Вінницький національний технічний університет

Факультет інтелектуальних інформаційних технологій та автоматизації

Кафедра комп'ютерних наук

Бакалаврська дипломна робота на тему:

ПРОГРАМНИЙ МОДУЛЬ ОЦІНЮВАННЯ ЯКОСТІ ТЕСТОВИХ ЗАВДАНЬ В ОСВІТНІХ ВИМІРЮВАННЯХ

Виконав: студент 4 курсу, групи КН-22мс
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Христянчук В. В.
(прізвище та ініціали)
Керівник: доцент к. ф. – м.н.
Шевчук О. Ф.
(прізвище та ініціали)
"07" 06 2024 р.
Рецензент: к.т.н., доцент каф САІТ
Горячев Г. В.
(прізвище та ініціали)
"07" 06 2024 р.

Допущено до захисту
Зав. кафедри: проф., д.т.н Яровий А.І.
"07" 06 2024 р.

Вінниця ВНТУ – 2024 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 "Інформаційні технології"
Спеціальність – 122 "Комп'ютерні науки"
Освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А.А.
" 07 " 03 2024 р

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Христянчуку Владиславу Валерійовичу

1. Тема роботи: Програмний модуль оцінювання якості тестових завдань в освітніх вимірюваннях.
Керівник роботи: доцент Шевчук Олександр Федорович.
Затверджені наказом вищого навчального закладу " 11 " 03 2024 року № 80

2. Термін подання студентом роботи 27.05.2024

3. Вихідні дані до роботи :
Кількість тестів для моніторингу - не менше 30;
Кількість учасників тестування - не менше 1000;
Мова програмування - об'єктно-орієнтована;
Кількість рекомендацій - не менше 3;

4. Зміст текстової частини (перелік питань, які потрібно розробити):
вступ; аналіз предметної області; опис структури програмного модуля оцінки якості тестових завдань; розробка структури програмного модуля оцінки якості тестових завдань; розробка складових програмного модуля оцінки якості тестових завдань; висновки; список використаних джерел; додатки.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень):
Структурна схема взаємодії блоків програмного модуля між собою; графік навчання моделі;

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Доцент к.ф – м.н. Шевчук О. Ф.		

7. Дата видачі завдання 09.02.2024

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
	Аналіз предметної області оцінки якості тестових завдань	06.05.24	10.05.24	Виконано
	Опис структури програмного модуля оцінки тестових завдань	11.05.24	15.05.24	Виконано
	Розробка структури програманого модуля оцінки тестових завдань	16.05.24	24.05.24	Виконано
	Розробка структури складових програмного модуля оцінки якості тестових завдань	27.05.24	29.05.24	Виконано
	Розробка інструкції користувача	30.05.24	03.06.24	Виконано
	Аналіз отриманих результатів та формування висновків	03.06.24	04.06.24	Виконано
	Оформлення бакалаврської дипломної роботи	04.06.24	06.06.24	Виконано

Студент (підпис) **Христянчук В. В.** (прізвище та ініціали)

Керівник роботи (підпис) **Шевчук О. Ф.** (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 88 сторінок формату А4, на яких є 34 рисунок, список використаних джерел містить 10 найменувань.

Ця робота присвячена розробці програмного модуля для оцінки якості тестових завдань в освітніх вимірюваннях, використовуючи нейромережеві технології та сучасні інструменти програмування.

У рамках бакалаврської роботи розглянуто наступні аспекти: проектування та розробка середовища для зберігання інформації про тестові завдання, створення інтерфейсу користувача для введення, редагування та аналізу тестових завдань, розробка нейромережевих алгоритмів для автоматичної оцінки якості тестових завдань, забезпечення безпеки та конфіденційності даних у системі.

Результатом даного дослідження є функціональний програмний модуль у вигляді веб додатку, що реалізований на мові Python у програмному середовищі Visual Studio Code, який дозволить освітнім установам оптимізувати процеси оцінювання якості тестових завдань, підвищити об’єктивність оцінювання, зменшити помилки та покращити загальний рівень освітнього процесу.

Ключові слова: оцінка якості, тестові завдання, освітні вимірювання, алгоритм, веб-додаток, Python, Django, зберігання інформації, інтерфейс користувача, нейромережа, автоматизація.

ABSTRACT

The bachelor thesis consists of 88 pages of A4 format, on which there is 34 figure, the list of used sources contains 10 names.

This work is devoted to the development of a software module for evaluating the quality of test tasks in educational measurements, using neural network technologies and modern programming tools.

The following aspects were considered in the framework of the bachelor's thesis: design and development of an environment for storing information about test tasks, creation of a user interface for entering, editing and analysis of test tasks, development of neural network algorithms for automatic assessment of the quality of test tasks, ensuring security and confidentiality of data in the system.

The result of this research is a functional software module in the form of a web application, implemented in Python in the Visual Studio Code programming environment, which will allow educational institutions to optimize the processes of evaluating the quality of test tasks, increase the objectivity of evaluation, reduce errors and improve the overall level of the educational process.

Keywords: quality assessment, test tasks, educational measurement, algorithm, web application, Python, Django, information storage, user interface, neural network, automation.

ЗМІСТ

ВСТУП

Актуальність дослідження. У сучасному світі знань, швидкість та якість їх засвоєння стає все більш важливою умовою успіху в будь-якій сфері життя. В освіті вимагається вища якість викладання та засвоєння навчального матеріалу, щоб готувати фахівців на високому рівні для різних галузей економіки та науки. Тестування знань є одним з основних інструментів для оцінювання рівня засвоєння матеріалу студентами, а також для визначення їхніх сильних та слабких сторін в навчанні.

Програмний модуль оцінки якості тестових завдань в освітніх вимірюваннях є актуальним у зв'язку з тим, що він дозволяє освітнім установам ефективно оцінювати якість тестових завдань та отримувати об'єктивну інформацію про їх відповідність освітнім стандартам. За допомогою даного модуля можна визначити, наскільки ефективно тестові завдання оцінюють знання студентів, а також виявити можливі проблеми у формулюванні запитань чи розподілі відповідей, що дозволяє підвищити якість освітнього процесу в цілому.

Оцінка якості тестових завдань [1] буде корисною для студентів будь-яких спеціальностей та рівнів навчання, а також для викладачів, які зможуть відслідковувати якість своїх тестових матеріалів та коригувати їх відповідно до потреб кожного студента. Таким чином, програмний модуль оцінки якості тестових завдань є актуальним та важливим інструментом для покращення якості освіти та досягнення більш ефективного навчання студентів. Даний модуль допоможе визначити найбільш складні для розуміння теми, знайти слабкі місця у тестових завданнях та надати рекомендації щодо їх удосконалення. Користувачі такої системи отримують перевагу в процесі оцінювання знань та можуть бути більш успішними в своїй освітній кар'єрі, що має значний вплив на їхню майбутню професійну діяльність. Таким чином, програмний модуль оцінки якості тестових завдань буде важливим інструментом для розвитку як студентів, так і навчальних закладів в цілому.

Метою дослідження є підвищення якості процесу моніторингу тестових завдань в освітніх вимірюваннях.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

- проаналізувати сучасні підходи до оцінки якості тестових завдань;
- розробити удосконалений алгоритм оцінки якості тестових завдань;
- розробити структуру нейромережевого модуля для оцінки якості тестових завдань;
- розробити складові нейромережевого модуля для оцінки якості тестових завдань.

Об’єктом дослідження є процес оцінки якості тестових завдань.

Предметом дослідження є програмні засоби оцінки якості тестових завдань.

Апробація результатів роботи.

Результати досліджень апробувались на такій конференції:

- «Науково-технічна конференція підрозділів Вінницького національного технічного університету (НТКП ВНТУ–2024)» [1].

Публікації: електронні тези конференції «Науково-технічна конференція підрозділів Вінницького національного технічного університету (НТКП ВНТУ–2024)» [1].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ОЦІНКИ ЯКОСТІ ТЕСТОВИХ ЗАВДАНЬ

1.1 Актуальність теми

Ефективна оцінка якості тестових завдань у сучасному освітньому середовищі є важливою складовою успішної стратегії для навчальних закладів будь-якого рівня та спеціалізації. Швидке зростання складності навчальних програм і постійний тиск щодо покращення якості освіти ставлять перед освітніми установами завдання ефективно оцінювати якість тестових завдань, щоб забезпечити об'єктивність і надійність оцінювання знань студентів. У цьому контексті використання нейромережевого програмного модуля для оцінки якості тестових завдань стає ключовим елементом стратегії підвищення якості освіти.

Нейромережевий програмний модуль допомагає навчальним закладам оптимізувати процес оцінювання тестових завдань, забезпечуючи ефективне управління всіма етапами їх перевірки, починаючи від створення, оцінювання та до валідації. З його допомогою витрачений час на аналіз якості тестових завдань значно зменшується, а інтегровані інтелектуальні алгоритми та автоматизовані процеси допомагають уникнути помилок та суб'єктивності в оцінюванні.

Нейромережеві алгоритми дозволяють автоматично оцінювати якість тестових завдань, визначати їх відповідність навчальним цілям та стандартам, а також виявляти слабкі місця у тестах. Це забезпечує зручність для всіх учасників освітнього процесу та прискорює процес оцінювання.

Програмний модуль також надає можливість відстежувати ефективність тестових завдань, автоматично генерувати звіти про якість завдань, а також нагадувати про необхідність перегляду та оновлення тестів, що спрощує контроль за якістю навчального матеріалу. Додатково, модуль забезпечує можливість проводити аналіз ефективності тестових завдань, виявляти та аналізувати помилки, а також генерувати звіти для внутрішньої та зовнішньої звітності.

Зростання вимог до якості освіти підсилює необхідність ефективного оцінювання тестових завдань. Нейромережевий програмний модуль допомагає навчальним закладам швидко та точно аналізувати тестові завдання, забезпечуючи високу якість оцінювання знань студентів. Крім того, в умовах інтенсивної конкуренції важливо не лише створювати якісні тести, але й ефективно їх використовувати. Нейромережевий модуль дозволяє відстежувати виконання умов тестування та автоматично генерувати звіти про їх ефективність, що спрощує контроль за якістю навчального процесу та дозволяє підтримувати високу якість освітніх послуг.

Необхідність зменшення ризиків, пов'язаних з оцінюванням тестових завдань, виникає з багатьох факторів, серед яких важливими є постійні зміни навчальних програм, конкурентний тиск на ринку освіти та необхідність забезпечення правової впевненості. Використання нейромережевого програмного модуля допомагає уникнути ризиків порушення освітніх стандартів, забезпечити правову впевненість та підвищити ефективність діяльності навчальних закладів.

Значна роль у сучасній освіті належить аналізу та звітності. Нейромережевий програмний модуль надає можливість проводити детальний аналіз ефективності тестових завдань, виявляти потенційні проблеми та приймати вчасні заходи для їх усунення. Аналітичні інструменти дозволяють здійснювати глибокий розгляд ефективності різних освітніх стратегій та управлінських рішень, що допомагає навчальним закладам удосконалювати свою діяльність та досягати стратегічних цілей.

Ефективне оцінювання якості тестових завдань набуває великої актуальності в сучасному світі, оскільки суспільство постійно зазнає впливу швидко змінюючихся технологій та освітніх стандартів. Навчальні заклади повинні бути готові адаптуватися до нових вимог і змінювати умови тестування відповідно до них. Крім того, конкурентна боротьба на ринку освіти змушує навчальні заклади шукати шляхи для оптимізації своїх процесів, включаючи оцінювання якості тестових завдань. Тільки ефективне оцінювання якості тестів

допомагає навчальним закладам не лише забезпечити високий рівень освітніх послуг, а й підтримувати конкурентні позиції.

1.2 Огляд актуальних методів оцінки якості тестових завдань

Оцінка якості тестових завдань є ключовим аспектом ефективного освітнього процесу в сучасному світі. Поява нейромережевих рішень, спрямованих на автоматизацію цього процесу, відкриває нові можливості для навчальних закладів у покращенні своєї діяльності та забезпеченні об'єктивного оцінювання знань студентів.

Переваги нейромережевого програмного модуля включають високий рівень автоматизації, точність і надійність. Використання нейромереж дозволяє швидко та ефективно аналізувати тестові завдання, що є критичним аспектом у сучасних умовах динамічного освітнього середовища. Крім того, нейромережі забезпечують високу точність оцінювання, що є надзвичайно важливим при аналізі якості тестових завдань.

Для розробки програмного модуля необхідно провести детальний аналіз предметної області, а саме методів оцінки якості тестових завдань. Основні підходи до цього включають:

Класичний тестовий аналіз [2] (Classical Test Theory, CTT) - цей підхід базується на статистичному аналізі відповідей учасників тестування для визначення надійності та валідності тестових завдань. Основні метрики включають коефіцієнт альфа Кронбаха для оцінки внутрішньої узгодженості та коефіцієнт кореляції для визначення валідності (рис. 1.1).

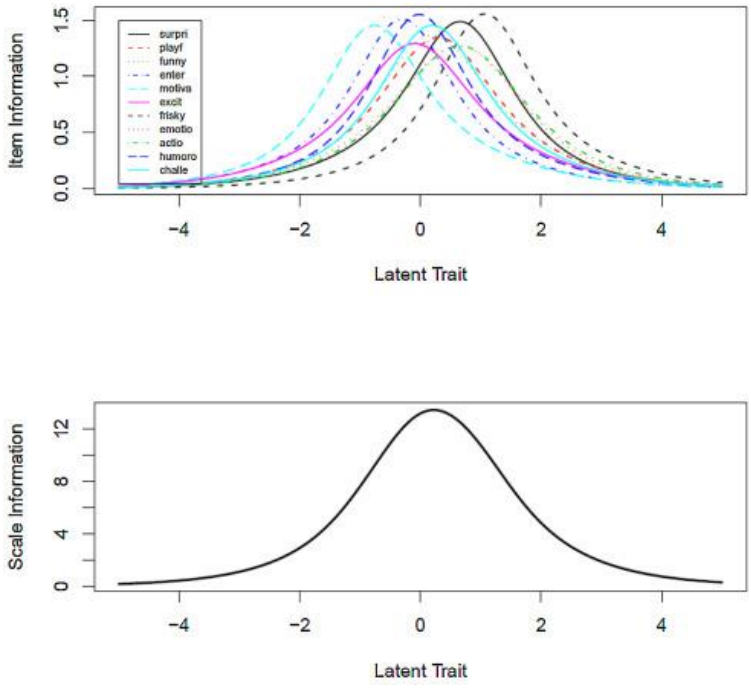


Рисунок 1.1 – Класичний тестовий аналіз

Теорія відповідей на запитання [2] (Item Response Theory, IRT) - цей метод оцінює якість тестових завдань на основі моделювання взаємозв'язку між латентною здатністю учасника тестування і ймовірністю правильної відповіді на завдання. IRT дозволяє створювати завдання з різними рівнями складності, дискримінаційною здатністю та ймовірністю вгадування (рис. 1.2).

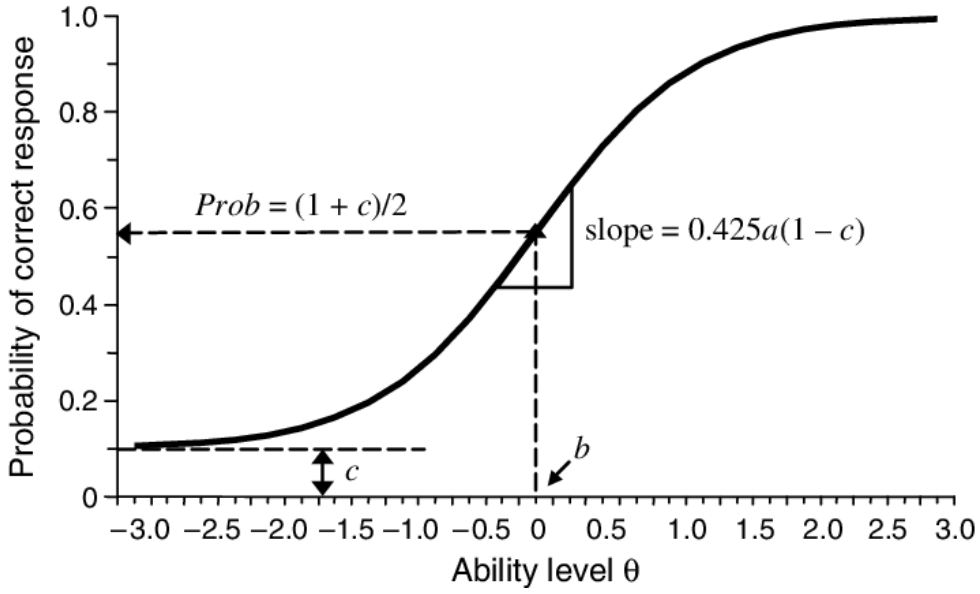


Рисунок 1.2 – Теорія відповідей на запитання

Аналіз за допомогою диференційованих функцій завдань [3] (Differential Item Functioning, DIF) - цей метод використовується для виявлення тестових завдань, які несправедливо оцінюють представників різних груп (наприклад, за статтю або етнічною приналежністю). DIF допомагає забезпечити рівноправність тестування та уникнути упереджених завдань (рис. 1.3).

Рисунок 1.3 – Аналіз диференційованих функцій завдань

Статистичні методи - використання статистичних методів, таких як лінійна регресія чи множинна регресія, а також алгоритмів машинного навчання, таких як Random Forest і Gradient Boosting, для аналізу взаємозв'язків між різними характеристиками тестових завдань та їхньою якістю. Ці методи дозволяють виявити фактори, що впливають на складність та дискримінаційну здатність завдань (рис. 1.4).

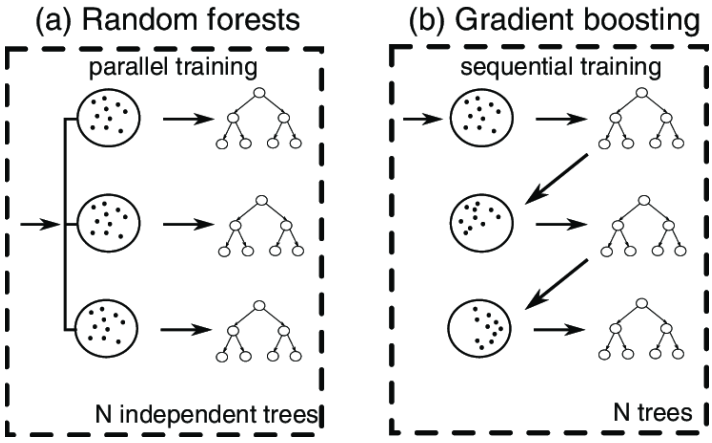


Рисунок 1.4 – Статистичні методи оцінки якості тестових завдань

Нейронні мережі - застосування рекурентних та згорткових нейронних мереж для аналізу великих обсягів даних, включаючи відповіді на тестові завдання та їхні характеристики. Моделі глибокого навчання, завдяки своїй здатності виявляти складні залежності в даних, виявляються особливо ефективними в задачах оцінки якості тестових завдань (рис. 1.5).

