі якого система формує фінальний звіт (рисунок 3.41).

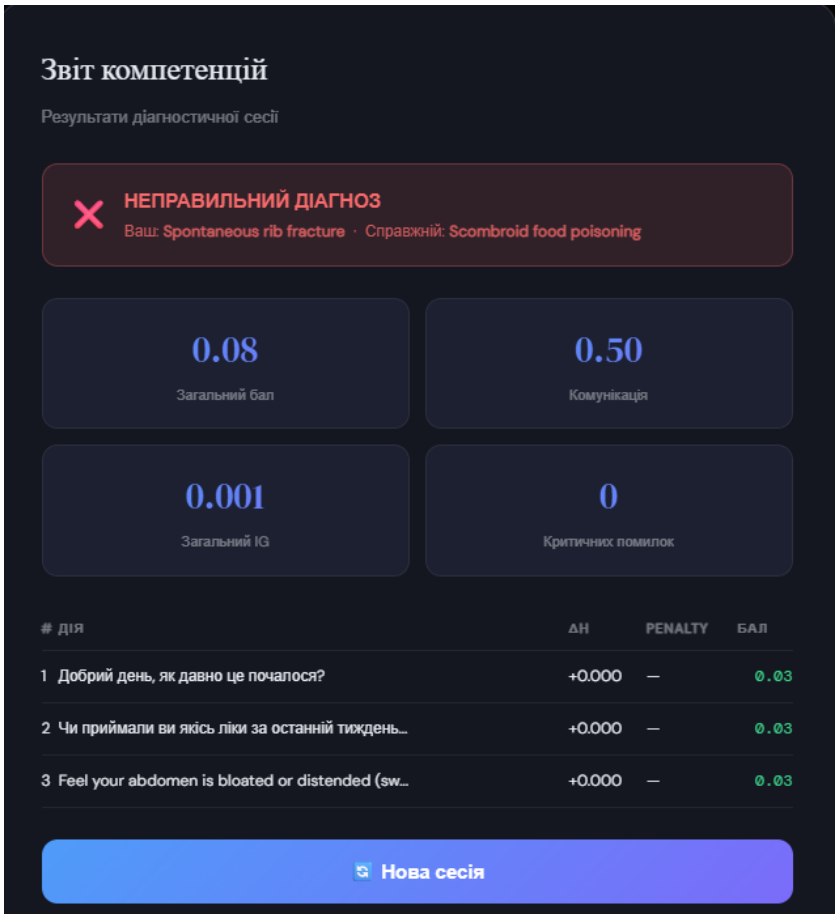



Рисунок 3.40 – Звіт компетенцій

Після постановки діагнозу система формує повний звіт компетенцій що відображає результати всієї діагностичної сесії. У верхній частині розміщений блок результату діагностики – при неправильному діагнозі він виділяється червоним кольором з відображенням як діагнозу студента так і істинного, що дозволяє одразу зрозуміти наскільки далекою була гіпотеза від правильної відповіді.

Нижче розміщені чотири зведені показники сесії у вигляді окремих карток. Загальний бал є сумою балів усіх кроків відповідно до виразу (2.11) і відображає інтегральну якість діагностичного процесу. Комунікативна оцінка є середнім значенням comm_score по всіх кроках і відображає загальний рівень культури спілкування студента з пацієнтом протягом сесії. Сумарний інформаційний прибуток показує наскільки ефективно студент звужував діагностичну невизначеність своїми питаннями – низьке значення свідчить про неструктурований підхід до збору анамнезу. Кількість критичних помилок відображає скільки разів студент звертався до небезпечних або протипоказаних втручань що спричинили нарахування штрафу.

У табличній частині розгортається покроковий аналіз кожної дії студента – питання або тест, зміна ентропії $\Delta H$ після кожного кроку, наявність штрафу та підсумковий бал кроку. Це дозволяє студенту детально проаналізувати свою діагностичну траєкторію – зрозуміти які питання були інформативними і суттєво знизили невизначеність, а які не принесли діагностичної цінності та лише витратили доступні кроки. Після ознайомлення зі звітом студент може розпочати нову сесію з новим випадково згенерованим пацієнтом.

