

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр

(назва освітнього ступеня)

на тему: Розробка програмного забезпечення для формування оптимальної конфігурації комп'ютерної мережі методом морфологічного моделювання

Виконав: студент

IV курсу, групи СН-41

спеціальності

122 Комп'ютерні науки

(шифр і назва спеціальності)

		Данилевич О.І. (прізвище та ініціали)
Керівник	(підпис)	Млинко Б.Б. (прізвище та ініціали)
Нормоконтроль	(підпис)	Шимчук Г.В. (прізвище та ініціали)
Завідувач кафедри	(підпис)	Боднарчук І.О. (прізвище та ініціали)
Рецензент	(підпис)	Тимощук Д.І. (прізвище та ініціали)

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ

Завідувач кафедри

Боднарчук І.О.
(підпис) (прізвище та ініціали)

« » червня 2025 р.

**ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ**

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)

Студенту Данилевичу Олегу Івановичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка програмного забезпечення для формування оптимальної конфігурації комп'ютерної мережі методом морфологічного моделювання

Керівник роботи Млинко Богдана Богданівна, кандидат технічних наук, доцент кафедри КН
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «27» січня 2025 року № 4/7-444

2. Термін подання студентом завершеної роботи 23 червня 2025р.

3. Вихідні дані до роботи Літературні джерела щодо методу морфологічного моделювання, його модифікацій.

4. Зміст роботи (перелік питань, які потрібно розробити)

Вступ. Розділ 1. Аналіз предметної області та постановка завдання. 1.1 Актуальність та теми. 1.2 Методи конфігурації та роль морфологічного аналізу. 1.3 Постановка задачі та формалізація технічного завдання. 1.4 Аналіз подібних програмних продуктів та їхня оцінка. Розділ 2. Проектування ПЗ для формування оптимальної конфігурації комп'ютерної мережі методом морфологічного моделювання. 2.1 Вибір і обґрунтування технологій. 2.2 Огляд середовища розробки. 2.3 Пошук актантів та їхнє виокремлення в МоМо. 2.4 Проектування ядра функціоналу системи МоМо. 2.5 Проектування архітектури графічного модуля МоМо. Розділ 3. Реалізація та тестування ПЗ для формування оптимальної конфігурації мережі методом морфологічного моделювання. 3.1 Структура ПЗ МоМо та його модульність. 3.2 Реалізація ядра ПЗ МоМо. 3.3 Реалізація графічного інтерфейсу. 3.4 Демонстрація розв'язання задачі оптимальної конфігурації комп'ютерної мережі Розділ 4. Безпека життєдіяльності, основи охорони праці. Висновки. Перелік джерел. Додатки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)

1. Титульний слайд. 2. Мета і завдання. 3. Метод морфологічного моделювання. 4. Архітектура та використані технології. 5. Ядро МоМо. 6.GUI МоМо. 7. AI-асистент МоМо. 8. Демонстрація прикладу. 9. Перспективи розвитку. 10. Висновки. 11. Подяка

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Сенчишин В.С., доцент кафедри МТ		

7. Дата видачі завдання 27 січня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	27.01.2025	
2.	Підбір джерел про метод морфологічного моделювання.	08.05.2025-13.05.2025	
3.	Опрацювання джерел по темі кваліфікаційної роботи	13.05.2025-20.05.2025	
4.	Виконання дослідження щодо технологій необхідних для розроблення програмного забезпечення.	20.05.2025-22.05.2025	
5.	Оформлення розділу «Аналіз предметної області та постановка завдання для програмного забезпечення для формування комп'ютерної мережі методом морфологічного моделювання»	21.05.2025-24.05.2025	
6.	Оформлення розділу «Проектування програмного забезпечення для формування комп'ютерної мережі методом морфологічного моделювання»	24.05.2025-01.06.2025	
7.	Оформлення розділу «Реалізація та тестування програмного забезпечення для формування комп'ютерної мережі методом морфологічного моделювання»	02.06.2025-05.06.2025	
8.	Виконання завдання до підрозділу «Безпека життєдіяльності»	02.06.2025-05.06.2025	
9.	Виконання завдання до підрозділу «Основи охорони праці»	06.06.2025-10.06.2025	
10.	Оформлення кваліфікаційної роботи	11.06.2025-15.06.2025	
11.	Нормоконтроль	16.06.2025	
12.	Перевірка на плагіат	18.06.2025	
13.	Попередній захист кваліфікаційної роботи		
14.	Захист кваліфікаційної роботи		

Студент

(підпис)

Данилевич О.І.

(прізвище та ініціали)

Керівник роботи

(підпис)

Млинко Б.Б.

(прізвище та ініціали)

АНОТАЦІЯ

Розробка програмного забезпечення для формування оптимальної конфігурації комп'ютерної мережі методом морфологічного моделювання // Кваліфікаційна робота освітнього рівня «Бакалавр» // Данилевич Олег Іванович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СН-41 // Тернопіль, 2025 // С. 73 , рис. – 27, табл. – 5, додат. – 2, бібліогр. – 58.

Ключові слова: морфологічне моделювання, підтримка прийняття рішень, конфігурація комп'ютерної мережі, графічний інтерфейс, штучний інтелект, прототип, модульне тестування.

Кваліфікаційна робота присвячена розробці програмного забезпечення для формування оптимальної конфігурації комп'ютерної мережі із використанням методу морфологічного моделювання.

Об'єктом дослідження є процес конфігурування мережевих систем (будь-яких складних систем) за допомогою розробленого програмного забезпечення.

Метою роботи є створення програмного інструменту, який дозволяє здійснювати перебір і оцінку альтернатив конфігурацій з урахуванням заданих критеріїв.

У першому розділі розглянуто теоретичні основи морфологічного аналізу та сучасні підходи до конфігурування складних систем в тому числі і комп'ютерних мереж.

У другому розділі подано архітектуру розробленого ПЗ, принципи його побудови та обґрунтування вибраних технологій.

У третьому розділі описано реалізацію компонентів системи, наведено приклад практичного застосування та результати тестування.

У четвертому розділі розглянуто питання охорони праці та безпеки життєдіяльності.

ANNOTATION

Software Development for Optimizing Computer Network Configurations Using Morphological Modeling // Qualification work of the educational level «Bachelor» // Danylevych Oleh // Ternopil Ivan Puluj National Technical University, Faculty of Computer and Information Systems and Software Engineering, Department of Computer Sciences, group SN-41 // Ternopil, 2025 // P. 73, fig. – 27, tabl. – 5, annexes – 2, references – 58.

Keywords: morphological analysis, decision support, computer network configuration, graphical interface, artificial intelligence, prototype, unit testing.

The qualification work is dedicated to the development of software for forming the optimal configuration of a computer network using the method of morphological analysis.

The object of the research is the process of configuring network systems (or any complex systems) using the developed software.

The goal of the work is to create a software tool that allows generating and evaluating configuration alternatives based on specified criteria.

The first section of the qualification paper considered the theoretical foundations of morphological analysis and modern approaches to the configuration of complex systems, including computer networks.

In the second section of the qualification work, the architecture of the developed software, the principles of its construction, and the rationale for the chosen technologies were presented.

The third section describes the implementation of the system components, provides an example of practical use, and includes the results of software testing.

The fourth section examines issues of occupational safety and life safety.

ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ

AI (англ. Artificial Intelligence) – штучний інтелект.

API (англ. Application Programming Interface) – інтерфейс прикладного програмування.

MIT License (англ. Massachusetts Institute of Technology) – ліцензія Массачусетського технологічного інституту на вільне програмне забезпечення.

GUI (англ. Graphical User Interface) – графічний інтерфейс користувача.

ПЗ – програмне забезпечення.

ШІ – штучний інтелект.

ЗМІСТ

ВСТУП	8
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ.....	10
1.1 Актуальність та обґрунтування обраної теми	10
1.2 Методи конфігурації та роль морфологічного аналізу.....	11
1.2.1 Підходи до вибору або побудови комп'ютерної мережі	11
1.2.2 Морфологічний аналіз як метод комбінаторного проектування для розв'язання задачі конфігурації комп'ютерної мережі.....	14
1.3 Постановка задачі та формалізація технічного завдання.....	16
1.4 Аналіз подібних програмних продуктів та їхня оцінка.....	18
1.5 Висновки до першого розділу	21
РОЗДІЛ 2. ПРОЄКТУВАННЯ ПЗ ДЛЯ ФОРМУВАННЯ ОПТИМАЛЬНОЇ КОНФІГУРАЦІЇ КОМП'ЮТЕРНОЇ МЕРЕЖІ МЕТОДОМ МОРФОЛОГІЧНОГО МОДЕЛЮВАННЯ	22
2.1 Вибір і обґрунтування технологій	22
2.2 Огляд середовища розробки.....	25
2.3 Пошук актантів та їхнє виокремлення в системі МоМо	26
2.4 Проектування ядра функціоналу системи МоМо	30
2.5 Проектування архітектури графічного модуля системи МоМо	33
2.5.1 Структура класів модуля МоМоGUI.....	37
2.6 Висновок до другого розділу	39
РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПЗ ДЛЯ ФОРМУВАННЯ ОПТИМАЛЬНОЇ КОНФІГУРАЦІЇ КОМП'ЮТЕРНОЇ МЕРЕЖІ МЕТОДОМ МОРФОЛОГІЧНОГО МОДЕЛЮВАННЯ.....	40
3.1 Структура програмного забезпечення МоМо та його модульність ...	40
3.2 Реалізація ядра програмного забезпечення МоМо	42
3.2.1 Структура класів модуля ядра МоМо	43
3.2.2 Тестування модуля ядра МоМо	44

3.3 Реалізація графічного інтерфейсу користувача	46
3.3.1 Структура та архітектура графічного інтерфейсу користувача ...	47
3.3.2 Стартове вікно МоМоGUI.....	48
3.3.3 Головне вікно МоМоGUI	49
3.3.4 Вікно AI-асистента в МоМоGUI	54
3.4 Демонстрація розв’язання задачі оптимальної конфігурації	
комп’ютерної мережі із використання ПЗ МоМо	56
3.5 Висновок до третього розділу	65
РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ	66
4.1 Оцінка ризику перевтоми при багатогодинній роботі з моделями	
конфігурацій у середовищах із високим когнітивним	
навантаженням	66
4.2 Профілактика мікротравм кистей та зап’ясть у фахівців при	
тривалому використанні периферійних пристроїв	68
4.3 Висновок до четвертого розділу	71
ВИСНОВКИ.....	72
ПЕРЕЛІК ДЖЕРЕЛ	74
ДОДАТКИ	

ВСТУП

Актуальність теми. У сучасних умовах стрімкого розвитку інформаційних технологій, зростання обсягу даних та ускладнення інфраструктурних вимог, особливого значення набуває проблема ефективного проєктування комп'ютерних мереж. Побудова оптимальної конфігурації таких систем є складним завданням, що вимагає врахування великої кількості технічних параметрів, взаємозалежностей між компонентами та обмежень за ресурсами.

У зв'язку з цим актуальним напрямком сучасних досліджень є розробка інструментів для автоматизованого конфігурування мереж з урахуванням формалізованих критеріїв. Метод морфологічного моделювання, що дозволяє системно перебирати комбінації технічних рішень і оцінювати їх відповідність заданим вимогам, набуває тут особливої цінності. Його інтеграція в програмне забезпечення, доповнене інтерфейсом користувача та можливостями штучного інтелекту, відкриває нові перспективи для підтримки прийняття рішень при таких задачах.

Мета і завдання дослідження. Метою цієї кваліфікаційної роботи освітнього рівня «Бакалавр» є створення програмного засобу, що дозволяє здійснювати формування оптимальної конфігурації комп'ютерної мережі методом морфологічного моделювання. Програмне забезпечення повинно бути гнучким, масштабованим, із підтримкою користувацького інтерфейсу й інтелектуального супроводу.

Для досягнення поставленої мети потрібно виконати ряд завдань, зокрема:

- проаналізувати сучасний стан досліджень у галузі автоматизованого конфігурування мережевих систем;
- дослідити теоретичні основи методу морфологічного моделювання;
- розробити архітектуру ПЗ та реалізувати ядро системи для генерування конфігурацій;

- створити графічний інтерфейс користувача для зручної взаємодії з системою;
- забезпечити можливість використання ШІ для вибору кращих варіантів та пояснення рішень;
- провести тестування системи та продемонструвати приклад її практичного застосування.

Практичне значення одержаних результатів. Розроблене програмне забезпечення дозволяє ефективно вирішувати задачі вибору конфігурації комп'ютерної мережі на основі формалізованих параметрів. Воно може застосовуватись у навчальних цілях, для підтримки інженерних рішень, у процесі підготовки інфраструктурних проєктів, а також у будь-яких сценаріях, де потрібна прозора й логічно обґрунтована побудова технічних систем.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Актуальність та обґрунтування обраної теми

На сьогодні комп'ютерні мережі виступають критичним елементом інформаційної інфраструктури підприємств, установ, освітніх закладів, державного управління, тощо. Із збільшенням кількості пристроїв, обсягів даних, складності архітектурних рішень зростає також і потреба у комплексному підході до проектування та оптимізації мереж. Неправильна конфігурація окремих вузлів мережі (маршрутизаторів, комутаторів, серверів, типів каналів зв'язку тощо) може призвести до надлишкових витрат, зниження продуктивності або критичних вразливостей у безпеці.

Завдання знаходження оптимальної конфігурації комп'ютерної мережі є, по суті, задачею комбінаторного вибору – коли з множини можливих альтернатив кожного компоненту (наприклад, вибір типу комутатора, маршрутизатора тощо) необхідно сформувати найкращу сукупність з точки зору заданих критеріїв. Зазвичай ці критерії включають вартість, продуктивність, надійність, споживання ресурсів, безпеку тощо [1]. У реальних умовах кількість можливих комбінацій становить більше сотні, а то й тисячі, що призводить до того, що порахувати і проаналізувати їх вручну важко або й взагалі – не можливо, тому традиційні ручні підходи оцінки стають неефективними.

У таких умовах актуальним є використання системного методу, що дозволяє враховувати весь простір можливих рішень і водночас зберігати гнучкість для різних конфігурацій. Одним із таких підходів є морфологічне моделювання – метод, що дозволяє формально подати всі варіанти структури складної системи, оцінити їх схожість до бажаної моделі (прототипу), і знайти оптимальні конфігурації на основі критеріїв [2]. Саме тому актуальним є створення інструменту, який дозволяє систематизувати всі можливі

конфігурації (системи), порівняти їх між собою та виявити найбільш наближені до заданого еталонного рішення (прототипу).

Тематика кваліфікаційної роботи є актуальною також тому, що в межах реалізованого програмного засобу було створено графічний інтерфейс, який дозволяє користувачам формувати морфологічні матриці порівняння систем, а також отримувати аналітичні інсайти завдяки вбудованому агенту, який працює на базі великих мовних моделей (анг. Large Language Model (LLM)).

1.2 Методи конфігурації та роль морфологічного аналізу

Як вже згадувалося задача побудови або реконфігурації комп'ютерної мережі є складною комбінаторною проблемою, при якій необхідно враховувати широкий спектр взаємозалежних факторів – від вибору типу обладнання і пропускної здатності каналів до вимог щодо безпеки, енергоспоживання, загальної вартості, вартості на встановлення та подальшу підтримку, можливостей масштабування та відмовостійкості, а також обмежень, пов'язаних з політикою підприємства.

На практиці для знаходження прийнятних рішень застосовуються різні підходи: від ручного проектування з використанням мережевих симуляторів до застосування мультикритеріальних методів аналізу, евристик і сучасних алгоритмів оптимізації, які дозволяють автоматично генерувати варіанти конфігурацій та оцінювати їх за визначеними критеріями.

1.2.1 Підходи до вибору або побудови комп'ютерної мережі

Одним із найпоширеніших способів проектування та перевірки комп'ютерних мереж є використання мережевих симуляторів – спеціалізованих програмних середовищ, які дозволяють створювати віртуальні топології, налаштовувати мережеве обладнання та проводити моделювання роботи протоколів. До найвідоміших таких інструментів належать Cisco Packet Tracer,

GNS3 (Graphical Network Simulator-3) та EVE-NG (Emulated Virtual Environment Next Generation) [9, 10] (див. рис. 1.1).



Рисунок 1.1 – Середовища для симуляції побудови та конфігурації комп'ютерних мереж

Cisco Packet Tracer базується на підході drag-and-drop інтерфейсу, який дозволяє будувати мережу із наявних компонентів. GNS3 є емулятором, який запускає реальні IOS-образи всередині віртуального середовища, чим саме пропонує певну долю реалістичності в розробці та тестуванні, але потребує більше обчислювальних ресурсів. EVE-NG поєднує переваги обох підходів у веб-інтерфейсі з підтримкою мультивендорних образів та інтеграцією з хмарними платформами. Усі ці інструменти чудово підходять для тестування протоколів (OSPF, BGP, EIGRP), налаштування VLAN, VPN та моделювання трафіку.

Однак ці симулятори мають одне важливе обмеження: вони не генерують автоматично варіанти конфігурацій мережі та не забезпечують їхню оцінку відповідно до заданих параметрів. Користувач повинен самостійно створити топологію, обрати обладнання, врахувати технічні обмеження і зробити вибір за сукупністю параметрів.

Іншим підходом до вибору конфігурацій мережі є застосування мультикритеріальних методів аналізу (Multi-Criteria Decision Making) [8]. Їхня суть полягає в системному порівнянні альтернатив за низкою критеріїв із

попереднім визначенням ваг кожного критерію експертним шляхом або за допомогою спеціальних алгоритмів.

Одним із найпоширеніших мультикритеріальних методів є метод аналізу ієрархій (Analysis Hierarchy Process (АНП)), який передбачає побудову ієрархічної структури задачі, виконання парних порівнянь між альтернативами та розрахунок вагових коефіцієнтів на основі експертних оцінок.

Крім АНП, у практиці також широко застосовуються такі методи, як:

- TOPSIS (Technique for Order Preference by Similarity to Ideal Solution) – вибір альтернативи здійснюється на основі схожості до так званого «ідеального» вектору.
- ELECTRE – метод виключає із набору, альтернативи чи комбінації, які є «явно» гіршими, іншими словами – дає змогу відкинути «слабкі» варіанти;
- SAW (Simple Additive Weighting) – кожен нормалізований оцінку альтернативи помножують на вагу критерію й підсумовують. Цей метод являється найшвидшим в реалізації.

Незважаючи на свою ефективність у задачах середньої складності, мультикритеріальні методи не масштабуються ефективно при великій кількості альтернатив та параметрів. Це зумовлено необхідністю формувати велику кількість матриць порівнянь і ручного налаштування ваг, що практично неможливо автоматизувати для систем з десятками або сотнями змінних.

Паралельно з МКМА популярні і евристичні методи оптимізації, які застосовуються у випадках, коли простір рішень занадто великий для повного перебору. До таких методів належать: генетичні алгоритми, метод рою частинок та жадібні стратегії

Ці підходи дозволяють отримати наближено оптимальні рішення у складних задачах, де неможливо перевірити всі варіанти. Проте їх ефективність залежить від правильного налаштування параметрів, які не завжди легко підібрати, особливо якщо критерії є нечітко формалізованими.

1.2.2 Морфологічний аналіз як метод комбінаторного проектування для розв'язання задачі конфігурації комп'ютерної мережі

На противагу вищезгаданим методам, морфологічний аналіз (ММ) є системним способом дослідження варіантів структури складного об'єкта на основі повного переліку можливих конфігурацій. Його було вперше запропоновано Фріцем Цвікі у 40-х роках ХХ століття для вирішення задач астрофізики [2, 3, 4]. Метод дозволяє побудувати морфологічну скриню (див. рис. 1.2), тобто таблицю або модель, яка включає: підсистеми, альтернативи та їхні ознаки.

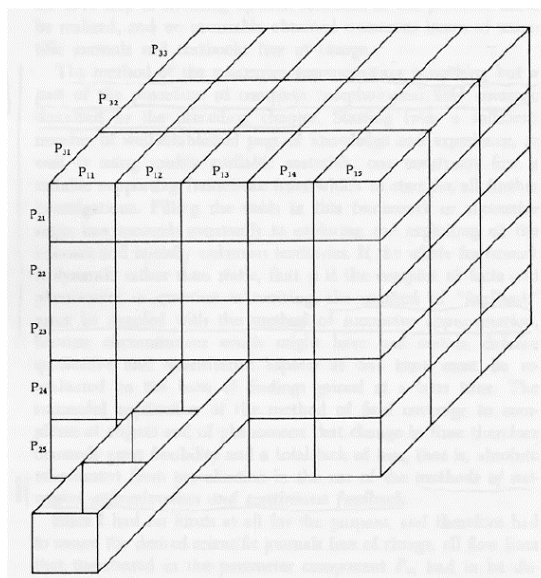


Рисунок 1.2 – Морфологічний скриня

Метод морфологічного моделювання у класичному формулюванні ґрунтується на декартовому добутку множин альтернатив підсистем, що дозволяє отримати повний простір конфігурацій; проте для інженерних задач (в тому числі і поставленої задачі щодо оптимального конфігурування комп'ютерної мережі), де кількість варіантів вимірюється тисячами або мільйонами, важливо не лише перебрати усі комбінації, а й мати механізм їх відбору відповідно до цілей користувача.

Модифікований підхід щодо реалізації методу морфологічного моделювання, використаний у цій кваліфікаційній роботі, доповнює традиційну схему введенням прототипу – еталонної конфігурації, яка фіксує бажані властивості системи, а також застосуванням бінарного кодування ознак і формальних метричних функцій для порівняння. Загалом модифікація методу морфологічного моделювання працює за підходом, який описаний нижче.

Для кожної можливої комбінації альтернатив із кожної підсистеми будується конфігурація. Нехай A_i – множина альтернатив i -тої підсистеми, а n – загальна кількість підсистем. Тоді розмір простору всіх конфігурацій C задається формулою:

$$|C| = \prod_{i=1}^n |A_i|, \quad (1.1)$$

Така побудова гарантує вичерпний аналіз усіх можливих рішень, оскільки жодна комбінація альтернатив не залишається поза розглядом.

Наступним етапом є формування прототипу – бажаної конфігурації, яка відображає ключові вимоги до системи. Прототип позначимо як P , так що $P = (p_1, p_2, \dots, p_n), p_i \in A_i$. p_i – обрана альтернатива для i -тої підсистеми, що задає ознаки «ідеальної» системи (наприклад, максимальна безпека чи енергоефективність). Прототип використовується як еталон для подальшого порівняння з усіма «можливими» варіантами з C .

Далі для кожної конфігурації $c \in C$ обчислюється міра схожості з прототипом. Найчастіше застосовуються дві метрики:

1. Коефіцієнт Жаккара

$$J(c, P) = \frac{|c \cap P|}{|c \cup P|}, \quad (1.2)$$

де $c \cap P$ – кількість спільних ознак;

$c \cup P$ – загальна кількість унікальних ознак.

2. Коефіцієнт Соренсена-Дайса

$$DSC(c, P) = \frac{2|c \cap P|}{|c| + |P|}. \quad (1.3)$$

Коефіцієнт Соренсена-Дайса (формула 1.3) більш чутливо реагує на кількість спільних ознак, подвоюючи їхній внесок.

Значення цих метрик впорядковують конфігурації за ступенем наближення до бажаного прототипу, що дозволяє обрати найрелевантніші варіанти для практичної реалізації.

Таким чином, у контексті задачі конфігурації комп'ютерної мережі морфологічне моделювання надає формальний, системний та прозорий підхід, і відрізняється від традиційних методів.

1.3 Постановка задачі та формалізація технічного завдання

У межах кваліфікаційної роботи розглядається задача розробки програмного забезпечення, яке дозволяє моделювати складні системи, що складаються з декількох підсистем, кожна з яких має набір альтернатив та ознак. Така постановка є типовою для широкого класу задач, зокрема задач проєктування та конфігурації комп'ютерних мереж, де компоненти можуть мати різні взаємозв'язки, функціональні властивості та технічні параметри.

Об'єктом розробки є програмне забезпечення для морфологічного моделювання складних багатокомпонентних систем під назвою МоМо.

Метою є створення інструменту, який дозволяє користувачеві описувати структуру системи, задавати можливі альтернативи її складових, порівнювати варіанти реалізацій, а також автоматизувати процес відбору найбільш прийнятних конфігурацій згідно із заданими критеріями.

Задача полягає у побудові системи, що надає можливість:

- Задавати структуру системи у вигляді переліку підсистем, кожна з яких містить альтернативи реалізації.
- Описувати ознаки альтернатив – уніфікованих характеристик, що представлені в бінарній формі.
- Формувати базу варіантів конфігурацій, які утворюються комбінацією альтернатив усіх підсистем.
- Порівнювати сформовані варіанти з певною «ідеальною» моделлю (прототипом) за допомогою критеріїв схожості або сумісності.
- Виводити результати аналізу у зручному для користувача вигляді: списку, таблиці, тощо.