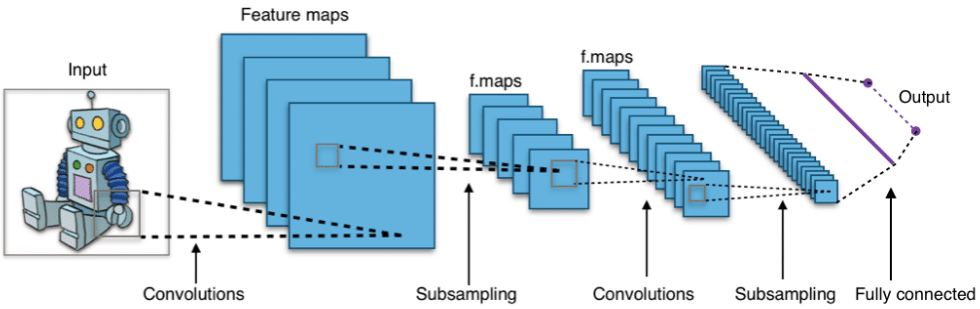


Рисунок 1.5 – Застосування нейронних мереж для оцінки якості тестових завдань

Залежно від специфіки тестових завдань, для аналізу можна використовувати різні типи нейронних мереж. Наприклад, мережі прямого поширення можуть бути оптимальними для аналізу статистичних даних, тоді як комбінація рекурентних і згорткових нейронних мереж може бути застосована для аналізу текстових та часових даних. Крім того, інтеграція передбачень з кількох спеціалізованих нейронних мереж може покращити загальну точність оцінки якості тестових завдань.

1.3 Визначення критеріїв оцінки якості тестових завдань

Ефективна оцінка якості тестових завдань в освітніх вимірюваннях вимагає дотримання певних ключових вимог. Основні критерії нейромережевого модуля оцінки якості тестових завдань включають:

1. Об'єктивність: оцінка повинна базуватися на реальних даних, отриманих від студентів під час тестування, без впливу суб'єктивних факторів.

Нейромережевий модуль має аналізувати відповіді студентів для визначення якості завдань, забезпечуючи об'єктивність результатів.

2. Точність: система повинна надавати високоточну оцінку якості тестових завдань. Використання нейромережевих алгоритмів дозволяє досягти точних результатів за рахунок глибокого аналізу даних і виявлення складних патернів.

3. Аналітика: оцінка якості повинна включати розширені аналітичні можливості, що дозволяють викладачам та адміністраторам отримувати докладну інформацію про ефективність тестових завдань та вносити необхідні зміни. Аналітичні інструменти повинні забезпечувати візуалізації, звіти та рекомендації.

4. Автоматизація: процес оцінки має бути автоматизованим, щоб забезпечити швидкість і ефективність. Нейромережевий модуль повинен автоматично збирати, обробляти та аналізувати дані без необхідності втручання з боку користувача.

5. Надійність: система повинна бути надійною та забезпечувати збереження даних про тестові завдання та результати оцінки в безпечному місці, захищеному від несанкціонованого доступу. Це гарантує збереження цілісності та конфіденційності даних.

6. Мотивація: система повинна включати елементи, що сприяють мотивації викладачів та студентів. Наприклад, надання детальних зворотних зв'язків щодо якості завдань, рекомендацій для їх покращення та індивідуальних звітів для студентів.

7. Гнучкість: система повинна бути адаптивною і легко налаштовуватися відповідно до різних освітніх стандартів та вимог. Це дозволить використовувати модуль в різних навчальних закладах та адаптувати його до специфічних потреб.

Виконання зазначених вимог дозволить створити ефективну систему оцінки якості тестових завдань, яка не тільки збирає та аналізує інформацію, але й активно впливає на процес створення та використання тестів. Це сприятиме

підвищенню якості освітніх вимірювань та забезпеченню об'єктивності й точності оцінювання знань студентів.

1.4 Виклики та постановка задачі оцінок якості тестових завдань

Оцінка якості тестових завдань є складним процесом, що включає в себе різні аспекти, кожен з яких може бути джерелом певних проблем та викликів.

Суб'єктивність оцінювання. Однією з головних проблем в оцінці якості тестових завдань є суб'єктивність, яка може впливати на результати оцінювання. Викладачі та освітні адміністратори можуть мати різні погляди на те, які завдання є якісними, а які ні. Це може призвести до варіацій у оцінці та вплинути на об'єктивність результатів. Навіть при використанні стандартних методик оцінювання людський фактор залишається значним джерелом суб'єктивності. Наприклад, різні експерти можуть по-різному оцінювати складність одного й того ж завдання.

Технологічні обмеження. Технологічні обмеження також можуть бути серйозним викликом в оцінці якості тестових завдань. Системи оцінки, що використовуються в навчальних закладах, часто залежать від наявних технічних засобів та інфраструктури. Обмеження можуть включати:

- низьку продуктивність систем обробки даних, що призводить до тривалого часу аналізу;
- обмеження у можливостях інтеграції з іншими освітніми платформами;
- недостатність підтримки сучасних алгоритмів аналізу, таких як нейронні мережі;
- аідсутність доступу до великих обсягів даних, необхідних для навчання та тестування нейронних мереж.

Питання конфіденційності даних. Конфіденційність даних є критично важливим аспектом у процесі оцінки якості тестових завдань. Збір, зберігання та

обробка великої кількості даних про студентів вимагає високого рівня безпеки. Основні виклики в цій сфері включають:

- забезпечення захисту персональних даних студентів відповідно до законодавства про конфіденційність;
- впровадження методів шифрування та безпечного зберігання даних;
- управління доступом до конфіденційних даних, щоб уникнути несанкціонованого доступу;
- врахування етичних аспектів під час збору та аналізу даних студентів.

Точність і валідність оцінок. Досягнення високої точності та валідності оцінок є важливим, але складним завданням. Помилки в оцінці можуть виникати через неправильно налаштовані алгоритми або недостатньо репрезентативні дані. Крім того, перевірка та валідація нових методів оцінки потребує значних ресурсів і часу.

Зміни в освітніх стандартах та вимогах. Освітні стандарти та вимоги можуть змінюватися, що створює додаткові виклики для систем оцінки якості тестових завдань. Системи повинні бути гнучкими та здатними швидко адаптуватися до нових стандартів та методик викладання і оцінювання.

Психологічні аспекти. Психологічний вплив на студентів також є важливим фактором. Занадто складні або незрозумілі тестові завдання можуть негативно вплинути на мотивацію студентів і їхнє ставлення до навчання. Тому необхідно враховувати психологічні аспекти під час розробки та оцінки тестових завдань, щоб забезпечити їх ефективність та позитивний вплив на навчальний процес.

Розглянуті проблеми та виклики підкреслюють необхідність комплексного підходу до оцінки якості тестових завдань, який враховує як технологічні, так і етичні, психологічні та організаційні аспекти. Це дозволить забезпечити об'єктивну, надійну та ефективну систему оцінки, яка відповідатиме сучасним вимогам освіти.

Постановка задачі. Нехай дано вхідний датасет оцінок опитуваних у тестуванні, тоді:

— якщо в тестуванні брали участь *M* осіб і проект тесту містить *N* завдань, то матриця результатів тестування буде прямокутною з розмірами *M*N*. Елементи матриці набувають значення "1", якщо відповідь правильна, або "0", якщо відповідь неправильна;

— перетворення матриці результатів полягає в її упорядкуванні за індивідуальними балами та легкістю завдань;

— підрахунок показників зв'язку тестових завдань між собою, а також тестових завдань таіндивідуальних балів осіб. Мірою такого зв'язку є коефіцієнт кореляції між відповідними величинами X_i та Y_i ;

$$r_{xy} = \frac{\sum_{i=1}^{N} X_i Y_i - \frac{1}{N}(\sum_{i=1}^{N} X_i * \sum_{i=1}^{N} Y_i)}{\sqrt{\sum_{i=1}^{N} X_i^2 - \frac{1}{N}(\sum_{i=1}^{N} X_i)^2}\sqrt{\sum_{i=1}^{N} Y_i^2 - \frac{1}{N}(\sum_{i=1}^{N} XY_i)^2}} \qquad (1)$$

де X_i - колонка відповідей на j-те завдання;

Y_i - колонку індивідуальних балів осіб;

r_{xy} є коефіцієнтом кореляції між j-м завданням та індивідуальними балами осіб;

— оброблення набору тестових завдань є визначення його надійності як інструменту для вимірювання знань з конкретної дисципліни.

$$r_{\text{н}T} = 1 - \frac{S_E^2}{S_X^2} \qquad (2)$$

де S^2_E – дисперсія похибок;

S^2_X – дисперсія балів по всьому тесті;

— визначення валідності тесту. Тест навчальної успішності називається валідним за його внутрішніми параметрами, якщо при додержанні інших категорій процесу вимірювання отримані результати відповідають критеріям об’єктивності;

— визначення валідності тестових завдань. Тестове завдання вважається валідним, якщо воно відповідає вимогам тесту, тобто забезпечує його валідність.

1.5 Висновки

У даному розділі було проаналізовано актуальні методи оцінки якості тестових завдань у контексті освітніх вимірювань. Розглянуто основні підходи до оцінки якості тестових завдань, включаючи класичний тестовий аналіз, теорію відповідей на запитання, аналіз диференційованих функцій завдань, статистичні методи та нейронні мережі. Надано їх коротку характеристику, проаналізовано переваги та недоліки кожного методу.

Визначено основні критерії оцінки якості тестових завдань, які включають об'єктивність, точність, аналітику, автоматизацію, надійність, мотивацію та гнучкість. Ці критерії дозволяють створити ефективну систему оцінки, яка не тільки збирає та аналізує інформацію, але й активно впливає на процес створення та використання тестів, забезпечуючи високий рівень якості освітніх вимірювань.

Також було розглянуто та виклики в оцінці якості тестових завдань, такі як суб'єктивність оцінювання, технологічні обмеження, питання конфіденційності даних, досягнення точності і валідності оцінок, зміни в освітніх стандартах та вимогах, а також психологічні аспекти та поставлено задачі на виконання . Всі ці аспекти підкреслюють необхідність комплексного підходу до оцінки якості тестових завдань, який враховує як технологічні, так і етичні, психологічні та організаційні аспекти.

2 ОПИС СТРУКТУРИ ПРОГРАМНОГО МОДУЛЯ ОЦІНКИ ЯКОСТІ ТЕСТОВИХ ЗАВДАНЬ

2.1 Вибір архітектури для оцінки якості тестових завдань

Вибір архітектури нейромережі є ключовим етапом у розробці програмного модуля для оцінки якості тестових завдань. У цьому підрозділі розглянуто різні архітектури нейронних мереж, їх особливості, переваги та недоліки, а також обґрунтовано вибір конкретної архітектури для вирішення завдань оцінки якості тестових завдань.

Типи нейронних мереж. Розглянемо основні типи нейронних мереж, які можуть бути використані для оцінки якості тестових завдань:

Перцептрон [5] (Perceptron): перцептрон є найпростішою формою нейронної мережі, що складається з одного шару нейронів. Він підходить для вирішення задач лінійної класифікації, але має обмежену здатність до виявлення складних залежностей у даних.

Мережі прямого поширення (Feedforward Neural Networks, FNN): цей тип нейронної мережі складається з кількох шарів нейронів (вхідного, прихованих і вихідного). FNN підходять для задач регресії та класифікації, оскільки вони можуть виявляти нелінійні залежності у даних. Однак, для аналізу послідовних даних їх застосування обмежене (рис. 2.1).

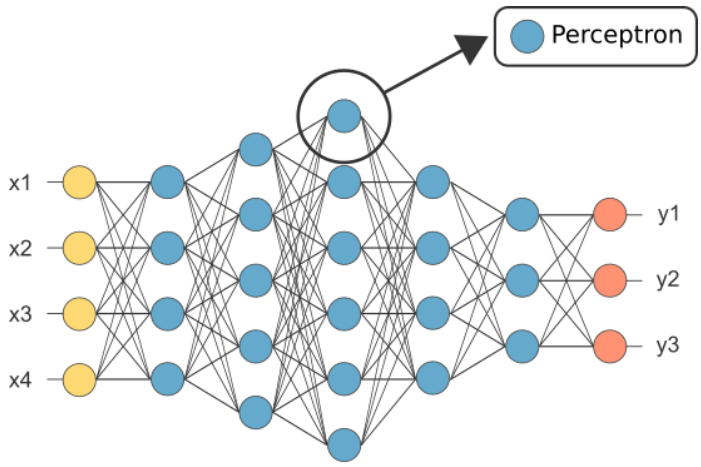


Рисунок 2.1 – Мережа прямого поширення

Рекурентні нейронні мережі (Recurrent Neural Networks, RNN): RNN спеціалізуються на обробці послідовних даних завдяки зворотним зв'язкам між нейронами. Вони підходять для аналізу тексту та часових рядів. Однак, стандартні RNN можуть мати проблеми з довготривалою пам'яттю (рис. 2.2).

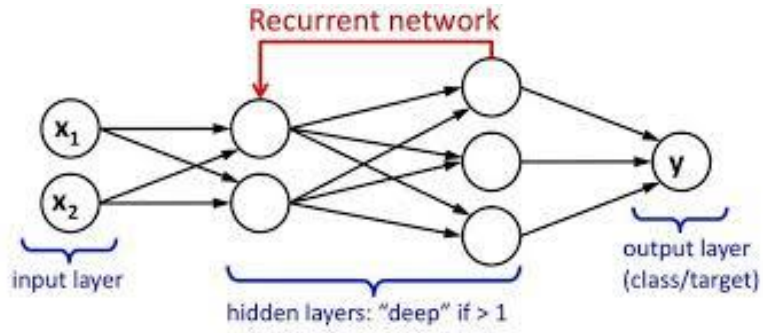


Рисунок 2.2 – Рекурентна нейронна мережа

Довго-короткотривала пам'ять (Long Short-Term Memory, LSTM): LSTM є вдосконаленою версією RNN, яка вирішує проблему довготривалої пам'яті завдяки спеціальним осередкам пам'яті. LSTM широко використовуються для аналізу тексту та складних послідовних даних (рис. 2.3).

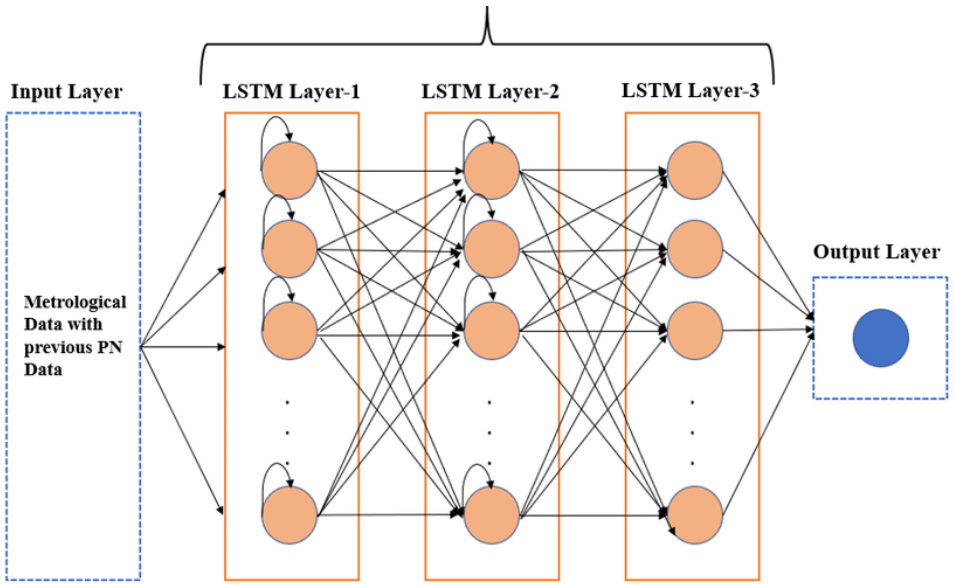


Рисунок 2.3 – Довго-короткотривала пам'ять

Згорткові нейронні мережі (Convolutional Neural Networks, CNN): CNN зазвичай використовуються для обробки зображень, але можуть застосовуватися і для аналізу текстових даних, наприклад, для виявлення патернів у текстах тестових завдань. CNN ефективні для виявлення локальних залежностей у даних (рис. 2.4).

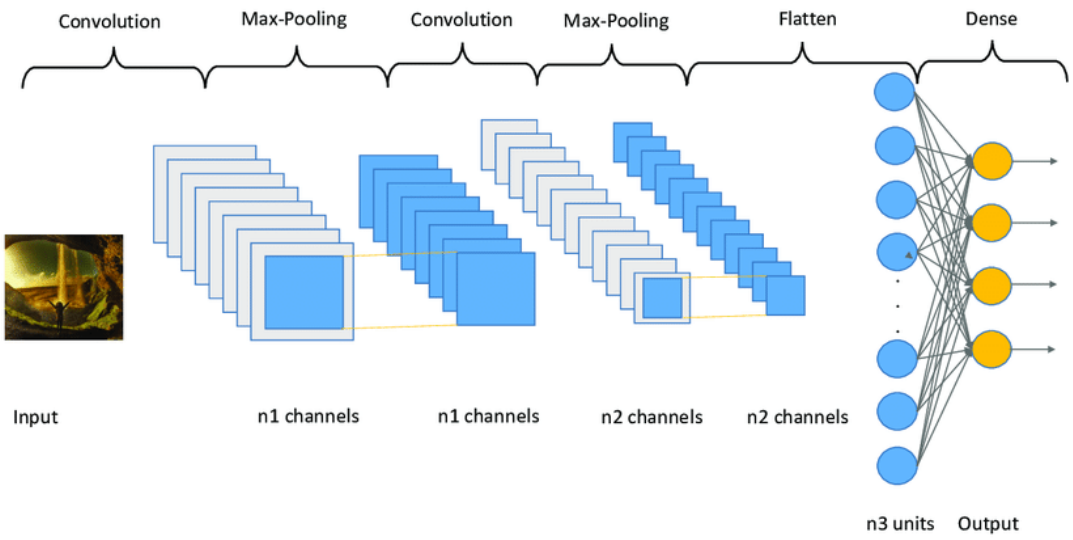


Рисунок 2.4 – Згорткова нейронна мережа

Вибір архітектури для оцінки якості тестових завдань. Для оцінки якості тестових завдань необхідно врахувати специфіку даних, які будуть аналізуватись, а саме: текстові дані, відповіді студентів, метадані про завдання тощо. Виходячи з цього, найбільш підходящими архітектурами є LSTM та CNN, або їх комбінація.

Комбінація LSTM та CNN:

1. Вхідний шар: приймає текстові дані завдань, відповіді студентів та інші метадані. Дані можуть бути попередньо оброблені та перетворені в числові вектори за допомогою методів векторизації тексту, таких як Word2Vec або GloVe.

2. Згортковий шар (CNN): використовується для виділення локальних патернів у текстових даних, таких як ключові слова або фрази, які можуть впливати на якість завдань.

3. Рекурентний шар (LSTM): обробляє послідовності даних для виявлення довготривалих залежностей та взаємозв'язків між різними частинами завдання.

4. Повнозв'язний шар [4] (Dense): забезпечує кінцеву класифікацію або регресію, перетворюючи вихідні дані з попередніх шарів у потрібний формат оцінки якості завдання.