# ВИСНОВКИ

У результаті виконання кваліфікаційної роботи вирішено актуальну задачу автоматизованої генерації адаптивних клінічних сценаріїв для систем медичної освіти в умовах семантичної свободи взаємодії студента з системою.

Проведений аналіз предметної галузі виявив ключові обмеження систем-аналогів. Full Code Medical Simulation оперує заздалегідь написаними кейсами від клініцистів і залишається статичною щодо генерації нових ситуацій, незважаючи на використання генеративного AI для діалогу. Ada Health демонструє потужність імовірнісних графів знань, однак не адаптована для навчального моделювання і позбавлена механізмів оцінювання когнітивної траєкторії студента. Виявлений технологічний розрив і визначив напрямок дослідження.

Наукова новизна полягає в комплексному поєднанні трьох математичних апаратів, що раніше не застосовувалося в системах медичного навчання. Модель пацієнта у вигляді динамічного зваженого графа знань із TF-IDF масштабуванням матриці суміжності забезпечує генерацію унікальних клінічних картин без прив'язки до фіксованих сценаріїв. Семантична векторизація на основі BioBERT дозволяє інтерпретувати вільний текстовий ввід студента без прив'язки до заздалегідь визначеного словника. Формалізація динаміки стану пацієнта апаратом керованих Марківських процесів із адаптивною перебудовою матриці суміжності за правилом Хебба унеможливлює завчене проходження за типовим шаблоном і змушує студента щоразу вибудовувати стратегію диференціальної діагностики самостійно.

Розроблений апарат оцінювання на основі ентропії Шеннона та метрик семантичної близькості до клінічних протоколів забезпечує об'єктивне вимірювання компетенцій одразу за трьома вимірами: діагностичною цінністю кроку, відповідністю протоколам і культурою

комунікації зі штрафами за небезпечні втручання. Такий підхід дозволяє отримати багатовимірний профіль студента, що відображає не лише правильність фінального діагнозу, а й якість клінічного мислення на кожному етапі обстеження.

Реалізований програмний комплекс включає дві версії віртуального пацієнта, адаптивний селектор питань та веб-інтерфейс із покроковим звітом компетенцій. Локальне розгортання LLM-компонента без залучення хмарних сервісів забезпечує конфіденційність медичних даних – критичний критерій для подальшого масштабування системи на реальні клінічні дані. Отримані результати підтверджують практичну застосовність запропонованого методу та його готовність до впровадження в процес підготовки майбутніх лікарів.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Abdulqader H. A., Abdulazeez A. M. Review on decision tree algorithm in healthcare applications. *Indonesian journal of computer science*. 2024. Vol. 13, no. 3. URL: https://doi.org/10.33022/ijcs.v13i3.4026 (date of access: 10.04.2026).

2. About simulation | SSH. *Home | SSH*. URL: https://www.ssih.org/about-simulation (date of access: 10.04.2026).

3. Bennett C. C., Hauser K. Artificial intelligence framework for simulating clinical decision-making: A Markov decision process approach. *Artificial intelligence in medicine*. 2013. Vol. 57, no. 1. P. 9–19. URL: https://doi.org/10.1016/j.artmed.2012.12.003 (date of access: 17.04.2026).

4. BioBERT: a pre-trained biomedical language representation model for biomedical text mining / J. Lee et al. *Bioinformatics*. 2019. URL: https://doi.org/10.1093/bioinformatics/btz682 (date of access: 09.04.2026).

5. DDXPlus: a new dataset for automatic medical diagnosis / A. F. Tchango et al. *Advances in neural information processing systems 35*, New Orleans, Louisiana, USA, 28 November – 9 December 2022. San Diego, California, USA, 2022. P. 31306–31318. URL: https://doi.org/10.52202/068431-2270 (date of access: 01.05.2026).

6. Digital implementation of the softmax activation function and the inverse softmax function / R. S et al. *2022 4th international conference on circuits, control, communication and computing (I4C)*, Bangalore, India, 21–23 December 2022. 2022. URL: https://doi.org/10.1109/i4c57141.2022.10057747 (date of access: 09.04.2026).