До основних функціональних вимог до ПЗ належать:

- Гнучке моделювання системи – можливість створення нових підсистем, додавання альтернатив і ознак.
- Редагування та збереження моделей – підтримка локальної роботи з моделями та базовий функціонал управління даними.
- Побудова комбінацій – автоматичне формування повного простору можливих конфігурацій.
- Інструменти порівняння – можливість обрати або задати орієнтир (прототип) для порівняння всіх варіантів.
- AI (Artificial Intelligent) асистент – як додаткова особливість цього проекту, що додала б можливості приймати швидкі рішення на основі прокалькульованих даних.

Таким чином, загальне технічне завдання полягає у створенні гнучкого, масштабованого та повторно використовуваного інструменту, що реалізує принципи морфологічного аналізу для задач структурної побудови та оцінки альтернатив складних систем в тому числі і побудови оптимальної конфігурації комп'ютерної мережі.

1.4 Аналіз подібних програмних продуктів та їхня оцінка

Сьогодні для підтримки процесу морфологічного моделювання існує низка як комерційних, так і відкритих інструментів. Найбільш відомими з них є MA/Carma, Morpheus, Morphological Box Builder та MorphR. Кожен із цих продуктів має свої сильні та слабкі сторони.

MA/Carma (Computer-Aided Morphological Analysis) (див. рис. 1.3) – це настільне програмне забезпечення для Windows, спеціалізоване на проведенні морфологічного аналізу і моделювання сценаріїв. Продукт розроблено Шведським морфологічним товариством (Swedish Morphological Society) [11]. MA/Carma є комерційною пропрієтарною системою, яка використовується як платформа для побудови морфологічних моделей та створення «лабораторій» сценарного і стратегічного аналізу.

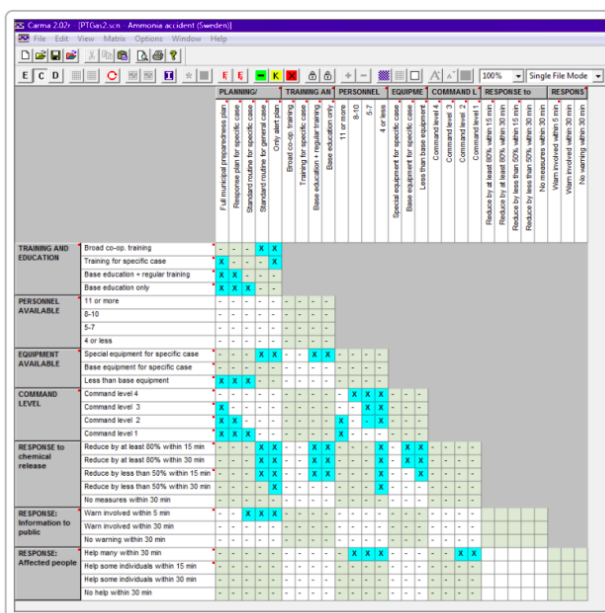


Рисунок 1.3 – Інтерфейс MA/Carma

Основний інтерфейс MA/Carma складається з трьох областей: поле редагування, матриця сумісності та поле відображення результатів (див. рис. 1.3). Користувач визначає параметри та їх значення у полі редагування, після чого в матриці здійснюється Cross-Consistency Assessment

(ССА) – перевірка всіх пар значень на сумісність. На скріншоті вище показано фрагмент матриці несумісностей: значення, які не можуть співіснувати в одній конфігурації, помічені символом «X». Після позначення несумісних пар система автоматично вилучає непрацездатні конфігурації з простору рішень.

Інструмент підтримує повний цикл морфологічного моделювання: від початкового аналізу і синтезу варіантів – до генерації сценаріїв, моделювання принципу «що буде, якщо...» (задавання вхідних умов і отримання вихідних) та оцінки наслідків.

Morpheus (див. рис. 1.4) – інструмент для створення морфологічних матриць, розроблений у Технічному університеті Чалмерса (Швеція) для навчальних і дослідницьких цілей. Початково Morpheus був реалізований як крос-платформний настільний додаток і використовувався студентами на Windows, macOS та Linux. У 2020-2021 роках інструмент випробували приблизно 500 студентів у межах курсових проєктів із розробки продуктів, що підтвердило його користь для систематизації генерування концептів. Станом на сьогодні Morpheus доступний як вебзастосунок, що повністю замінює настільну версію (вихідний код відкритий на GitHub) [12].

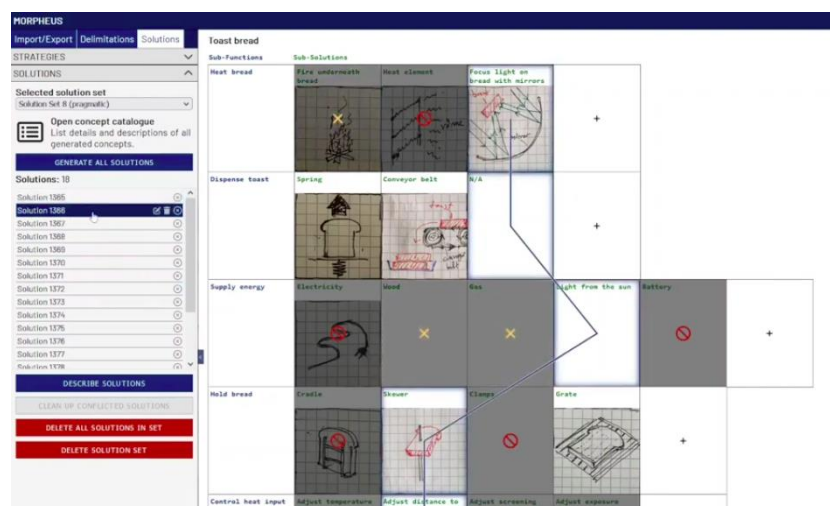


Рисунок 1.5 – Інтерфейс Morpheus

Morpheus дозволяє створювати та редагувати морфологічні матриці в браузері. Користувач може додавати субфункції (параметри) і варіанти їх

реалізації, позначати несумісні пари значень, автоматично генерувати всі можливі конфігурації та експортувати результати у CSV або зображення. За словами авторів, інструмент дозволив студентам швидко генерувати та відслідковувати дуже велику кількість концептів, чого важко досягти вручну.

MorphR – це пакет для мови R, що реалізує морфологічний аналіз з інтерактивним веб-інтерфейсом на базі Shiny (через браузер) [13]. На відміну від класичних настільних рішень, MorphR працює на будь-якій операційній системі, де доступний R та веб-браузер, що робить його крос-платформеним рішенням. Проект був представлений Markus Voge, як перша спроба перенести метод морфологічного аналізу у сучасне веб-середовище з відкритим кодом.

У таблиці 1.1 наведено порівняння ключових параметрів: доступності, підтримки перевірки на сумісність, наявності веб-інтерфейсу, можливостей експорту, відкритості коду та наявності інтелектуальної підтримки.

Таблиця 1.1 – Порівняння альтернатив

Назва	Доступ	Cross-Consistency	Веб	Експорт	Відкритий код
MA/Carma	Пропріетарне	Так	Ні	Обмежено	Ні
Morpheus	Відкритий	Так	Так	Так	Так
MorphR	Відкритий	Так	Так	Обмежено	Так

Як видно з таблиці 1.1, сучасні інструменти морфологічного аналізу демонструють різноманіття підходів до реалізації цього методу – від класичних настільних рішень до веб застосунків і статистичних пакетів. Крім того, жоден із розглянутих інструментів не позиціонується як інтуїтивно зрозумілий універсальний конструктор, який може бути використаний неспеціалістами у довільній предметній області без глибокої попередньої підготовки.

Таким чином, актуальною є розробка нового засобу, який не лише відтворює класичну логіку морфологічного аналізу, але й адаптує її до вимог поставленого завдання, конфігурації комп'ютерної мережі.

1.5 Висновки до першого розділу

У першому розділі обґрунтовано актуальність автоматизованого підходу до проєктування конфігурацій складних систем, зокрема комп'ютерних мереж.

Проведено огляд традиційних методів – симуляційних інструментів (для симуляцій роботи комп'ютерної мережі), багатокритеріального аналізу та евристичних алгоритмів, і показано їхні обмеження в аспектах повного перебору варіантів та прозорості результатів.

Розглянуто метод морфологічного моделювання як системний засіб формалізації підсистем, альтернатив і ознак, що забезпечує структурований перебір усіх комбінацій і оцінювання їх за формальними метриками.

Сформульовано загальні технічні вимоги до програмного забезпечення для морфологічного аналізу, яке повинно бути предметно-незалежним, гнучким і масштабованим.

Проведено порівняльний аналіз наявних інструментів і виявлено прогалини в їхній універсальності.

Отже, існує необхідність розробити новий засіб, що поєднуватиме класичний морфологічний підхід із сучасними інтерфейсними та інтелектуальними компонентами.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ПЗ ДЛЯ ФОРМУВАННЯ ОПТИМАЛЬНОЇ КОНФІГУРАЦІЇ КОМП'ЮТЕРНОЇ МЕРЕЖІ МЕТОДОМ МОРФОЛОГІЧНОГО МОДЕЛЮВАННЯ

2.1 Вибір і обґрунтування технологій

Першочерговим завданням при розробці програмного забезпечення є обґрунтований вибір набору технологій, оскільки саме він визначає як зручність реалізації, так і якість кінцевого програмного продукту. Вибір інструментальних засобів для проєктування та розробки складних систем повинен спиратися на чітко визначені критерії, які відповідають поставленим вимогам.

У межах цієї кваліфікаційної роботи стояло завдання створити програмне забезпечення, яке реалізує метод морфологічного моделювання для конфігурації багатокomпонентних систем, зокрема комп'ютерних мереж. Вибір технологій в конкретному випадку був важливим, оскільки система мала забезпечувати не лише математично коректний і ефективний розрахунок, але й інтуїтивно зрозумілий інтерфейс користувача (UI) та інтегровані функції штучного інтелекту. Окрім цього, важливими були такі фактори, як продуктивність, масштабованість, зручність тестування.

З огляду на поставлені вимоги до програмного продукту, було здійснено аналіз та відібрано оптимальний набір технологій. Зокрема, було обрано мову програмування Python, бібліотеки для роботи з даними, фреймворк для побудови графічного інтерфейсу, технології для інтеграції штучного інтелекту, формати даних для імпорту та експорту, а також засоби автоматизації процесів тестування.

Мова Python обрана в якості основної мови програмування для реалізації цього проєкту з кількох причин. По-перше, синтаксис її є інтуїтивно зрозумілим та лаконічним, що значно прискорює розробку і полегшує підтримку коду. По-друге, Python має надзвичайно розвинену екосистему

пакетів для різного формату роботи із даними, зокрема: pandas і NumPy [15]. По-третє, Python є однією із найпопулярніших мов програмування за даними індексу TIOBE, у травні 2025 року частка використання Python становила 25,35 % (див. рис. 2.1), що є найвищим показником за всю історію індексу серед решти популярних мов [14].

May 2025	May 2024	Change	Programming Language		Ratings	Change
1	1			Python	25.35%	+9.02%
2	3	▲		C++	9.94%	+0.41%
3	2	▼		C	9.71%	-0.27%
4	4			Java	9.31%	+0.62%
5	5			C#	4.22%	-2.27%
6	6			JavaScript	3.68%	+0.66%

Рисунок 2.1 – Рейтинг популярності мов програмування станом на 2025

Ураховуючи всі ці фактори Python – став основним та єдиним фаворитом при розробці ПЗ для оптимального конфігурування комп’ютерної мережі із використанням морфологічного моделювання (надалі будемо використовувати назву продукту МоМо).

Для обробки та аналізу табличних даних ядро проєкту використовує бібліотеки pandas і NumPy. Бібліотека pandas забезпечує структури даних DataFrame та Series, які дозволяють ефективно оперувати таблицями альтернатив і ознак, виконувати фільтрацію, агрегацію та злиття наборів даних. NumPy, у свою чергу, надає оптимізовані багатовимірні масиви і векторизовані операції, що значно прискорює масові обчислення під час генерації комбінацій і обчислення метрик схожості [16].

Для створення крос-платформного графічного інтерфейсу обрано фреймворк PyQt5 [17]. Він забезпечує набір багатофункціональних віджетів, реалізованих на основі Qt, та підтримує механізм сигнал-слот, що дозволяє відокремити логіку обробки подій від представлення даних.

Використання PyQt5 гарантує однакову поведінку інтерфейсу на Windows, macOS і Linux. Для забезпечення неблокувального виконання обчислень застосовано QThread, модуль, що дозволяє розподілити важкі завдання між потоками та зберегти реактивність інтерфейсу.

Для реалізації інтеграції штучного інтелекту обрано бібліотеку CrewAI, яка забезпечує обгортку для взаємодії з великими мовними моделями (LLM) із використання ReAct (Reasoning Action) технології, тобто підкріплення власних спостережень за допомогою виклику наданих інструментів [22]. Саме це й робить CrewAI ідеальним вибором при створенні AI систем, агентів або чат-ботів.

Було прийнято рішення використовувати вбудованого асистента на базі пропрієтарних моделей OpenAI з кількох причин. По-перше, моделі OpenAI стабільно посідають провідні позиції в галузевих бенчмарках за якістю генерованого тексту і різноманітністю задач. По-друге, у порівнянні з відкритим моделями, затримка відповіді (latency) пропрієтарних моделей через хмарний API значно нижча, це пов'язано із тим, що відкриті моделі запускаються локально, а це означає, що використовуються ресурси конкретного ПК. До комерційних ж доступ здійснюється через API інтерфейс. По-третє, моделлю із найнижчою затримкою на ринку зараз є саме модель від OpenAI gpt-4.1 (станом на 22 травня 2025 року) [25].

Узагальнюючи, вибір стеку технологій для розробки МоМо базується на принципах продуктивності, гнучкості та крос-платформеності. Python забезпечує швидке прототипування та широку екосистему модулів, тоді як pandas і NumPy гарантують ефективну обробку великих обсягів табличних даних. PyQt5 дозволяє створювати привабливий інтерфейс. Інтеграція AI через CrewAI і пропрієтарні моделі OpenAI забезпечує високу якість і низьку затримку генерації відповідей без навантаження локальних ресурсів.

2.2 Огляд середовища розробки

У процесі створення програмного забезпечення для оптимального конфігурування комп'ютерної мережі із використанням методу морфологічного моделювання було обрано два ключові інструменти для розробки: Visual Studio Code та GitHub. Ці середовища були використані як на етапі проєктування ПЗ, так і під час активної розробки, тестування та підтримки.

Visual Studio Code (див. рис. 2.2) – це текстовий редактор, розроблений компанією Microsoft, який підтримує розширення функціоналу за допомогою плагінів, зокрема для роботи з Python [26].

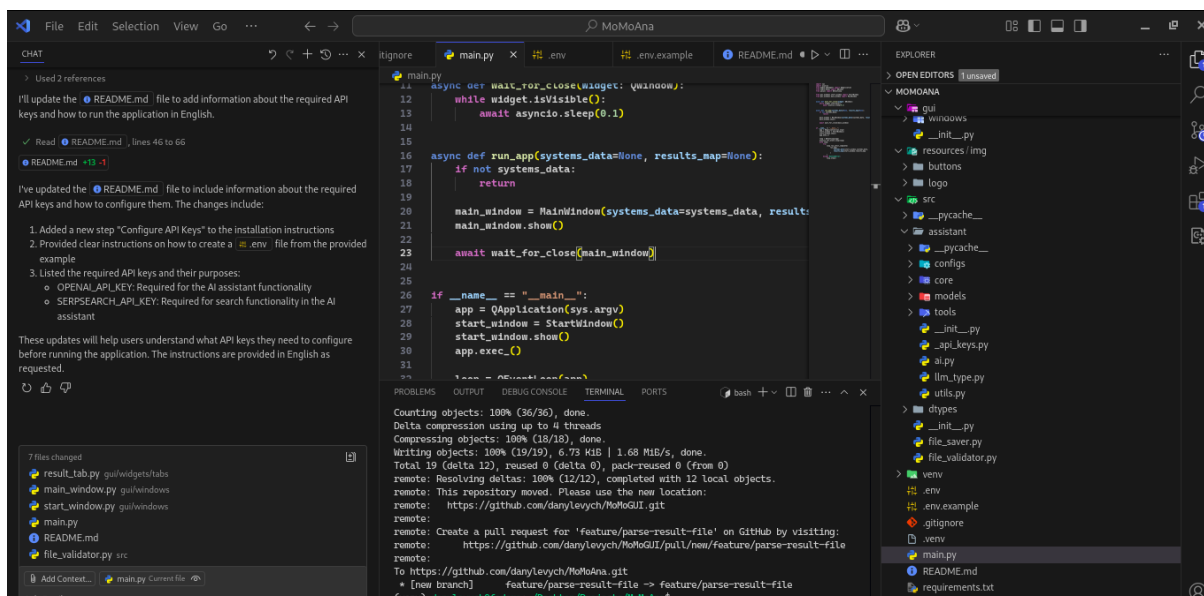


Рисунок 2.2 – Інтерфейс VS Code

Його було обрано як основне середовище розробки через, в більшій мірі, досвід роботи та розробки саме в VS Code, проте, якщо позбутись суб'єктивності то на це також вплинули і ряд наступних особливостей: підтримка багатовіконного режиму, вбудований термінал, інтеграції з системами контролю версій (Git), кросплатформеність цього текстового редактору, і нарешті – вбудований ШІ асистент (GitHub Copilot), який допомагав автодоповненнями коду, та виправленню багів. Також варто

відмітити сотні розширень, які доступні в екосистемі VS Code, що дозволяють проводити автоматичну перевірку коду, форматування, перегляд змін, запуск тестів, а також керування середовищами виконання Python.

GitHub – це платформа для зберігання проєктів, які використовують систему контролю версій Git [27]. Проєкт МоМо зберігається у віддаленому репозиторії GitHub (покликання: <https://github.com/danylevyach/MoMoGUI>), що дозволяє зручно відслідковувати зміни, керувати гілками та створювати контрольовані версії. Основні переваги використання GitHub у межах цього проєкту полягають у можливості веденні історії змін, організації пул-реквестів та фіксації завдань через систему Issues (див. рис. 2.3).

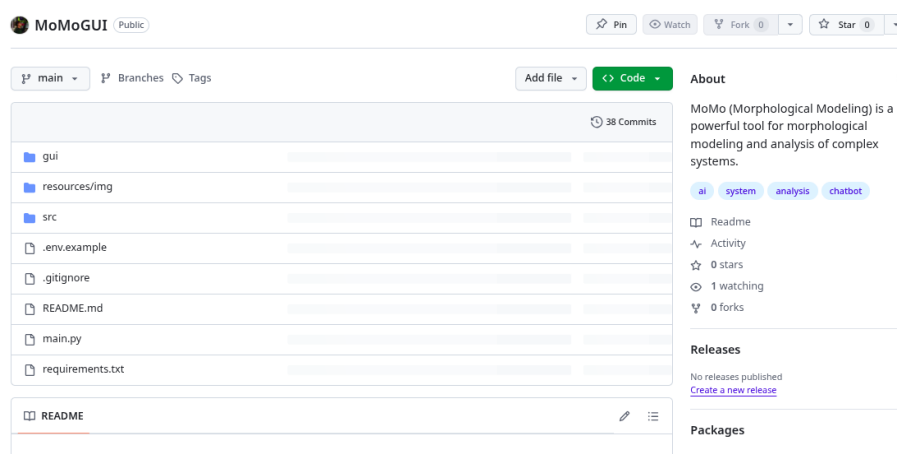


Рисунок 2.3 – Головна сторінка репозиторію МоМо на GitHub

Використання Visual Studio Code у тандемі з GitHub дало змогу створити зручне, контрольоване та відтворюване середовище розробки, в якому можна ефективно реалізовувати та підтримувати ПЗ МоМо.

2.3 Пошук актантів та їхнє виокремлення в системі МоМо

У цьому розділі буде розглянуто застосування мови UML (Unified Modeling Language) для графічного відображення поведінки користувача в програмному забезпеченні для оптимальної конфігурації комп'ютерної системи

із використанням методу морфологічного моделювання. Ми опишемо основні сценарії взаємодії, продемонструємо побудову діаграми варіантів використання та пояснимо, які актори (актани) беруть участь у цих сценаріях, а також які функції вони можуть здійснювати в межах системи.

Загалом мова UML є міжнародним стандартом, затвердженим консорціумом Object Management Group, і використовується для формальної специфікації, візуалізації та документування програмних систем на всіх етапах життєвого циклу розробки. Серед різних типів діаграм, визначених стандартом UML 2.5.1, особливе місце посідає діаграма варіантів використання. Вона слугує інструментом фіксації функціональних вимог і відображає взаємодію зовнішніх акторів із системою через сукупність варіантів її поведінки (use cases) [29]. Такі діаграми для програмного забезпечення застосовуються на ранніх етапах аналізу вимог для окреслення контексту системи, визначення її меж. Вони формують базис для подальшого деталізованого проектування, зокрема для побудови послідовнісних і класових діаграм [31, 32].

На діаграмі із рисунку 2.4 виокремлено двох акторів. Користувач взаємодіє з ядром функціональності та ініціює всі основні сценарії роботи, та AI-агент забезпечує відповіді на запити, що передаються з модуля взаємодії.

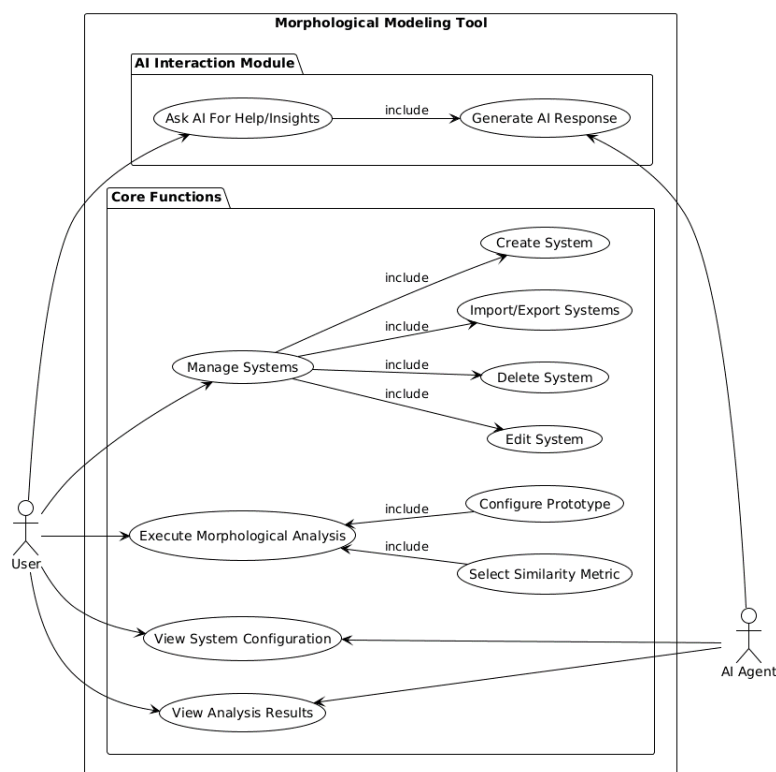


Рисунок 2.5 – Діаграма варіантів використання МоМо

Для детального розуміння взаємодії акторів із системою розглянемо окремо дії користувача та функціональні можливості AI-агента, виходячи з UML-діаграми варіантів використання (див. рис. 2.4).

Спочатку зосередимося на користувачеві. Функціональність, доступна йому через графічний інтерфейс системи, умовно поділяється на три основні групи: керування конфігурацією, параметризація аналізу, та перегляд результатів.

1. Керування конфігурацією системи. Користувач має змогу створювати нові системи, редагувати або видаляти існуючі, а також імпортувати та експортувати у вигляді таблиць (у форматі Excel). Ці дії охоплюють усі аспекти роботи з підсистемами та їхніми альтернативами та ознаками.

2. Параметризація та запуск аналізу. Після створення або імпорту системи, користувач визначає прототип (еталонну конфігурацію), що служить точкою порівняння, та обирає метрику схожості (Жаккар або Соренсен-Дайс).

Далі ініціюється морфологічний аналіз, в межах якого система генерує всі можливі конфігурації та обчислює ступінь їх відповідності до прототипу.