5. Вихідний шар: видає оцінку якості тестового завдання, яка може бути у формі ймовірності, рейтингу або іншої метрики, що використовується для оцінювання.

Обґрунтування вибору. Комбінація FNN та LSTM є оптимальним вибором для завдання з наступних причин:

— здатність виявляти локальні патерни (FNN): FNN ефективно виділяють важливі локальні характеристики у тексті, що допомагає зрозуміти контекст та значимість окремих елементів завдань;
— обробка послідовних даних (LSTM): LSTM дозволяють аналізувати послідовності тексту, враховуючи контекст та взаємозв'язок між різними частинами завдання, що є важливим для повної оцінки його якості;
— гнучкість та масштабованість: комбінація цих двох архітектур дозволяє адаптувати модель до різних типів даних та завдань, забезпечуючи високу точність та надійність оцінки.

Вибір архітектури нейромережі є ключовим етапом у розробці ефективного програмного модуля для оцінки якості тестових завдань. Комбінація FNN та LSTM дозволяє врахувати всі необхідні аспекти аналізу текстових даних та забезпечити високу точність оцінки, що є критично важливим для покращення якості освітнього процесу.

2.2 Навчання та валідація нейромережевої моделі

Процес навчання та валідації нейромережевої моделі є критично важливим етапом у розробці програмного модуля для оцінки якості тестових

завдань. Цей процес включає в себе збір та підготовку даних, налаштування параметрів моделі, навчання моделі на тренувальних даних, валідацію моделі на тестових даних та оцінку її продуктивності.

1. Збір даних: Для навчання нейромережевої моделі необхідний великий обсяг даних, що включають тестові завдання, відповіді студентів та відповідні метадані. Джерела даних можуть включати існуючі бази даних освітніх установ, онлайн платформи для тестування та інші ресурси.

2. Попередня обробка даних: зібрані дані потребують попередньої обробки для забезпечення їхньої якості та відповідності вимогам моделі:

— очищення даних: видалення шуму, таких як зайві пробіли, спеціальні символи та помилки;

— нормалізація: приведення числових даних до єдиного масштабу.

3. Навчання: на основі зібраних данних, модель навчається.

4. Обрахунок: повністю готова та навчена модель отримує на виконання, попередньо згенерований датасет:

— обрахунок: за алгоритмом, нейромережа, використовуючи математичні та статистичні формули, виконує обрахунок;

— аналіз результатів: впорядкування та виведення результатів, а також рекомендації. Якщо % проходження тесту занадто малий (менше 16%), або занадто великий (більше 84 %) – ці тести вважаються невалідними.

5. Виведення результатів: у зручний для користувача веб-інтерфейс виводяться результати.

Навчання моделі

1. Розподіл даних: дані розподіляються на тренувальний, валідаційний та тестовий набори. 70% даних використовуються для навчання, 15% для валідації та 15% для тестування.

2. Налаштування параметрів моделі

— вибір гіперпараметрів: для налаштування моделі необхідні оптимальні гіперпараметри, такі як кількість шарів, кількість

нейронів у кожному шарі, розмір вікна згортки (для FNN), розмір кроку (для LSTM), активаційні функції, розмір батчу, швидкість навчання та кількість епох;

— ініціалізація моделі: створення та ініціалізація нейромережевої моделі з використанням обраних архітектурних компонентів (FNN та LSTM), відповідно до вимог задачі.

3. Процес навчання: модель навчається на тренувальних даних за допомогою алгоритму зворотного поширення помилки (backpropagation). На кожній ітерації модель оптимізує свої ваги для мінімізації функції втрат. Основні етапи включають:

— прямий прохід (forward pass): вхідні дані проходять через всі шари моделі, генеруючи вихідні значення;

— обчислення втрат: визначення різниці між передбаченими значеннями та реальними мітками;

— зворотний прохід (backward pass): оновлення ваг моделі на основі обчислених втрат за допомогою алгоритму градієнтного спуску.

Валідація моделі

— моніторинг продуктивності: під час навчання модель перевіряється на валідаційному наборі даних для моніторингу її продуктивності. Це дозволяє виявити можливі проблеми, такі як перенавчання (overfitting) або недонавчання (underfitting);

— регуляризація: для запобігання перенавчанню використовуються методи регуляризації, такі як Dropout, L2-регуляризація та інші;

— налаштування гіперпараметрів: на основі результатів проведена настройка гіперпараметрів моделі для покращення її продуктивності.

Оцінка продуктивності моделі

— тестування на тестовому наборі даних: після завершення навчання модель перевіряється на тестовому наборі даних, які не

використовувалися під час навчання та валідації. Це дозволяє отримати об'єктивну оцінку її продуктивності;

— метрики оцінки: для оцінки продуктивності моделі використовуються різні метрики, такі як точність (accuracy), повнота (recall), F1-міра (F1-score) та інші. ці метрики дозволяють оцінити якість передбачень моделі;

— аналіз результатів: результати тестування аналізуються для визначення сильних та слабких сторін моделі.

Навчання та валідація нейромережевої моделі є складним та багатоетапним процесом, який вимагає ретельного підходу та обґрунтованих рішень на кожному етапі. Вибір правильних даних, налаштування гіперпараметрів, регулярний моніторинг продуктивності та адекватна оцінка результатів є ключовими факторами успіху у створенні ефективного програмного модуля для оцінки якості тестових завдань.

2.3 Оптимізація алгоритмів оцінки якості тестових завдань

Оптимізація алгоритмів оцінки якості тестових завдань є критично важливим етапом, що дозволяє підвищити точність, швидкість і ефективність нейромережевого модуля. У цьому розділі розглянуто основні підходи до оптимізації алгоритмів, включаючи налаштування гіперпараметрів, регуляризацію, використання сучасних технік оптимізації та паралельні обчислення.

Налаштування гіперпараметрів

1. Вибір гіперпараметрів: Оптимальні значення гіперпараметрів, таких як кількість шарів, кількість нейронів у кожному шарі, розмір вікна згортки (для CNN), розмір кроку (для LSTM), швидкість навчання, розмір батчу та кількість епох, суттєво впливають на продуктивність моделі.

2. Методи налаштування:

— Grid Search: вичерпний пошук усіх можливих комбінацій гіперпараметрів;

— Random Search: випадковий вибір комбінацій гіперпараметрів у визначених діапазонах;

— Bayesian Optimization: використання байєсівських методів для оптимізації гіперпараметрів на основі попередніх результатів.

Регуляризація

— Dropout: випадкове виключення нейронів під час навчання для запобігання перенавчанню (overfitting). Це сприяє генералізації моделі та покращенню її продуктивності на нових даних;

— L1 та L2 регуляризація: додавання штрафу до функції втрат за великі ваги нейронів. L1-регуляризація сприяє розрідженості моделі, тоді як L2-регуляризація зменшує абсолютні значення ваг.

Сучасні техніки оптимізації

1. Адаптивні методи оптимізації (рис. 2.5 – 2.6):

— Adam: комбінує переваги методів Adagrad та RMSprop для ефективного та швидкого навчання;

— RMSprop: адаптивний метод градієнтного спуску, який враховує середньоквадратичну похибку попередніх градієнтів;

— Adagrad: адаптує швидкість навчання для кожного параметра на основі частоти його оновлення.

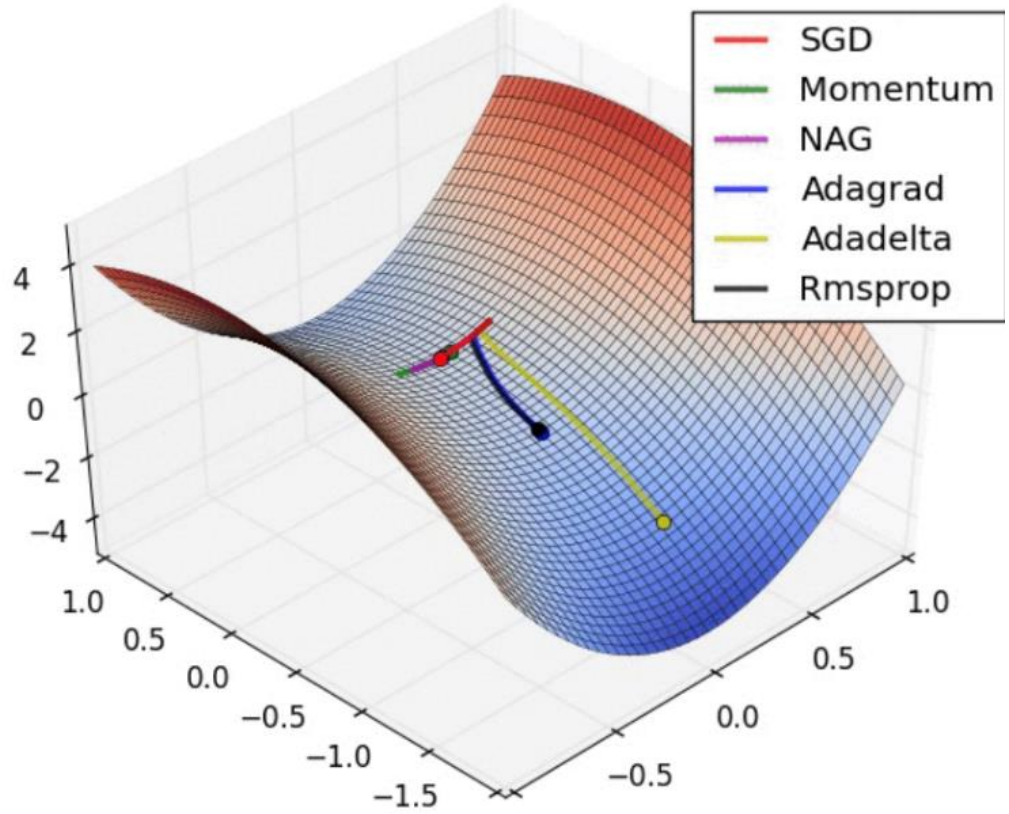


Рисунок 2.5 – Адаптивні методи оптимізації Adagrad та RMSprop

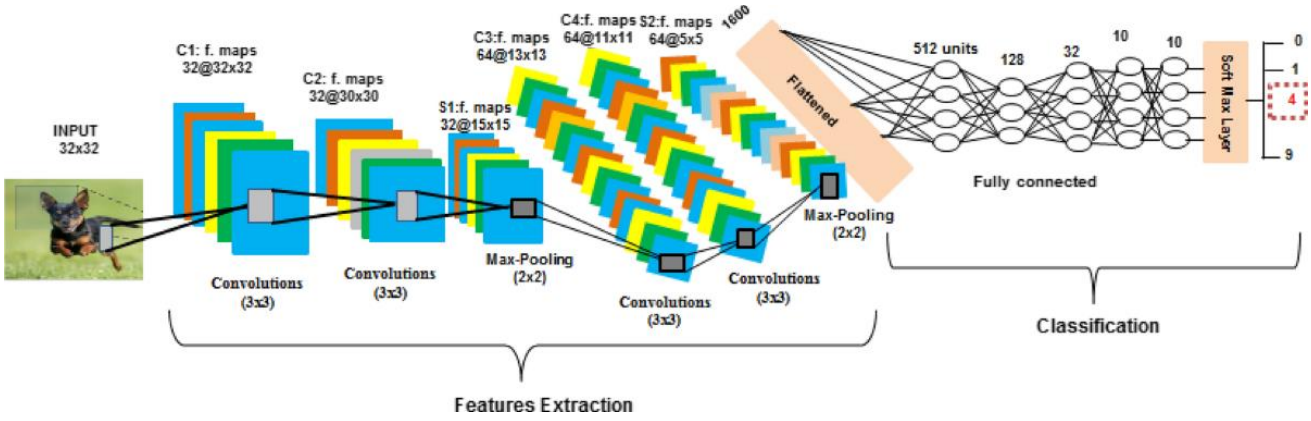


Рисунок 2.6 – Адаптивний метод оптимізації Adam

2. Експоненціальне згладжування: Використання експоненціально зважених середніх для градієнтів та квадратів градієнтів, що дозволяє стабілізувати процес навчання.

Паралельні обчислення та використання GPU [10]

— використання GPU: застосування графічних процесорів (GPU) для навчання моделей значно прискорює процес обробки великих обсягів даних та виконання складних обчислень;
— розподілене навчання: навчання моделі на декількох машинах або кластері серверів, що дозволяє обробляти ще більші обсяги даних і скорочує час навчання.

Використання попередньо натренованих моделей

— Transfer Learning: застосування попередньо натренованих моделей для вирішення схожих задач. Це дозволяє значно скоротити час навчання та підвищити точність моделі, використовуючи знання, отримані з інших задач;
— Fine-tuning: налаштування попередньо натренованої моделі на нових даних для адаптації до специфічних вимог завдання.

Постійний моніторинг та адаптація

— моніторинг продуктивності: постійне відстеження продуктивності моделі на тренувальних та валідаційних даних дозволяє вчасно виявляти та вирішувати проблеми, що виникають під час навчання;
— адаптивне навчання: внесення змін у процес навчання на основі аналізу результатів. Це включає динамічну зміну гіперпараметрів або застосування нових методів оптимізації.

Оптимізація алгоритмів оцінки якості тестових завдань є важливим етапом, який впливає на загальну ефективність нейромережевого модуля. Використання різних підходів до налаштування гіперпараметрів, регуляризації, сучасних методів оптимізації, паралельних обчислень, попередньо натренованих моделей та постійного моніторингу дозволяє досягти високої точності та надійності оцінки, що сприяє покращенню якості освітніх вимірювань.

2.4 Висновки

У процесі розробки програмного модуля для оцінки якості тестових завдань важливу роль відіграють кілька ключових етапів: вибір архітектури нейронної мережі, навчання та валідація моделі, а також оптимізація алгоритмів.

Було розглянуто різні типи нейронних мереж, такі як перцептрон, мережі прямого поширення (FNN), рекурентні нейронні мережі (RNN), довго-короткотривала пам'ять (LSTM) та згорткові нейронні мережі (CNN).

Найбільш підходящою для оцінки якості тестових завдань виявилася комбінація LSTM та CNN, що дозволяє ефективно обробляти текстові дані, виявляти локальні патерни та аналізувати послідовності.

Процес навчання моделі включає збір та попередню обробку даних, налаштування параметрів, навчання на тренувальних даних та валідацію на тестових даних.

Важливим етапом є розподіл даних на тренувальний, валідаційний та тестовий набори, що забезпечує об'єктивну оцінку продуктивності моделі.

Регуляризація та налаштування гіперпараметрів сприяють покращенню якості моделі та запобіганню перенавчанню.

Оптимізація включає налаштування гіперпараметрів, регуляризацію, використання сучасних технік оптимізації, таких як Adam, RMSprop та Adagrad, а також паралельні обчислення.

Застосування попередньо натренованих моделей (Transfer Learning) дозволяє скоротити час навчання та підвищити точність моделі.

Постійний моніторинг продуктивності моделі та адаптивне навчання дозволяють вчасно виявляти та вирішувати проблеми, що виникають під час навчання.

Таким чином, комбінація цих підходів дозволяє досягти високої точності та надійності оцінки, що сприяє покращенню якості освітнього процесу та забезпечує об'єктивну оцінку знань студентів.

3 РОЗРОБКА СТРУКТУРИ ПРОГРАМНОГО МОДУЛЯ ОЦІНКИ ЯКОСТІ ТЕСТОВИХ ЗАВДАНЬ

Програмний модуль оцінки якості тестових завдань призначений для аналізу ефективності та надійності тестових завдань на основі їх виконання студентами. Модуль повинен виконувати:

- оцінку точності прогнозів моделі;
- аналіз кореляцій;
- оцінкювати якості завдань;
- аналіз розподілу балів студентів;
- виявлення складних та легких завдань;
- візуалізацію результатів;
- розраховувати надійність за методом дисперсійного аналізу;
- оцінювати валідность завдань;
- моделювання Раша.

Перша задача модуля полягатиме у розділенні даних на тренувальний та тестовий набори. Це буде необхідно для побудови моделей, які використовуватимуться для прогнозування результатів на нових даних. Розділення даних дозволить оцінити, наскільки добре модель зможе узагальнювати інформацію, а не просто запам'ятовувати її. Тренувальний набір використовуватиметься для навчання моделі, тоді як тестовий набір використовуватиметься для перевірки її точності.

Після розділення даних модель навчатиметься на тренувальному наборі та будуватиме прогнози на тестових даних. Це дозволить оцінити ефективність моделі та порівняти прогнозовані значення з реальними результатами тестування. Візуалізація результатів порівняння допоможе виявити будь-які значні розбіжності та оцінити точність прогнозів.

Для візуалізації результатів буде використовуватися графік, на якому відображатимуться реальні значення результатів тестування на осі абсцис та прогнозовані значення на осі ординат. Це дозволить швидко оцінити, наскільки

добре модель прогнозує результати. Якщо прогнозовані значення добре співпадатимуть з реальними, точки на графіку будуть розташовані вздовж діагоналі.

Кореляційна матриця дозволить оцінити, наскільки відповіді на різні завдання взаємопов'язані між собою. Вона будуватиметься шляхом розрахунку коефіцієнтів кореляції між усіма парами завдань. Візуалізація кореляційної матриці у вигляді теплової карти дозволить швидко ідентифікувати завдання, які матимуть високий або низький ступінь кореляції.

Діаграми розподілу балів студентів будуватимуться до та після видалення невалідних завдань. Це дозволить оцінити вплив виключення завдань, які є надто легкими або надто складними, на загальний розподіл балів. Розподіл балів до видалення невалідних завдань може показати більш широку варіативність, тоді як після видалення невалідних завдань розподіл стане більш однорідним і відображатиме реальні знання студентів.

Легкість завдань визначатиметься як відсоток студентів, які відповіли на завдання правильно. Завдання з високим рівнем легкості можуть бути занадто простими, тоді як завдання з низьким рівнем легкості можуть бути надто складними. Дискримінація завдань вимірюватиме, наскільки добре завдання розрізняє студентів з різними рівнями знань. Завдання з високою дискримінацією будуть більш ефективними для оцінки знань студентів.

Аналіз кількості правильних відповідей дозволить оцінити, які завдання мають найбільшу і найменшу кількість правильних відповідей. Це допоможе виявити завдання, які можуть бути занадто легкими або занадто складними. Побудова графіків, що відображатимуть ці показники, надасть візуальне представлення ефективності кожного завдання.

На основі показників легкості завдання класифікуватимуться як надто легкі або надто складні. Завдання, які мають легкість вище певного порогу, вважатимуться надто легкими, а завдання з легкістю нижче певного порогу - надто складними. Виявлення таких завдань дозволить виключити їх з тесту для підвищення його валідності.

Візуалізація розподілу легкості та дискримінації завдань дозволить оцінити складність та ефективність завдань наочно. Гістограми та теплові карти, які відображатимуть ці показники, допоможуть зрозуміти, як розподіляються завдання за складністю та дискримінацією.

Надійність тесту оцінюватиметься шляхом розрахунку дисперсії балів та середнього квадратичного відхилення похибки вимірювання. Це дозволить визначити, наскільки стабільними є результати тестування. Чим менша дисперсія похибки вимірювання, тим вища надійність тесту.