7. Feutrill A., Roughan M. A review of shannon and differential entropy rate estimation. *Entropy*. 2021. Vol. 23, no. 8. P. 1046. URL: https://doi.org/10.3390/e23081046 (date of access: 20.04.2026).

8. Full Code Medical Simulations. *Fullcodemedical*. URL: https://fullcodemedical.com/. (date of access: 23.03.2026).

9. Gemma 3. *Google DeepMind*. URL: https://deepmind.google/models/gemma/gemma-3/ (date of access: 15.04.2026).

10. GitHub - dmis-lab/biobert: bioinformatics'2020: BioBERT: a pre-trained biomedical language representation model for biomedical text mining. *GitHub*. URL: https://github.com/dmis-lab/biobert (date of access: 09.04.2026).

11. GitHub - ihorsikorskyi/clinicaladaptivescenariogenerator: master's degree diploma project. *GitHub*. URL: https://github.com/IhorSikorskyi/ClinicalAdaptiveScenarioGenerator (dat e of access: 23.04.2026).

12. GitHub - mila-iqia/ddxplus. *GitHub*. URL: https://github.com/mila-iqia/ddxplus?tab=readme-ov-file#ddxplus-a-new-dataset-for-automatic-medical-diagnosis (date of access: 20.04.2026).

13. Health. powered by ada. *Ada*. URL: https://ada.com/ (date of access: 28.03.2026).

14. Healthy simulation. *Healthy simulation*. URL: http://healthysimulation.com/. (date of access: 09.04.2026).

15. Heerema M., Leeuwen W. A. v. Derivation of Hebb's rule. *Journal of physics A: mathematical and general*. 1999. Vol. 32, no. 2. P. 263–286. URL: https://doi.org/10.1088/0305-4470/32/2/004 (date of access: 15.04.2026).

16. Kononenko I. Machine learning for medical diagnosis: history, state of the art and perspective. *Artificial intelligence in medicine*. 2001. Vol. 23, no. 1. P. 89–109. URL: https://doi.org/10.1016/s0933-3657(01)00077-x (date of access: 29.03.2026).

17. Kushniruk A. W. Analysis of complex decision-making processes in health care: cognitive approaches to health informatics. *Journal of biomedical informatics*. 2001. Vol. 34, no. 5. P. 365–376. URL: https://doi.org/10.1006/jbin.2001.1021 (date of access: 10.04.2026).

18. L2 normalization and geodesic distance for enhanced information preservation in visualizing high-dimensional single-cell sequencing data / Z. Rong et al. *BCB '24: 15th ACM international conference on bioinformatics, computational biology and health informatics*, Shenzhen China. New York, NY, USA, 2024. P. 1–11. URL: https://doi.org/10.1145/3698587.3701361 (date of access: 10.04.2026).

19. Learning a health knowledge graph from electronic medical records / M. Rotmensch et al. *Scientific reports*. 2017. Vol. 7, no. 1. URL: https://doi.org/10.1038/s41598-017-05778-z (date of access: 08.04.2026).

20. Mimic-iv. *PhysioNet*. URL: https://physionet.org/content/mimiciv/3.1/ (date of access: 07.04.2026).

21. Ollama. *Ollama*. URL: https://ollama.com/ (date of access: 10.04.2026).

22. Open-source AI models for any application | llama 3. *Open-source AI Models for Any Application | Llama 3*. URL: https://www.llama.com/models/llama-3/ (date of access: 14.04.2026).

23Peng Y. Inequalities for the Frobenius norm. *Journal of mathematical inequalities*. 2015. No. 2. P. 493–498. URL: https://doi.org/10.7153/jmi-09-43 (date of access: 15.04.2026).

24. Safavian S. R., Landgrebe D. A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*. 1991. Vol. 21, no. 3. P. 660–674. URL: https://doi.org/10.1109/21.97458 (date of access: 06.04.2026).

25. SympGraph / P. Sondhi et al. *The 18th ACM SIGKDD international conference*, Beijing, China, 12–16 August 2012. New York, New York, USA, 2012. URL: https://doi.org/10.1145/2339530.2339712 (date of access: 05.04.2026).