3. Перегляд результатів та структури. За результатами аналізу користувач може ознайомитись із ранжованим списком конфігурацій та оцінками їх схожості. Звичайно загальну поточну конфігурацію можна також експортувати, для того, щоб при наступному підході можна було працювати із вже створеною системою.

Також варто зауважити, що користувач має доступ до комунікації із AI агентом. Це в свою чергу дає можливість запитати про допомогу або про те, які інсайти може надати помічник стосовно результатів обчислення схожості.

Функціональність AI-агента у системі реалізована як підтримка користувача на етапі аналізу результатів і прийняття рішень. Його роль полягає в супроводі – так званий «last mile customization», коли на основі вже згенерованих даних користувач отримує їхню інтерпретацію (інсати), пояснення або поради. Агент працює з контекстом, отриманим із графічного інтерфейсу, а також за потреби залучає додаткові джерела, такі як веб-пошук. Основні функції AI-агента включають:

- Генерацію пояснень щодо отриманих результатів аналізу.
- Надання рекомендацій з покращення або адаптації конфігурацій.
- Відповіді на уточнюючі питання користувача у вільній формі.
- Доступ до зовнішніх джерел інформації для розширення контексту.

Таким чином, агент не дублює базову логіку аналізу, а доповнює її, надаючи інтерпретацію результатів і взаємодію із користувачем із використанням природної мови.

Загалом розділення на два класи акторів – User і AI Agent – дозволяє чітко окреслити, які функціональні блоки реалізуються в UI-модулі, а які – в модулі взаємодії зі штучним інтелектом.

2.4 Проектування ядра функціоналу системи МоМо

На момент початку розробки програмного забезпечення для розв'язання задачі оптимальної конфігурації комп'ютерної мережі із використанням методу морфологічного моделювання на Python не існувало відкритих або комерційних бібліотек, які б реалізовували модифікований метод морфологічного аналізу, про який розповідалося в першому розділі. Існуючі інструменти надавали обмежені можливості автоматизації та розширення через власні формати даних. Відповідно, виникла необхідність створити Python-модуль, який поєднав би:

- генерацію всіх комбінацій альтернатив із заданих систем;
- фільтрацію за принципом Cross-Consistency Assessment;
- обчислення метрик схожості (Жаккара, Соренсена-Дайса);
- чіткий програмний інтерфейс для подальшої інтеграції з GUI.

Для відображення структури ядра програмного забезпечення було використано діаграму класів, яка є основним засобом структурного моделювання в UML [32]. Цей тип діаграм дозволяє описати програмну систему у вигляді класів із відповідними атрибутами, методами та зв'язками між ними.

Діаграма класів фокусується на статичному аспекті системи, що дає змогу зафіксувати логіку взаємодії між компонентами та провести декомпозицію функціональності на незалежні одиниці. У проектуванні складних програмних модулів вона дозволяє визначити, які об'єкти є незалежними, які формують композиційні структури, а які взаємодіють через зовнішні інтерфейси.

На рисунку 2.6 подано реалізацію діаграми класів ядра МоМо, яка демонструє ключові об'єкти, включно з моделями систем, прототипом та модулем розрахунків. Загалом ця модель класів реалізує модифікований підхід із використання методу морфологічного моделювання.

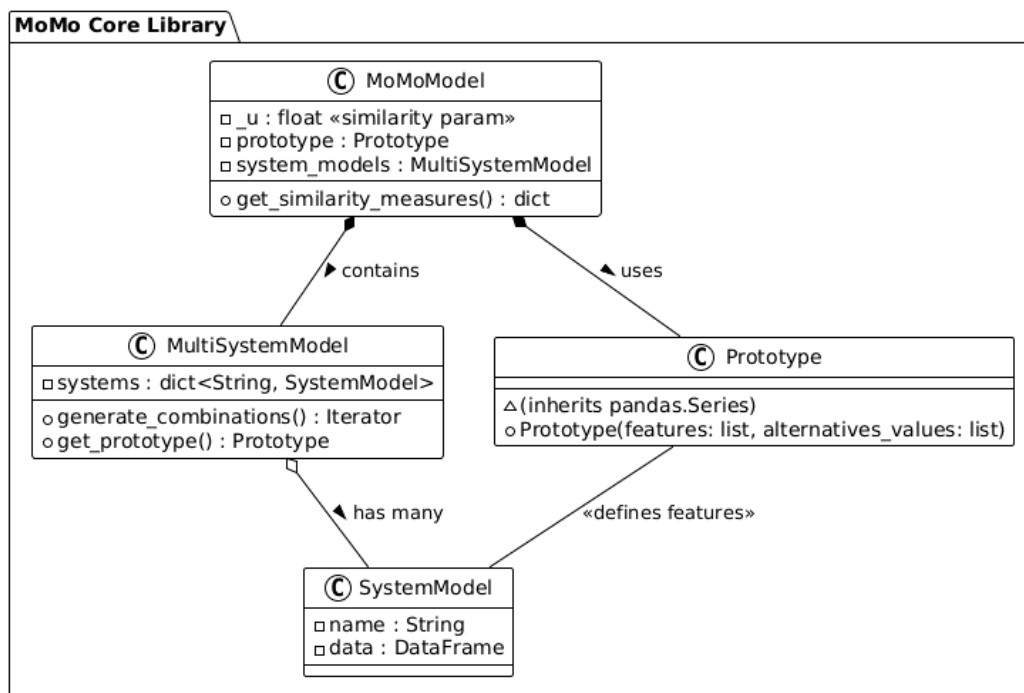


Рисунок 2.6 – Діаграма класів ядра системи МоМо

У процесі проектування архітектури ядра модуля МоМо було обрано підхід, що ґрунтується на принципах модульності, єдиної відповідальності (Single Responsibility Principle) та композиції як домінантного механізму взаємодії між компонентами.

Для головного модуля було використано дещо модифікований патерн «Фасад», у ролі якого виступає клас MoMoModel. Цей патерн є структурним шаблоном проектування, що забезпечує спрощений інтерфейс до складної підсистеми або набору взаємопов'язаних компонентів. Метою цього патерну є зменшення складності взаємодії користувача або зовнішніх модулів із внутрішніми елементами системи шляхом делегування викликів і абстрагування від деталей реалізації [37].

Архітектура ядра логічно поділяється на три пов'язані рівні. На найнижчому рівні перебуває об'єкт SystemModel, який є базовим представленням окремої підсистеми. Цей об'єкт інкапсулює всі дані щодо ознак та відповідних альтернатив. На цьому рівні зберігається структурована

інформація, яка у подальшому слугує будівельним матеріалом для більш складних операцій.

Наступним рівнем у ієрархії є `MultiSystemModel`, який агрегує набір об'єктів `SystemModel`. Його завданням є управління множиною підсистем і реалізація механізмів створення повного простору конфігурацій. Іншими словами, саме цей компонент відповідає за побудову логіки комбінування можливих варіантів усіх підсистем у єдине узгоджене представлення. Такий підхід дозволяє масштабувати систему шляхом простого додавання нових підсистем, не змінюючи при цьому основну логіку аналізу. `MultiSystemModel` не має знання про прототип чи метрику, а лише виконує свою частину – побудову простору розв'язків (варіантів комбінації).

Третім важливим компонентом є `Prototype`, який виконує роль абстрактного представлення бажаної конфігурації. Прототип не виконує жодних обчислень, але є ключовим у системі, оскільки забезпечує орієнтир для аналізу. Всі комбінації, згенеровані `MultiSystemModel`, оцінюються відносно цього прототипу. Концептуально, прототип визначає, «яким має бути ідеальний варіант», а система виконує пошук найближчих до нього реалізацій.

На найвищому рівні архітектури перебуває `MoMoModel`, який об'єднує в собі все попереднє. Цей компонент координує взаємодію між `MultiSystemModel` та `Prototype`, виступає зовнішнім інтерфейсом ядра та реалізує головну функцію – обчислення ступеня схожості кожної конфігурації до прототипу. Таким чином, `MoMoModel` можна розглядати як точку доступу до логіки всієї системи. Цей клас приймає вхідні параметри, формує на їх основі запит до `MultiSystemModel`, а далі здійснює обчислення, результати яких можуть бути використані як для виводу на екран, так і для формування звітів чи подальшої обробки.

Слід підкреслити, що в архітектурі дотримано принципу низької зв'язаності між класами. Усі компоненти мають вузько окреслені зони відповідальності та можуть замінюватися або тестуватися незалежно один від одного. Крім того, обраний підхід передбачає відкритість до розширення, тобто

у майбутньому можуть бути додані нові метрики, інші способи порівняння або методи фільтрації.

Загалом така архітектура дозволяє забезпечити не лише стабільність та масштабованість рішення, але й його адаптивність до майбутніх вимог, включаючи інтеграцію з іншими модулями, як-от графічний інтерфейс або системи штучного інтелекту, що розширюють можливості користувача у взаємодії з результатами аналізу.

2.5 Проектування архітектури графічного модуля системи МоМо

Графічний інтерфейс МоМоGUI забезпечує зв'язок між користувачем і обчислювальним ядром системи, реалізуючи механізми керування конфігураційними даними, ініціації морфологічного аналізу, візуалізації результатів та інтеграції з AI-асистентом. Архітектурно інтерфейс організовано у вигляді окремих вкладок, кожна з яких відповідає за окрему групу функціональних можливостей, що дозволяє розмежувати презентаційну логіку від обчислювального модуля. При цьому взаємодія з ядром здійснюється через визначені інтерфейси, що гарантує незалежність GUI-вимог від внутрішньої реалізації алгоритмів моделювання та полегшує підтримку й розширення системи.

Для формалізації сценаріїв взаємодії між компонентами графічного інтерфейсу, ядром системи та зовнішніми сервісами використано діаграми послідовності UML. Цей тип діаграм дозволяє відобразити часову впорядкованість повідомлень між об'єктами системи під час виконання певного сценарію [35]. На відміну від діаграм класів, які показують статичну структуру, діаграми послідовності зосереджені на динаміці процесів: вони дозволяють виявити залежності між викликами, обробкою подій та передачею даних на часовій осі.

У межах цього підрозділу будуть розглянуті такі ключові сценарії використання інтерфейсу МоМоGUI:

- створення конфігурації та запуск морфологічного аналізу;
- імпорт конфігураційних даних із зовнішнього файлу формату Excel;
- взаємодія користувача з AI-асистентом через графічний інтерфейс.

Зараз перейдемо до розгляду першого із виокремлених сценаріїв – створення конфігурації та ініціацію морфологічного аналізу. Цей сценарій ілюструє, як користувач через інтерфейс задає вихідні дані та запускає обчислювальний процес у ядрі системи. Діаграму, яка зображає потік виконання можна побачити на рисунку 2.7.

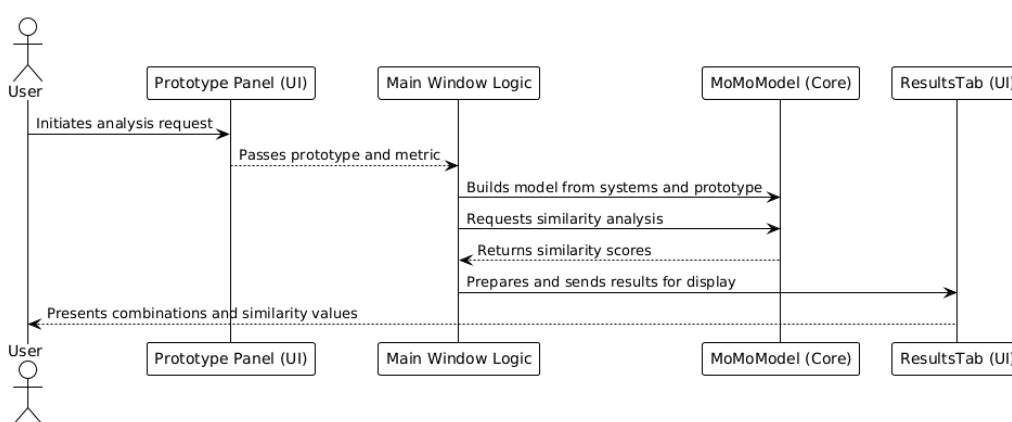


Рисунок 2.7 – Діаграма послідовності для сценарію створення конфігурації та ініціацію морфологічного аналізу

Діаграма послідовності із рисунку 2.7 ілюструє основні (абстраговані) кроки виконання сценарію створення та початку морфологічного аналізу:

1. Користувач ініціює запит на аналіз із панелі.
2. Панель передає зібрані параметри (список ознак прототипу та вибрану метрику) до логіки головного вікна (Main Window Logic).
3. Головне вікно формує об'єкт MoMoModel, ініціалізуючи його даними про підсистеми й прототип.
4. Викликається метод обчислення схожості в MoMoModel.
5. Після завершення розрахунків результати повертаються до Main Window Logic і передаються у вкладку результатів (ResultsTab).

6. Користувач переглядає згенеровані конфігурації та їхні оцінки схожості.

Таким чином, реалізовано чіткий поділ обов'язків між презентаційною частиною, контролером і ядром застосунку.

Далі перейдемо до другого ключового сценарію – імпорт даних із зовнішнього файлу формату Excel (див. рис. 2.8).

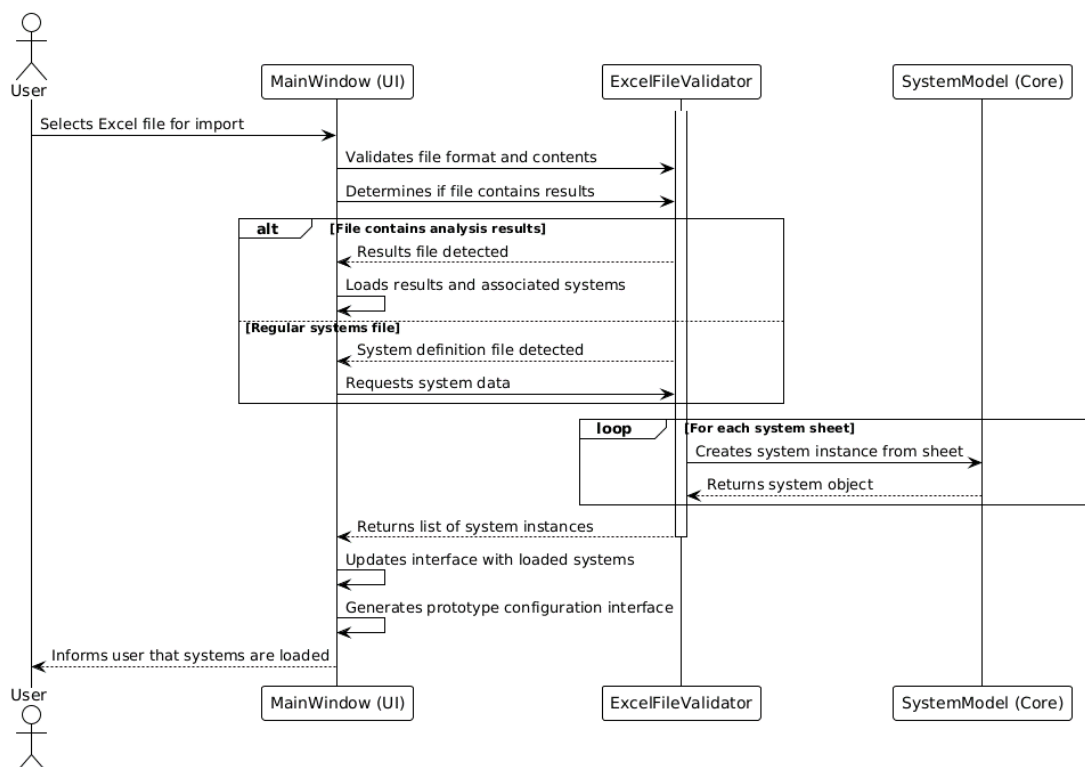


Рисунок 2.8 – Діаграма послідовності: імпорт систем із Excel-файлу

Відповідно до діаграми, імпорт конфігураційних даних створюється через основний компонент графічного інтерфейсу користувача (Main Window), який відповідає за взаємодію з файловою системою. Після вибору файлу користувачем виконується перевірка його структури та вмісту за допомогою спеціалізованого валідатора даних (Excel File Validator). Результатом цього етапу є визначення типу імпортованих даних (конфігурації систем або результати аналізу). Далі система здійснює послідовне зчитування вмісту

кожного аркуша файлу, формуючи відповідні об'єкти, які представляють моделі системи.

Після завершення імпортування дані передаються компоненту графічного інтерфейсу, який оновлює інформацію у відповідних вкладках. Така послідовність дій дозволяє керувати великими обсягами інформації, скорочуючи час на підготовку до подальшого аналізу, а використання окремого компонента для валідації імпортованих файлів сприяє забезпеченню коректності вхідних даних та мінімізації потенційних помилок під час роботи з ними.

Останнім сценарієм взаємодії з графічним інтерфейсом є звернення користувача до інтегрованого AI-асистента. Взаємодія компонентів у цьому сценарії ілюструється діаграмою послідовності, наведеною на рисунку 2.9.

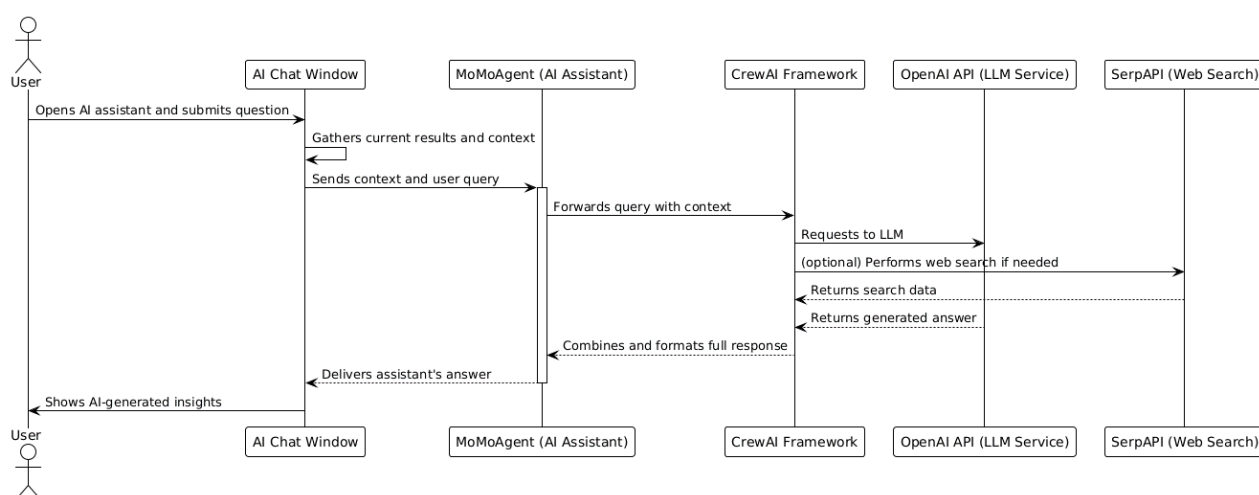


Рисунок 2.9 – Діаграма послідовності для взаємодії із AI-агентом

Загальна послідовність взаємодії з AI-асистентом у межах інтерфейсу MoMoGUI представлена у вигляді таких етапів:

- Користувач ініціює звернення до асистента, вводячи текстовий запит у відповідному вікні інтерфейсу.
- Запит передається разом із даними поточного аналізу (результати, параметри, прототип) у модуль AI-агента.

- Агент перетворює вхідні дані у формат, необхідний для зовнішньої мовної моделі.
- У разі необхідності уточнення агент додатково звертається до зовнішніх сервісів, зокрема пошукових API.
- Згенерована відповідь надходить до інтерфейсу та відображається у формі тексту, пояснення або рекомендації.

Такий підхід забезпечує користувачу можливість Last Mile Customization, дозволяючи мати динамічну підтримку користувача в процесі прийняття рішень та інтерпретації результатів аналізу.

2.5.1 Структура класів модуля MoMoGUI

Діаграма класів на рисунку 2.10 відображає логічну організацію основних елементів інтерфейсу MoMoGUI, їхню структуру та взаємодію з ядром програми. Вона дозволяє виявити, які компоненти відповідають за конкретні функціональні блоки, як дані передаються між представленням та обробкою, а також які об'єкти координують виконання користувацьких сценаріїв. Нижче подано стислий опис основних класів, представлених на діаграмі.

- **MainWindow** – координує роботу всіх підлеглих ресурсів інтерфейсу, відповідає за маршрутизацію подій, ініціалізацію вкладок і передачу даних до ядра MoMoModel.
- **SystemsTab** – надає засоби відображення, імпорту та базового редагування конфігурацій систем; містить таблицю підсистем та їх альтернатив.
- **PrototypePanel** – забезпечує формування користувацького прототипу, включаючи вибір характеристик і налаштування параметра метрики.
- **ResultsTab** – приймає результати аналізу від ядра і відтворює їх у вигляді сортированої таблиці з показниками схожості.
- **AssistantWindow** – інкапсулює інтерфейс діалогу з AI-агентом; взаємодіє з користувачем безпосередньо.

- MoMoAgent – програмний посередник між AssistantWindow та зовнішнім AI-сервером; формує запити, обробляє відповіді й доповнює їх контекстом поточного аналізу.

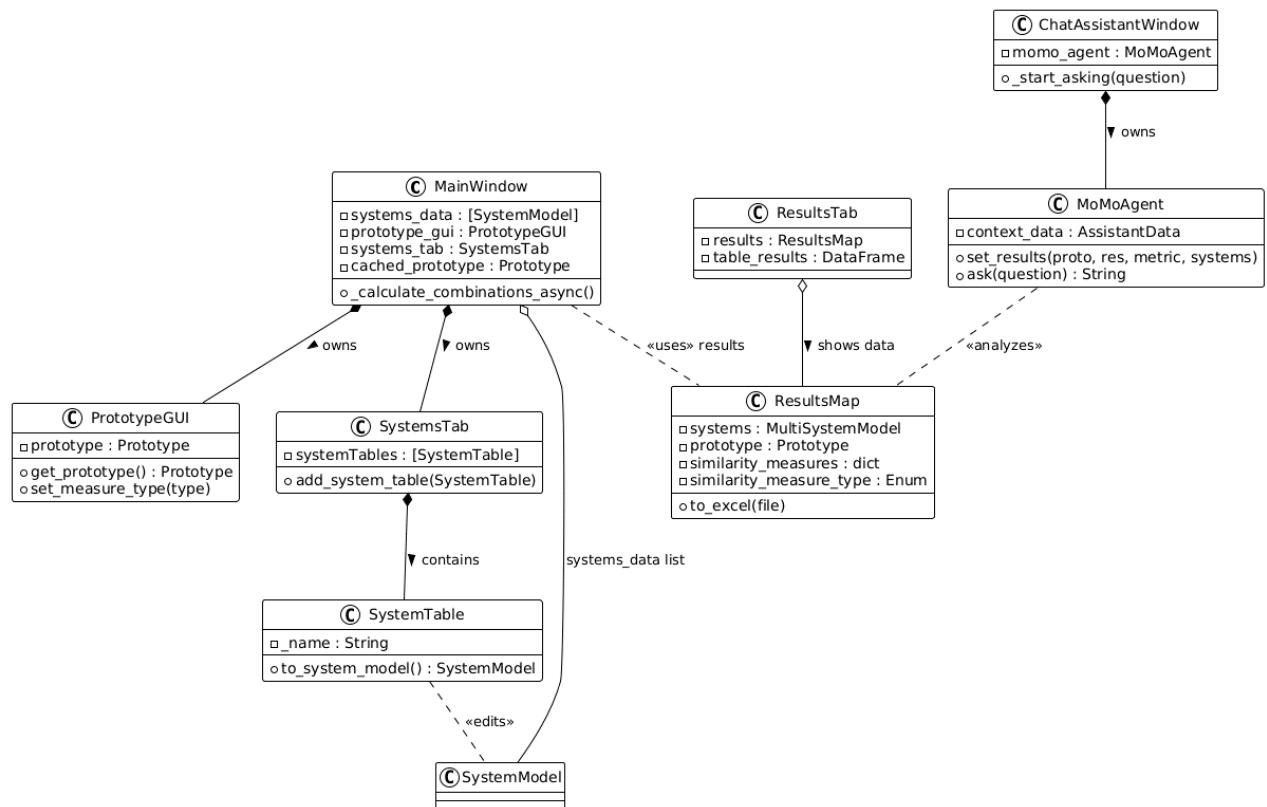


Рисунок 2.10 – UML-діаграма класів графічного інтерфейсу MoMoGUI

Загальна структура реалізує підхід, близький до Model-View-Controller (MVC), де View представлено класами вкладок інтерфейсу, Controller – логікою в MainWindow, а Model – зовнішнім обчислювальним ядром (MoMoModel), що функціонує незалежно від GUI-рівня.