Середній бал обчислюватиметься як середнє значення балів студентів за всіма завданнями. Крім того, оцінюватиметься кількість валідних завдань, тобто завдань, які знаходитимуться в межах допустимих порогів легкості. Це дозволить визначити, наскільки добре тест оцінює знання студентів.

Використання моделі Раша [6] дозволить розрахувати ймовірності правильної відповіді на завдання. Ця модель враховуватиме рівень знань студентів та складність завдань, що дозволить оцінити, як ці фактори впливатимуть на ймовірність правильної відповіді. Крім того, розраховуватимуться відношення шансів правильної відповіді, які допоможуть детальніше зрозуміти взаємозв’язки між студентами та завданнями.

Структура програмного модуля складається з блоків, що взаємодіють між собою для забезпечення максимально ефективної роботи і комфорту користувача:

- блок оцінки прогнозів;
- блок аналізу даних;
- блок виявлення невалідних завдань;
- блок візуалізації;
- блок розрахунку надійності;
- блок оцінки валідності;
- блок моделювання Раша.

На рисунку 3.1 наведено структурну схему взаємодії блоків

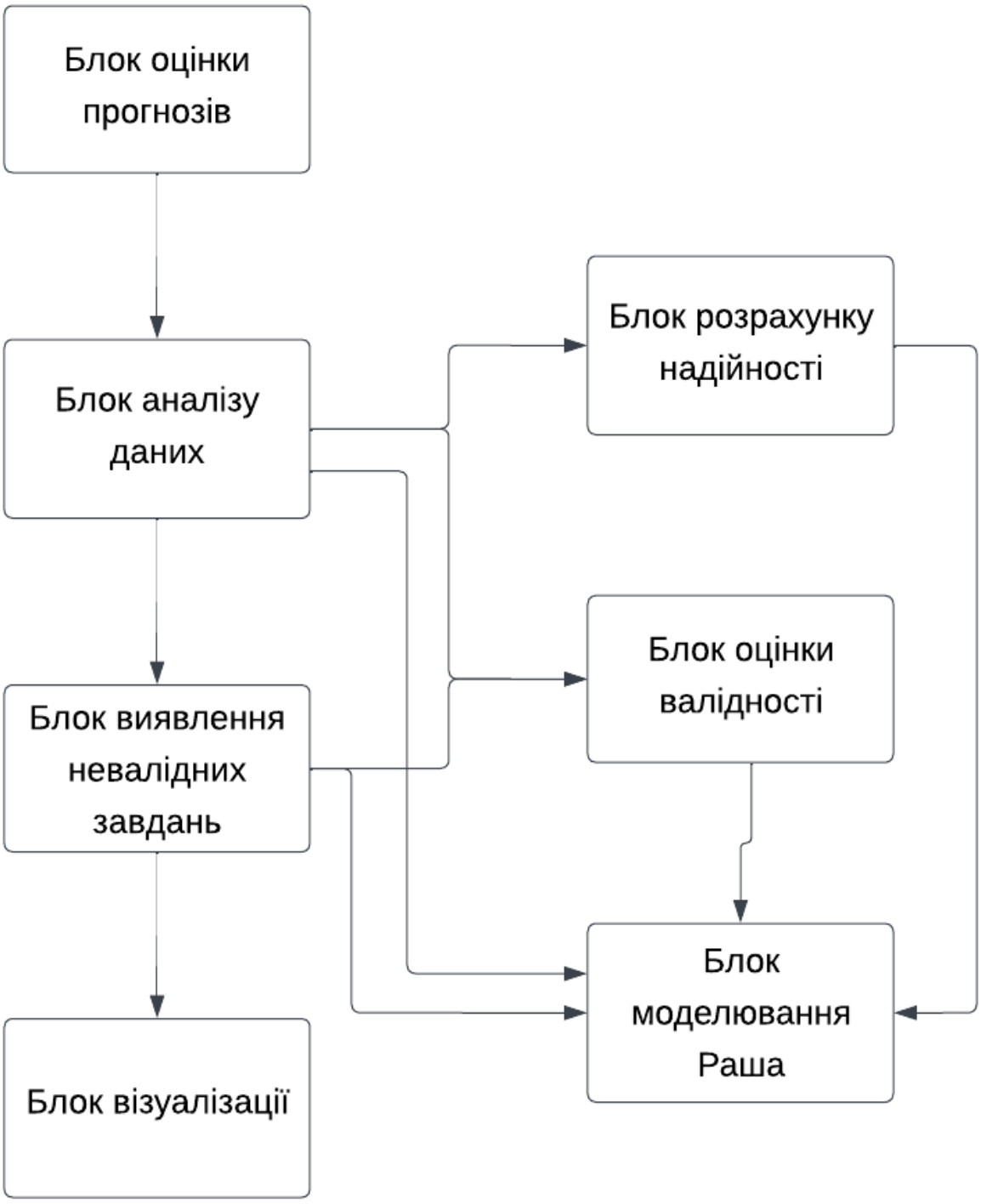


Рисунок 3.1 – Структурна схема взаємодії блоків

Ця структура забезпечує комплексний підхід до аналізу тестових завдань, дозволяючи підвищити їх ефективність, надійність та валідність. Кожен блок виконує свою специфічну роль, забезпечуючи точність та надійність оцінки тестових завдань.

3.1 Висновки

Програмний модуль оцінки якості тестових завдань розроблений для комплексного аналізу ефективності та надійності тестових завдань на основі виконання студентами. Основні висновки щодо цього модулю включають наступне:

Модуль виконує оцінку точності прогнозів моделі, аналіз кореляцій, оцінку якості завдань, аналіз розподілу балів студентів, виявлення складних та легких завдань, візуалізацію результатів, розрахунок надійності за методом дисперсійного аналізу, оцінку валідності завдань та моделювання Раша.

Першочерговим завданням модуля є розділення даних на тренувальний та тестовий набори, що дозволяє будувати моделі для прогнозування результатів на нових даних. Це дає змогу оцінити, наскільки добре модель може узагальнювати інформацію.

Після розділення даних модель навчається на тренувальному наборі та будує прогнози на тестових даних, що дозволяє оцінити її ефективність та точність.

Візуалізація результатів порівняння реальних та прогнозованих значень дозволяє швидко оцінити точність моделі.

Кореляційна матриця та її візуалізація у вигляді теплової карти дозволяють оцінити взаємозв’язки між різними завданнями.

Аналіз розподілу балів до та після видалення невалідних завдань дозволяє оцінити вплив виключення надто легких або складних завдань на загальний розподіл балів.

Легкість завдань визначається як відсоток студентів, які відповіли на завдання правильно, а дискримінація завдань вимірює, наскільки добре завдання розрізняє студентів з різними рівнями знань.

Надійність тесту оцінюється шляхом розрахунку дисперсії балів та середньоквадратичного відхилення похибки вимірювання, що дозволяє визначити стабільність результатів тестування.

4 РОЗРОБКА СКЛАДОВИХ ПРОГРАМНОГО МОДУЛЯ ОЦІНКИ ЯКОСТІ ТЕСТОВИХ ЗАВДАНЬ

4.1 Обґрунтування вибору середовища розробки

Для розробки програмного модуля оцінки якості тестових завдань необхідно вибрати зручне та ефективне середовище розробки, яке забезпечить швидкість та надійність розробки програмного забезпечення. Далі наведено обґрунтування вибору та порівняння з іншим популярним середовищем розробки PyCharm.

Visual Studio Code (VS Code).

Є безкоштовним, відкритим середовищем розробки, яке було створено компанією Microsoft. Воно надає широкі можливості для розробки на різних мовах програмування, включаючи Python, який використовується для даного проекту. VS Code включає в себе:

1. Легкість і Швидкість: VS Code є легким і швидким середовищем, що дозволяє ефективно працювати навіть на менш потужних комп'ютерах. Завдяки оптимізованому коду та невеликому розміру встановлювального файлу, VS Code завантажується та працює дуже швидко.

2. Розширюваність: VS Code має велику кількість розширень, які можна легко встановити для підтримки різних мов програмування та інструментів. Для Python доступні розширення, такі як Python Extension, Jupyter, Pylance, які полегшують розробку та тестування коду. Доступні також розширення для Docker, Kubernetes, ESLint та інших інструментів, що робить VS Code дуже гнучким.

3. Інтеграція з Git: VS Code має вбудовану підтримку систем контролю версій, таких як Git, що дозволяє зручно працювати з репозиторіями, робити комміти, переглядати історію змін та вирішувати конфлікти. Крім того, можна встановити додаткові розширення для розширених можливостей роботи з Git.

4. Налаштовуваність: середовище дозволяє налаштовувати інтерфейс та функціонал відповідно до потреб користувача, використовуючи JSON-конфігурації. Це дозволяє створювати індивідуальні налаштування для різних проєктів та забезпечує високу гнучкість у роботі.

5. Підтримка різних платформ: VS Code доступний для Windows, macOS та Linux, що робить його універсальним вибором для розробників на різних операційних системах. Це дозволяє легко переключатися між різними платформами без необхідності вивчати нове середовище розробки.

Переваги:

1. Безкоштовність: VS Code є повністю безкоштовним, без обмежень у функціоналі.

2. Велика кількість розширень: можливість додавання нових функцій та підтримка різних мов програмування завдяки розширенням.

3. Висока продуктивність: легка вага та швидка робота навіть на менш потужних комп'ютерах.

4. Гнучкість у налаштуванні: можливість налаштування середовища під конкретні потреби проекту.

5. Активна спільнота: велика спільнота користувачів та розробників, що активно підтримують та розширюють можливості VS Code.

Недоліки:

1. Потребує налаштування: для максимальної ефективності потрібне початкове налаштування та встановлення необхідних розширень.

2. Обмежена вбудована функціональність: деякі функції, такі як розширене налагодження або підтримка баз даних, потребують встановлення додаткових розширень.

PyCharm:

Це інтегроване середовище розробки (IDE), розроблене компанією JetBrains спеціально для Python. Воно також є популярним серед розробників Python через його потужні можливості та інтеграцію. PyCharm включає в себе:

1. Інтегровані Інструменти для Розробки Python: PyCharm надає розширені інструменти для розробки на Python, такі як рефакторинг коду, автоматичне завершення коду, навігація по коду, перевірка помилок у реальному часі. Це робить процес розробки більш зручним та ефективним.

2. Інтеграція з Django і Flask: PyCharm має спеціальні інструменти для розробки веб-додатків на Python з використанням фреймворків Django і Flask, що робить його ідеальним вибором для веб-розробників. Ці інструменти включають шаблони проектів, налагодження та управління базами даних.

3. Вбудована Підтримка баз даних: PyCharm включає інструменти для роботи з базами даних, що дозволяє легко керувати базами даних та виконувати SQL-запити безпосередньо з IDE. Це особливо корисно для розробників, які працюють з великими обсягами даних.

4. Підтримка Тестування: PyCharm надає вбудовані інструменти для написання та запуску тестів, що дозволяє забезпечити високу якість коду. Інструменти для тестування включають підтримку unittest, pytest та інших фреймворків для тестування.

Переваги:

1. Потужні інструменти для розробки на Python: Всі необхідні інструменти для розробки, налагодження та тестування коду.

2. Спеціалізація на Python: оптимізоване для розробки на Python, включаючи підтримку популярних фреймворків та інструментів.

3. Вбудована підтримка баз даних: легке керування базами даних та виконання SQL-запитів.

4. Інструменти для тестування: підтримка різних фреймворків для тестування, що дозволяє забезпечити високу якість коду.

Недоліки:

1. Ресурсомісткість: потребує більше ресурсів у порівнянні з VS Code, що може впливати на продуктивність на старих або менш потужних комп'ютерах.

2. Вартість: професійна версія PyCharm є платною.

3. Менш гнучке налаштування: хоча PyCharm має багато вбудованих інструментів, його налаштування менш гнучке у порівнянні з VS Code.

Порівняння Visual Studio Code та PyCharm

Продуктивність

— VS Code: швидке та легке середовище, яке не потребує багато ресурсів. Це дозволяє ефективно працювати навіть на менш потужних комп'ютерах.

— PyCharm: більш ресурсомістке середовище, що може бути менш ефективним на старих або менш потужних комп'ютерах. Потребує більше оперативної пам'яті та процесорного часу.

Розширюваність

— VS Code: має велику кількість розширень для різних мов програмування та інструментів. Це дозволяє легко додавати нові функції та підтримку мов програмування за потреби.

— PyCharm: спеціалізоване середовище для Python з багатьма вбудованими інструментами, але менш гнучке у порівнянні з VS Code. Для підтримки інших мов потребує встановлення додаткових плагінів.

Підтримка Різних Мов Програмування

— VS Code: підтримує широкий спектр мов програмування через розширення. Це робить його універсальним інструментом для розробників, які працюють з різними мовами. Ви можете легко налаштувати VS Code для роботи з JavaScript, TypeScript, Java, C++, C#, Go, Ruby, PHP та багатьма іншими мовами, просто встановивши відповідні розширення.

— PyCharm: оптимізоване для Python, хоча і підтримує інші мови через додаткові плагіни. PyCharm надає відмінні можливості для розробки на Python, але якщо вам потрібно працювати з іншими

мовами програмування, вам може знадобитися встановити додаткові плагіни або використовувати інші IDE.

Інтеграція з Інструментами Розробки

— VS Code: інтеграція з Git, Docker, Jupyter Notebooks та багатьма іншими інструментами через розширення. Це дозволяє використовувати VS Code для широкого спектра завдань, таких як контроль версій, управління контейнерами, аналіз даних та машинне навчання. Завдяки великій кількості розширень, ви можете налаштувати VS Code під конкретні потреби вашого проекту.

— PyCharm: вбудована підтримка роботи з базами даних, фреймворками Django і Flask, інструменти для тестування. PyCharm спеціально розроблений для Python-розробників, забезпечуючи всі необхідні інструменти для розробки веб-додатків, роботи з базами даних та тестування коду. Це робить його ідеальним вибором для тих, хто працює з Python і пов'язаними технологіями.

Вартість

— VS Code: безкоштовне середовище. Це робить його доступним для всіх розробників незалежно від бюджету. Всі основні функції та можливості доступні безкоштовно, що дозволяє використовувати його для будь-яких проектів без додаткових витрат.

— PyCharm: є безкоштовна версія (Community Edition) з обмеженим функціоналом, та платна версія (Professional Edition) з повним набором функцій. Community Edition підходить для базових завдань, але для доступу до всіх розширених функцій, потрібна платна версія.

Таблиця 4.1 – Порівняння характеристик сучасних засобів тестування

Параметр	VS Code	PyCharm
Продуктивність	+	-
Розширюваність	+	-
Підтримка Різних Мов Програмування	+	-
Інтеграція з Інструментами Розробки	-	+
Вартість	+	-

При виборі середовища розробки для програмного модуля оцінки якості тестових завдань основними критеріями були гнучкість, продуктивність, можливість інтеграції з різними інструментами та вартість. Visual Studio Code (VS Code) виявився оптимальним вибором завдяки своїй легкості, швидкості, великій кількості доступних розширень та можливості налаштування під конкретні потреби проекту. Крім того, безкоштовність та підтримка різних платформ роблять його універсальним та ідеальним вибором.

PyCharm, хоча і є потужним середовищем для розробки на Python, має деякі обмеження, такі як висока ресурсомісткість та вартість професійної версії. Він ідеально підходить для спеціалізованих задач у сфері Python-розробки, особливо для веб-додатків та роботи з базами даних, але може бути менш гнучким у порівнянні з VS Code для проектів, що вимагають створення нейромережевих програмних модулів.

Таким чином, для даного проєкту Visual Studio Code був обраний як середовище розробки завдяки своїм численним перевагам, що забезпечують ефективну та зручну роботу над програмним модулем оцінки якості тестових завдань.

4.2 Обґрунтування вибору мови програмування

Для розробки програмного модуля оцінки якості тестових завдань необхідно вибрати мову програмування, яка забезпечить ефективну розробку, тестування та підтримку програмного забезпечення. Нижче наведено обґрунтування вибору та порівняння з іншими популярними мовами програмування, що використовуються для розробки нейронних мереж та машинного навчання.

Python є однією з найпопулярніших мов програмування у сфері машинного навчання, аналізу даних та розробки нейронних мереж. Його популярність зумовлена кількома ключовими перевагами:

1. Python відомий своєю простою та зрозумілою синтаксичною структурою, що робить його легкою для вивчення як для початківців, так і для досвідчених програмістів. Лаконічний код та багата стандартна бібліотека дозволяють швидко розробляти та тестувати.

2. Python має широкий набір бібліотек та фреймворків для машинного навчання та нейронних мереж, таких як TensorFlow, Keras, PyTorch, Scikit-learn, Pandas, NumPy та багато інших. Ці бібліотеки надають всі необхідні інструменти для швидкої та ефективної розробки моделей машинного навчання.

3. Python має одну з найбільших та найактивніших спільнот розробників. Це забезпечує швидку підтримку та доступ до великої кількості навчальних матеріалів, документації, форумів та відкритих проектів, що значно полегшує розробку та вирішення проблем.

4. Python легко інтегрується з іншими мовами програмування, такими як C/C++ (через Cython та інші інструменти), що дозволяє оптимізувати критичні частини коду для досягнення максимальної продуктивності.

C++ — це мова програмування загального призначення, яка відома своєю високою продуктивністю та ефективним управлінням пам’яттю. Вона широко використовується у розробці системного програмного забезпечення, ігор, високопродуктивних додатків та вбудованих систем. C++ забезпечує

низькорівневий контроль над апаратним забезпеченням і має високу швидкість виконання програм завдяки компіляції в машинний код.

Переваги:

1. Висока продуктивність: C++ забезпечує високу швидкість виконання програм завдяки компіляції в машинний код та ефективному управлінню пам’яттю.

2. Контроль над ресурсами: C++ дозволяє розробникам більш точно контролювати управління пам’яттю та ресурсами, що може бути критичним для деяких високопродуктивних додатків.

Недоліки:

1. Складність синтаксису: C++ має складний синтаксис

2. Відсутність бібліотек для ML: хоча існують бібліотеки для машинного навчання на C++, вони менш популярні та мають менше функціональних можливостей у порівнянні з Python.

Java — це об’єктно-орієнтована мова програмування, яка розроблена для створення кросплатформних додатків. Вона використовується у великій кількості корпоративних додатків, веб-додатків, мобільних додатків (Android) та наукових обчислень. Java відома своєю стабільністю, безпекою та багатою екосистемою.

Переваги:

1. Портативність: Java є кросплатформною мовою, що дозволяє розробляти додатки, які можуть виконуватися на будь-якій платформі з встановленою JVM.

2. Популярність: Java широко використовується у великих корпораціях для розробки різноманітних додатків, включаючи системи машинного навчання.

Недоліки:

1. Менш інтуїтивний синтаксис: у порівнянні з Python, Java має більш складний синтаксис, що може збільшити час розробки.

2. Обмежена підтримка бібліотек для ML: хоча існують бібліотеки для машинного навчання на Java, такі як Weka та Deeplearning4j, вони менш популярні та мають менше можливостей у порівнянні з Python.