Запропонована структура класів реалізує принципи низької зв’язаності та високої згуртованості: кожен графічний компонент має чітко окреслену зону відповідальності, а взаємодія з ядром здійснюється через мінімальний інтерфейс.

2.6 Висновок до другого розділу

У другому розділі кваліфікаційної роботи було здійснено повний аналіз архітектурної організації та технічної реалізації програмного забезпечення, створеного для оптимальної конфігурації комп'ютерних мереж із використанням методу морфологічного моделювання.

Було обґрунтовано вибір мови програмування, бібліотек, фреймворків і допоміжних інструментів, що застосовувалися в процесі розробки. Показано, що обрані технології, бібліотеки забезпечують достатню гнучкість, масштабованість у межах одного програмного середовища.

Особливу увагу приділено модулю обчислювального ядра (MoMo), який реалізує алгоритмічну частину методу морфологічного аналізу. У процесі розгляду було проаналізовано логіку його побудови, принципи побудови комбінацій, використання метрик схожості, а також взаємодію між його класами. Архітектурні рішення модуля базуються на принципах модульності, ізоляції обов'язків та мінімізації зв'язаності, що дозволяє застосовувати ядро незалежно від графічного середовища.

Окремий підрозділ було присвячено аналізу структури модуля графічного інтерфейсу (MoMoGUI). На основі UML-діаграм послідовності та класів розглянуто ключові сценарії використання, такі як створення конфігурацій, імпорт даних та взаємодія з AI-асистентом. Показано, що інтерфейс реалізовано згідно з принципами логічної ізоляції та розділення обов'язків між компонентами, що в подальшому може являтися плацдармом для інтеграції нових функцій.

Таким чином, реалізована система має чітко структуровану архітектуру, яка забезпечує узгоджену взаємодію між інтерфейсом, ядром та зовнішніми сервісами, та відповідає вимогам до сучасного програмного забезпечення як з точки зору технічної реалізації, так і з погляду масштабованості.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПЗ ДЛЯ ФОРМУВАННЯ ОПТИМАЛЬНОЇ КОНФІГУРАЦІЇ КОМП'ЮТЕРНОЇ МЕРЕЖІ МЕТОДОМ МОРФОЛОГІЧНОГО МОДЕЛЮВАННЯ

3.1 Структура програмного забезпечення МоМо та його модульність

У попередніх розділах уже неодноразово зазначалося, що програмне забезпечення для оптимальної конфігурації комп'ютерної мережі із використанням методу морфологічного моделювання було реалізовано у вигляді двох окремих, концептуально ізольованих компонентів – обчислювального ядра та графічного інтерфейсу користувача. Розділення обчислювальної логіки та рівня представлення підвищує гнучкість системи, спрощує її підтримку, полегшує процес модульного тестування та сприяє поступовому розвитку окремих компонентів.

Чітке розмежування ролей компонентів гарантує, що ядро системи відповідає виключно за математичні розрахунки та обробку даних, а інтерфейс – за організацію взаємодії з користувачем. Така архітектура дозволяє оновлювати або замінювати окремі модулі без ризику порушити стабільність решти системи і створює можливості для автономного використання ядра, наприклад у тому випадку, якщо захочеться розробити веб-версію програмного забезпечення МоМо.

Обчислювальне ядро системи МоМо Core реалізовано як самостійний Python-модуль, доступний для встановлення через стандартний пакетний менеджер `pip` [43]. Його архітектура спирається на чітку модульність: в межах ядра виділено компоненти для опису моделей систем та прототипів, генерації простору можливих конфігурацій і розрахунку метрик схожості.

Логіка обчислень побудована на загальній схемі морфологічного аналізу, яка дозволяє працювати з довільною кількістю альтернатив та ознак. Підтримка різних показників схожості, зокрема коефіцієнтів Жаккара та Соренсена-Дайса, дає змогу адаптувати методику під специфіку конкретної задачі. Алгоритм

автоматично формує всі допустимі комбінації з урахуванням несумісних пар значень, а результати повертає у форматі словників, що спрощує їхню подальшу обробку та візуалізацію.

Особливий акцент у MoMo Core зроблено на інкапсуляції та стабільності API: кожен клас не містить прихованих залежностей і взаємодіє з іншими компонентами через чітко визначені методи. Такий підхід значно полегшує написання юніт-тестів. У наступних підрозділах буде детально розглянуто реалізацію ключових класів.

Графічний інтерфейс MoMoGUI створено на основі фреймворку PyQt5 і організовано у вигляді окремих вкладок, кожна з яких відповідає за конкретний набір завдань: управління списком систем, формування прототипу, відображення результатів аналізу та взаємодію з AI-агентом.

Обмін інформацією між GUI і обчислювальним ядром здійснюється через внутрішні об'єктні моделі, що дозволяє користувачу формувати конфігураційні дані, запускати аналіз і отримувати результати у вигляді таблиць.

Вкладка інтеграції з AI-агентом реалізована як автономний текстовий чат, у якому користувач може вводити довільні запити щодо проведеного аналізу та отримувати роз'яснення або рекомендації. Завдяки ізольованій архітектурі цього модуля можлива підміна або розширення AI-сервісів без зміни основної логіки GUI.

Отже, проєкт побудовано на принципах чіткої модульності: обчислювальне ядро та графічний інтерфейс розмежовані як самостійні репозиторії, що забезпечує чисту архітектуру, спрощує незалежне оновлення й повторне використання коду та закладає основу для подальшого нарощування функціонала. У наступних підрозділах розглянемо кожен із компонентів детальніше.

3.2 Реалізація ядра програмного забезпечення МоМо

Модуль МоМо Core розроблено як незалежний Python-пакет, який іде разом із MIT ліцензією, що дозволяє безперешкодне використання коду із подальшою його модифікацією, як для комерційних так і для не комерційних проєктів (див. рис. 3.1) [41], призначений для формалізації процесу морфологічного моделювання та обчислення конфігураційних комбінацій систем з набору характеристик і альтернатив. Він виконує роль обчислювального двигуна: отримує на вхід опис систем у вигляді структур даних, формує повний простір комбінацій альтернатив, та обчислює показники схожості кожної конфігурації з довільно заданим прототипом.

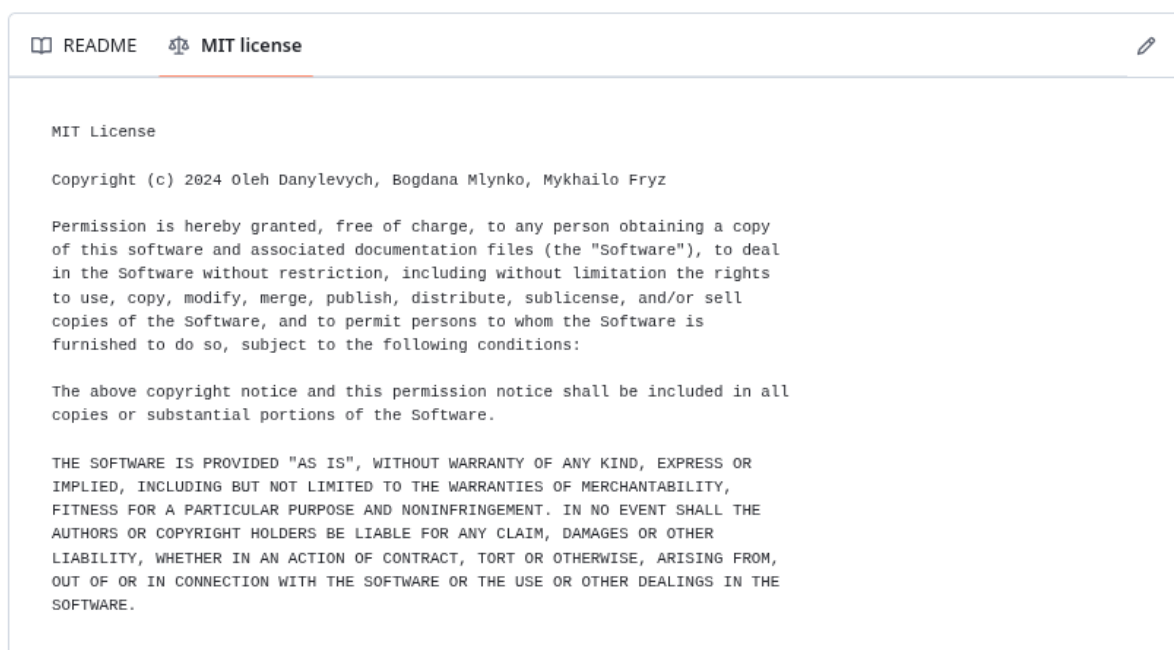


Рисунок 3.1 – MIT ліцензія ядра модуля МоМо

Архітектурно ядро побудовано навколо об'єктно-орієнтованої моделі, де кожна система представлена власним класом із чітко визначеними властивостями ознак та альтернатив.

Замість розробки власних контейнерів для даних, було вирішено взяти за основу контейнери по типу Series та DataFrame із бібліотеки Pandas. Деякі із

класів використовують ці контейнери, або в якості власних полів, або успадковуються від них, чим розширюють і надають потрібну функціональність. Функціональний інтерфейс ядра закриває клас `MoMoModel`, через який користувач задає набір систем, налаштовує прототип і обирає метрику схожості.

3.2.1 Структура класів модуля ядра МоМо

У цьому розділі розглядається логічна організація ядра функціоналу програмного забезпечення для оптимальної конфігурації комп'ютерної мережі із використанням морфологічного моделювання, що складається з чотирьох взаємопов'язаних класів: `SystemModel`, `MultiSystemModel`, `Prototype` та `MoMoModel`. Вони формують повний цикл морфологічного аналізу – від зберігання вихідних параметрів до обчислення показників схожості. Тепер розглянемо кожен із класів детальніше.

Почнемо з `SystemModel`. Він подібний до розширеного `pandas.DataFrame` і описує одну підсистему, де рядки відповідають ознакам, а стовпці – альтернативам. Код реалізації цього класу можна побачити у лістингу 1 додатка А.

Цей клас також використовує два приватних класи для своєї роботи - `_FeaturesAppender`, який додає до даних системи ознаки, та `_AlternativeAppender`, який відповідно додає альтернативи до даних, які зберігає конкретна система. Ці два класи в собі містять логіку, яка допомагає уникнути помилок при додаванні нових альтернатив чи ознак – відповідно.

Логічним продовженням є `MultiSystemModel`, який агрегує колекцію об'єктів `SystemModel` та генерує декартовий добуток альтернатив, інакше кажучи всі можливі комбінації. У коді (див. лістинг 2 додатка А) метод `generate_combinations` вибудовує ітератор конфігурацій, уникаючи створення зайвих проміжних структур, що знижує споживання пам'яті при великій кількості підсистем, ця оптимізація була додана в процесі тестування модуля

на великій кількості даних, до цього використовувався стандартний перебір із подальшим повернення всіх результатів користувачу. На жаль, останній підхід зазнав краху оскільки очевидно, що із збільшенням кількості альтернатив, кількість комбінацій зростає експоненціально.

Клас `MultiSystemModel` також виступає як клас-фабрика для створення об'єкта класу `Prototype`. Він створюється на основі наявних систем, а точніше їхніх ознак.

Далі – `Prototype`. Це підклас `pandas.Series`, який фіксує еталонну конфігурацію і служить відправною точкою для порівняння. У конструкторі (див. лістинг 3 додаток А) передбачено автоматичне приведення альтернатив до базового типу даних ядра (рядок, число або булеве значення), що уніфікує подальші побітові операції.

Завершальним елементом є `MoMoModel` – фасад, який інкапсулює всі попередні структури й надає єдину точку доступу до функціональності ядра. У середині нього відбувається ініціалізація `MultiSystemModel`, під'єднання `Prototype`, а також вибір метрики схожості. Метод `get_similarity_measures` (лістинг 4 додатка А) виконує ітеративний прорахунок коефіцієнтів схожості та повертає результат у вигляді `pandas.DataFrame` для подальшої візуалізації або експорту.

У підсумку, описана структура класів гарантує модульність та розширюваність `MoMo Core`. Чіткий поділ обов'язків спрощує тестування, підтримку й подальший розвиток алгоритмів, а також дозволяє інтегрувати пакет у зовнішні проєкти без втрати цілісності логіки.

3.2.2 Тестування модуля ядра MoMo

Модульне та юніт тестування [45, 46] є важливою частиною процесу розробки модуля ядра програмного забезпечення для оптимальної конфігурації комп'ютерної мережі із використанням методу морфологічного моделювання, оскільки воно дозволяє перевірити стабільність і коректність роботи ключових

компонентів незалежно від графічного інтерфейсу. Усі тести розміщено у каталозі tests/ (див. рис. 3.2) основного репозиторію, і вони охоплюють як окремі класи, так і фасадний інтерфейс MoMoModel.

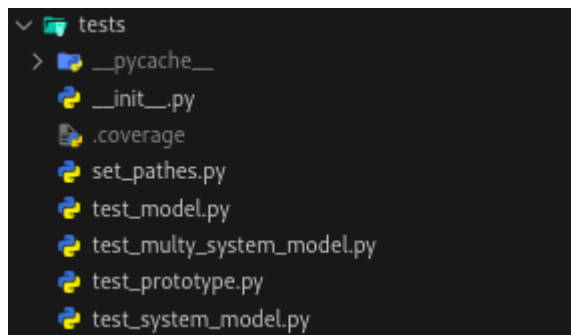


Рисунок 3.2 – Директорія із модульними тестами

Загалом структура тестів побудована за класичним принципом unit-тестування: кожен тестовий випадок перевіряє одну незалежну функцію або сценарій взаємодії. Зокрема, для класу SystemModel тести перевіряють, чи правильно створюється об'єкт при подачі допустимих та недопустимих параметрів, а також чи вірно працює механізм додавання нових альтернатив. Для MultiSystemModel основну увагу приділено генерації комбінацій – кількість згенерованих варіантів має збігатися з добутком кількостей альтернатив по кожній ознаці, що підтверджується на фіксованих вхідних даних. Крім того, перевіряється поведінка при відсутності альтернатив або конфліктних системах.

Особливе місце в тестуванні займає клас MoMoModel, який реалізує логіку обчислення метрик схожості. Для нього написано низку тестів, які перевіряють обчислення коефіцієнтів Жаккара та Соренсена-Дайса. Наприклад, якщо конфігурація повністю збігається з прототипом, результат повинен бути рівним 1.0, а якщо не має спільних ознак – 0. Також тестується поведінка методу get_similarity_measures при порожньому прототипі, відсутності систем або за неправильного типу метрики.

```

• (venv) danylevych@Fedora:~/Desktop/Projects/MoMo/momo/tests$ coverage run -m pytest
===== test session starts =====
platform linux -- Python 3.12.10, pytest-8.3.5, pluggy-1.5.0
rootdir: /home/danylevych/Desktop/Projects/MoMo
plugins: anyio-4.4.0, typeguard-4.3.0, docker-3.1.2
collected 40 items

test_model.py ..... [ 15%]
test_multy_system_model.py ..... [ 52%]
test_prototype.py ... [ 60%]
test_system_model.py ..... [100%]

===== 40 passed in 2.63s =====
• (venv) danylevych@Fedora:~/Desktop/Projects/MoMo/momo/tests$ coverage report
Name                                                                    Stmts  Miss  Cover
-----
/home/danylevych/Desktop/Projects/MoMo/momo/model.py                    67     24    64%
/home/danylevych/Desktop/Projects/MoMo/momo/prototype.py                 16      5    69%
/home/danylevych/Desktop/Projects/MoMo/momo/system_models/_item_apender.py 58     10    83%
/home/danylevych/Desktop/Projects/MoMo/momo/system_models/system_models.py 165     20    88%
TOTAL                                                                    306     59    81%
❖ (venv) danylevych@Fedora:~/Desktop/Projects/MoMo/momo/tests$

```

Рисунок 3.3 – Результати запуску тестів та результати покриття коду тестами

Рисунок 3.3 демонструє приклад запуску автоматизованих модульних тестів для ядра MoMo за допомогою фреймворку `pytest` у зв'язці з інструментом `coverage`. На першому етапі відображено результат проходження 40 тестових кейсів, які охоплюють логіку всіх основних модулів. Усі тести успішно пройдені, що підтверджує коректність реалізації.

У другій частині ілюстрації представлено зведений звіт `coverage`, який деталізує ступінь покриття вихідного коду тестами. Усього охоплено 306 операторів, з яких 81 % були протестовані.

3.3 Реалізація графічного інтерфейсу користувача

Графічний інтерфейс користувача є важливою складовою будь-якого програмного забезпечення, не винятком є і програмне забезпечення для оптимальної конфігурації комп'ютерної мережі із використанням методу морфологічного моделювання, адже саме він дозволяє взаємодіяти з алгоритмами ядра, не звертаючись безпосередньо до програмного коду. Завдяки GUI користувач може зручно створювати моделі, змінювати їхні параметри, виконувати аналіз та інтерпретувати результати за допомогою засобів візуалізації.

Під час розробки графічного інтерфейсу особливу увагу було приділено модульності, чіткому поділу відповідальностей між класами та забезпеченню швидкої реакції на дії користувача. Структура MoMo GUI відповідає наступній логіці: кожен окремий екран або панель представляє певний етап роботи, починаючи від завантаження вихідних даних і закінчуючи візуалізацією фінальних результатів або взаємодією з інтегрованим AI-асистентом.

3.3.1 Структура та архітектура графічного інтерфейсу користувача

Графічний модуль структурований у межах директорії gui репозиторію, що поділяється на три піддиректорії: styles, widgets та windows. Піддиректорія styles містить файли оформлення, які визначають стилі елементів інтерфейсу, і задають чорно білий стиль для всіх елементів програмного забезпечення для оптимальної конфігурації комп'ютерної мережі із використанням методу морфологічного моделювання. У піддиректорії widgets зосереджено окремі компоненти інтерфейсу, які повторно використовуються у різних вікнах. Зокрема, тут реалізовано такі елементи, як таблиця систем (SystemTable), панель налаштування прототипу (PrototypeGUI), вкладки результатів (ResultsTab), віджет чату AI-асистента тощо. Поділ на віджети сприяє модульності: кожен компонент відповідає за власну частину інтерфейсу і логіку взаємодії. Нарешті, піддиректорія windows містить основні вікна застосунку, а саме: стартове вікно і головне вікно програми. Класи StartWindow та MainWindow визначають структуру та поведінку відповідно стартового і головного вікон інтерфейсу.

Архітектура GUI тісно взаємодіє з ядром програми, проте залишається від нього незалежною. Взаємозв'язок здійснюється через структури даних та моделі, що визначені у модулі src. Головне вікно отримує від стартового вікна початкові дані (список систем або файл із результатами) і створює на їх основі відповідні об'єкти моделей. Обчислені результати (об'єкт ResultsMap) також передаються до GUI для відображення у вигляді таблиць. Цей підхід дає змогу

розділити бізнес-логіку, яка зосереджена в src та GUI частини, які відповідають за відображення даних та результатів.

Також, для роботи програми застосовується і подіє-орієнтований підхід програмування, а точніше MoMo використовує сигнали і слоти PyQt [47] для обробки дій користувача (наприклад, сигнал про додання або видалення вкладок систем, запуск обчислень тощо). У наступних пунктах кваліфікаційної роботи ми детальніше розглянемо кожну із компонент.

3.3.2 Стартове вікно MoMoGUI

Стартове вікно програми виконує роль початкової точки взаємодії користувача з морфологічним аналізом. Клас StartWindow, що наслідується від QMainWindow фреймворку PyQt5, відповідає за створення нової системи або завантаження існуючих даних із файлу. На рисунку 3.8 наведено структуру цього вікна: заголовок, далі коротко викладено призначення програми, після чого розміщено дві кнопки – Create та Load File.

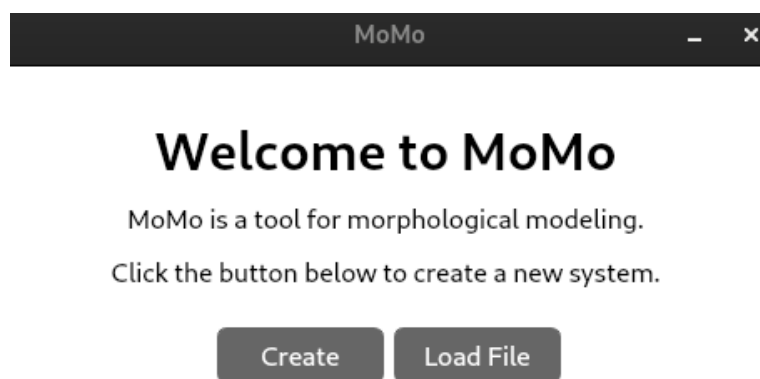


Рисунок 3.4 – Стартове вікно ПЗ MoMo

Натискання кнопки Create відкриває діалогове вікно, у якому користувач повідомляє назву нової системи; після підтвердження формується порожня модель (лістинг 1 додаток Б). У разі вибору Load File виконується імпорт Excel-файлу, що може містити як вихідні дані (таблиця ознак та альтернатив), так і

результати попереднього аналізу. Якщо в обраному документі присутній об'єкт ResultsMap, він автоматично завантажується для подальшого відображення в головному вікні; у протилежному випадку, за відсутності жодної системи, користувач отримує повідомлення про помилку вводу даних.

Після успішного створення нової системи або завантаження файлу стартове вікно закривається і передає інформацію класу MainWindow.

3.3.3 Головне вікно MoMoGUI

Головне вікно є центральним компонентом MoMoGUI, в якому відбувається основна взаємодія користувача з системою. Саме тут користувач конфігурує морфологічні моделі, виконує розрахунок можливих комбінацій та аналізує отримані результати. Клас MainWindow (наслідує QMainWindow) реалізує логіку цього вікна, включно зі створенням усіх вкладених панелей та віджетів (див. лістинг 2 додаток Б).

Архітектурно головне вікно розділене на дві основні області за допомогою горизонтального розділювача (QSplitter). Ліва частина інтерфейсу відведена під панель прототипу системи (Prototype Panel), а права – під панель систем і результатів (Systems/Results Panel). Така компоновка дозволяє одночасно налаштовувати прототип (вибір значень характеристик для цільової комбінації) та переглядати списки систем чи результати розрахунків.

Головне вікно загалом може перебувати у трьох станах залежно від етапу роботи:

1. порожнє вікно (не додано жодної системи);
2. вікно із завантаженими системами та панеллю прототипу;
3. вікно з відображенням результатів аналізу. Розглянемо кожен із цих станів окремо.

У порожньому стані головне вікно ініціалізується без попередньо завантажених даних, тобто список систем на момент запуску є порожнім. Такий сценарій передбачено для випадків, коли користувач не виконав імпорт на

стартовому етапі або видалив усі системи із списку систем в головному вікні. У цьому режимі права частина інтерфейсу відображає службове повідомлення (див. рис. 3.5), під яким розміщено кнопки: Add та Load File. Цей елемент реалізовано у вигляді окремого віджета EmptySystemsTab (див. лістинг 3 додаток Б), спеціально розробленого для відображення порожнього стану.

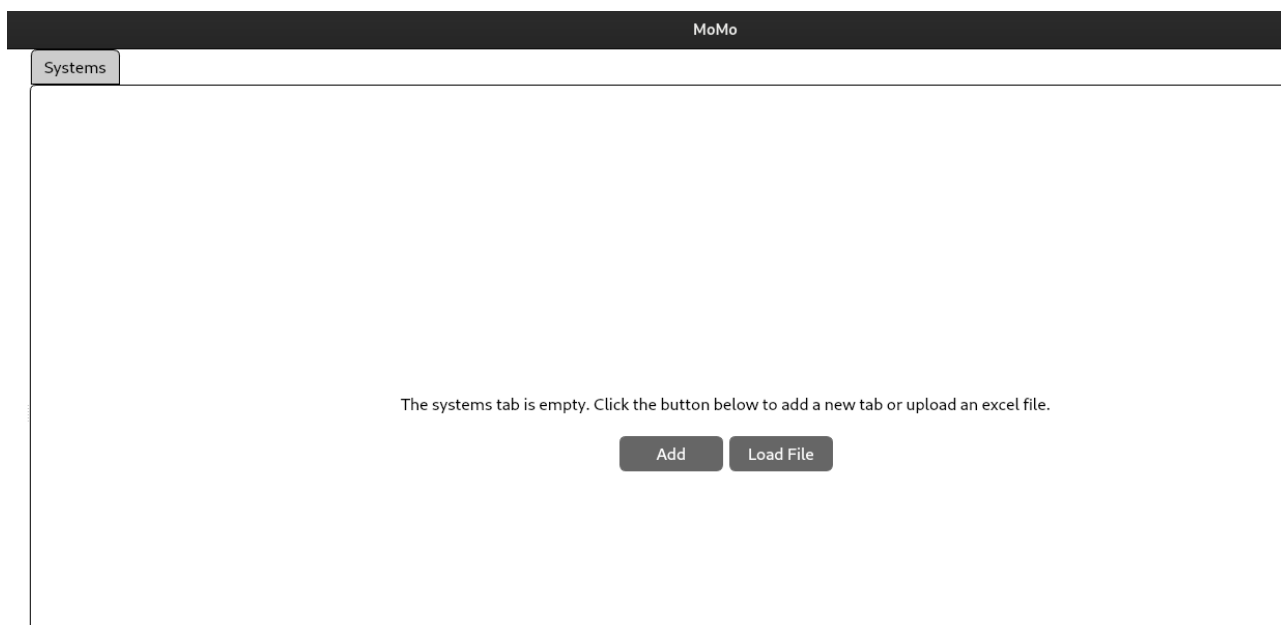


Рисунок 3.5 – Головне вікно у порожньому стані

Ліва панель прототипу в цьому режимі залишається прихованою або мінімізованою, оскільки немає жодної системи, з якою можна було б здійснювати порівняння. Програма автоматично налаштовує ширину розділювача таким чином, щоб уся доступна область інтерфейсу була відведена під вкладку Systems, а панель прототипу займала 0% ширини.

Користувач має змогу натиснути кнопку Add, щоб викликати діалогове вікно введення назви нової системи. Після підтвердження вміст вікна змінюється: порожній інтерфейс замінюється на повноцінну вкладку з таблицею ознак і альтернатив. Натискання Load File імпортує результати або списки систем із Excel-документа.

У другому стані (див. рис. 3.6), який активується після створення або завантаження щонайменше однієї системи, головне вікно переходить у

повноцінний робочий режим. У цьому випадку інтерфейс ініціалізує віджет SystemsTab (див. лістинг 4 додаток Б), який замінює попередній EmptySystemsTab і відкриває доступ до вкладок із системами.

Права частина головного вікна відображає список активних систем у вигляді таблиць альтернатив та ознак, організованих у вкладкову структуру (на рисунку можна побачити дві системи test та test2). У верхній частині розташована панель керування з кнопками Add new system, Read from Excel, Save all, Save selected та Delete, що дозволяють створювати, зберігати або видаляти системи безпосередньо в межах цього інтерфейсу.

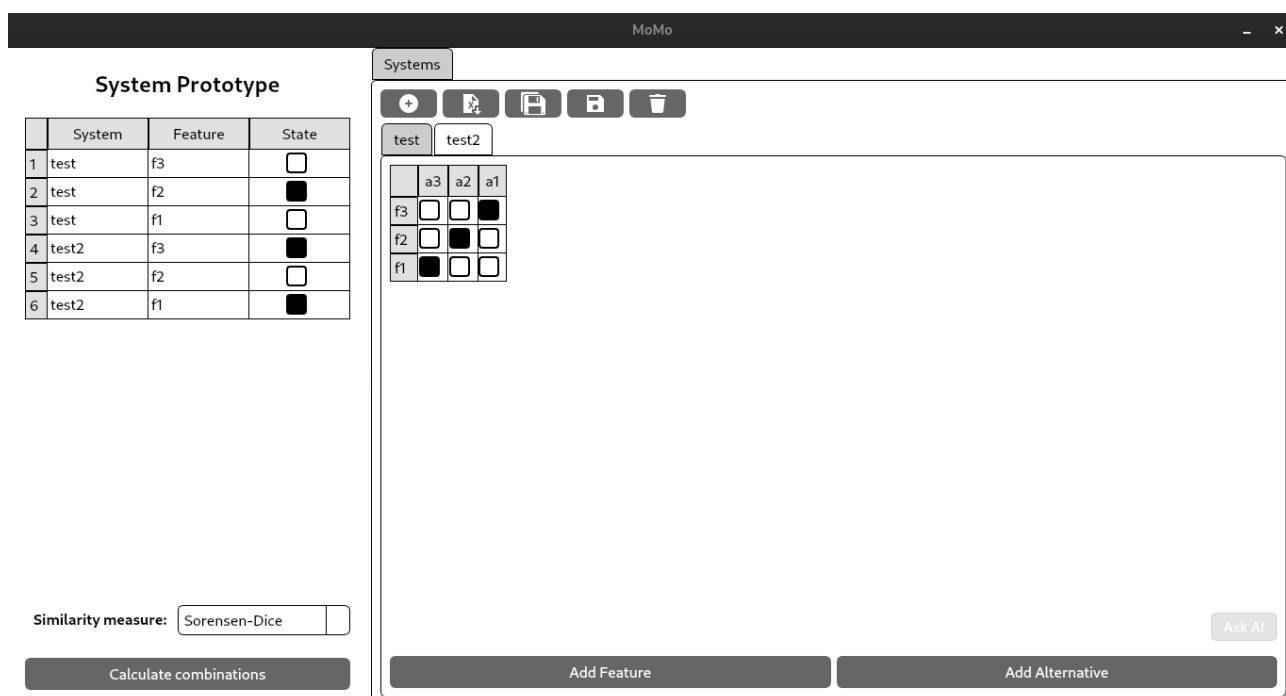


Рисунок 3.6 – Головне вікно програми у стані із заповненими системами

На рисунку 3.6 зображено інтерфейс головного вікна у другому стані: видно відкриту вкладку системи, панель інструментів і активну ліву частину з прототипом.

Ліва панель у цьому стані активується та відображає прототипову конфігурацію (яка постійно оновлюється при зміні систем). Цю частину інтерфейсу реалізовано віджетом PrototypeGUI (див. лістинг 5 додаток Б), що формує таблицю з усіма характеристиками, які використовуються у системах, і

дозволяє вказати значення (альтернативи), які складають прототип. Для кожної характеристики передбачено прапорець (checkbox), що визначає, чи включено цю характеристику до поточного прототипу. Над таблицею розташований заголовок, а під таблицею – випадючий список для вибору методу обчислення подібності (Jaccard або Sorensen-Dice) і кнопка Calculate combinations, яка запускає аналіз.

Після редагування прототипу користувач може розпочати аналіз, натиснувши кнопку обчислення. Програма формує повний простір можливих комбінацій альтернатив (використовуючи всі активні системи), порівнює кожну конфігурацію з прототипом і обчислює значення метрики схожості. Процес виконується асинхронно – це дозволяє зберегти реактивність інтерфейсу й уникнути його блокування під час виконання обчислень. По завершенні аналізу створюється нова вкладка з результатами, а інтерфейс автоматично переходить до наступного стану.

У третьому стані (див. рис. 3.7) головне вікно переходить до режиму представлення результатів морфологічного аналізу. Після завершення розрахунку, ініційованого користувачем, програма створює нову вкладку типу ResultsTab (див. лістинг 6 додаток Б). Таким чином, окрім вкладки Systems, в інтерфейсі з'являється додаткова область, призначена для виводу сформованих конфігурацій та відповідних значень метрик подібності.

Віджет ResultsTab, що відповідає за цей режим, реалізує таблицю результатів на основі компонента QTableWidgetItem. У верхній частині вкладки розміщено заголовок і кнопку Save to Excel, яка відкриває діалогове вікно для збереження результатів у форматі Excel. Таблиця містить по одному стовпцю для кожної характеристики системи, а також додатковий стовпець Similarity, у якому відображається значення вибраної метрики. Кожен рядок у таблиці відповідає одній згенерованій конфігурації, а останній стовпець показує ступінь її схожості з прототипом.

System Prototype

	System	Feature	State
1	test	f3	☐
2	test	f2	☒
3	test	f1	☐
4	test2	f3	☒
5	test2	f2	☐
6	test2	f1	☒

Similarity measure:

Results

	test	test2	Similarity
1	a3	a2	0.5
2	a2	a2	0.5
3	a1	a2	0.5
4	a3	a3	0.0
5	a3	a1	0.0
6	a2	a3	0.0
7	a2	a1	0.0
8	a1	a3	0.0
9	a1	a1	0.0

Рисунок 3.7 – Головне вікно програми у стані результатів

Сортування рядків таблиці за замовчуванням виконується за спаданням значення схожості: найподібніші до заданого прототипу конфігурації виводяться на початку списку. Це дозволяє швидко зорієнтуватися у структурі результатів та виділити найрелевантніші варіанти для подальшого аналізу або прийняття рішень. Якщо протягом одного сеансу роботи з програмою MoMo буде ініційовано кілька розрахунків (наприклад, із різними версіями прототипу або зміненими системами), кожен новий набір результатів буде представлено окремою вкладкою з автоматично згенерованою назвою.

Для того, аби зберегти дані програмне забезпечення, що розроблялося для розв'язання задачі оптимальної конфігурації комп'ютерної мережі із використанням методу морфологічного моделювання, використовує спеціально визначений формат даних (див. рис. 3.8).

A) Metadata

	A	B	C	D
1	file_type	version	date	similarity_measure_type
2	MoMo_Results	1.0	2025-05-29 22:00:42	sorensen_dice
3				
4				

B) Similarity_Results

	A	B	C
1	test	test2	Similarity
2	a3	a2	0.5
3	a1	a2	0.5
4	a2	a2	0.5
5	a3	a3	0
6	a3	a1	0
7	a2	a3	0
8	a2	a1	0
9	a1	a3	0
10	a1	a1	0

C) Prototype

	A	B	C	D
1			0	
2	test	f3	0	
3		f2	1	
4		f1	0	
5	test2	f3	1	
6		f2	0	
7		f1	1	
8				
9				
10				
11				

D) Systems

	A	B	C	D
1		a3	a2	a1
2	f3	0	0	1
3	f2	0	1	0
4	f1	0	0	0
5				

Рисунок 3.8 – Збережені результати: А) вкладка Metadata; Б) вкладка Similarity_Results; С) вкладка Prototype; Д) вкладки із системами

Вкладка Metadata зберігає в собі всю порібну інформацію для подальшого розпаршування результатів в ПЗ МоМо. Вкладка Similarity_Results зберігає в собі дані, які користувач бачив у вкладці ResyltTab. Сторінка Prototype відповідає за збереження стану прототипу, який привів до цих результатів, і фінальними є вкладки систем, які зберігають в собі інформацію про ознаки та альтернативи, а також їхні стани.

3.3.4 Вікно AI-асистента в MoMoGUI

Окремим компонентом графічного інтерфейсу МоМо є вікно інтегрованого AI-асистента, що надає користувачеві last mile customization. Призначення цього модуля полягає в поясненні результатів аналізу, формулюванні порад щодо редагування прототипу, а також у наданні відповідей на загальні запитання, що стосуються методу морфологічного моделювання. Доступ до AI-агента здійснюється через плаваючу кнопку Ask AI, розташовану у правому нижньому куті головного вікна. Натискання на цю

кнопку відкриває окреме вікно чату, що функціонує як незалежний інтерфейс на основі класу ChatAssistantWindow.

Клас ChatAssistantWindow є нащадком QWidget і включає всі необхідні елементи для реалізації чат-взаємодії. Інтерфейс вікна візуально наближений до традиційних чат-застосунків по типу ChatGPT чи Claude AI (див. рис. 3.9). У нижній частині розміщено текстове поле для введення запитів природною мовою.

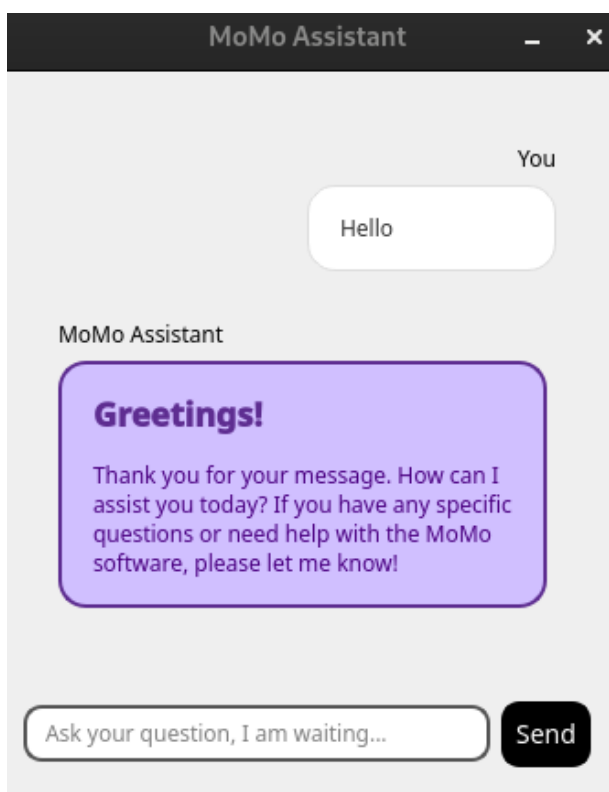


Рисунок 3.9 – Вигляд вікна AI помічника

Після введення запиту, інтерфейс перемикається на режим активного діалогу. Запит користувача відображається у вигляді діалогової «бульбашки», після чого запускається обробка – у вікні з’являється індикатор активності «AI is thinking...». Коли відповідь сформовано, вона виводиться у вигляді нового повідомлення в чаті. Таким чином реалізовано двосторонню взаємодію між користувачем і асистентом.

Технічно AI-агент може «переглядати» результати і стан головного вікна. Однак ці деталі приховані від користувача – він взаємодіє лише з чат-інтерфейсом.

AI-асистент здатен пояснювати значення коефіцієнтів схожості, допомагати у виборі метрики, інтерпретувати вплив окремих альтернатив на загальну модель, пропонувати варіанти покращення конфігурації чи надавати додаткові інсайти використовуючи пошук.

3.4 Демонстрація розв’язання задачі оптимальної конфігурації комп’ютерної мережі із використання ПЗ МоМо

Для демонстрації практичної цінності системи MoMoGUI розглянемо реальний сценарій: IT-відділ середньостатистичної компанії планує оновити мережеву інфраструктуру, щоб забезпечити зростаючі вимоги до пропускної здатності, надійності та енергоефективності. Перед інженерами постало завдання вибору оптимальної комбінації обладнання, що включає маршрутизатор, комутатор, брандмауер та канал передачі даних, з урахуванням жорстких бюджетних і технологічних обмежень. Для автоматизації цього процесу було застосовано інструментарій морфологічного моделювання МоМо, що дозволяє формалізувати вимоги, побудувати повний простір можливих конфігурацій та обрати найбільш відповідний варіант.

Значущість обраного підходу полягає в його здатності врахувати численні технічні характеристики та обмеження одночасно, а також забезпечити прозорість прийняття рішення. Реальні моделі обладнання, включені до аналізу, відрізняються за продуктивністю, вартістю, рівнем безпеки і витратами на експлуатацію, що робить задачу комбінування доволі складною. Використовуючи MoMoGUI, інженери можуть задати порогові значення для кожної з ознак, швидко згенерувати всі можливі конфігурації та обчислити їхню відповідність ідеальному прототипу, що суттєво прискорює процес прийняття рішення та підвищує його обґрунтованість.

У моделі комп'ютерної мережі підсистема «Маршрутизатор» відповідає за спрямування трафіку між сегментами і вихід у зовнішні мережі. У задачі були обрані три типові рішення, що відрізняються за цими параметрами:

- Cisco ISR4331: маршрутизатор з пропускнуою здатністю до 1 Gbps; має високу стабільність та помірне енергоспоживання.
- Juniper MX5: маршрутизатор з пропускнуою здатністю до 10 Gbps; володіє широким набором функцій, але характеризується значними витратами та підвищеним енергоспоживанням.
- Mikrotik CCR1009: бюджетне рішення середнього рівня з пропускнуою здатністю до 3 Gbps; має менший рівень відмовостійкості та відносно недорогий.

Оцінювання альтернатив виконано за такими чотирма ознаками:

- Пропускна здатність ($TP \geq 5 \text{ Gbps}$) – визначає здатність маршрутизатора обробляти великі обсяги даних без затримок.
- Вартість ($\text{Cost} \leq 2000 \text{ USD}$) – бюджетна межа, що встановлює максимальну прийнятну ціну придбання.
- Надійність ($\text{Rel} \geq 4$) – умовна шкала від 1 до 5 для оцінки безвідмовної роботи та терміну служби.
- Енергоспоживання ($\text{Power} \leq 100 \text{ W}$) – критеріальний показник для оцінки експлуатаційних витрат і екологічності рішення.

У таблиці 3.1 можна побачити вигляд підсистеми «Маршрутизатор» в бінаризованому вигляді, тобто готовому до запуску в MoMo

Таблиця 3.1 – Бінаризовані ознаки для підсистеми «Маршрутизатор»

Альтернатива	$TP \geq 5?$	$\text{Cost} \leq 2000?$	$\text{Rel} \geq 4?$	$\text{Pwr} \leq 100?$
Cisco ISR4331	0	1	1	1
Juniper MX5	1	0	1	0
Mikrotik CCR1009	0	1	0	1

У проміжному підсумку слід зазначити, що жодна з альтернатив не задовольняє одночасно всі чотири вимоги. Cisco ISR4331 виявляється

економічно вигідним і енергоефективним, але не досягає заданого порогу пропускної здатності. Juniper MX5 забезпечує необхідну швидкість, але перевищує бюджет і ліміт за споживанням. Надоступніший варіант – Mikrotik CCR1009 – відповідає лише двом із чотирьох критеріїв. Це підкреслює необхідність компромісного підходу при виборі маршрутизатора залежно від пріоритету критеріїв.

Комутатор відіграє центральну роль у локальному переміщенні мережевого трафіку та визначає масштабованість інфраструктури. При проектуванні мережі є важливим знайти баланс між достатньою кількістю портів, підтримкою живлення по Ethernet (PoE), високою пропускною здатністю та прийнятною вартістю.

У нашому завданні було обрано три моделі комутаторів, які відрізняються набором функцій та ціновим діапазоном. Cisco Catalyst 9300 пропонує 24 порти з повною підтримкою PoE і загальну пропускну здатність до 80 Gbps, але його вартість значно перевищує бюджетні обмеження. TP-Link TL-SG3210 є бюджетним рішенням з восьми портів та без PoE, яке підходить для невеликих офісів із мінімальними вимогами до швидкодії. Між ними знаходиться Ubiquiti US-16, що надає 16 портів з PoE та пропускну здатність 70 Gbps за значно доступнішою ціною.

Після ознайомлення з характеристиками альтернатив переходимо до критичних ознак, за якими проводиться їхня бінарна оцінка. По-перше, кількість портів ($\text{Ports} \geq 16$) визначає гнучкість майбутніх розширень мережі; по-друге, наявність PoE є ключовою для живлення периферійних пристроїв без додаткових джерел живлення. Висока сумарна пропускна здатність ($\text{TP} \geq 50$ Gbps) гарантує відсутність «вузьких місць» при пікових навантаженнях, а обмеження вартості ($\text{Cost} \leq 500$ USD) забезпечує дотримання бюджету без значних компромісів у функціональності.

Таблиця 3.2 – Бінаризовані ознаки для підсистеми «Комутатор»

Альтернатива	Ports ≥ 16 ?	PoE?	TP ≥ 50 ?	Cost ≤ 500 ?
Cisco Catalyst 9300	1	1	1	0
TP-Link TL-SG3210	0	0	0	1
Ubiquiti US-16	1	1	1	1

У результаті аналізу бачимо, що серед розглянутих варіантів лише Ubiquiti US-16 одночасно задовольняє всі чотири ознаки, що робить його пріоритетним кандидатом для включення до фінальної конфігурації. При цьому Cisco Catalyst 9300 демонструє відмінні технічні параметри, але не відповідає бюджетним обмеженням, а TP-Link TL-SG3210 задовольняє бюджет і пропускну здатність, але поступається за кількістю портів і PoE.

Підсистема «Брандмауер» забезпечує захист мережі від несанкціонованого доступу, тому ключовими характеристиками є пропускна здатність, рівень безпеки, затримка та вартість. Розглянуто три рішення:

- pfSense є безкоштовним та гнучким, але має занижений рівень захисту;
- Cisco ASA 5506-X має високий захист і мінімальну затримку, але перевищує бюджет і не витримує вимоги пропускну здатності;
- FortiGate 60E являє собою збалансоване комерційне рішення з достатньою швидкістю та захистом, проте дорожче.

Наступним кроком є визначення ознак, а точніше їхніх бінарних відповідників. По-перше, пропускна здатність (TP ≥ 3 Gbps) гарантує швидку обробку трафіку. По-друге, вартість (Cost ≤ 500 USD) обмежує капітальні витрати. По-третє, рівень безпеки (Sec ≥ 5) відображає здатність брандмауера протидіяти сучасним загрозам. І нарешті, затримка (Latency ≤ 2 ms) визначає вплив на час відгуку мережі.

Таблиця 3.3 – Бінаризовані ознаки для підсистеми «Брандмауер»

Альтернатива	$TP \geq 3?$	$Cost \leq 500?$	$Sec \geq 5?$	$Lat \leq 2?$
pfSense	1	1	0	1
Cisco ASA 5506-X	0	0	1	1
FortiGate 60E	1	0	1	1

Проміжні висновки вказують, що жодне з рішень не задовольняє всі чотири критерії одночасно. pfSense є найбільш економічним та енерго-ефективним, але його рівень безпеки не досягає порогу. Cisco ASA 5506-X гарантує найвищий рівень захисту та найнижчу затримку, проте не відповідає вимогам бюджету та пропускної здатності. FortiGate 60E забезпечує потрібний рівень безпеки та пропускну здатність, однак перевищує ліміт бюджету. Отже, остаточний вибір потребуватиме компромісу між бюджетом і рівнем захисту.

Канал передачі даних відповідає за фізичне переміщення інформації між пристроями і має бути надійним, швидким та економічно виправданим. Розглянуті альтернативи включають традиційний мідний кабель, оптоволокну та бездротове середовища для передачі.

Серед альтернатив CAT6 Copper вирізняється низькою вартістю та поширеністю, але обмеженою пропускну здатністю і надійністю. Fiber Optic забезпечує високу швидкість передачі та максимальну стабільність, хоча коштує дорожче. Wireless 5 GHz пропонує гнучкість розгортання та відсутність кабельних обмежень, проте має найнижчі показники надійності та пропускної здатності.

Для бінарної оцінки було визначено чотири ознаки. Пропускна здатність ($TP \geq 5 \text{ Gbps}$) необхідна для високошвидкісного обміну даними. Вартість ($Cost \leq 1 \text{ USD/м}$) обмежує інвестиції в прокладку каналу. Надійність ($Rel \geq 4$) відображає стабільність передачі та втрати сигналу. Безпроводність (Wireless?) вказує на відсутність фізичного з'єднання, що може бути перевагою при мобільному або тимчасовому розгортанні.

Таблиця 3.4 – Бінаризовані ознаки для підсистеми «Канал передачі даних»

Альтернатива	$TP \geq 5?$	$Cost \leq 1?$	$Rel \geq 4?$	Wireless?
CAT6 Copper	0	1	0	0
Fiber Optic	1	0	1	0
Wireless 5 GHz	0	0	0	1

З отриманих даних видно, що Fiber Optic є єдиним рішенням, яке задовольняє критеріям пропускної здатності та надійності, але перевищує ліміт за вартістю. CAT6 Copper є бюджетним варіантом з обмеженою продуктивністю, а Wireless 5 GHz відзначається мобільністю, але меншою стабільністю. Остаточний вибір залежатиме від пріоритету між вартістю, швидкістю та надійністю каналу.

Після визначення альтернатив і їхніх ознак наступним кроком є побудова прототипу – еталонної конфігурації, з якою порівнюватимуться всі згенеровані поєднання. У нашому прикладі прототип відповідає максимальним очікуванням ІТ-відділу: для кожної підсистеми встановлено бажаний стан усіх чотирьох критеріїв, крім тих, які зазнали технічних або бюджетних компромісів. Так, для маршрутизатора критерію « $TP \geq 5 \text{ Gbps}$ » відведено значення 0, оскільки навіть високопріоритетні моделі не гарантують одночасного задоволення всіх вимог до пропускної здатності та інших показників, тоді як інші три ознаки зафіксовано як виконані. Аналогічні рішення ухвалено для комутаторів, брандмауерів та каналів передачі даних: у прототипі відображено найважливіші для бізнесу характеристики, що дозволяє сфокусуватися на ключових параметрах під час аналізу.

На рисунку 3.10 наведено вікно із прототипом системи, у якому затемнені клітини відповідають активним (1) ознакам, а пусті – неактивним (0).