R — це мова програмування та середовище програмування, спеціально розроблене для статистичного аналізу даних та візуалізації. Вона широко використовується серед статистиків, дослідників даних та аналітиків для виконання статистичних обчислень та створення графіків.

Переваги:

1. Потужний інструмент для статистики: R спеціалізується на статистичному аналізі даних, що робить його ідеальним для деяких видів машинного навчання та аналізу даних.

2. Широкий набір бібліотек для статистики та графіки: R має багатий набір бібліотек для статистичного аналізу та візуалізації даних, таких як ggplot2, dplyr та caret.

Недоліки:

1. Менша продуктивність: R менш продуктивний у порівнянні з Python та іншими мовами для великих обчислювальних задач.

2. Обмежена підтримка загальних ML завдань: хоча R добре підходить для статистичних завдань, його підтримка для розробки нейронних мереж та загальних ML завдань менш розвинена у порівнянні з Python.

Julia — це мова програмування високого рівня, розроблена для високопродуктивних числових та наукових обчислень. Вона поєднує легкість використання з високою продуктивністю, що робить її ідеальною для наукових досліджень, машинного навчання та аналізу даних.

Переваги:

1. Висока продуктивність: Julia поєднує легкість використання з високою продуктивністю, яка наближається до рівня C.

2. Оптимізована для числових обчислень: Julia спеціально розроблена для числових та наукових обчислень, що робить її ідеальною для машинного навчання.

Недоліки:

1. Молодість мови: Julia є відносно молодою мовою, що означає меншу кількість бібліотек та менш розвинену екосистему у порівнянні з Python.

2. Менша спільнота: через молодість мови спільнота користувачів Julia значно менша.

Таблиця 4.2 – Порівняння характеристика мов програмування

Характеристика	Python	C++	Java	R	Julia
Продуктивність	+	+ +	+	-	++
Легкість написання коду	++	+	+	+	+
Інструменти для розробки нейромереж	+ +	-	-	+	+
Кількість бібліотек/фреймворків	+ +	-	-	++	-

Вибір мови програмування для розробки програмного модуля оцінки якості тестових завдань базується на кількох ключових факторах, таких як легкість у використанні, наявність потужних бібліотек для машинного навчання, активність спільноти та підтримка різноманітних інструментів і фреймворків. Python виявився найкращим вибором завдяки своїй простоті, широкому набору бібліотек і фреймворків для машинного навчання та активній спільноті розробників.

У порівнянні з іншими мовами програмування, такими як C++, Java, R та Julia, Python забезпечує оптимальний баланс між продуктивністю, легкістю використання та функціональністю, що робить його ідеальним вибором для розробки модулів машинного навчання та аналізу даних.

4.3 Програмна реалізація модуля генерації датасету

У цьому розділі описується програмна реалізація модуля генерації датасету для тестування студентів. Модуль забезпечує створення великого набору даних, який включає результати відповідей студентів на тестові завдання, а також додаткові дані, такі як імена студентів, їхні класи та кластери відповідно до рівня знань. Основною метою цього модуля є створення симуляції реального тестування, де кожен студент має унікальний набір відповідей на тестові завдання, що відповідають його кластеру знань (від "Дуже слабкого" до "Відмінника"). Це дозволяє аналізувати результати тестування, ідентифікувати складні та легкі питання, а також оцінювати ефективність тесту.

Першим кроком є завантаження необхідних даних, таких як списки прізвищ та імен, які будуть використовуватися для генерації випадкових імен студентів. Також визначається кількість студентів і питань, що будуть використовуватись для створення датасету. Для імітації різних класів студентів генерується випадковий клас, який буде спільним для всіх студентів у цьому наборі даних (рис. 4.1).

```
import pandas as pd
import matplotlib.pyplot as plt
import random
import os

with open('utils/surnames.txt', 'r', encoding='utf-8') as f:
    surnames = f.read().splitlines()

with open('utils/names.txt', 'r', encoding='utf-8') as f:
    names = f.read().splitlines()

num_students = 10000
num_questions = 40

class_number = np.random.randint(5, 13)
class_letter = np.random.choice(['А', 'Б', 'В', 'Г'])
student_class = f'{class_number}-{class_letter}'
```

Рисунок 4.1 – Завантаження списків імен та прізвищ

На цьому етапі завантажуються списки прізвищ та імен, які зберігаються в текстових файлах. Це дозволяє створити унікальні імена для кожного студента. Також визначається загальна кількість студентів та кількість питань у тесті. Для симуляції реальних умов клас кожного студента генерується випадковим чином.

Наступним кроком є генерація унікальних записів для кожного студента, включаючи їхні ПІБ та класифікація за рівнем знань (кластеризація). Студенти розподіляються на кластери відповідно до заданих ймовірностей. Цей етап необхідний для створення реалістичного розподілу студентів з різними рівнями знань (рис. 4.2).

```
class_number = np.random.randint(5, 13)
class_letter = np.random.choice(['А', 'Б', 'В', 'Г'])
student_class = f'{class_number}-{class_letter}'

def generate_full_name():
    surname = np.random.choice(surnames)
    name = np.random.choice(names)
    return f'{surname} {name}'

students_set = set()
students = []

while len(students) < num_students:
    full_name = generate_full_name()
    if full_name not in students_set:
        students_set.add(full_name)
        students.append(f'{student_class} {full_name}')

clusters = ['Дуже слабкий', 'Слабкий', 'Середній', 'Хорошист', 'Відмінник']
cluster_probabilities = [0.1, 0.1, 0.4, 0.3, 0.1]
students_per_cluster = np.random.choice(clusters, num_students, p=cluster_probabilities)
```

Рисунок 4.2 – Генерація унікальних імен та прізвищ

Тут створюється функція generate_full_name, яка генерує випадкові комбінації прізвищ та імен. Унікальність записів забезпечується шляхом перевірки наявності згенерованого імені у множині students_set. Після цього

студенти розподіляються на кластери за допомогою ймовірностей, що дозволяє відобразити реальний розподіл знань у класі.

Для кожного кластеру встановлюються ймовірності правильної відповіді на питання. Також визначаються дуже легкі та дуже складні питання. Цей етап включає в себе корекцію базових ймовірностей для кожного кластера та генерацію відповідей на основі цих ймовірностей. Це дозволяє створити набір даних, який відображає різний рівень складності питань для різних груп студентів.

Після генерації відповідей обчислюються сумарні бали для кожного студента, і результати зберігаються в Excel-файл. Цей етап включає створення DataFrame з відповідями студентів, додавання інформації про їхні імена та кластери, а також збереження всього набору даних в зручному для подальшого аналізу формат (рис. 4.3 – 4.4).

Рисунок 4.3 – Розподіл кількості успіших відповідей студентів

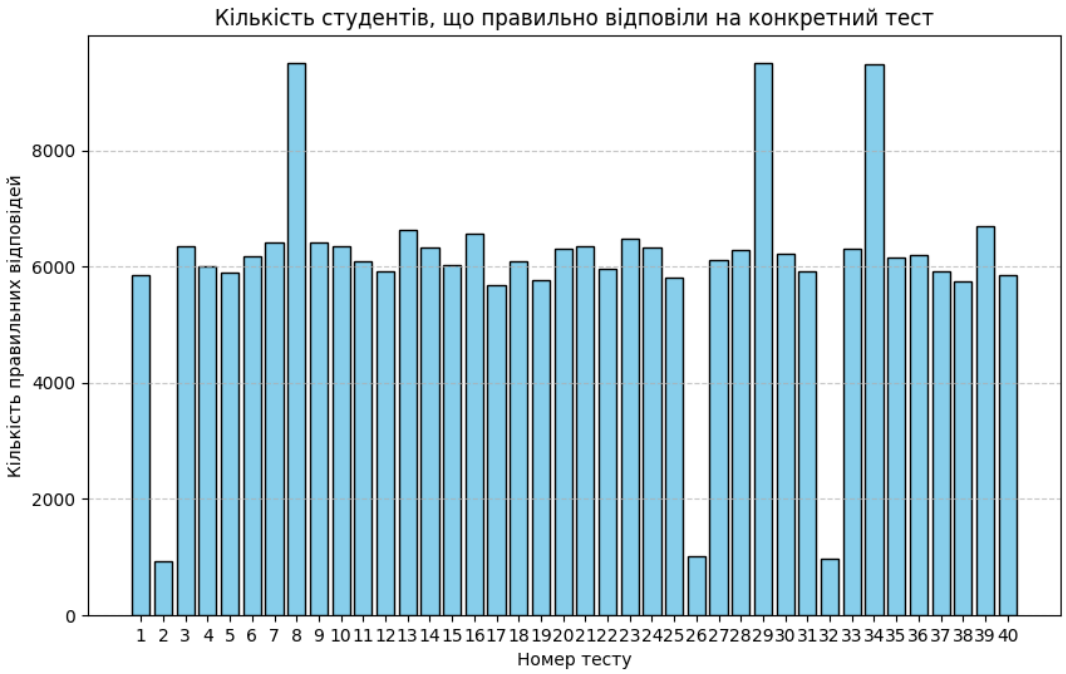


Рисунок 4.4 – Графік кількості студентів, що відповіли на конкретний тест

4.4 Програмна реалізація навчального модуля нейромережі

Модуль використовує дані, згенеровані у попередніх розділах, та включає кілька основних етапів: підготовку даних, створення і налаштування моделі, тренування моделі, збереження результатів та візуалізацію процесу навчання.

Перед початком навчання нейромережі необхідно підготувати дані. Вони включають зчитування даних з файлу, видалення непотрібних стовпців, розділення даних на навчальний та валідаційний набори, а також масштабування даних (рис. 4.5).

```
df = pd.read_excel('data/матриця_результатів_тестування.xlsx')

df = df.drop(columns=['Особи, яких тестують, I'])
df = df.drop(columns=['Кластер'])

total_scores = df.pop('Сумарний бал особи')

X_train, X_val, y_train, y_val = train_test_split(df, total_scores, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
```

Рисунок 4.5 – Підготовка даних

На цьому етапі дані завантажуються з Excel файлу, видаляються непотрібні стовпці, такі як імена студентів та їхні кластери, оскільки ці дані не впливають на результати тестування. Цільова змінна [7] (сумарний бал) відокремлюється від решти даних. Дані розділяються на навчальний (80%) та валідаційний (20%) набори для подальшого навчання та оцінки моделі. Для покращення продуктивності нейромережі дані масштабуються за допомогою StandardScaler.

Для побудови нейронної мережі використовується бібліотека TensorFlow та її високорівневий API Keras. Модель складається з кількох шарів Dense, Dropout та BatchNormalization, що дозволяє забезпечити високий рівень узагальнення та стабільності моделі (рис. 4.6).

```
model = Sequential()
model.add(Dense(256, activation='relu', input_shape=(X_train.shape[1],)))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(BatchNormalization())
model.add(Dense(32, activation='relu'))
model.add(Dense(1))
```

Рисунок 4.6 – Створення та налаштування моделі

Нейронна мережа складається з кількох шарів:

- Dense (256 нейронів): Перший шар з 256 нейронами та функцією активації ReLU;
- BatchNormalization: Нормалізація вхідних даних, що допомагає стабілізувати процес навчання;
- Dropout (0.5): Випадкове відключення 50% нейронів для запобігання перенавчанню;

— Dense (128 нейронів): Другий шар з 128 нейронами та функцією активації ReLU;

— BatchNormalization: Нормалізація вхідних даних;

— Dropout (0.5): Випадкове відключення 50% нейронів;

— Dense (64 нейрони): Третій шар з 64 нейронами та функцією активації ReLU;

— BatchNormalization: Нормалізація вхідних даних;

— Dense (32 нейрони): Четвертий шар з 32 нейронами та функцією активації ReLU;

— Dense (1 нейрон): Вихідний шар з одним нейроном для прогнозування сумарного балу.

Модель компілюється з використанням оптимізатора Adam, функції втрат MSE (mean squared error) та метрики MAE (mean absolute error).

Для навчання моделі використовуються дані навчального та валідаційного наборів. Процес навчання контролюється за допомогою колбеків EarlyStopping та ReduceLROnPlateau, що допомагає уникнути перенавчання та автоматично зменшує швидкість навчання при стабілізації втрат на валідаційному наборі (рис. 4.7).

```
early_stopping = EarlyStopping(monitor='val_loss', patience=20, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=10, min_lr=0.0001)

history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=500,
    batch_size=16,
    callbacks=[early_stopping, reduce_lr]
)
```

Рисунок 4.7 – Навчання моделі

Колбек EarlyStopping контролює процес навчання та зупиняє його, якщо втрати на валідаційному наборі не зменшуються протягом 20 епох. Колбек ReduceLROnPlateau автоматично зменшує швидкість навчання на фактор 0.2,

якщо втрати не зменшуються протягом 10 епох. Модель навчається протягом 500 епох з використанням розміру батчу 16.

Після завершення навчання модель зберігається у файл, а також генеруються графіки процесу навчання для подальшого аналізу (рис. 4.8).

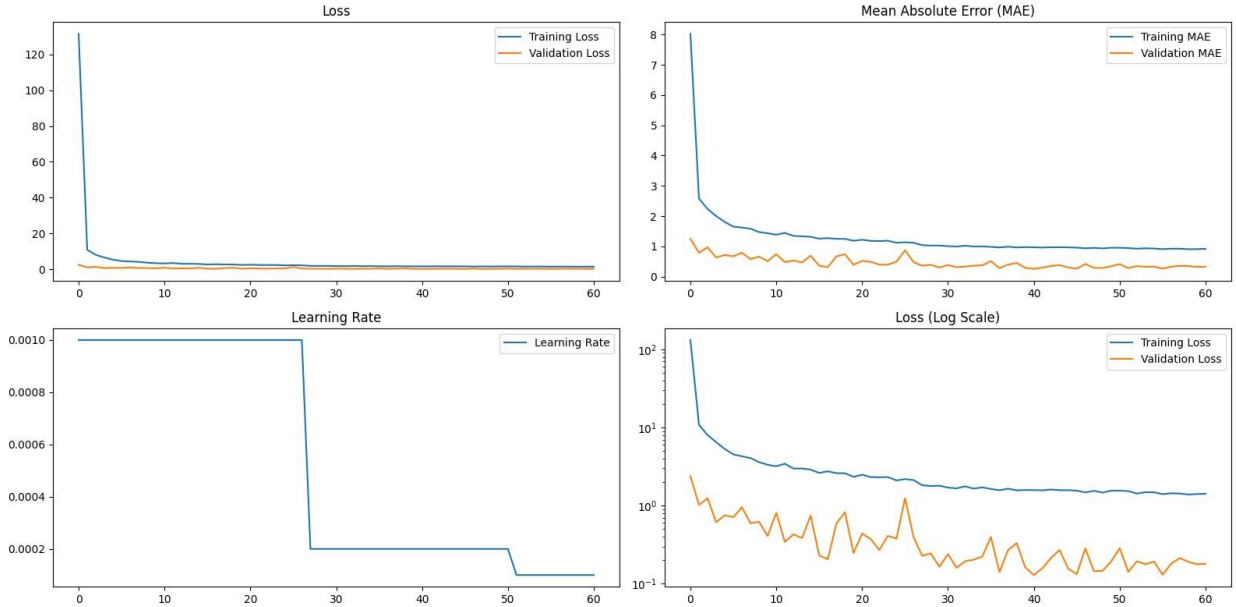


Рисунок 4.8 – Графік процесу навчання моделі

4.5 Програмна реалізація аналітичного модуля нейромережі

Аналітичний модуль виконує декілька ключових функцій, включаючи оцінку точності моделей, аналіз якості завдань, візуалізацію результатів та виявлення невалідних завдань. Цей модуль дозволяє здійснити всебічний аналіз результатів тестування та отримати корисну інформацію для покращення тестів. Спочатку створюється клас TestAnalysis, який інкапсулює всі методи для аналізу. Клас отримує на вхід DataFrame з відповідями студентів та їхні сумарні бали. Також створюється директорія для збереження графіків, якщо вона не існує. Це дозволяє забезпечити збереження та організацію візуалізацій у зручному місці (рис. 4.9).

```
class TestAnalysis:
    def __init__(self, df, total_scores):
        self.df = df
        self.total_scores = total_scores
        self.plot_dir = 'static/plots'
        self.messages = []
        if not os.path.exists(self.plot_dir):
            os.makedirs(self.plot_dir)
```

Рисунок 4.9 – Ініціалізація та підготовка даних

На початку імпортуються необхідні бібліотеки: numpy та pandas для обробки даних, matplotlib та seaborn для побудови графіків, sklearn для розділення даних, scipy для статистичних обчислень та os для роботи з файловою системою. Потім створюється клас TestAnalysis, що ініціалізується з DataFrame результатів тестування та сумарними балами студентів. Директорія для збереження графіків створюється, якщо вона ще не існує.

Метод predict_and_evaluate використовується для оцінки точності моделей. Він розділяє дані на тренувальний та тестовий набори, використовує модель для прогнозування результатів на тестовому наборі, а також будує графік порівняння прогнозованих та реальних значень. Це дозволяє візуально оцінити точність моделі та виявити можливі розбіжності між реальними та прогнозованими результатами (рис. 4.10).

```
def predict_and_evaluate(self, model):
    self.messages.append("Running predict_and_evaluate function.")
    X_train, X_test, y_train, y_test = train_test_split(self.df, self.total_scores, test_size=0.2, random_state=42)
    y_pred = model.predict(X_test)

    plt.figure(figsize=(10, 6))
    plt.scatter(y_test, y_pred, alpha=0.3)
    plt.xlabel('Реальні значення')
    plt.ylabel('Прогнозовані значення')
    plt.title('Прогнозовані vs Реальні значення')
    plt.savefig(os.path.join(self.plot_dir, '1predicted_vs_actual.png'))
    plt.close()
```

Рисунок 4.10 – Оцінка точності моделей та візуалізація результатів

Метод починається з розділення даних на навчальний та тестовий набори з використанням функції train_test_split з бібліотеки sklearn. Потім модель використовує тренувальні дані для прогнозування результатів на тестових даних. Побудований графік порівнює реальні та прогнозовані значення, що дозволяє оцінити ефективність моделі.

Метод correlation_matrix генерує кореляційну матрицю для всіх тестових завдань. Кореляційна матриця дозволяє виявити зв'язки між різними питаннями тесту та ідентифікувати можливі кореляції, які можуть вплинути на результати тестування. Це важливо для виявлення завдань, які можуть бути надто схожими або надто відрізнятись за складністю (рис. 4.11).

```
def correlation_matrix(self):
    self.messages.append("Generating correlation matrix.")
    correlation_matrix = self.df.corr()
    plt.figure(figsize=(24, 20))
    sns.heatmap(correlation_matrix, annot=False, cmap='coolwarm')
    plt.title('Кореляційна матриця тестових завдань')
    plt.savefig(os.path.join(self.plot_dir, '2correlation_matrix.png'), dpi=150)
    plt.close()
```

Рисунок 4.11 – Кореляційний аналіз

Метод quality_analysis виконує аналіз якості тестових завдань, розраховуючи їхню легкість та дискримінацію. Легкість завдання визначається як частка студентів, які правильно відповіли на питання, тоді як дискримінація показує, наскільки добре завдання розрізняє студентів з різними рівнями знань. Цей аналіз допомагає виявити завдання, які є надто легкими або надто складними для більшості студентів (рис. 4.12).

```
def quality_analysis(self):
    self.messages.append("Running quality analysis.")
    total_students = self.df.shape[0]
    correct_answers = self.df.sum(axis=0)
    difficulty = correct_answers / total_students
    discrimination = self.df.corrwith(self.total_scores)

    analysis_df = pd.DataFrame({
        'Завдання': self.df.columns,
        'Легкість': difficulty,
        'Дискримінація': discrimination,
        'Кількість правильних відповідей': correct_answers
    }).sort_values(by='Кількість правильних відповідей', ascending=False)

    self.messages.append(str(analysis_df))

    return analysis_df, difficulty, discrimination
```

Рисунок 4.12 – Аналіз якості завдань

Цей метод обчислює легкість та дискримінацію для кожного завдання. Легкість визначається як частка студентів, які правильно відповіли на питання, а дискримінація – як кореляція кожного завдання з сумарними балами студентів. Результати зберігаються в DataFrame, що дозволяє легко відсортувати та проаналізувати дані.

Метод plot_student_scores_distribution будує графіки розподілу балів студентів. Це дозволяє візуально оцінити розподіл балів до та після видалення невалідних завдань. Така візуалізація допомагає зрозуміти, як зміна набору завдань впливає на загальний розподіл результатів студентів (рис. 4.13).

```
def plot_student_scores_distribution(self, df, total_scores, title, filename):
    self.messages.append(f"Plotting student scores distribution: {title}")
    correct_answers_per_student = df.sum(axis=1)
    max_possible_score = df.shape[1]
    percentage_correct = (correct_answers_per_student / max_possible_score) * 100
    bins = [0, 60, 70, 80, 90, 100]
    labels = ['F', 'D', 'C', 'B', 'A']
    grade_counts = pd.cut(percentage_correct, bins=bins, labels=labels, include_lowest=True)
    total_students = percentage_correct.shape[0]
    grade_percentages = (grade_counts / total_students) * 100

    plt.figure(figsize=(10, 6))
    sns.barplot(x=grade_percentages.index, y=grade_percentages.values, palette='viridis')
    for index, value in enumerate(grade_percentages):
        plt.text(index, value + 0.5, f'{value:.2f}%', ha='center', fontsize=10)
    plt.title(title)
    plt.xlabel('Оцінка')
    plt.ylabel('Відсоток студентів')
    plt.ylim(0, 100)
    plt.savefig(os.path.join(self.plot_dir, filename))
    plt.close()
```

Рисунок 4.13 – Аналіз якості завдань

Метод починається з обчислення кількості правильних відповідей для кожного студента та розрахунку відсотку правильних відповідей. Потім будується гістограма, яка показує розподіл студентів за оцінками від F до A. Графік зберігається у визначеній директорії. Це дозволяє швидко оцінити загальну успішність студентів та ідентифікувати можливі проблеми з розподілом оцінок.

Метод analyze_tasks ідентифікує завдання, які є надто легкими або надто складними. Це дозволяє виключити ці завдання з тесту, щоб підвищити його валідність. Метод базується на порогових значеннях легкості завдань, які визначаються на основі розподілу відповідей студентів (рис. 4.14).

```
def analyze_tasks(self, analysis_df):
    self.messages.append("Analyzing tasks.")
    total_students = self.df.shape[0]
    easy_tasks = []
    hard_tasks = []

    for index, row in analysis_df.iterrows():
        n_responses = row['Кількість правильних відповідей']
        easy_threshold, hard_threshold = self.calculate_thresholds(n_responses)
        if easy_threshold is None or hard_threshold is None:
            continue

        adjustment = 0.1 * n_responses / total_students
        adjusted_easy_threshold = easy_threshold - adjustment
        adjusted_hard_threshold = hard_threshold + adjustment

        if row['Легкість'] >= adjusted_easy_threshold:
            easy_tasks.append(index)
        if row['Легкість'] <= adjusted_hard_threshold:
            hard_tasks.append(index)

    self.messages.append("Легкі завдання:")
    self.messages.append(str(easy_tasks))

    self.messages.append("Складні завдання:")
    self.messages.append(str(hard_tasks))

    return easy_tasks, hard_tasks
```

Рисунок 4.14 – Виявлення невалідних завдань

Цей метод аналізує кожне завдання, використовуючи обчислені легкість та дискримінацію. Завдання класифікуються як "легкі" або "складні" на основі порогових значень. Завдання з легкістю вище скоригованого порогу вважаються надто легкими, а завдання з легкістю нижче скоригованого порогу – надто складними. Це дозволяє покращити загальну валідність тесту, виключивши екстремальні завдання.

Методи visualize_task_difficulty та visualize_task_discrimination будують графіки розподілу легкості та дискримінації завдань. Це дозволяє наочно оцінити складність та ефективність завдань, що є важливим для оптимізації тесту та забезпечення його валідності (рис. 4.15).

```
def visualize_task_difficulty(self, difficulty, easy_tasks, hard_tasks):
    self.messages.append("Visualizing task difficulty.")
    plt.figure(figsize=(12, 8))

    easy_hist = plt.hist(difficulty[easy_tasks].dropna(), bins=20, edgecolor='blue', alpha=0.5, label='Легкі завдання')
    for i in range(len(easy_hist[0])):
        y_offset = 0
        for j in range(int(easy_hist[0][i])):
            x = easy_hist[1][i] + (easy_hist[1][1] - easy_hist[1][0])
            y = y_offset + 0.1
            plt.annotate(
                f'Легке завдання',
                xy=(x, y),
                xytext=(x + 0.005 * i, y + i * 0.02 + 0.02),
                textcoords='data',
                arrowprops=dict(facecolor='blue', arrowstyle='->'),
                fontsize=9,
                ha='left'
            )
            y_offset += 1
```

Рисунок 4.15 – Візуалізація розподілу легкості завдань

Метод anova_reliability обчислює надійність тесту за допомогою дисперсійного аналізу. Це дозволяє визначити, наскільки стабільними є результати тестування. Дисперсійний аналіз оцінює варіабельність результатів та визначає частку, що пояснюється справжніми відмінностями між студентами, а не випадковими коливаннями (рис. 4.16).

```
def anova_reliability(self, data):
    self.messages.append("Calculating ANOVA reliability.")
    even_columns = data.columns[::2]
    odd_columns = data.columns[1::2]
    even_scores = data[even_columns].sum(axis=1)
    odd_scores = data[odd_columns].sum(axis=1)

    E = even_scores - odd_scores
    X_plus_Y = even_scores + odd_scores

    M = len(even_scores)

    S_E2 = (1 / (M - 1)) * (np.sum(E**2) - (1 / M) * (np.sum(E)**2))
    S_X2 = (1 / (M - 1)) * (np.sum(X_plus_Y**2) - (1 / M) * (np.sum(X_plus_Y)**2))

    reliability = 1 - (S_E2 / S_X2)
    return reliability, S_X2
```

Рисунок 4.16 – Оцінка надійності тесту

Метод розділяє питання на парні та непарні стовпці, обчислює сумарні бали для кожної групи, а потім розраховує різницю між ними. Використовуючи

ці різниці, обчислюються дисперсії S_E2 та S_X2 [8], які використовуються для розрахунку надійності тесту за допомогою формули

Цей підхід дозволяє визначити частку загальної варіабельності, яка пояснюється випадковими коливаннями.

Метод calculate_validity оцінює валідність завдань, визначаючи середній бал, кількість валідних завдань, а також кількість завдань, які є надто легкими або надто складними. Валідність завдань визначається на основі їхньої легкості: завдання з легкістю вище певного порогу вважаються надто легкими, а ті, що нижче порогу – надто складними (рис. 4.17).

```
def calculate_validity(self, df):
    self.messages.append("Calculating validity.")
    average_score = df.mean().mean()
    difficulty = df.mean(axis=0)
    discrimination = df.corrwith(self.total_scores)

    valid_tasks = (difficulty > 0.16) & (difficulty < 0.84)
    invalid_tasks_easy = difficulty >= 0.84
    invalid_tasks_hard = difficulty <= 0.16

    return average_score, valid_tasks.sum(), invalid_tasks_easy.sum(), invalid_tasks_hard.sum()
```

Рисунок 4.17 – Оцінка надійності тесту

Метод починається з обчислення середнього балу для всіх завдань. Далі обчислюються легкість та дискримінація для кожного завдання. Завдання класифікуються як валідні, якщо їхня легкість знаходиться в межах від 0.16 до 0.84. Завдання, які не відповідають цим критеріям, класифікуються як надто легкі або надто складні.

Метод rasch_model реалізує модель Раша, яка дозволяє розрахувати ймовірності правильної відповіді на завдання, відношення шансів та інші показники для детального аналізу рівнів знань студентів та складності завдань. Модель Раша використовує логістичну функцію для моделювання ймовірності правильної відповіді залежно від рівня знань студента та складності завдання (рис. 4.18).

```
def rasch_model(self, A=None, C=None):
    self.messages.append("Running Rasch model analysis.")
    def logit(p):
        p = np.clip(p, 1e-5, 1 - 1e-5)
        return np.log(p / (1 - p))

    theta = logit(self.total_scores / self.total_scores.max()).values
    beta = self.df.apply(lambda col: logit(col.mean()), axis=0).values

    if A is None:
        A = np.ones(len(beta))
    if C is None:
        C = np.zeros(len(beta))
```

Рисунок 4.18 – Модель Раша

Метод починається з визначення логіт-функції, яка використовується для перетворення ймовірностей у логіти. Потім обчислюються рівні знань студентів theta та складність завдань beta. Модель Раша використовує ці параметри для обчислення ймовірностей правильної відповіді P_matrix та відповідних відношень шансів odds_ratios. Метод також визначає похідну ймовірності правильної відповіді щодо рівнів знань студентів.

Метод run_full_analysis об'єднує всі попередні методи для виконання повного аналізу тестових даних. Він включає оцінку точності моделей, кореляційний аналіз, аналіз якості завдань, виявлення невалідних завдань, оцінку надійності та валідності тесту, а також застосування моделі Раша. Цей метод забезпечує комплексний підхід до аналізу результатів тестування та надає всебічну інформацію для вдосконалення тесту (рис. 4.19).

```
def run_full_analysis(self, model):
    self.predict_and_evaluate(model)
    self.correlation_matrix()
    analysis_df, difficulty, discrimination = self.quality_analysis()
    easy_tasks, hard_tasks = self.analyze_tasks(analysis_df)
    self.visualize_task_difficulty(difficulty, easy_tasks, hard_tasks)
    self.visualize_task_discrimination(discrimination, easy_tasks, hard_tasks)
    anova_rel, total_variance = self.anova_reliability(self.df)
    self.messages.append(f'Надійність тесту за методом дисперсійного аналізу: {anova_rel:.3f}')
    self.messages.append(f'Дисперсія балів (S_X^2): {total_variance:.3f}')
    standard_error = np.sqrt(total_variance * (1 - anova_rel))
    self.messages.append(f'Середнє квадратичне відхилення похибки вимірювання (S_E): {standard_error:.3f}')
    confidence_intervals = []
    t_value = t.ppf(0.975, self.df.shape[0] - 1)
    for score in self.total_scores:
        ci_lower = score - t_value * standard_error
        ci_upper = score + t_value * standard_error
        confidence_intervals.append((ci_lower, ci_upper))
```

Рисунок 4.19 – Виконання повного аналізу

Цей метод об'єднує всі попередні методи для виконання комплексного аналізу тестових даних. Починаючи з оцінки точності моделей і кореляційного аналізу, метод переходить до аналізу якості завдань, виявлення невалідних завдань, та візуалізації їхніх характеристик. Після цього проводиться оцінка надійності тесту, що дозволяє визначити, наскільки стабільними є результати тестування. Далі метод оцінює валідність завдань, визначаючи середній бал, кількість валідних завдань, а також кількість надто легких і надто складних завдань. Довірчі інтервали для балів студентів обчислюються як до, так і після видалення невалідних завдань, що дозволяє оцінити вплив цих завдань на загальну валідність тесту. Метод rasch_model застосовується для детального аналізу рівнів знань студентів та складності завдань за допомогою моделі Раша. На завершення, метод будує кілька візуалізацій, включаючи гістограми розподілу рівнів знань студентів theta та складності завдань beta, теплову карту відношень шансів правильної відповіді odds_ratios, та теплову карту похідних ймовірності правильної відповіді P_derivative. Це забезпечує комплексний підхід до аналізу тестових даних і надає вичерпну інформацію для вдосконалення тесту. Таким чином, аналітичний модуль забезпечує всебічний аналіз результатів тестування, що включає в себе:

— оцінку точності моделей та візуалізацію результатів;

- кореляційний аналіз для виявлення зв'язків між завданнями;
- аналіз якості завдань, що включає обчислення легкості та дискримінації;
- виявлення невалідних завдань для підвищення валідності тесту;
- візуалізацію розподілу легкості та дискримінації завдань;
- оцінку надійності тесту за допомогою дисперсійного аналізу;
- оцінку валідності завдань;
- застосування моделі Раша для детального аналізу рівнів знань студентів та складності завдань.

Результати аналізу та візуалізації дозволяють отримати глибоке розуміння якості тестових завдань і забезпечити основу для їхнього покращення.

4.6 Інтеграція нейромережі у веб-застосунок

Інтеграція нейромережі у веб-застосунок є важливим етапом, який дозволяє кінцевим користувачам взаємодіяти з моделлю через зручний інтерфейс. У цьому розділі розглядається процес інтеграції нейромережі у веб-застосунок за допомогою Flask – мікрофреймворка [9] для розробки веб-додатків на Python. Flask був обраний завдяки своїй простоті, легкості та гнучкості, що робить його ідеальним вибором для створення прототипів та невеликих проектів.

Спочатку створюється веб-застосунок з використанням Flask. Для цього імпортується необхідні бібліотеки, налаштовуються конфігурації та визначаються основні маршрути. Flask дозволяє легко організувати маршрути, які відповідають за різні сторінки та функції веб-застосунку (рис. 4.20).

```
import os
import numpy as np
import pandas as pd
from flask import Flask, render_template
from neuro.test_analysis import TestAnalysis
from silence_tensorflow import silence_tensorflow
silence_tensorflow()
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
os.environ['TF_ENABLE_ONEDNN_OPTS'] = '0'
from tensorflow.keras.models import load_model

app = Flask(__name__)
```

Рисунок 4.20 – Ініціалізація веб-застосунку

На початку імпортуються необхідні бібліотеки. os використовується для роботи з файловою системою, numpy та pandas – для обробки даних, flask – для створення веб-застосунку, silence_tensorflow – для приглушення повідомлень TensorFlow, та tensorflow.keras.models – для завантаження моделі нейронної мережі. Flask застосунок ініціалізується шляхом створення об'єкта app.

Для аналізу використовуються дані з Excel файлу, який завантажується та обробляється. Далі модель нейронної мережі завантажується з файлу, і проводиться повний аналіз даних з використанням методів класу TestAnalysis (рис. 4.21).

```
df = None
total_scores = None
analysis = None
rdf = pd.read_excel('data/матриця_результатів_тестування.xlsx')
def run_analysis():
    global df, total_scores, analysis
    df = pd.read_excel('data/матриця_результатів_тестування.xlsx')
    df = df.drop(columns=['Особи, яких тестують, I'])
    df = df.drop(columns=['Кластер'])
    total_scores = df.pop('Сумарний бал особи')
    model = load_model('data/neural_network_model.keras')
    analysis = TestAnalysis(df, total_scores)
    analysis.run_full_analysis(model)
```

Рисунок 4.21 – Завантаження та обробка даних

Цей блок коду визначає глобальні змінні для DataFrame (df), сумарних балів (total_scores) та об'єкта аналізу (analysis). Функція run_analysis завантажує дані з Excel файлу, видаляє непотрібні стовпці, завантажує модель нейронної мережі та виконує повний аналіз даних за допомогою класу TestAnalysis.

Flask дозволяє легко визначити маршрути, які відповідають за різні сторінки веб-застосунку. Кожен маршрут відповідає певній функції, яка виконується при зверненні до відповідної URL-адреси. Після визначення всіх маршрутів, веб-застосунок запускається. Перед запуском виконується функція run_analysis(), яка завантажує дані та проводить повний аналіз (рис. 4.22).

```
if __name__ == '__main__':
    run_analysis()
    app.run(debug=False)
```

Рисунок 4.22 – Запуск веб-застосунку

Інтеграція нейромережі у веб-застосунок за допомогою Flask дозволяє створити зручний інтерфейс для взаємодії з моделлю та аналізу результатів тестування. Використання Flask надає гнучкість у налаштуванні маршрутів,

обробці даних та відображенні результатів. Завдяки цьому підходу, користувачі можуть легко отримувати доступ до результатів аналізу та рекомендацій щодо покращення тестів, а також візуалізувати важливі показники та графіки.

4.7 Тестування роботи розробленного програмного забезпечення та аналіз результатів

Тестування програмного забезпечення є важливим етапом у процесі розробки, оскільки воно дозволяє виявити помилки та недоліки на ранніх стадіях і забезпечити надійну та стабільну роботу кінцевого продукту

Тестування програмного забезпечення було виконано за кількома ключовими напрямками, кожен з яких забезпечує різні аспекти перевірки функціональності та стабільності:

Функціональне тестування: Перевірка всіх основних функцій програмного забезпечення для забезпечення їхньої правильної роботи відповідно до специфікацій.

Інтеграційне тестування: Перевірка взаємодії між різними модулями системи для забезпечення їхньої коректної роботи в складі єдиного продукту.

Тестування користувацького інтерфейсу: Перевірка зручності та коректності роботи веб-інтерфейсу, включаючи рендеринг даних, навігацію та інтерактивні елементи.

Регресійне тестування: Перевірка коректності роботи системи після внесення змін або виправлення помилок для забезпечення відсутності нових помилок.

Функціональне тестування включало перевірку всіх основних функцій програмного забезпечення:

— завантаження даних: перевірка коректності завантаження даних з Excel файлу та їхньої попередньої обробки;

- аналіз даних: перевірка роботи методів класу TestAnalysis, включаючи розрахунок легкості, дискримінації, кореляційних матриць, та інших статистичних показників;
- модель нейромережі: перевірка коректності завантаження та використання моделі нейромережі для прогнозування результатів;
- генерація візуалізацій: перевірка побудови та збереження всіх графіків і візуалізацій, включаючи розподіли балів, кореляційні матриці та теплові карти.

Результати функціонального тестування показали, що всі функції працюють коректно, дані завантажуються та обробляються без помилок, а всі статистичні показники та візуалізації генеруються належним чином. Також було проведено тестування програмного модуля. Для тестування на достовірність правидвості визначення було залучено 10 експертів, використовуючи реальні тестові завдання, в яких деякі із завдань були аномально складні та легкі. Оцінки виставлялись по шкалі від 1 до 10, в залежності від того, наскільки програмний модуль якісно зміг проаналізувати тестові завдання.

Результати тестування програмого модуля наведені в таблиці 4.2.