	System	Feature	State
1	Routers	TP $\geq$ 5?	☐
2	Routers	Cost $\leq$ 2000?	☒
3	Routers	Rel $\geq$ 4?	☒
4	Routers	Pwr $\leq$ 100?	☒
5	Switches	Ports $\geq$ 16?	☒
6	Switches	PoE?	☒
7	Switches	TP $\geq$ 50?	☐
8	Switches	Cost $\leq$ 500?	☒
9	Firewalls	TP $\geq$ 3?	☒
10	Firewalls	Cost $\leq$ 500?	☒
11	Firewalls	Sec $\geq$ 5?	☒
12	Firewalls	Lat $\leq$ 2?	☐
13	Data Links	TP $\geq$ 5?	☒
14	Data Links	Cost $\leq$ 1?	☒
15	Data Links	Rel $\geq$ 4?	☒
16	Data Links	Wireless?	☒

Рисунок 3.10 – Прототип комп'ютерної мережі

Після натискання Calculate combinations MoMoGUI автоматично згенерувала всі 81 можливу конфігурацію ($3 \times 3 \times 3 \times 3$) та відобразила їх у вкладці Results (див. рисунок 3.17). Першими у списку з'явилися варіанти зі значенням схожості близько 0,64, що свідчить про відповідність семи з одинадцяти обраних ознак прототипу. У цих конфігураціях маршрутизатор представлено моделлю Cisco ISR4331, а комутатор – Cisco Catalyst 9300 або трохи бюджетнішим Ubiquiti US-16; брандмауер – pfSense, а канал передачі даних – CAT6 Copper чи Wireless 5 GHz.

Ключовим фактором такого результату є поєднання недорогого, але надійного маршрутизатора з економічним брандмауером і доступним каналом передачі.

System Prototype

	System	Feature	State
1	Routers	TP ≥ 5?	☐
2	Routers	Cost ≤ ...	☒
3	Routers	Rel ≥ 4?	☒
4	Routers	Pwr ≤ 100?	☒
5	Switches	Ports ≥ 16?	☒
6	Switches	PoE?	☒
7	Switches	TP ≥ 50?	☐
8	Switches	Cost ≤ ...	☒
9	Firewalls	TP ≥ 3?	☒
10	Firewalls	Cost ≤ ...	☒
11	Firewalls	Sec ≥ 5?	☒
12	Firewalls	Lat ≤ 2?	☐
13	Data Links	TP ≥ 5?	☒
14	Data Links	Cost ≤ 1?	☒
15	Data Links	Rel ≥ 4?	☒
16	Data Links	Wireless?	☒

Similarity measure: Jaccard

Calculate combinations

Results

	Routers	Switches	Firewalls	Data Links	Similarity
1	Cisco ISR4331	Cisco Catalyst 9300	pfSense	CAT6 Copper	0.6363636363636364
2	Cisco ISR4331	Cisco Catalyst 9300	pfSense	Wireless 5 GHz	0.6363636363636364
3	Cisco ISR4331	Cisco Catalyst 9300	FortiGate 60E	CAT6 Copper	0.6363636363636364
4	Cisco ISR4331	Cisco Catalyst 9300	FortiGate 60E	Wireless 5 GHz	0.6363636363636364
5	Cisco ISR4331	Ubiquiti US-16	pfSense	CAT6 Copper	0.6363636363636364
6	Cisco ISR4331	Ubiquiti US-16	pfSense	Wireless 5 GHz	0.6363636363636364
7	Cisco ISR4331	Ubiquiti US-16	FortiGate 60E	CAT6 Copper	0.6363636363636364
8	Cisco ISR4331	Ubiquiti US-16	FortiGate 60E	Wireless 5 GHz	0.6363636363636364
9	Mikrotik CCR1009	Cisco Catalyst 9300	pfSense	CAT6 Copper	0.6363636363636364
10	Mikrotik CCR1009	Cisco Catalyst 9300	pfSense	Wireless 5 GHz	0.6363636363636364
11	Mikrotik CCR1009	Cisco Catalyst 9300	FortiGate 60E	CAT6 Copper	0.6363636363636364
12	Mikrotik CCR1009	Cisco Catalyst 9300	FortiGate 60E	Wireless 5 GHz	0.6363636363636364
13	Mikrotik CCR1009	Ubiquiti US-16	pfSense	CAT6 Copper	0.6363636363636364
14	Mikrotik CCR1009	Ubiquiti US-16	pfSense	Wireless 5 GHz	0.6363636363636364
15	Mikrotik CCR1009	Ubiquiti US-16	FortiGate 60E	CAT6 Copper	0.6363636363636364
16	Mikrotik CCR1009	Ubiquiti US-16	FortiGate 60E	Wireless 5 GHz	0.6363636363636364
17	Cisco ISR4331	Cisco Catalyst 9300	pfSense	Fiber Optic	0.6086956521739131
18	Cisco ISR4331	Cisco Catalyst 9300	FortiGate 60E	Fiber Optic	0.6086956521739131
19	Cisco ISR4331	Ubiquiti US-16	pfSense	Fiber Optic	0.6086956521739131
20	Cisco ISR4331	Ubiquiti US-16	FortiGate 60E	Fiber Optic	0.6086956521739131
21	Mikrotik CCR1009	Cisco Catalyst 9300	pfSense	Fiber Optic	0.6086956521739131
22	Mikrotik CCR1009	Cisco Catalyst 9300	FortiGate 60E	Fiber Optic	0.6086956521739131
23	Mikrotik CCR1009	Ubiquiti US-16	pfSense	Fiber Optic	0.6086956521739131
24	Mikrotik CCR1009	Ubiquiti US-16	FortiGate 60E	Fiber Optic	0.6086956521739131
25	Cisco ISR4331	Cisco Catalyst 9300	Cisco ASA 5506-X	CAT6 Copper	0.5714285714285714
26	Cisco ISR4331	Cisco Catalyst 9300	Cisco ASA 5506-X	Wireless 5 GHz	0.5714285714285714

Save to Excel

Рисунок 3.11 – Результати виконання морфологічного аналізу

Як тільки в канал передачі даних підставити дорожчий, але надійніший Fiber Optic, схожість падає до $\sim 0,61$, оскільки порушується ціновий поріг. Ще нижче, близько 0,57, розташувалися комбінації з брандмауером Cisco ASA 5506-X: він чудово закриває параметр безпеки, зате «просідає» за вартістю та пропускну здатністю. Таким чином, жодна конфігурація не дотягнула до ідеальної одиниці, але верхівка списку показує, де сьогодні проходить практичний компроміс між швидкістю, бюджетом і захистом.

Після ознайомлення з результатами морфологічного аналізу користувач може звернутися до вбудованого AI-асистента за додатковими консультаціями. Наприклад, у нашому прикладі ми ввели запит: «Який із перелічених результатів найкраще підійде для мережі IT-відділу середньостатистичної компанії (15-20 осіб)?». Асистент оцінює перелік конфігурацій, звертаючи увагу на відповідність ключовим показникам прототипу та практичним вимогам масштабів організації.

У відповідь агент виводить повноцінний текстовий аналіз (рис. 3.12), який починається з узагальнення рівня сумісності конфігурації: підкреслюється, що, хоча маршрутизатор Cisco ISR4331 не відповідає порогу пропускну

здатності, загальна збалансованість компонентів дозволяє забезпечити надійну роботу мережі. Далі AI-асистент детально описує переваги вибраних пристроїв, зокрема комутатора Cisco Catalyst 9300 через його PoE-підтримку та енергоефективного брандмауера pfSense.

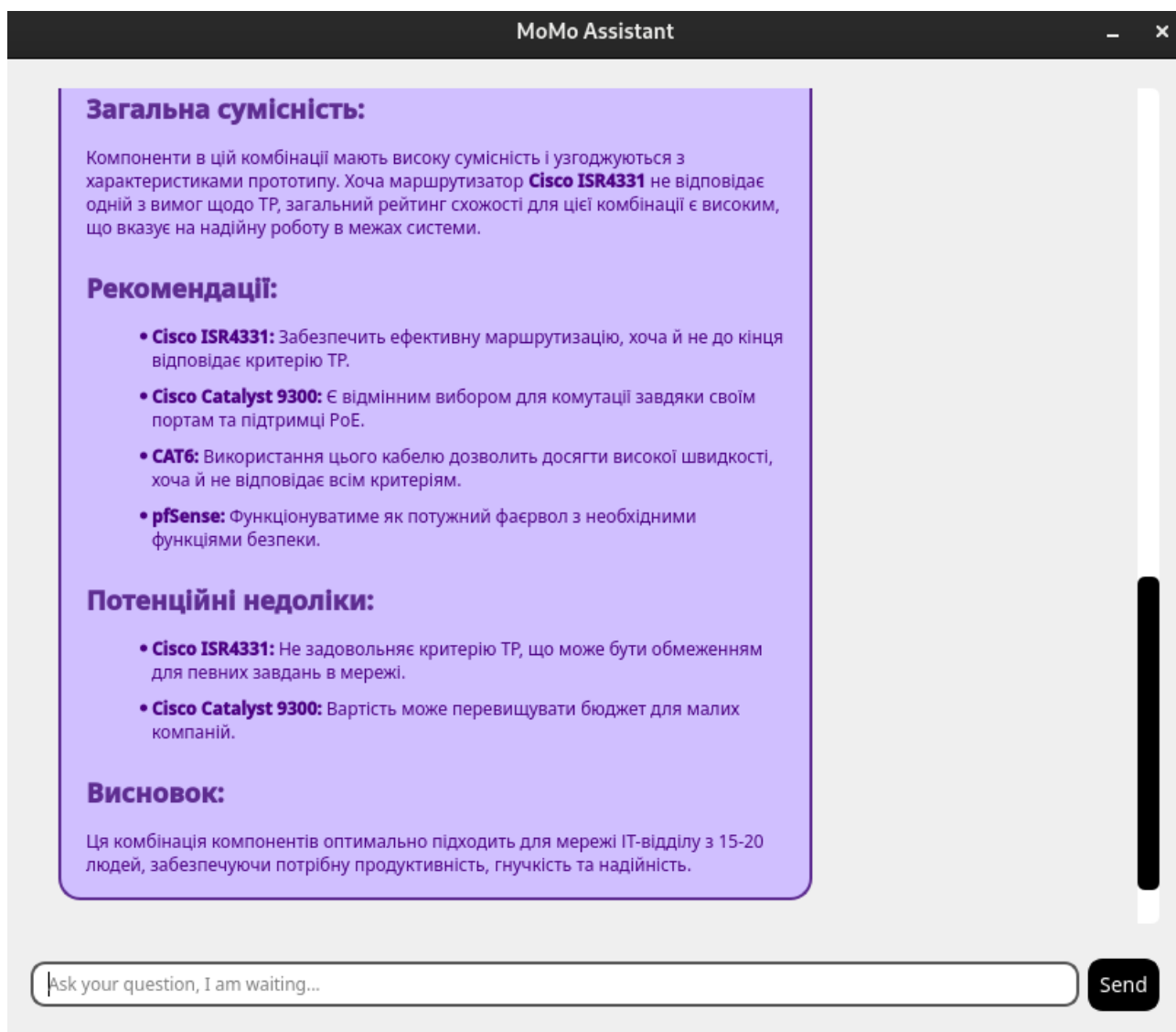


Рисунок 3.12 – Інсайти від AI-асистента

У наступному абзаці асистент зазначає можливі недоліки – наприклад, ризик виникнення «вузьких місць» у каналі передачі або перевищення бюджету при використанні комерційних рішень – і завершує висновком про доцільність обраної конфігурації для вказаного сценарію. Такий інтегрований підхід

дозволяє не лише отримати оцінку технічної коректності рішення, а й одразу сформулювати практичні рекомендації з урахуванням бізнес-вимог.

3.5 Висновок до третього розділу

У третьому розділі кваліфікаційної роботи виконано пояснення стовоно реалізації ключових компонентів програмного забезпечення МоМо, яке складається з двох автономних модулів: обчислювального ядра (MoMo Core) та графічного інтерфейсу користувача (MoMoGUI). У ході проєктування ядро було оформлено як незалежний Python-пакет із чітко визначеною структурою класів для побудови простору морфологічних конфігурацій та обчислення метрик подібності. Архітектурний аналіз і реалізація цього модуля включали використання патерна «Фасад» для інкапсуляції складних алгоритмічних процесів та надання простого інтерфейсу для зовнішніх споживачів.

Графічний інтерфейс розроблено на основі PyQt5 із розподілом на стартове вікно, головне вікно та вікно AI-асистента. Кожне з них реалізовано окремим класом із відповідними віджетами та сигналами-слотами для обробки подій. Блокова архітектура GUI дозволила ізолювати логіку відображення й керування даними, що значно спростило тестування й майбутню підтримку. Інтеграція AI-асистента через модуль MoMoAgent забезпечила можливість «останніх штрихів» (last mile customization), який доповнює традиційні етапи морфологічного аналізу інтелектуальними порадами.

Значною складовою розділу стало тестування ядра та інтеграційні перевірки GUI, що підтвердили коректність розрахунків і стабільність інтерфейсу при різних сценаріях роботи. Продемонстрований приклад застосування із конфігурацією комп'ютерної мережі ілюструє практичну цінність розробленого ПЗ та його готовність до використання в реальних умовах.

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Оцінка ризику перевтоми при багатогодинній роботі з моделями конфігурацій у середовищах із високим когнітивним навантаженням

У сучасних умовах стрімкого розвитку інформаційних технологій фахівці дедалі частіше зіштовхуються з необхідністю виконання складних когнітивних завдань, що супроводжуються високим рівнем інтелектуального навантаження. Зокрема, морфологічне моделювання конфігурацій комп'ютерних мереж вимагає багатоетапного аналізу альтернатив, тривалого утримання уваги та прийняття рішень у ситуаціях невизначеності. Такий режим діяльності спричиняє інтенсивне розумове напруження, яке за умови тривалої безперервної роботи може призводити до втоми, а в окремих випадках – до перевтоми.

Згідно з міжнародним стандартом ISO 10075-1 [48], психічна втома визначається як тимчасове зниження розумової та фізичної функціональної здатності, зумовлене інтенсивністю, тривалістю і характером попереднього навантаження, усунення якого можливе лише шляхом відпочинку. Отже, надмірне розумове навантаження викликає тимчасове зниження функціональної ефективності мозку, відновлення якої потребує відповідних пауз у роботі.

Ризики розвитку перевтоми у процесі використання програмного забезпечення для оптимальної конфігурації комп'ютерної мережі із використання морфологічного моделювання пов'язані насамперед із накопиченням розумової втоми. Об'єктивними проявами цього стану є зниження продуктивності інтелектуальної праці, суб'єктивними – почуття виснаженості та труднощі з концентрацією. Дослідження в галузі гігієни праці засвідчують, що розумова втома супроводжується зменшенням продуктивності мислення, зниженням уваги та уповільненням когнітивних процесів [49]. Особливо швидко втома розвивається за умов монотонної, нецікавої роботи. У

випадку морфологічного моделювання це означає необхідність тривалого підтримання високого рівня когнітивної активності, що без відповідної регуляції може призвести до перевтоми.

Перевтома є більш небезпечним станом, ніж звичайна втома, оскільки має хронічний характер і проявляється у формі стійкого зниження працездатності, порушень сну, підвищеної дратівливості та фізіологічних порушень, таких як ослаблення імунітету та загострення хронічних захворювань [49].

Для збереження високої продуктивності та запобігання перевтомі при інтенсивній розумовій праці необхідно впроваджувати профілактичні заходи, що спираються на принципи ергономіки та гігієни праці. Перш за все, робочий день слід планувати з урахуванням регулярних коротких пауз: кожної години варто відводити 3–5 хвилин на відновлення концентрації, а після двох-трьох годин безперервної інтелектуальної роботи доцільно робити більш тривалу перерву. Для зменшення монотонності та рівномірного розподілу розумового навантаження слід чергувати складні аналітичні завдання з рутинною документацією або невеликими фізичними вправами, що активізують різні ділянки мозку та допомагають уникнути одноманітної втоми. Крім того, оптимальне робоче середовище, у якому ергономічне розташування обладнання, комфортне освітлення, відсутність зайвого шуму й зручні меблі забезпечують мінімізацію психофізіологічного навантаження, сприяє тривалому збереженню уваги. Практика самомасажу кистей та шиї або виконання легких вправ офісної йоги під час мікропауз істотно покращує кровообіг і розслаблює напружені м'язи, що прискорює відновлення ментальної активності та знижує рівень стресу.

Організація робочого місця ІТ-спеціаліста повинна відповідати вимогам ДСТУ 7950:2015 «Дизайн і ергономіка. Робоче місце під час виконання робіт стоячи. Загальні ергономічні вимоги» [50]. Згідно з цим стандартом, поверхня столу має розташовуватися на висоті, що забезпечує згин рук у ліктях під кутом близько 90 градусів, а клавіатура та інші пристрої введення – на одному рівні для уникнення надмірного згинання зап'ясть. Монітор слід встановлювати так,

щоб його верхня межа була на рівні очей або трохи нижче, підтримуючи нейтральне положення шиї і зменшуючи напругу м'язів шиї та плечей. Стілець повинен мати регульовану висоту, підтримку поперекового вигину та м'яку, але достатньо тверду основу, аби рівномірно розподіляти навантаження на стегна й попереки. Важливим є створення вільного простору під столом для вільного розміщення ніг і зміни поз, а також забезпечення оптимального кута між стегнами та корпусом (90-110 градусів). Усі елементи робочого місця повинні мати плавні контури без гострих кутів, а відстань до екрана – 50-70 см, щоб виключити перенапруження зорового апарату. Дотримання цих рекомендацій гарантує мінімізацію фізичного дискомфорту та створює умови для максимально ефективної та безпечної роботи ІТ-фахівця.

Профілактика розумової втоми та перевтоми є необхідною умовою ефективної та безпечної праці спеціалістів у галузі інформаційних технологій. Забезпечення балансу між працею і відпочинком, належна організація робочого середовища та регулярні перерви сприяють збереженню когнітивної продуктивності й загального здоров'я працівників. У контексті морфологічного моделювання, де тривалі навантаження є нормою, застосування превентивних заходів дозволяє не лише підтримувати високий рівень професійної діяльності, а й уникати довготривалих негативних наслідків для організму.

4.2 Профілактика мікротравм кистей та зап'ясть у фахівців при тривалому використанні периферійних пристроїв

У процесі професійної діяльності фахівці у сфері комп'ютерних технологій значну частину робочого часу проводять у статичних позах, виконуючи численні дрібні рухи пальцями та кистями під час взаємодії з клавіатурою, мишею або тачпадом. Такий характер діяльності пов'язаний із підвищеним ризиком розвитку патологічних змін у структурах верхніх кінцівок. Повторювані рухи, статичне навантаження на суглоби зап'ясть і

передпліччя можуть призводити до поступового формування кумулятивних ушкоджень м'язів, сухожиль і нервів – так званих мікротравм.

У контексті охорони праці ці порушення класифікуються як професійні м'язово-скелетні розлади, зокрема травми від повторного навантаження (RSI, Repetitive Strain Injuries) [51]. Найбільш відомим прикладом є синдром зап'ясткового каналу, зумовлений компресією серединного нерва на тлі запалення сухожиль у карпальному каналі. До інших поширених патологій належать тендініти, епіконділіти, а також загальний синдром перенапруження кистей. Особливо виразим це є у людей, які протягом тривалого часу інтенсивно працюють з мишею або здійснюють великий обсяг набору тексту.

Симптоматика таких порушень розвивається поступово і включає біль, втому, оніміння пальців, поколювання та зниження сили хвату. У разі відсутності належної профілактики вказані явища можуть прогресувати до стадій, що потребують тривалого медичного втручання.

Мікротравми верхніх кінцівок є одними з найпоширеніших форм професійних захворювань серед працівників інтелектуальної праці. За статистичними даними, до 65% нових випадків професійних патологій пов'язані з повторною травматизацією м'яких тканин. Зокрема, розлади кистей і зап'ясть є основною причиною тривалих тимчасових втрат працездатності. Епідеміологічні дослідження засвідчують, що понад 5-7 годин щоденної роботи за комп'ютером значно підвищують ризик появи болю, оніміння та інших порушень у ділянці кистей. Особливо шкідливою є надмірна експлуатація комп'ютерної миші – використання її понад 20 годин на тиждень суттєво корелює з підвищеною імовірністю розвитку синдрому зап'ясткового каналу [52].

Такі порушення не лише спричиняють дискомфорт, а й призводять до зниження загальної продуктивності праці, потребують значних витрат на лікування та можуть стати причиною втрати професійної придатності. З огляду на це, профілактика мікротравм кистей слід розглядати як пріоритетну складову системи охорони праці. Ергономічна організація робочого місця

полягає в тому, щоб забезпечити нейтральне положення зап'ясть під час роботи: клавіатуру, мишу та монітор слід розташовувати таким чином, щоб кисті не зазнавали зайвого напруження (про це описано в попередньому розділі), а доступ до всіх периферійних пристроїв був зручним і не вимагав ротацій кистей. Виконання регулярних мікропауз – зокрема, згідно з вимогами ДСанПіН 3.3.2.007-98 «Гігієнічні вимоги до роботи за дисплеями ПЕОМ», робочий час операторів, які тривалий час виконують роботу перед відеотерміналами, слід організовувати з обов'язковими внутрішньозмінними перервами. Зокрема, при роботі з інтенсивним набором текст рекомендовано після кожної години безперервної роботи відпочивати 10 хвилин. Такий підхід дозволяє тканинам відновлюватися та запобігти накопиченню мікропошкоджень й своєчасно давати кистям необхідний відпочинок.

Щоб зменшити м'язове напруження та покращити кровообіг, рекомендується періодично виконувати спеціальні вправи для розтягнення передпліч, згинання й обертання зап'ясть, а також самомасаж долонь і зап'ястків. Використання сучасних ергономічних засобів – наприклад, вертикальних мишок, розділених клавіатур, підкладок із гелевою вставкою під зап'ястя – сприяє рівномірному розподілу навантаження на опорно-руховий апарат і суттєво знижує ризик перевантажень. Усі перелічені заходи в сукупності формують ефективну систему профілактики професійних травм кистей і сприяють збереженню здоров'я та працездатності ІТ-спеціалістів.

Мікротравми кистей і зап'ясть становлять суттєву загрозу для здоров'я спеціалістів, що тривалий час працюють з комп'ютерною технікою. Їх розвиток тісно пов'язаний із недотриманням ергономічних норм, відсутністю перерв та фізичної активності. Запровадження комплексу профілактичних заходів – від правильної організації робочого місця до впровадження регулярних мікропауз і вправ – є ефективним засобом зниження професійних ризиків.

4.3 Висновок до четвертого розділу

У четвертому розділі кваліфікаційної роботи описано основні аспекти безпеки життєдіяльності та охорони праці при виконанні професійної діяльності, пов'язаної з морфологічним моделюванням комп'ютерних мереж. Розглянуто два ключові напрями: когнітивне навантаження під час роботи зі складними конфігураційними моделями, що зумовлює ризики перевтоми, та фізичне навантаження на кисті та зап'ястки, що може призводити до мікротравм верхніх кінцівок.

У межах розділу було проаналізовано характерні фактори ризику, пов'язані з розумовою втомою і перенапруженням опорно-рухового апарату, їхній вплив на працездатність і стан здоров'я фахівця. Наведено сучасні наукові підходи до профілактики цих станів, серед яких – ергономічна організація робочого місця, дотримання режиму праці й відпочинку, виконання релаксаційних і фізичних вправ.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було розроблено програмне забезпечення для формування оптимальної конфігурації комп'ютерної мережі методом морфологічного моделювання. Реалізована система дозволяє автоматизовано генерувати всі можливі комбінації технічних рішень, порівнювати їх із заданим прототипом і надавати зручний інтерфейс для взаємодії з результатами аналізу. Запропонований підхід дозволяє суттєво скоротити час і зусилля, необхідні для проєктування мережових інфраструктур, а також зробити процес вибору технічної конфігурації прозорим і керованим. Вбудований AI-асистент підсилює функціональність системи, допомагаючи користувачу з інтерпретацією результатів та прийняттям рішень.

У першому розділі кваліфікаційної роботи освітнього рівня «Бакалавр»:

- подано обґрунтування актуальності автоматизованого підходу до конфігурування комп'ютерних мереж;
- розглянуто традиційні симуляційні, багатокритеріальні та евристичні методи та показано їхні обмеження;
- висвітлено концепцію морфологічного моделювання як структурованого способу перебору варіантів;
- проаналізовано чинні програмні інструменти й сформульовано вимоги до нового ПЗ.

У другому розділі кваліфікаційної роботи:

- досліджено архітектуру розробленої системи, що включає ядро МоМо та інтерфейс МоМоGUI;
- обґрунтовано вибір Python, PyQt5 та супутніх бібліотек для реалізації функціональних вимог;
- сформовано UML-діаграми, які описують взаємодію компонентів та основні сценарії роботи.

У третьому розділі кваліфікаційної роботи:

- розроблено й інтегровано обчислювальне ядро з фасадним API та модульний графічний інтерфейс;
- запропоновано й реалізовано AI-асистент для експертної підтримки користувача;
- спроектовано прототип реальної мережевої задачі та здійснено його морфологічний аналіз;
- протестовано модулі ядра й GUI, що підтвердило коректність обчислень та стабільність роботи.

У розділі «Безпека життєдіяльності, охорона праці» висвітлено висвітлено основні ризики, що виникають під час виконання інтелектуально та фізично навантажених завдань у сфері інформаційних технологій, а також окреслено заходи профілактики розумової перевтоми та мікротравм верхніх кінцівок шляхом впровадження ергономічних, гігієнічних і організаційних рішень.

ПЕРЕЛІК ДЖЕРЕЛ

- 1 What is Network Design? URL: <https://www.anticlockwise.com.au/blog/what-is-network-design/> (дата зверення: 02.06.2025)
- 2 Fritz Zwicky: Morphological astronomy URL: https://mathshistory.st-andrews.ac.uk/Extras/Morphological_astronomy/ (дата зверення: 02.06.2025)
- 3 General Morphological Analysis (GMA). URL: <https://www.swemorph.com/ma.html> (дата зверення: 02.06.2025)
- 4 Futures Studies using General Morphological Analysis. [PDF] URL: <https://www.swemorph.com/pdf/futures.pdf> (дата зверення: 02.06.2025)
- 5 Fryz M., Mlynko B. Property Analysis of Conditional Linear Random Process as a Mathematical Model of Cyclostationary Signal. 2nd International Workshop on Information Technologies: Theoretical and Applied Problems (ITTAP 2022). Ternopil, Ukraine: CEUR Workshop Proceedings, 2022. Vol. 3309. P. 77–82.
- 6 Problem Structuring using Computer-Aided Morphological Analysis. Journal of the Operational Research Society, 2006. URL: <https://www.swemorph.com/pdf/psm-gma.pdf> (дата зверення: 02.06.2025)
- 7 A morphological analysis approach for cloud services selection. International Journal of Mathematical, Engineering and Management Sciences. URL: <https://www.ijmems.in/cms/storage/app/public/uploads/volumes/30-IJMEMS-23-0687-9-3-581-615-2024.pdf> (дата зверення: 02.06.2025)
- 8 Morphological Zwicky Boxes vs Feature Model. URL: <https://recontextualization.medium.com/morphological-zwicky-boxes-vs-feature-model-2348e5d978ad> (дата зверення: 02.06.2025)
- 9 GNS3 vs EVE-NG vs Cisco Packet Tracer: Detailed Comparison. URL: <https://www.networkstraining.com/gns3-vs-eve-ng-vs-cisco-packet-tracer> (дата зверення: 02.06.2025)

10 6 Best Network Simulators for Windows 10 and 11 (Free and Paid). URL: <https://umatechnology.org/6-best-network-simulators-for-windows-10-and-11> (дата зверення: 02.06.2025)

11 MA/Carma – Computer-Aided Morphological Analysis. URL: <https://www.swemorph.com/macarma.html> (дата зверення: 02.06.2025)

12 Morpheus: Morphological Matrix Tool. URL: <https://github.com/johnmartins/morpheus> (дата зверення: 02.06.2025)

13 MorphR: R Shiny Package for Morphological Analysis. URL: <https://github.com/sgrubsmyon/morphr> (дата зверення: 02.06.2025)

14 Python rules the world. TIOBE Index for May 2025. URL: <https://www.tiobe.com/tiobe-index/> (дата зверення: 02.06.2025)

15 What is Python? Executive Summary. Python.org. URL: <https://www.python.org/doc/essays/blurb/> (дата зверення: 02.06.2025)

16 Python for Data Analysis: Data Wrangling with pandas, NumPy, and IPython. 2nd ed. – O'Reilly Media, 2017.

17 Riverbank Computing Limited. PyQt5 Reference Guide. URL: <https://www.riverbankcomputing.com/static/Docs/PyQt5/> (дата зверення: 02.06.2025)

18 Poetry Docs: Publishing to PyPI. URL: <https://python-poetry.org/docs/libraries/> (дата зверення: 02.06.2025)

19 Qt for Python (PySide 6) Documentation. URL: <https://doc.qt.io/qtforpython-6/> (дата зверення: 02.06.2025)

20 QThread Class Reference (Qt for Python). URL: <https://doc.qt.io/qtforpython-6.5/PySide6/QtCore/QThread.html> (дата зверення: 02.06.2025)

21 CrewAI – Contextual Role-Driven AI Integration. URL: <https://www.crewai.com/> (дата зверення: 02.06.2025)

22 CrewAI GitHub Repository. URL: <https://github.com/crewAIInc/crewAI> (дата зверення: 02.06.2025)

23 CrewAI Quickstart Guide. URL: <https://docs.crewai.com/quickstart> (дата зверення: 02.06.2025)

24 CrewAI Examples. URL: <https://docs.crewai.com/examples/example> (дата зверення: 02.06.2025)

25 OpenAI API. URL: <https://openai.com/api/> (дата зверення: 02.06.2025)

26 Visual Studio Code. URL: <https://code.visualstudio.com/> (дата зверення: 02.06.2025)

27 Where the world builds software. URL: <https://github.com/> (дата зверення: 02.06.2025)

28 Object Management Group. Unified Modeling Language (UML), Version 2.5.1 Specification. – OMG, 2017. URL: <https://www.omg.org/spec/UML/2.5.1> (дата зверення: 02.06.2025)

29 Visual Paradigm. What is Use Case Diagram? URL: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/> (дата зверення: 02.06.2025)

30 Visual Paradigm Guides. A Comprehensive Guide to Use Case Modeling. URL: <https://guides.visual-paradigm.com/a-comprehensive-guide-to-use-case-modeling/> (дата зверення: 02.06.2025)

31 UML Use Case Diagram: Tutorial with Example. URL: <https://www.guru99.com/use-case-diagrams-example.html> (дата зверення: 02.06.2025)

32 Class Diagram - What is it? URL: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/> (дата зверення: 02.06.2025)

33 Creately. UML Diagram Types: Learn About All 14 Types of UML Diagrams. URL: <https://creately.com/blog/diagrams/uml-diagram-types-examples/> (дата зверення: 02.06.2025)

34 UML Diagrams Examples. URL: <https://www.uml-diagrams.org/index-examples.html> (дата зверення: 02.06.2025)

35 Creating Sequence Diagrams in UML: A Comprehensive Tutorial. URL: <https://guides.visual-paradigm.com/creating-sequence-diagrams-in-uml-a-comprehensive-tutorial/> (дата зверення: 02.06.2025)

36 Wikipedia. List of Unified Modeling Language tools. URL: https://en.wikipedia.org/wiki/List_of_Unified_Modeling_Language_tools (дата зверення: 02.06.2025)

37 Unified Modeling Language (UML) Diagrams. URL: <https://www.geeksforgeeks.org/unified-modeling-language-uml-introduction/> (дата зверення: 02.06.2025)

38 Facade – Шаблони проєктування. URL: <https://refactoring.guru/uk/design-patterns/facade> (дата зверення: 02.06.2025)

39 GeeksforGeeks. Facade Method Design Pattern. URL: <https://www.geeksforgeeks.org/facade-design-pattern-introduction/> (дата зверення: 02.06.2025)

40 DEV Community. The Facade Pattern: A Simplified Beginner Guide. URL: <https://dev.to/devleader/the-facade-pattern-a-simplified-beginner-guide-440i> (дата зверення: 02.06.2025)

41 The MIT License. Open Source Initiative. URL: <https://opensource.org/license/MIT> (дата зверення: 02.06.2025)

42 Apache License, Version 2.0. URL: <https://www.apache.org/licenses/LICENSE-2.0.html> (дата зверення: 02.06.2025)

43 Packaging Python Projects. Python Packaging User Guide. URL: <https://packaging.python.org/en/latest/tutorials/packaging-projects/> (дата зверення: 02.06.2025)

44 How to Publish a Python Package to PyPI з Twine. URL: <https://www.geeksforgeeks.org/how-to-publish-python-package-at-pypi-using-twine-module/> (дата зверення: 02.06.2025)

45 GeeksforGeeks. Unit Testing - Software Testing. URL: <https://www.geeksforgeeks.org/unit-testing-software-testing/> (дата зверення: 02.06.2025)

- 46 GeeksforGeeks. Module Testing. URL: <https://www.geeksforgeeks.org/module-testing/> (дата зверення: 02.06.2025)
- 47 How to Use Signals and Slots - Qt Wiki. URL: https://wiki.qt.io/How_to_Use_Signals_and_Slots (дата зверення: 02.06.2025)
- 48 ISO 10075:1995. Ergonomic principles related to mental workload — General terms and definitions. URL: <https://www.scribd.com/document/268051954/ISO-10075-1995-Ergonomic-Mental-Workload-General-Terms> (дата зверення: 02.06.2025)
- 49 Державна служба України з питань праці. Втома та її профілактика. URL: <https://dp.dsp.gov.ua/novyny/vtoma-ta-ii-profilaktyka> (дата зверення: 02.06.2025)
- 50 ДСТУ 7950:2015 Дизайн і ергономіка. Робоче місце під час виконання робіт стоячи. Загальні ергономічні вимоги. - Чинний від 2016-09-01. - Київ : УкрНДНЦ, 2016. - III, 7 с. : рис. - (Національний стандарт України).
- 51 Профілактика «офісного синдрому» у осіб першого періоду зрілого віку засобами оздоровчого фітнесу. URL: <https://reposit.unisport.edu.ua/server/api/core/bitstreams/58b6fcb8-901d-49de-a275-79461f2cf21d/content> (дата зверення: 02.06.2025)
- 52 Work-Related Musculoskeletal Disorders of the Hand and Wrist: Epidemiology, Pathophysiology, and Sensorimotor Changes. URL: <https://pmc.ncbi.nlm.nih.gov/articles/PMC1557630/> (дата зверення: 02.06.2025)
- 53 Надання першої допомоги потерпілим URL: <http://referatu.net.ua/newreferats/448/187221> (дата зверення: 02.06.2025)
- 54 Желібо Є. П. Безпека Життєдіяльності / Є. П. Желібо, Н. М. Заверуха, В. В. Задарний. – Київ: Каравела, 2008. – 344 с.
- 55 Zaporozhets, Y. Kuts, B. Mlynko, M. Fryz, and L. Scherbak, “EEG Signal Classification Using Linear Process Model-Based Feature Extraction and Supervised Learning,” in Advanced System Development Technologies II. Studies in Systems, Decision and Control, M. Bezuglyi, N. Bouraou, V. Mykytenko, G. Tymchyk, and A.

Zaporozhets, Eds., Cham: Springer Nature Switzerland, 2025, pp. 235–257. doi: 10.1007/978-3-031-82035-9_7.

56 Fryz M., Scherbak L., Mlynko B., Mykhailovych T. Linear Random Process Model-Based EEG Classification Using Machine Learning Techniques. Proceedings of the 1st International Workshop on Computer Information Technologies in Industry 4.0 (CITI 2023). Ternopil, Ukraine: CEUR Workshop Proceedings, 2023. Vol. 3468. P. 126–132.

57 M. Fryz, B. Mlynko, Property analysis of multivariate conditional linear random processes in the problems of mathematical modelling of signals, Technology Audit and Production Reserves, 3/2(65), 2022, pp. 29–32.

58 Бабак В. П., Марченко М. Є., Фриз. Б. Г. Теорія ймовірностей, випадкові процеси та математична статистика. К.: Техніка, 2004. 288 с.

ДОДАТКИ

Лістинги MoMo Core

Лістинг 1 – Код SystemModel

```

class SystemModel:
    def __init__(self, name: str, data=None, features=None,
alternatives=None):
        self._init_name(name)
        self._validate_as_list_string(features, "features")
        self._validate_as_list_string(alternatives,
"alternatives")

self._validate_measures_of_data_features_alternatives(data,
features, alternatives)
        self.data = pd.DataFrame(data=data, index=features,
columns=alternatives)
        self.__features_appender = _FeaturesAppender(self.data)
        self.__alternatives_appender =
_AlternativesAppender(self.data)

    def _init_name(self, name):
        if not isinstance(name, str):
            raise ValueError("The name parameter must be a
string.")
        self.name = name

    def _validate_measures_of_data_features_alternatives(self,
data, features, alternatives):
        if data is not None:
            if features is not None and len(data) !=
len(features):
                raise ValueError("The number of rows in data must
match the length of features.")
            if alternatives is not None and len(data[0]) !=
len(alternatives):
                raise ValueError("The number of columns in data
must match the length of alternatives.")

    def _validate_as_list_string(self, value, name):
        if value is not None and not all(isinstance(v, str) for v
in value):
            raise ValueError(f"All elements of {name} must be
strings.")

    def _validate_as(self, value, name, expected_type):
        if not isinstance(value, expected_type):
            raise ValueError(f"The {name} parameter must be of
type {expected_type.__name__}.")

    def _validate_length(self, value, name, expected_length):

```

```

        if value is not None and len(value) != expected_length:
            raise ValueError(f"The {name} parameter must have a
length of {expected_length}.")

    def add_feature(self, feature_name: str, alternatives: list |
dict):
        self.__features_appender.data_frame = self.data
        self.data =
self.__features_appender.add_item(feature_name, alternatives)

    def add_alternative(self, alternative_name, features: list |
dict):
        self.__alternatives_appender.data_frame = self.data
        self.data =
self.__alternatives_appender.add_item(alternative_name, features)

    def remove_feature(self, feature_name):
        self._validate_as(feature_name, "feature_name", str)
        if feature_name not in self.data.index:
            raise ValueError(f"Feature '{feature_name}' does not
exist.")
        self.data = self.data.drop(index=feature_name)

    def remove_alternative(self, alternative_name):
        self._validate_as(alternative_name, "alternative_name",
str)
        if alternative_name not in self.data.columns:
            raise ValueError(f"Alternative '{alternative_name}'
does not exist.")
        self.data = self.data.drop(columns=alternative_name)

    def get_features(self):
        return tuple(self.data.index)

    @property
    def features(self):
        return self.get_features()

    def get_alternatives(self):
        return tuple(self.data.columns)

    @property
    def alternatives(self):
        return self.get_alternatives()

    @property
    def loc(self):
        return self.data.loc

    def __getitem__(self, key):
        return self.data[key]

    def __setitem__(self, key, value):

```

```
        self.data[key] = value

def __str__(self):
    return f'"{self.name}"\n{self.data}'
```

Лістинг 2 – Код MultiSystemModel

```

class MultiSystemModel:
    def __init__(self, system_models: list | tuple | set | None =
None):
        self.systems = {}

        if isinstance(system_models, MultiSystemModel):
            self.systems = system_models.systems
        elif isinstance(system_models, list):
            self._constructor_from_list(system_models)
        elif isinstance(system_models, tuple):
            self._constructor_from_tuple(system_models)
        elif isinstance(system_models, set):
            self._constructor_from_set(system_models)
        elif system_models is None:
            pass
        else:
            raise ValueError("The system_models parameter must be
a list, tuple or set.")

        def _constructor_from_list(self, system_models):
            if all(isinstance(system, SystemModel) for system in
system_models):
                for system in system_models:
                    self.add_system(system)
            else:
                raise ValueError("All elements in the list must be
SystemModel instances.")

        def _constructor_from_tuple(self, system_models):
            if all(isinstance(system, SystemModel) for system in
system_models):
                for system in system_models:
                    self.add_system(system)
            else:
                raise ValueError("All elements in the tuple must be
SystemModel instances.")

        def _constructor_from_set(self, system_models):
            if all(isinstance(system, SystemModel) for system in
system_models):
                for system in system_models:
                    self.add_system(system)
            else:
                raise ValueError("All elements in the set must be
SystemModel instances.")

        def add_system(self, system_model: SystemModel):
            if not isinstance(system_model, SystemModel):
                raise ValueError("The system_model parameter must be a
SystemModel instance.")
            self.systems[system_model.name] = system_model

```

```

def add_systems(self, system_models: list | tuple | set):
    if not isinstance(system_models, (list, tuple, set)):
        raise ValueError("The system_models parameter must be
a list, tuple or set.")
    for system in system_models:
        self.add_system(system)

def remove_system(self, system_name):
    if self.__has_system(system_name):
        del self.systems[system_name]
    else:
        raise ValueError(f"System '{system_name}' does not
exist.")

def __has_system(self, system_name):
    return system_name in self.systems

def get_system_names(self):
    return tuple(self.systems.keys())

def get_all_combinations(self):
    system_names = list(self.systems.keys())
    system_data = [self.systems[name].data for name in
system_names]
    related_features = self.get_features_related_to_system()
    alternatives_name = list(product(*[data.columns for data
in system_data]))
    result_df = pd.DataFrame(index=related_features)

    for combination in alternatives_name:
        result_df[combination] = 0
        for system_name, system_df in zip(system_names,
system_data):
            for feature in system_df.index:
                if related_feature :=
self.__get_related_feature(feature, related_features):
                    value = system_df.loc[feature,
combination[system_names.index(system_name)]]
                    result_df.at[related_feature, combination]
= value

    return result_df

def generate_combinations(self):
    system_names = list(self.systems.keys())
    system_data = [self.systems[name].data for name in
system_names]
    related_features = self.get_features_related_to_system()

    for combination in product(*[data.columns for data in
system_data]):
        new_column = pd.Series(0, index=related_features)

```

```

        for idx, (system_name, system_df) in
enumerate(zip(system_names, system_data)):
            for feature in system_df.index:
                if related_feature :=
self.__get_related_feature(feature, related_features):
                    value = system_df.loc[feature,
combination[idx]]
                                new_column[related_feature] = value
                                yield tuple(combination), new_column

def __get_related_feature(self, feature, related_features):
    for related_feature in related_features:
        if feature in related_feature:
            return related_feature
    raise ValueError(f"Feature {feature} is not related to any
system.")

def get_features_related_to_system(self):
    systems = self.get_system_names()
    features = [self.systems[name].data.index for name in
systems]
    related_features = []
    for system, feature_list in zip(systems, features):
        for feature in feature_list:
            related_features.append((system, feature))
    return tuple(related_features)

def get_all_features(self):
    all_features = list()
    for system_data in self.systems.values():
        all_features.extend(system_data.data.index.to_list())
    return tuple(all_features)

def get_prototype(self):
    if not self.systems:
        return Prototype()
    related_features = self.get_features_related_to_system()
    prototype = Prototype(data=[0 for _ in related_features],
index=related_features)
    return prototype

def __str__(self):
    return self.get_all_combinations().__str__()

```

Лістинг 3 – Код Prototype

```
class Prototype(pd.Series):
    def __init__(self, data=None, index=None):
        super().__init__(data=data, index=index)

    def set_marks(self, marks_list: dict | list):
        if isinstance(marks_list, dict):
            self._set_marks_dict(marks_list)
        elif isinstance(marks_list, list):
            self._set_marks_list(marks_list)
        else:
            raise ValueError("The marks_list parameter must be a
dictionary or a list.")

    def _set_marks_dict(self, marks_dict: dict):
        for outer_key, inner_dict in marks_dict.items():
            for inner_key, value in inner_dict.items():
                self[(outer_key, inner_key)] = value

    def _set_marks_list(self, marks_list: list):
        self[:] = marks_list
```


Лістинг 4 – Код MoMoModel

```

from .prototype import Prototype
from .system_models.system_models import MultiSystemModel

class MoMoModel:
    def __init__(
        self,
        system_models: MultiSystemModel | list | tuple | set,
        prototype: Prototype | None = None,
    ):
        self._u = 0
        self.system_models = MultiSystemModel(system_models)
        self._init_prototype(prototype)

    def _init_prototype(self, value):
        if value is None:
            self.prototype = self.system_models.get_prototype()
        elif isinstance(value, Prototype):
            if
value.index.equals(self.system_models.get_prototype().index):
                self.prototype = value
            else:
                raise ValueError("The prototype must have the same
index as the system models.")
        else:
            raise ValueError("The prototype must be a Prototype or
None instance.")

    def get_prototype(self):
        return self.prototype

    def set_prototype(self, prototype):
        self._init_prototype(prototype)

    @property
    def prototype_(self):
        return self.get_prototype()

    @prototype_.setter
    def prototype_(self, prototype):
        self.set_prototype(prototype)

    def get_system_models(self):
        return self.system_models

    def set_system_models(self, system_models):
        self.system_models = MultiSystemModel(system_models)
        self.prototype = self.system_models.get_prototype()

    @property
    def system_models_(self):

```

```

        return self.get_system_models()

    @system_models.setter
    def system_models_(self, system_models):
        self.set_system_models(system_models)

    @property
    def u(self):
        return self._u

    @u.setter
    def u(self, value: str|int|float):
        if isinstance(value, str):
            value = value.lower()
            if value == "sorensen_dice":
                self._u = 0
            elif value == "dice":
                self._u = 1
            else:
                raise ValueError("The value must be 'jaccard' or
'dice'.")

        elif isinstance(value, (int, float)):
            if 0 <= value <= 1:
                self._u = value
            else:
                raise ValueError("The value must be between 0 and
1.")

    def __calculate_similarity_measures(self, prototype,
compared_system):
        intersection_num = (prototype & compared_system).sum()
        card_prototype = prototype.sum()
        card_compared_system = compared_system.sum()
        u = self._u
        return 2 * intersection_num / ((1 + u) * (card_prototype +
card_compared_system) - 2 * u * intersection_num)

    def get_similarity_measures(self):
        similarity_measures = {}

        for combination, new_column in
self.system_models.generate_combinations():
            similarity_measures[combination] =
self.__calculate_similarity_measures(
                self.prototype, new_column
            )

        return similarity_measures

    def __str__(self):
        output = f"Prototype:\n{self.prototype}\n\n"
        output += f"System Models:\n{self.system_models}"
        return output

```

Лістинги

Лістинг 1 – Код StartWindow

```

class StartWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self._systems_data = []
        self._results_map = None
        self._window_widget = QWidget()
        self._init_ui()
        self.setCentralWidget(self._window_widget)
        self.setStyleSheet(load_window_style())
        self.setFixedSize(470, 250)
        self.setWindowIcon(QIcon("resources/img/logo/logo.ico"))

    def _init_ui(self):
        systems_layout = QVBoxLayout()
        buttons_layout = QHBoxLayout()

        self.setWindowTitle("MoMo")

        label = QLabel()
        label.setText("""
<h1>Welcome to MoMo</h1>
<p>MoMo is a tool for morphological modeling.</p>
<p>Click the button below to create a new system.</p>
""")
        label.setAlignment(Qt.AlignTop | Qt.AlignCenter)
        systems_layout.addWidget(label)

        create_button = QPushButton("Create")
        create_button.setFixedWidth(100)
        create_button.clicked.connect(self._create_action)

        upload_file_button = QPushButton("Load File")
        upload_file_button.setFixedWidth(100)

        upload_file_button.clicked.connect(self._load_from_file_action)

        buttons_layout.addWidget(create_button)
        buttons_layout.addWidget(upload_file_button)
        buttons_layout.setAlignment(Qt.AlignCenter)

        systems_layout.addLayout(buttons_layout)
        systems_layout.setContentsMargins(20, 20, 20, 60)

        self._window_widget.setLayout(systems_layout)

    def _load_from_file_action(self):

```

```

        file_name, _ = QFileDialog.getOpenFileName(self, 'Open
File', filter="Excel Files (*.xlsx *.xls)")

        if file_name:
            file_validator = ExcelFileValidator()
            is_results_file =
file_validator.is_results_file(file_name)
            self._systems_data = load_systems_data(file_name)

            if not self._systems_data:
                QMessageBox.warning(self, "Error", "No systems
found in the file. Please ensure the file has the correct format
or data.")
                return

            if is_results_file:
                try:
                    self._results_map =
ResultsMap.from_excel(file_name)
                except Exception as e:
                    print(f"Error loading results: {e}")

            self.close()

    def _create_action(self):
        system_name, ok = QInputDialog.getText(self, "New System",
"Enter the name of the new system:")

        if ok and system_name:
            self._systems_data.append(SystemModel(system_name))
            self.close()

    @property
    def systems_data(self):
        return self._systems_data

    @property
    def results_map(self):
        return self._results_map