Таблиця 4.2 – Результати тестування програмного модуля

Експерт	1	2	3	4	5	6	7	8	9	10
Оцінка	9	8	10	9	7	8	9	9	7	9

Проаналізувавши дані, маємо загальну оцінку 85 із 100. Отже, можна дійти висновку, що ефективність роботи програмного модуля оцінювання якості тестових завдань в освітніх вимірюваннях є достатньо високою. Це свідчить про підвищення ефективності процесу моніторингу тестових завдань в освітніх вимірюваннях.

Інтеграційне тестування включало перевірку взаємодії між різними модулями програмного забезпечення:

- взаємодія між модулем завантаження даних та аналітичним модулем: перевірка коректності передачі даних між цими модулями;
- взаємодія між аналітичним модулем та модулем нейромережі: перевірка коректності використання результатів аналізу для тренування та оцінки моделі нейромережі;
- взаємодія між веб-інтерфейсом та бекендом: перевірка коректності відображення результатів аналізу та візуалізацій на веб-сторінках.

Інтеграційне тестування показало, що всі модулі коректно взаємодіють між собою, дані передаються та обробляються без помилок, а результати відображаються належним чином.

Тестування користувацького інтерфейсу включало перевірку зручності та коректності роботи веб-застосунку:

- рендеринг даних: перевірка коректного відображення даних на веб-сторінках, включаючи таблиці з результатами тестування;
- навігація: перевірка роботи навігації між різними сторінками веб-застосунку;
- інтерактивні елементи: перевірка роботи інтерактивних елементів, таких як кнопки, посилання та інші елементи керування.

Результати тестування користувацького інтерфейсу показали, що всі елементи інтерфейсу працюють належним чином, дані відображаються коректно, а навігація між сторінками є інтуїтивною та зручною.

Регресійне тестування включало перевірку коректності роботи системи після внесення змін або виправлення помилок:

- перевірка коректності роботи функцій після внесення змін: перевірка, чи не з'явилися нові помилки після внесення змін до коду.
- перевірка відсутності негативного впливу нових функцій: перевірка, чи не впливають нові функції на стабільність роботи існуючих функцій.

Регресійне тестування показало, що система продовжує працювати стабільно після внесення змін, нові функції не спричиняють нових помилок, а існуючі функції продовжують працювати належним чином.

Результати тестування показали, що програмне забезпечення для оцінки якості тестових завдань працює стабільно та коректно. Усі основні функції були перевірені та продемонстрували належний рівень продуктивності. Взаємодія між різними модулями системи відбувається без помилок, а користувацький інтерфейс забезпечує зручну та інтуїтивну взаємодію з користувачем.

Навантажувальне тестування підтвердило здатність системи витримувати високі навантаження та працювати стабільно навіть під час пікових навантажень. Регресійне тестування показало, що система продовжує працювати стабільно після внесення змін, що свідчить про високу якість коду та його гнучкість.

Проведене тестування підтвердило, що розроблене програмне забезпечення для оцінки якості тестових завдань відповідає вимогам та працює стабільно. Система забезпечує точний аналіз даних.

4.8 Висновки

У процесі розробки програмного модуля для оцінки якості тестових завдань були розглянуті та реалізовані кілька ключових аспектів, які забезпечують його ефективність та надійність.

Було обрано зручне та потужне середовище розробки, яке підтримує всі необхідні інструменти та бібліотеки для реалізації нейронних мереж та їх інтеграції у веб-застосунок. Мова програмування обрана з урахуванням її популярності, підтримки бібліотек для машинного навчання, простоти використання та інтеграції з іншими інструментами. Це забезпечило ефективну розробку та підтримку модуля. Було розроблено модуль для збору та попередньої обробки даних, що включає очищення, нормалізацію та векторизацію текстових даних, що є критично важливим для якісного навчання нейронної мережі.

ВИСНОВКИ

Під час виконання досліджень за тематикою бакалаврської дипломної роботи було обґрунтовано актуальність задачі оцінки якості тестових завдань в освітніх вимірюваннях. Метою даного дослідження було підвищення якості процесу моніторингу тестових завдань в освітніх вимірюваннях, що включало аналіз сучасних підходів до оцінки якості тестових завдань, розробку та впровадження удосконаленого алгоритму оцінки якості з використанням нейромережевих технологій, розробку самого нейромережевого модуля, а також його складових. Об’єктом дослідження був процес оцінки якості тестових завдань, а предметом дослідження були програмні засоби оцінки якості тестових завдань. Результати тестування доводять, що розробка є гідним конкурентом наявних аналогічних розробок.

У ході дослідження було проведено всебічний аналіз існуючих сучасних підходів до оцінки якості тестових завдань, а також визначено ключові критерії оцінки якості тестів. Було розроблено удосконалений алгоритм оцінки якості тестових завдань та доведено ефективність нової розробки, в порівнянні із наявними аналогами. Описано структуру програмного модуля для оцінки якості тестових завдань. Обрано архітектуру нейромережі для аналізу даних та визначення якості завдань, проведено навчання та валідацію моделі, а також оптимізацію алгоритмів для досягнення максимальної точності. Проведено обґрунтування вибору середовища розробки та мови програмування. Обрано Visual Studio Code як середовище розробки та Python як мову програмування. Розроблено модуль генерації датасету, модуль забезпечує генерацію унікальних записів для кожного студента, кластеризацію за рівнем знань, а також визначення легких та складних питань для аналізу. Розроблено навчальний модуль нейромережі, який забезпечує навчання моделі на основі згенерованих даних. Спроєктовано аналітичний модуль, а також інтегровано нейромережу у веб-застосунок за допомогою Flask, що забезпечило зручний інтерфейс для взаємодії з користувачами. Результати тестування підтвердили стабільну та коректну роботу програмного забезпечення, що свідчить про його високу якість та надійність.

Отже, всі поставлені задачі було успішно виконано, а мету дослідження досягнуто.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Шевчук О.Ф., Христянчук В.В. Програмний модуль оцінки якості тестових завдань в освітніх вимірюваннях. *Матеріали LIII науково-технічної конференції підрозділів Вінницького національного технічного університету* (НТКП ВНТУ–2024), 20-22 березня 2024 р., ВНТУ, 2024. [Електронний ресурс]. Режим доступу: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2024/paper/view/20367/16847 (дата звернення 01.06.2024).
2. Continuous Integration: CircleCI vs Travis CI vs Jenkins [Електронний ресурс]. – Режим доступу: https://djangostars.com/blog/continuous-integration-circleci-vs-travisci-vs-jenkins/ (дата звернення 29.05.2024).
3. Python Doc [Електронний ресурс]. — Режим доступу до ресурсу: https://www.python.org (дата звернення 22.05.2024).
4. Stackoverflow [Електронний ресурс]. — Режим доступу до ресурсу: https://stackoverflow.com/ (дата звернення 27.05.2024).
5. Scikit-learn [Електронний ресурс]. — Режим доступу до ресурсу: https://scikit-learn.org/stable/ (дата звернення 24.05.2024).
6. Kaggle [Електронний ресурс]. — Режим доступу до ресурсу: https://www.kaggle.com/datasets/kaushil268/disease-prediction-using-machine-learning/?select=Training.csv (дата звернення 27.05.2024).
7. Medium [Електронний ресурс]. — Режим доступу до ресурсу: https://towardsdatascience.com/document-your-machine-learning-project-in-a-smart-way-35c68aa5fc0e (дата звернення 25.05.2024).
8. CRM BLOG [Електронний ресурс]. — Режим доступу до ресурсу: https://gowombat.team/blog/posts/7-reasons-to-use-a-customer-relationship-management-crm-system (дата звернення 26.05.2024).
9. CRM PULSE [Електронний ресурс]. — Режим доступу до ресурсу: https://sendpulse.com/support/glossary/crm (дата звернення 22.05.2024).

10. CRM BUILD [Електронний ресурс]. — Режим доступу до ресурсу: https://bambooagile.eu/insights/how-to-build-a-crm-system-from-scratch (дата звернення 21.05.2024).
11. Кухаренко В. М., Перхун Л. П., Товмаченко Н. М. Методика комплексного оцінювання якості тестів. Частина 1. *Статистика України*. 2018. № 3. С. 40–48 (дата звернення 21.05.2024).
12. О.А. Костіков, Т. Ю. Соломко. Оцінка якості тестових задвань методами сучасної теорії тестування. *IT synergy.* 2023. Т. 1, № 4. С. 109-117. *DOI:* https://doi.org/10.53920/ITS-2023-1-7 (дата звернення 22.05.2024).
13. Кухаренко В. М., Перхун Л. П., Товмаченко Н. М. Методика комплексного оцінювання якості тестів. Частина 2 . *Статистика України*. 2018. № 4. С. 72–79. (дата звернення 18.05.2024).
14. В.П. Бригінець, С.О. Подласов, О.В. Матвійчук. Апробація тестових завдань та аналіз їх якості для нестандартизованих тестів. *IVET.* 2018. Т. 12, №6. С. 160-166. DOI: https://doi.org/10.32835/2707-3092.2020.20.160-166 (дата звернення 12.05.2024).
15. Методичні вказівки до виконання бакалаврської дипломної роботи для студентів денної та заочної форм навчання, спеціальності 122 «Комп’ютерні науки» / уклад. О. В. Сілагін. Вінниця : ВНТУ, 2018. 28 с.

ДОДАТКИ

ДОДАТОК А

ПРОТОКОЛ ПЕРЕВІРКИ ДИПЛОМНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Програмний модуль оцінювання якості тестових завдань в освітніх вимірюваннях.

Тип роботи: бакалаврська дипломна робота
(БДР, МКР)

Підрозділ кафедра комп'ютерних наук, ФІТА
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 97,12% Схожість 2,88%

Аналіз звіту подібності (відмітити потрібне):

- ✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.
- □ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.
- □ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку Озеранський В.С.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи Христянчук В. В

Керівник роботи Шевчук О.Ф.

ДОДАТОК Б
ФРАГМЕНТ ЛІСТИНГУ ПРОГРАМНОГО КОДУ

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import random
import os

with open('utils/surnames.txt', 'r', encoding='utf-8') as f:
    surnames = f.read().splitlines()

with open('utils/names.txt', 'r', encoding='utf-8') as f:
    names = f.read().splitlines()

num_students = 10000
num_questions = 40

class_number = np.random.randint(5, 13)
class_letter = np.random.choice(['А', 'Б', 'В', 'Г'])
student_class = f'{class_number}-{class_letter}'

def generate_full_name():
    surname = np.random.choice(surnames)
    name = np.random.choice(names)
    return f'{surname} {name}'

students_set = set()
students = []

while len(students) < num_students:
    full_name = generate_full_name()
    if full_name not in students_set:
        students_set.add(full_name)
```

```
        students.append(f'{student_class} {full_name}')

clusters = ['Дуже слабкий', 'Слабкий', 'Середній', 'Хорошист', 'Відмінник']
cluster_probabilities = [0.1, 0.1, 0.4, 0.3, 0.1]
students_per_cluster = np.random.choice(clusters, num_students, p=cluster_probabilities)

cluster_probs = {
    'Дуже слабкий': np.random.uniform(0.1, 0.3, size=num_questions),
    'Слабкий': np.random.uniform(0.3, 0.5, size=num_questions),
    'Середній': np.random.uniform(0.5, 0.7, size=num_questions),
    'Хорошист': np.random.uniform(0.7, 0.8, size=num_questions),
    'Відмінник': np.random.uniform(0.8, 1.0, size=num_questions)
}

num_very_easy_questions = random.randint(3, 3)
num_very_hard_questions = random.randint(3, 3)

very_easy_questions = np.random.choice(num_questions, num_very_easy_questions,
replace=False)
remaining_questions = [q for q in range(num_questions) if q not in very_easy_questions]
very_hard_questions = np.random.choice(remaining_questions, num_very_hard_questions,
replace=False)

responses = []

easy_increase = {
    'Дуже слабкий': 0.7,
    'Слабкий': 0.7,
    'Середній': 0.8,
    'Хорошист': 0.9,
    'Відмінник': 1
}

hard_decrease = {
    'Дуже слабкий': 0.1,
```

```
    'Слабкий': 0.2,
    'Середній': 0.3,
    'Хорошист': 0.4,
    'Відмінник': 0.5
}

for i, cluster in enumerate(students_per_cluster):
    base_prob = cluster_probs[cluster]
    p = base_prob
    p[very_easy_questions] = np.clip(base_prob[very_easy_questions] + easy_increase[cluster],
0.95, 0.95)
    p[very_hard_questions] = np.clip(base_prob[very_hard_questions] - hard_decrease[cluster], 0.1,
0.1)
    p = np.clip(p, 0, 1)
    student_responses = np.random.binomial(1, p, num_questions)
    responses.append(student_responses)

responses = np.array(responses)

total_scores = responses.sum(axis=1)

columns = [str(i+1) for i in range(num_questions)]
df_responses = pd.DataFrame(responses, columns=columns)

df_responses.insert(0, 'Особи, яких тестують, I', students)
df_responses['Сумарний бал особи'] = total_scores

df_responses['Кластер'] = students_per_cluster

df_responses.to_excel('data/матриця_результатів_тестування.xlsx', index=False)

train_plots_dir = 'static/train'
os.makedirs(train_plots_dir, exist_ok=True)

plt.figure(figsize=(10, 6))
```

```
plt.hist(total_scores, bins=np.arange(num_questions+2)-0.5, edgecolor='black')
plt.title('Розподіл кількості успішних відповідей студентів')
plt.xlabel('Кількість успішних відповідей')
plt.ylabel('Кількість студентів')
plt.xticks(np.arange(0, num_questions + 1, 1))
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.savefig(os.path.join(train_plots_dir, 'distribution_of_total_scores.png'))
plt.close()

correct_answers_per_question = responses.sum(axis=0)

plt.figure(figsize=(10, 6))
plt.bar(range(1, num_questions + 1), correct_answers_per_question, color='skyblue',
edgecolor='black')
plt.title('Кількість студентів, що правильно відповіли на конкретний тест')
plt.xlabel('Номер тесту')
plt.ylabel('Кількість правильних відповідей')
plt.xticks(range(1, num_questions + 1))
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.savefig(os.path.join(train_plots_dir, 'correct_answers_per_question.png'))
plt.close()

print("Файл успішно збережено і графікb побудовано!")


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from scipy.stats import t
import os

class TestAnalysis:
    def __init__(self, df, total_scores):
        self.df = df
```

```
        self.total_scores = total_scores
        self.plot_dir = 'static/plots'
        self.messages = []
        if not os.path.exists(self.plot_dir):
            os.makedirs(self.plot_dir)

    def predict_and_evaluate(self, model):
        self.messages.append("Running predict_and_evaluate function.")
        X_train, X_test, y_train, y_test = train_test_split(self.df, self.total_scores, test_size=0.2,
random_state=42)
        y_pred = model.predict(X_test)

        plt.figure(figsize=(10, 6))
        plt.scatter(y_test, y_pred, alpha=0.3)
        plt.xlabel('Реальні значення')
        plt.ylabel('Прогнозовані значення')
        plt.title('Прогнозовані vs Реальні значення')
        plt.savefig(os.path.join(self.plot_dir, '1predicted_vs_actual.png'))
        plt.close()

    def correlation_matrix(self):
        self.messages.append("Generating correlation matrix.")
        correlation_matrix = self.df.corr()
        plt.figure(figsize=(24, 20))
        sns.heatmap(correlation_matrix, annot=False, cmap='coolwarm')
        plt.title('Кореляційна матриця тестових завдань')
        plt.savefig(os.path.join(self.plot_dir, '2correlation_matrix.png'), dpi=150)
        plt.close()

    def quality_analysis(self):
        self.messages.append("Running quality analysis.")
        total_students = self.df.shape[0]
        correct_answers = self.df.sum(axis=0)
        difficulty = correct_answers / total_students
        discrimination = self.df.corrwith(self.total_scores)
```

```
        analysis_df = pd.DataFrame({
            'Завдання': self.df.columns,
            'Легкість': difficulty,
            'Дискримінація': discrimination,
            'Кількість правильних відповідей': correct_answers
        }).sort_values(by='Кількість правильних відповідей', ascending=False)

        self.messages.append(str(analysis_df))

        return analysis_df, difficulty, discrimination

    def plot_student_scores_distribution(self, df, total_scores, title, filename):
        self.messages.append(f"Plotting student scores distribution: {title}")
        correct_answers_per_student = df.sum(axis=1)
        max_possible_score = df.shape[1]
        percentage_correct = (correct_answers_per_student / max_possible_score) * 100
        bins = [0, 60, 70, 80, 90, 100]
        labels = ['F', 'D', 'C', 'B', 'A']
        grade_counts = pd.cut(percentage_correct, bins=bins, labels=labels,
include_lowest=True).value_counts().sort_index()
        total_students = percentage_correct.shape[0]
        grade_percentages = (grade_counts / total_students) * 100

        plt.figure(figsize=(10, 6))
        sns.barplot(x=grade_percentages.index, y=grade_percentages.values, palette='viridis')
        for index, value in enumerate(grade_percentages):
            plt.text(index, value + 0.5, f'{value:.2f}%', ha='center', fontsize=10)
        plt.title(title)
        plt.xlabel('Оцінка')
        plt.ylabel('Відсоток студентів')
        plt.ylim(0, 100)
        plt.savefig(os.path.join(self.plot_dir, filename))
        plt.close()
```

```
def calculate_thresholds(self, n, base_easy=0.84, base_hard=0.16):
    if n < 10:
        return None, None
    easy_corridor = (10 / n) * (base_easy - 0.16)
    hard_corridor = (10 / n) * (0.84 - base_hard)
    return base_easy - easy_corridor, base_hard + hard_corridor

def analyze_tasks(self, analysis_df):
    self.messages.append("Analyzing tasks.")
    total_students = self.df.shape[0]
    easy_tasks = []
    hard_tasks = []

    for index, row in analysis_df.iterrows():
        n_responses = row['Кількість правильних відповідей']
        easy_threshold, hard_threshold = self.calculate_thresholds(n_responses)
        if easy_threshold is None or hard_threshold is None:
            continue

        adjustment = 0.1 * n_responses / total_students
        adjusted_easy_threshold = easy_threshold - adjustment
        adjusted_hard_threshold = hard_threshold + adjustment

        if row['Легкість'] >= adjusted_easy_threshold:
            easy_tasks.append(index)
        if row['Легкість'] <= adjusted_hard_threshold:
            hard_tasks.append(index)

    self.messages.append("Легкі завдання:")
    self.messages.append(str(easy_tasks))

    self.messages.append("Складні завдання:")
    self.messages.append(str(hard_tasks))

    return easy_tasks, hard_tasks
```

```
    def visualize_task_difficulty(self, difficulty, easy_tasks, hard_tasks):
        self.messages.append("Visualizing task difficulty.")
        plt.figure(figsize=(12, 8))

        easy_hist = plt.hist(difficulty[easy_tasks].dropna(), bins=20, edgecolor='blue', alpha=0.5,
label='Легкі завдання')
        for i in range(len(easy_hist[0])):
            y_offset = 0
            for j in range(int(easy_hist[0][i])):
                x = easy_hist[1][i] + (easy_hist[1][1] - easy_hist[1][0])
                y = y_offset + 0.1
                plt.annotate(
                    f'Легке завдання',
                    xy=(x, y),
                    xytext=(x + 0.005 * i, y + i * 0.02 + 0.02),
                    textcoords='data',
                    arrowprops=dict(facecolor='blue', arrowstyle='->'),
                    fontsize=9,
                    ha='left'
                )
                y_offset += 1

        hard_hist = plt.hist(difficulty[hard_tasks].dropna(), bins=20, edgecolor='red', alpha=0.5,
label='Складні завдання')
        for i in range(len(hard_hist[0])):
            y_offset = 0
            for j in range(int(hard_hist[0][i])):
                x = hard_hist[1][i] + (hard_hist[1][1] - hard_hist[1][0]) / 2
                y = y_offset + 0.1
                plt.annotate(
                    f'Складне завдання',
                    xy=(x, y),
                    xytext=(x + 0.005 * i, y + i * 0.02 + 0.02),
                    textcoords='data',
```