```

Лістинг 2 – Код MainWindow

```

class MainWindow(QMainWindow):
    systemsLoaded = pyqtSignal(list)

    def __init__(self, systems_data: list[SystemModel] = None,
results_map: ResultsMap = None):
        super().__init__()
        self.systems_data = systems_data or []
        self.results_map = results_map
        self.systems_tab = None
        self.empty_systems_tab = None
        self.prototype_gui = None
        self.cached_prototype = None
        self.chat_window = None

        self._setup_ui()

        if self.results_map and self.results_map.prototype is not
None:
            self.cached_prototype = self.results_map.prototype

        self.setStyleSheet(load_window_style())
        self.setWindowIcon(QIcon("resources/img/logo/logo.ico"))

        if self.results_map:

self.tabs_manager.add_result_tab(ResultsTab(results=self.results_m
ap))

    def _setup_ui(self):
        self.splitter = QSplitter(Qt.Horizontal, parent=self)
        self.prototype_widget = QWidget(parent=self)
        self.tabs_manager = TabManager(parent=self)

        self.setCentralWidget(self.splitter)

        self.prototype_widget.setSizePolicy(QSizePolicy.Expanding,
QSizePolicy.Preferred)
        self.prototype_widget.setLayout(QVBoxLayout())

        self.splitter.addWidget(self.prototype_widget)

        self.splitter.addWidget(self.tabs_manager)

        self._create_systems_tabs()

        self.setWindowTitle("MoMo")

self.setGeometry(*QDesktopWidget().screenGeometry().getRect())

        if self.systems_data:
            self._resize_splitter()

```

```

else:
    self._resize_splitter(left_ratio=0, right_ratio=1)

self._ask_ai_button()

def _ask_ai_button(self):
    self.ask_ai_button = FloatingButton(parent=self)
    self.ask_ai_button.setStyleSheet(load_window_style())
    self.ask_ai_button.button.setText("Ask AI")

self.ask_ai_button.button.setStyleSheet(load_ask_ai_style())

self.ask_ai_button.button.clicked.connect(self._show_momo_agent_widget)
    self.ask_ai_button.show()

def _show_momo_agent_widget(self):
    # code...

def closeEvent(self, event):
    # code...

def _resize_splitter(self, left_ratio=0.28, right_ratio=0.72):
    # code...

def _create_systems_tabs(self):
    if not self.systems_data:
        self._init_empty_systems_tab()

self.tabs_manager.insert_system_tab(self.empty_systems_tab)
    else:
        self._init_filled_systems_tab()
        self.tabs_manager.insert_system_tab(self.systems_tab)

def _init_filled_systems_tab(self):
    self.systems_tab = SystemsTab(parent=self.tabs_manager,
main_window=self, on_content_change=self._update_window)

self.systems_tab.noTabsLeft.connect(self._reset_to_empty_systems_tab)

    for system_model in self.systems_data:
        table_widget = SystemTable(system_model)
        self.systems_tab.add_system_table(table_widget)

def _init_empty_systems_tab(self):
    self.empty_systems_tab =
EmptySystemsTab(parent=self.tabs_manager)

```

```

self.empty_systems_tab.add_tab_button.clicked.connect(self._create_
_first_system_tab)

self.empty_systems_tab.load_from_file_button.clicked.connect(self.
_upload_file)

def _create_prototype_gui(self):
    if not self.systems_data:
        return

    if self.results_map and self.results_map.prototype is not
None:
        prototype = self.results_map.prototype
        measure_type =
self.results_map.similarity_measure_type
    else:
        prototype =
MoMoModel(self.systems_data).get_prototype()
        measure_type = None

        if self.cached_prototype is not None and not
self.cached_prototype.empty:
            overlapping_idx =
prototype.index.intersection(self.cached_prototype.index)
            prototype.loc[overlapping_idx] =
self.cached_prototype.loc[overlapping_idx]

        self.prototype_gui = PrototypeGUI(prototype)
        self.prototype_gui.set_measure_type(measure_type)
        self.prototype_gui.calculate_button.clicked.connect(
            lambda:
asyncio.create_task(self._calculate_combinations_async())
        )

self.prototype_widget.layout().addWidget(self.prototype_gui)
self._resize_splitter()

def _create_first_system_tab(self):
    self.systems_tab = SystemsTab(parent=self.tabs_manager,
main_window=self, on_content_change=self._update_window)

self.systems_tab.noTabsLeft.connect(self._reset_to_empty_systems_t
ab)

    if self.systems_tab.add_system_tab_via_dialog_window():
        self.tabs_manager.remove_insert_tab(self.systems_tab,
"Systems", 0)

```

```

def _reset_to_empty_systems_tab(self):
    self._init_empty_systems_tab()

self.tabs_manager.remove_insert_tab(self.empty_systems_tab,
"Systems", 0)

    if self.prototype_gui:
        try:

self.prototype_gui.calculate_button.clicked.disconnect()
        except TypeError:
            print("Already disconnected")
            pass

self.prototype_widget.layout().removeWidget(self.prototype_gui)
    self.prototype_gui.hide()
    self.prototype_gui.deleteLater()
    self.prototype_gui = None
    self.cached_prototype = None

    QApplication.processEvents()

    self._resize_splitter(left_ratio=0.01, right_ratio=0.99)

def _update_window(self):
    self.tabs_manager.setCurrentIndex(0)

    if not self.systems_tab:
        return

    if self.prototype_gui:
        self.cached_prototype =
self.prototype_gui.get_prototype()

    if hasattr(self, 'systems_tab'):
        self.systems_data = []
        for i in range(self.systems_tab.tabs.count()):
            tab_widget = self.systems_tab.tabs.widget(i)
            if isinstance(tab_widget, SystemTable):

self.systems_data.append(tab_widget.to_system_model())

        self.systemsLoaded.emit(self.systems_data)
        self._recreate_prototype_gui()

def _recreate_prototype_gui(self):
    if self.prototype_gui:

self.prototype_widget.layout().removeWidget(self.prototype_gui)
        self.prototype_gui.deleteLater()
        self.prototype_gui = None

```



```

        if self.systems_data:
            self._create_prototype_gui()

    def _upload_file(self):
        file_path, _ = QFileDialog.getOpenFileName(self, "Open
File", "", "Excel Files (*.xlsx *.xls)")

        if not file_path:
            return

        file_validator = ExcelFileValidator()
        is_results_file =
file_validator.is_results_file(file_path)

        # If this is a results file, load it first
        if is_results_file:
            try:
                self.results_map =
ResultsMap.from_excel(file_path)
                self.systems_data = load_systems_data(file_path)
                # Always update cached prototype with results
prototype
                if self.results_map and self.results_map.prototype
is not None:
                    self.cached_prototype =
self.results_map.prototype
            except Exception as e:
                print(f"Error loading results: {e}")
                return
        else:
            # Regular system file
            self.systems_data = load_systems_data(file_path)

        if not self.systems_data:
            return

        self.systems_tab = SystemsTab(parent=self.tabs_manager,
main_window=self, on_content_change=self._update_window)

        self.systems_tab.noTabsLeft.connect(self._reset_to_empty_systems_t
ab)

        self.tabs_manager.remove_insert_tab(self.systems_tab,
"Systems", 0)

        for system_model in self.systems_data:

self.systems_tab.add_system_table(SystemTable(system_model))

        self._recreate_prototype_gui()

        if is_results_file and self.results_map:

```

```

self.tabs_manager.add_result_tab(ResultsTab(results=self.results_m
ap))

def _read_from_excel(self):
    file_path, _ = QFileDialog.getOpenFileName(self, "Open
File", "", "Excel Files (*.xlsx *.xls)")

    if not file_path:
        return

    file_validator = ExcelFileValidator()
    is_results_file =
file_validator.is_results_file(file_path)

    # If this is a results file, load it first
    if is_results_file:
        try:
            self.results_map =
ResultsMap.from_excel(file_path)
            new_systems = load_systems_data(file_path)
            # Always update cached prototype with results
prototype
            self.cached_prototype = self.results_map.prototype
        except Exception as e:
            print(f"Error loading results: {e}")
            return
    else:
        # Regular system file
        new_systems = load_systems_data(file_path)

    if not new_systems:
        return

    # Add the new systems to the existing systems tab
    for system_model in new_systems:

self.systems_tab.add_system_table(SystemTable(system_model))
        self.systems_data.append(system_model)

    self.results_map = None
    self._recreate_prototype_gui()

def _calculate_combinations_sync(self):
    if not self.systems_data:
        return None
    prototype = self.prototype_gui.get_prototype()
    model = MoMoModel(self.systems_data, prototype)
    model.u = self.prototype_gui.get_similarity_measure_type()

    pd.set_option('display.max_columns', None)
    results_map = ResultsMap(

```

```

        systems=model.system_models,
        similarity_measures=model.get_similarity_measures(),
        prototype=prototype,

similarity_measure_type=self.prototype_gui.get_similarity_measure_
type()
    )

    return results_map

    async def _calculate_combinations_async(self):
        if not self.systems_data:
            return

        progress_dialog = QProgressDialog("Calculating...",
"Cancel", 0, 0, parent=self)
        progress_dialog.setFixedSize(300, 100)
        progress_dialog.setWindowTitle("Please wait")
        progress_dialog.setWindowModality(Qt.WindowModal)

        future = None

        def cancel_calculation():
            if future:
                future.cancel()
                progress_dialog.close()

        progress_dialog.canceled.connect(cancel_calculation)

        try:
            loop = asyncio.get_running_loop()
            future = loop.run_in_executor(None,
self._calculate_combinations_sync)

            await asyncio.sleep(0.5)
            if not future.done():
                progress_dialog.show()
            results_map = await future
            if progress_dialog.wasCanceled():
                return
            if results_map is None:
                return

self.tabs_manager.add_result_tab(ResultsTab(results=results_map))

        except Exception as e:
            print(f"Error during calculation: {e}")
        finally:
            progress_dialog.close()

    def get_current_result_tab(self):
        return self.tabs_manager.get_current_result_tab()

```


Лістинг 3 – Код EmptySystemTab

```

class EmptySystemsTab(QWidget):
    def __init__(self, parent):
        super().__init__(parent=parent)
        self.init_ui()

    def init_ui(self):
        systems_layout = QVBoxLayout()
        buttons_layout = QHBoxLayout()

        label = QLabel("The systems tab is empty. Click the button
below to add a new tab or upload an excel file.")
        label.setAlignment(Qt.AlignCenter)

        self.add_tab_button = QPushButton("Add")
        self.add_tab_button.setFixedWidth(100)
        self.load_from_file_button = QPushButton("Load File")
        self.load_from_file_button.setFixedWidth(100)

        buttons_layout.addWidget(self.add_tab_button)
        buttons_layout.addWidget(self.load_from_file_button)
        buttons_layout.setAlignment(Qt.AlignCenter)

        systems_layout.addStretch()
        systems_layout.addWidget(label, alignment=Qt.AlignCenter)
        systems_layout.addLayout(buttons_layout)
        systems_layout.addStretch()

        self.setLayout(systems_layout)

    def connect_action_to_add_button(self, action):
        self.add_tab_button.clicked.connect(action)

    def connect_action_to_upload_button(self, action):
        self.load_from_file_button.clicked.connect(action)

```

Лістинг 4 – Код SystemTab

```

class SystemsTab(QWidget):
    noTabsLeft = pyqtSignal()

    def __init__(self, parent, main_window=None,
on_content_change=None):
        super().__init__(parent)
        self.main_window = main_window
        self.on_content_change = on_content_change
        self.tabs = QTabWidget()
        self._init_ui()

    def _init_ui(self):
        buttons_layout = QHBoxLayout()

        for btn in self._create_toolbar_buttons():
            buttons_layout.addWidget(btn)

        buttons_layout.setAlignment(Qt.AlignLeft)
        main_layout = QVBoxLayout(self)
        main_layout.addLayout(buttons_layout)
        main_layout.addWidget(self.tabs)

        self.setLayout(main_layout)

self.tabs.tabBar().setContextMenuPolicy(Qt.CustomContextMenu)

self.tabs.tabBar().customContextMenuRequested.connect(self._show_c
ontext_menu)

    def _create_toolbar_buttons(self):
        def create_button(icon_path, tooltip, size=(60, 30),
action=None):
            btn = QPushButton()
            pixmap = QPixmap(icon_path)
            btn.setIcon(QIcon(pixmap))
            btn.setIconSize(pixmap.size())
            btn.setFixedSize(*size)
            btn.setToolTip(tooltip)

            if action:
                btn.clicked.connect(action)
            return btn

        add_button =
create_button('resources/img/buttons/add.png', 'Add new system',
action=self.add_system_tab_via_dialog_window)
        read_button =
create_button('resources/img/buttons/import.png', 'Read system
from Excel', action=self.main_window._read_from_excel)

```

```

        save_all_button =
create_button('resources/img/buttons/save_all.png', 'Save all
systems', action=self._save_all_tabs)
        save_button =
create_button('resources/img/buttons/save.png', 'Save selected
system', action=self._save_selected_table)
        delete_button =
create_button('resources/img/buttons/delete.png', 'Delete selected
system', action=self._confirm_delete_selected_table)

        return [add_button, read_button, save_all_button,
save_button, delete_button]

    def add_system_table(self, system_table: SystemTable):
        self.tabs.addTab(system_table, system_table._name)
        self.tabs.setCurrentIndex(self.tabs.count() - 1)

        if self.on_content_change is not None:
system_table.dataChanged.connect(self.on_content_change)
            self.on_content_change()

    def add_system_tab_via_dialog_window(self) -> bool:
        tab_name, ok = InputText.getText(self, "New Tab", "Enter
the name of the new tab:")

        if ok and tab_name.strip():
            if self._is_tab_name_exists(tab_name):
                QMessageBox.warning(self, "Error", f"A system with
the name '{tab_name}' already exists.")
                return False

            system_model = SystemModel(name=tab_name)
            table_widget = SystemTable(system_model)
            self.add_system_table(table_widget)
            return True
        return False

    def _show_context_menu(self, pos):
        tab_index = self.tabs.tabBar().tabAt(pos)
        if tab_index == -1:
            return

        menu = QMenu(self)
        rename_action = QAction("Rename", self)
        rename_action.triggered.connect(lambda:
self._rename_table(tab_index))
        menu.addAction(rename_action)

        delete_action = QAction("Delete", self)
        delete_action.triggered.connect(lambda:
self._confirm_delete_table(tab_index))
        menu.addAction(delete_action)

```

```

        save_action = QAction("Save Selected System", self)
        save_action.triggered.connect(lambda:
self._save_table(tab_index))
        menu.addAction(save_action)

        menu.exec_(self.tabs.tabBar().mapToGlobal(pos))

    def _rename_table(self, index):
        old_name = self.tabs.tabText(index)
        new_name, ok = InputText.getText(self, "Rename System",
"Enter new tab name:", text=old_name)

        if ok and new_name.strip():
            if new_name != old_name and
self._is_tab_name_exists(new_name):
                QMessageBox.warning(self, "Error", f"A system with
the name '{new_name}' already exists.")
                return

            self.tabs.setTabText(index, new_name)
            self.tabs.widget(index)._name = new_name

            if self.on_content_change is not None:
                self.on_content_change()

    def _confirm_delete_table(self, index):
        reply = QMessageBox.question(self, 'Confirm Delete', "Are
you sure you want to delete this tab?",
                                   QMessageBox.Yes |
QMessageBox.No, QMessageBox.No)
        if reply == QMessageBox.Yes:
            self._delete_table(index)

            if self.on_content_change is not None:
                self.on_content_change()

    def _delete_table(self, index):
        self.tabs.removeTab(index)
        if self.tabs.count() == 0:
            self.noTabsLeft.emit()

    def _confirm_delete_selected_table(self):
        tab_index = self.tabs.currentIndex()
        if tab_index != -1:
            self._confirm_delete_table(tab_index)

    def _save_selected_table(self):
        tab_index = self.tabs.currentIndex()
        if tab_index != -1:
            self._save_table(tab_index)

    def _save_table(self, index):

```



```

        tab_widget = self.tabs.widget(index)
        if not tab_widget:
            return

        if tab_widget.is_empty():
            QMessageBox.warning(self, "No data", "This tab is
empty. Nothing to save.")
            return

        data = tab_widget.to_system_model().data
        file_path, _ = QFileDialog.getSaveFileName(self, "Save Tab
as Excel", "", "Excel Files (*.xlsx)")
        if file_path:
            saver = ExcelSaver(file_path)
            saver.save_tab(self.tabs.tabText(index), data)

    def _save_all_tabs(self):
        file_path, _ = QFileDialog.getSaveFileName(self, "Save All
Tabs as Excel", "", "Excel Files (*.xlsx)")

        if file_path.strip():
            self.save_to_file(file_path)

    def save_to_file(self, file_path):
        for i in range(self.tabs.count()):
            tab_widget = self.tabs.widget(i)
            if isinstance(tab_widget, SystemTable):
                data = tab_widget.to_system_model().data
                saver = ExcelSaver(file_path)
                saver.save_tab(self.tabs.tabText(i), data)

    def _is_tab_name_exists(self, name: str) -> bool:
        for i in range(self.tabs.count()):
            if self.tabs.tabText(i).strip().lower() ==
name.strip().lower():
                return True
        return False

```

Лістинг 5 – Код PrototypeGUI

```

class PrototypeGUI(QWidget):
    def __init__(self, prototype: Prototype, parent=None):
        super().__init__(parent)
        self.prototype = prototype
        self._init_ui()

    def _init_ui(self):
        self.main_layout = QVBoxLayout()
        self.setLayout(self.main_layout)

        self.label = QLabel("<h2>System Prototype</h2>")
        self.label.setAlignment(Qt.AlignCenter)
        self.main_layout.addWidget(self.label)

        self.table = QTableWidget()
        self.main_layout.addWidget(self.table)

        similarity_layout = QFormLayout()
        similarity_layout.setSpacing(2)

        self.u_label = QLabel("Similarity measure:")
        font = self.u_label.font()
        font.setBold(True)
        self.u_label.setFont(font)

        self.u_type = QComboBox()
        self.u_type.addItem(["Sorensen-Dice", "Jaccard"])

        similarity_layout.addRow(self.u_label, self.u_type)
        self.main_layout.addLayout(similarity_layout)

        self.calculate_button = QPushButton("Calculate
combinations")
        self.main_layout.addSpacing(10)
        self.main_layout.addWidget(self.calculate_button)

        self.populate_table()

    def populate_table(self):
        self.table.setRowCount(len(self.prototype))
        self.table.setColumnCount(3)

        self.table.setSizePolicy(QSizePolicy.Expanding,
QSizePolicy.Expanding)

        self.table.horizontalHeader().setSectionResizeMode(QHeaderView.Str
etch)

        self.table.setHorizontalHeaderLabels(["System", "Feature",
"State"])

```

```

self.table.verticalHeader().setSectionResizeMode(QHeaderView.Fixed
)
    self.table.verticalHeader().setDefaultSectionSize(30)

    for row_index, (index, state) in
enumerate(self.prototype.items()):
        system, feature = index

        system_item = QTableWidgetItem(system)
        system_item.setFlags(system_item.flags() &
~Qt.ItemIsEditable)
        self.table.setItem(row_index, 0, system_item)

        feature_item = QTableWidgetItem(feature)
        feature_item.setFlags(feature_item.flags() &
~Qt.ItemIsEditable)
        self.table.setItem(row_index, 1, feature_item)

        checkbox = CenteredCheckbox(state=bool(state))
        checkbox.stateChanged(lambda state, index=index:
self.on_state_changed(index, state))
        self.table.setCellWidget(row_index, 2, checkbox)

    self.table.cellClicked.connect(self.on_cell_clicked)

def on_state_changed(self, index, state):
    self.prototype[index] = 1 if state == Qt.Checked else 0

def on_cell_clicked(self, row, column):
    if column == 2: # State column
        checkbox = self.table.cellWidget(row, column)
        checkbox.toggle()

def get_prototype(self):
    return self.prototype

def set_measure_type(self, measure_type: str):
    if measure_type == "sorensen_dice":
        self.u_type.setCurrentIndex(0)
    elif measure_type == "jaccard":
        self.u_type.setCurrentIndex(1)
    else:
        pass

def get_similarity_measure_type(self):
    return self.u_type.currentText().replace("-", "_").lower()

```

Лістинг 6 – Код ResultTab

```

class ResultsTab(QWidget):
    def __init__(self, results: ResultsMap, parent=None):
        super().__init__(parent)
        self._init_fields(results)
        self._init_ui()

    def _init_fields(self, results: ResultsMap):
        self._results = results
        self.columns_len = len(results.systems_names) + 1

    def _init_ui(self):
        self.main_layout = QVBoxLayout()
        self.header_layout = QHBoxLayout()

        self.label = QLabel("<h2>Results</h2>")
        self.save_button = QPushButton("Save to Excel")

        # Button layout
        buttons_layout = QHBoxLayout()
        buttons_layout.addWidget(self.save_button)
        buttons_layout.setAlignment(Qt.AlignRight)

        self.table = QTableWidgetItem()
        self.table.setSizePolicy(QSizePolicy.Expanding,
QSizePolicy.Expanding)
        header = self.table.horizontalHeader()
        header.setSectionResizeMode(QHeaderView.Stretch)

        self.header_layout.addWidget(self.label)
        self.header_layout.addLayout(buttons_layout)
        self.main_layout.addLayout(self.header_layout)

        self.main_layout.addWidget(self.table)

        self.setLayout(self.main_layout)

        self.save_button.clicked.connect(self._save_to_excel)

        self._setup_table()
        self._set_up_table_data()

    def _setup_table(self):
        self.table.setColumnCount(self.columns_len)

    self.table.setHorizontalHeaderLabels(list(self._results.systems_names) + ["Similarity"])
    self.table.setSortingEnabled(True)

    def _set_up_table_data(self):
        self.table.clearContents()
        self.table.setRowCount(0)

```

```

        results_df = self._results.results
        self.table.setRowCount(len(results_df))

        for row_index, row in results_df.iterrows():
            for col_index, value in enumerate(row):
                self.table.setItem(row_index, col_index,
QTableWidgetItem(str(value)))

        self.table.sortByColumn(self.columns_len - 1,
Qt.DescendingOrder)
        self.table.resizeRowsToContents()

    def _save_to_excel(self):
        file_path, _ = QFileDialog.getSaveFileName(self, "Save
Results", "", "Excel Files (*.xlsx)")
        if file_path:
            if not file_path.endswith(".xlsx"):
                file_path += ".xlsx"
            self._results.to_excel(file_path)

    @property
    def results(self):
        """
        ResultsMap: ResultsMap object.
        """
        return self._results

    @results.setter
    def results(self, results: ResultsMap):
        """
        Set the results to be displayed in the table.
        """
        self._results = results
        self._setup_table()
        self._set_up_table_data()

    @property
    def table_results(self):
        """
        DataFrame with the results.
        """
        return self._results.results

    @property
    def data(self):
        """
        Dict with the parameters used to generate the results.
        """
        return self._results.data

```

Лістинг 7 – Код асистента

```

class MoMoAgent:
    def __init__(self):
        self.llm = self._create_llm()
        self.context_data = AssistantData()
        self.crew = self._create_crew()

    def _create_llm(self) -> Any:
        if CONFIGS.openai_api_key:
            return LLM(
                api_key=CONFIGS.openai_api_key,
                model=CONFIGS.model_to_use,
            )
        elif CONFIGS.anthropic_api_key:
            return LLM(
                api_key=CONFIGS.anthropic_api_key,
                model=CONFIGS.model_to_use,
            )
        else:
            raise ValueError("No API keys available for language
models")

    def _create_crew(self) -> Crew:
        tools = []
        if CONFIGS.serper_api_key:
            import os
            os.environ["SERPER_API_KEY"] = CONFIGS.serper_api_key
            tools.append(SerperDevTool())

        agent, tasks = self._create_agent_and_tasks(tools)

        return Crew(
            agents=[agent],
            tasks=tasks,
            verbose=True,
            memory=True
        )

    def _create_agent_and_tasks(self, tools: List) -> tuple:
        agent_config, task_configs =
AgentSettingsLoader.load_settings(
    "src/assistant/configs/settings/answer_agent.yaml"
)

        agent = Agent(
            name=agent_config.name,
            role=agent_config.role,
            goal=agent_config.goal,
            backstory=agent_config.backstory,
            verbose=agent_config.verbose,
            allow_delegation=agent_config.allow_delegation,
            llm=self.llm,

```

```

        tools=tools,
        max_retry_limit=3
    )

    tasks = []
    for task_config in task_configs:
        tasks.append(Task(
            name=task_config.name,
            description=task_config.description,
            expected_output=task_config.expected_output,
            agent=agent,
            tools=tools
        ))

    return agent, tasks

    def set_results(self, prototype: str, results: str, metric:
str, systems: str):
        self.context_data.prototype = prototype
        self.context_data.result_

```