```
                arrowprops=dict(facecolor='red', arrowstyle='->'),
                fontsize=9,
                ha='left'
            )
            y_offset += 1

    plt.title('Розподіл легкості тестових завдань')
    plt.xlabel('Легкість')
    plt.ylabel('Кількість завдань')
    plt.legend()
    plt.grid(True)
    plt.xticks(np.arange(0, 1.1, 0.03))
    plt.savefig(os.path.join(self.plot_dir, '3task_difficulty_distribution.png'))
    plt.close()

def visualize_task_discrimination(self, discrimination, easy_tasks, hard_tasks):
    self.messages.append("Visualizing task discrimination.")
    easy_discrimination = discrimination[easy_tasks].dropna()
    hard_discrimination = discrimination[hard_tasks].dropna()

    plt.figure(figsize=(10, 6))
    plt.hist(easy_discrimination, bins=10, edgecolor='blue', alpha=0.7, label='Легкі завдання')
    plt.hist(hard_discrimination, bins=10, edgecolor='red', alpha=0.7, label='Складні завдання')
    plt.title('Розподіл дискримінації тестових завдань')
    plt.xlabel('Дискримінація')
    plt.ylabel('Кількість завдань')
    plt.legend()
    plt.savefig(os.path.join(self.plot_dir, '4task_discrimination_distribution.png'))
    plt.close()

def anova_reliability(self, data):
    self.messages.append("Calculating ANOVA reliability.")
    even_columns = data.columns[::2]
    odd_columns = data.columns[1::2]
    even_scores = data[even_columns].sum(axis=1)
```

```
        odd_scores = data[odd_columns].sum(axis=1)

        E = even_scores - odd_scores
        X_plus_Y = even_scores + odd_scores

        M = len(even_scores)

        S_E2 = (1 / (M - 1)) * (np.sum(E**2) - (1 / M) * (np.sum(E)**2))
        S_X2 = (1 / (M - 1)) * (np.sum(X_plus_Y**2) - (1 / M) * (np.sum(X_plus_Y)**2))

        reliability = 1 - (S_E2 / S_X2)
        return reliability, S_X2

    def calculate_validity(self, df):
        self.messages.append("Calculating validity.")
        average_score = df.mean().mean()
        difficulty = df.mean(axis=0)
        discrimination = df.corrwith(self.total_scores)

        valid_tasks = (difficulty > 0.16) & (difficulty < 0.84)
        invalid_tasks_easy = difficulty >= 0.84
        invalid_tasks_hard = difficulty <= 0.16

        return average_score, valid_tasks.sum(), invalid_tasks_easy.sum(), invalid_tasks_hard.sum()

    def rasch_model(self, A=None, C=None):
        self.messages.append("Running Rasch model analysis.")
        def logit(p):
            p = np.clip(p, 1e-5, 1 - 1e-5)
            return np.log(p / (1 - p))

        theta = logit(self.total_scores / self.total_scores.max()).values
        beta = self.df.apply(lambda col: logit(col.mean()), axis=0).values

        if A is None:
```

```
            A = np.ones(len(beta))
        if C is None:
            C = np.zeros(len(beta))

        P = lambda theta, beta, A, C: C + (1 - C) * (np.exp(A * (theta[:, None] - beta)) / (1 + np.exp(A * (theta[:, None] - beta))))
        Q = lambda theta, beta, A, C: 1 - P(theta, beta, A, C)

        P_matrix = P(theta, beta, A, C)
        Q_matrix = Q(theta, beta, A, C)

        assert np.all((P_matrix >= 0) & (P_matrix <= 1)), "Неправильні значення в матриці P"
        assert np.all((Q_matrix >= 0) & (Q_matrix <= 1)), "Неправильні значення в матриці Q"

        odds_ratios = P_matrix / Q_matrix

        def individual_odds_ratio(g, m, j):
            return np.exp(theta[g] - theta[m])

        def task_odds_ratio(i, k, n):
            return np.exp(beta[n] - beta[k])

        def derivative_P_theta(theta, beta, A):
            exp_diff = np.exp(A * (theta[:, None] - beta))
            return A * exp_diff / ((1 + exp_diff) ** 2)

        P_derivative = derivative_P_theta(theta, beta, A)

        return P_matrix, Q_matrix, odds_ratios, theta, beta, individual_odds_ratio, task_odds_ratio, P_derivative

    def run_full_analysis(self, model):
        self.predict_and_evaluate(model)
        self.correlation_matrix()
        analysis_df, difficulty, discrimination = self.quality_analysis()
```

```
        easy_tasks, hard_tasks = self.analyze_tasks(analysis_df)
        self.visualize_task_difficulty(difficulty, easy_tasks, hard_tasks)
        self.visualize_task_discrimination(discrimination, easy_tasks, hard_tasks)
        anova_rel, total_variance = self.anova_reliability(self.df)
        self.messages.append(f'Надійність тесту за методом дисперсійного аналізу: {anova_rel:.3f}')
        self.messages.append(f'Дисперсія балів (S_X^2): {total_variance:.3f}')
        standard_error = np.sqrt(total_variance * (1 - anova_rel))
        self.messages.append(f'Середнє квадратичне відхилення похибки вимірювання (S_E): {standard_error:.3f}')
        confidence_intervals = []
        t_value = t.ppf(0.975, self.df.shape[0] - 1)
        for score in self.total_scores:
            ci_lower = score - t_value * standard_error
            ci_upper = score + t_value * standard_error
            confidence_intervals.append((ci_lower, ci_upper))

        self.messages.append("Перші 5 довірчих інтервалів для балів:")
        for i, (lower, upper) in enumerate(confidence_intervals[:5]):
            self.messages.append(f"Бал {i+1}: [{lower:.3f}, {upper:.3f}]")

        self.plot_student_scores_distribution(self.df, self.total_scores, "Розподіл студентів за оцінками (до видалення невалідних завдань)", "student_scores_distribution_before.png")
        valid_tasks = [col for col in self.df.columns if col not in easy_tasks and col not in hard_tasks]
        df_valid = self.df[valid_tasks]
        self.plot_student_scores_distribution(df_valid, df_valid.sum(axis=1), "Розподіл студентів за оцінками (після видалення невалідних завдань)", "student_scores_distribution_after.png")
        anova_rel_valid, total_variance_valid = self.anova_reliability(df_valid)
        self.messages.append(f'Надійність тесту після видалення невалідних завдань: {anova_rel_valid:.3f}')
        self.messages.append(f'Дисперсія балів після видалення невалідних завдань (S_X^2): {total_variance_valid:.3f}')
        standard_error_valid = np.sqrt(total_variance_valid * (1 - anova_rel_valid))
        self.messages.append(f'Середнє квадратичне відхилення похибки вимірювання після видалення невалідних завдань (S_E): {standard_error_valid:.3f}')
```

```
        average_score, valid_tasks, invalid_tasks_easy, invalid_tasks_hard =
self.calculate_validity(self.df)
        self.messages.append(f'Середній бал: {average_score:.2f}')
        self.messages.append(f'Кількість валідних завдань: {valid_tasks}')
        self.messages.append(f'Кількість легких невалідних завдань: {invalid_tasks_easy}')
        self.messages.append(f'Кількість складних невалідних завдань: {invalid_tasks_hard}')

        confidence_intervals_valid = []
        for score in self.total_scores:
            ci_lower = score - t_value * standard_error_valid
            ci_upper = score + t_value * standard_error_valid
            confidence_intervals_valid.append((ci_lower, ci_upper))

        self.messages.append("Перші 5 довірчих інтервалів для балів після видалення невалідних
завдань:")
        for i, (lower, upper) in enumerate(confidence_intervals_valid[:5]):
            self.messages.append(f"Бал {i+1}: [{lower:.3f}, {upper:.3f}]")

        A = np.ones(self.df.shape[1])
        C = np.zeros(self.df.shape[1])
        P_matrix, Q_matrix, odds_ratios, theta, beta, individual_odds_ratio, task_odds_ratio,
P_derivative = self.rasch_model(A, C)
        self.messages.append('P матриця:\n' + str(P_matrix))
        self.messages.append('Q матриця:\n' + str(Q_matrix))

        plt.figure(figsize=(10, 6))
        plt.hist(theta, bins=20, edgecolor='black', alpha=0.7, label='Рівні знань (theta)')
        plt.hist(beta, bins=20, edgecolor='black', alpha=0.7, label='Складність завдань (beta)')
        plt.legend()
        plt.title('Розподіл рівнів знань та складності завдань')
        plt.xlabel('Значення')
        plt.ylabel('Кількість')
        plt.savefig(os.path.join(self.plot_dir, '5theta_beta_distribution.png'))
        plt.close()
```

```
        plt.figure(figsize=(10, 6))
        sns.heatmap(odds_ratios, cmap='coolwarm', cbar=True, vmin=0.5, vmax=2.0)
        plt.title('Відношення шансів правильної відповіді')
        plt.xlabel('Завдання')
        plt.ylabel('Особи')
        plt.savefig(os.path.join(self.plot_dir, '6odds_ratios.png'), dpi=150)
        plt.close()

        g, m, j = 0, 1, 0
        odds_ratio_example = individual_odds_ratio(g, m, j)
        self.messages.append(f'Відношення шансів правильної відповіді для осіб {g} і {m} на
завдання {j}: {odds_ratio_example:.3f}')

        i, k, n = 0, 0, 1
        task_odds_ratio_example = task_odds_ratio(i, k, n)
        self.messages.append(f'Відношення шансів правильної відповіді для особи {i} на
завдання {k} і {n}: {task_odds_ratio_example:.3f}')

        self.messages.append(str(P_derivative))
        plt.figure(figsize=(10, 6))
        sns.heatmap(P_derivative, cmap='YlGnBu', cbar=True)
        plt.title('Похідна ймовірності правильної відповіді')
        plt.xlabel('Завдання')
        plt.ylabel('Особи')
        plt.savefig(os.path.join(self.plot_dir, '7probability_derivative.png'), dpi=150)
        plt.close()

import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
```

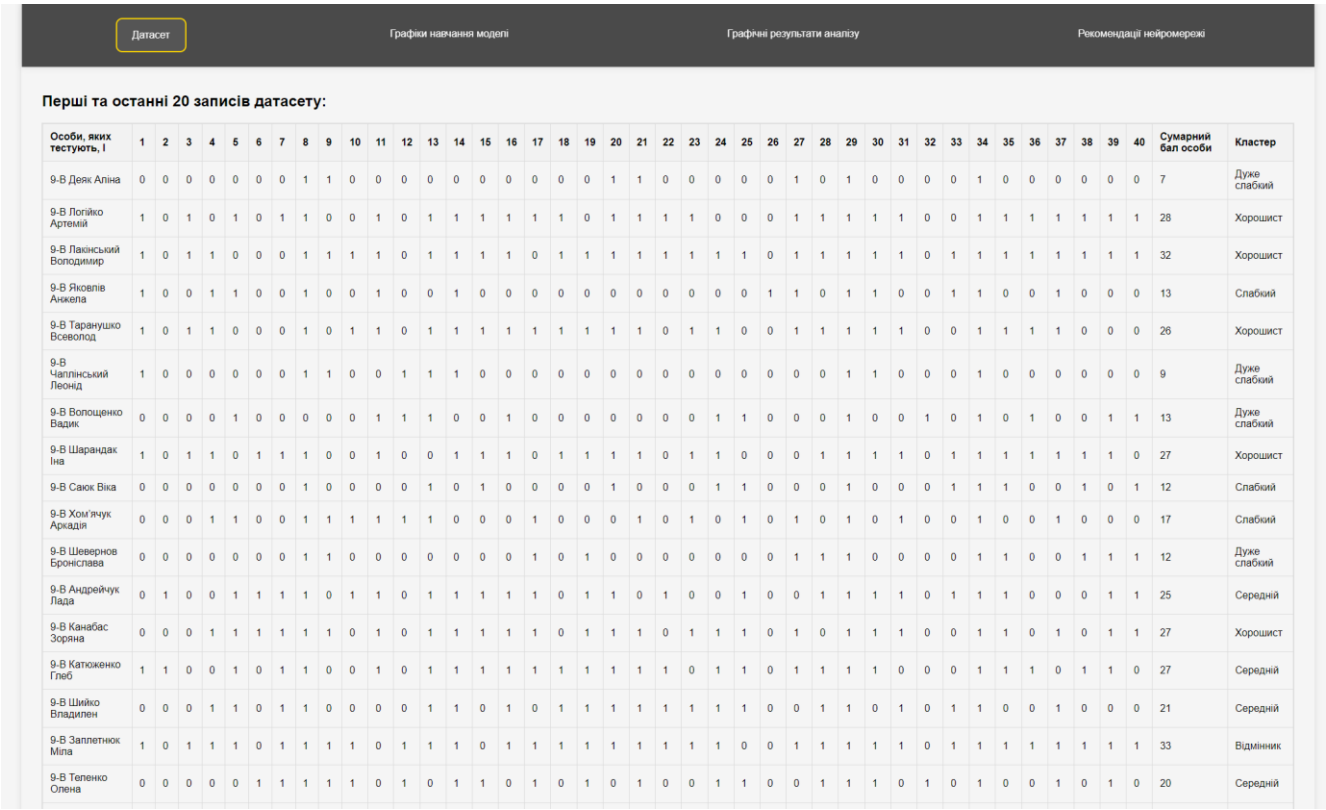
```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import os

df = pd.read_excel('data/матриця_результатів_тестування.xlsx')

df = df.drop(columns=['Особи, яких тестують, I'])
df = df.drop(columns=['Кластер'])

total_scores = df.pop('Сумарний бал особи')

X_train, X_val, y_train, y_val = train_test_split(df, total_scores, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)

model = Sequential()
model.add(Dense(256, activation='relu', input_shape=(X_train.shape[1],)))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(BatchNormalization())
model.add(Dense(32, activation='relu'))
model.add(Dense(1))

model.compile(optimizer=Adam(learning_rate=0.001), loss='mse', metrics=['mae'])

early_stopping = EarlyStopping(monitor='val_loss', patience=20, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=10, min_lr=0.0001)
```

```
history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=500,
    batch_size=16,
    callbacks=[early_stopping, reduce_lr]
)

model.save('data/neural_network_model.keras')

train_plots_dir = 'static/train'
os.makedirs(train_plots_dir, exist_ok=True)

plt.figure(figsize=(16, 8))

plt.subplot(2, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.title('Loss')

plt.subplot(2, 2, 2)
plt.plot(history.history['mae'], label='Training MAE')
plt.plot(history.history['val_mae'], label='Validation MAE')
plt.legend()
plt.title('Mean Absolute Error (MAE)')

plt.subplot(2, 2, 3)
plt.plot(history.history['lr'], label='Learning Rate')
plt.legend()
plt.title('Learning Rate')

plt.subplot(2, 2, 4)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
```

```
plt.yscale('log')
plt.legend()
plt.title('Loss (Log Scale)')

plt.tight_layout()

plt.savefig(os.path.join(train_plots_dir, 'training_plots.png'))
plt.close()

print(f"Остання втрата на тренувальному наборі: {history.history['loss'][-1]}")
print(f"Остання MAE на тренувальному наборі: {history.history['mae'][-1]}")
print(f"Остання втрата на валідаційному наборі: {history.history['val_loss'][-1]}")
print(f"Остання MAE на валідаційному наборі: {history.history['val_mae'][-1]}")
```

ДОДАТОК В
ІНСТРУКЦІЯ КОРИСТУВАЧА

Після запуску проєкту, користувача направить на головну сторінку, на які буде відображено витяг із датасету та панель навігації, по якій можна перейти на інші сторінки.



Особи, яких тестують, І	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	Сумарний бал особи	Кластер
9-В Деяк Аліна	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	0	0	0	0	7	Дуже слабкий
9-В Логійко Артемій	1	0	1	0	1	0	1	1	0	0	1	0	1	1	1	1	1	1	0	1	1	1	1	0	0	0	1	1	1	1	1	0	0	1	1	1	1	1	1	1	28	Хорошист
9-В Лакінський Володимир	1	0	1	1	0	0	0	1	1	1	1	0	1	1	1	1	0	1	1	1	1	1	1	1	1	0	1	1	1	1	1	0	1	1	1	1	1	1	1	1	32	Хорошист
9-В Яковлів Анжела	1	0	0	1	1	0	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	1	0	0	1	1	0	0	1	0	0	0	13	Слабкий
9-В Таранушко Всеволод	1	0	1	1	0	0	0	1	0	1	1	0	1	1	1	1	1	1	1	1	1	0	1	1	0	0	1	1	1	1	1	0	0	1	1	1	1	0	0	0	26	Хорошист
9-В Чаплінський Леонід	1	0	0	0	0	0	0	1	1	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	9	Дуже слабкий
9-В Волощенко Вадик	0	0	0	0	1	0	0	0	0	0	1	1	1	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	1	0	0	1	0	1	0	1	0	0	1	1	13	Дуже слабкий
9-В Шарандак Іна	1	0	1	1	0	1	1	1	0	0	1	0	0	1	1	1	0	1	1	1	1	0	1	1	0	0	0	1	1	1	1	0	1	1	1	1	1	1	1	0	27	Хорошист
9-В Саюк Віка	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	1	0	0	0	1	1	0	0	0	1	0	0	0	1	1	1	0	0	1	0	1	12	Слабкий
9-В Хом'ячук Аркадія	0	0	0	1	1	0	0	1	1	1	1	1	1	0	0	0	1	0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	0	0	1	0	0	0	17	Слабкий
9-В Шевернов Броніслава	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	1	1	0	0	0	0	1	1	0	0	1	1	1	12	Дуже слабкий
9-В Андрейчук Лада	0	1	0	0	1	1	1	1	0	1	1	0	1	1	1	1	1	0	1	1	0	1	0	0	1	0	0	1	1	1	1	0	1	1	1	0	0	0	1	1	25	Середній
9-В Канабас Зоряна	0	0	0	1	1	1	1	1	1	0	1	0	1	1	1	1	1	0	1	1	1	0	1	1	1	0	1	0	1	1	1	0	0	1	1	0	1	0	1	1	27	Хорошист
9-В Катюженко Глеб	1	1	0	0	1	0	1	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	0	1	1	0	1	1	1	1	0	0	0	1	1	1	0	1	1	0	27	Середній
9-В Шийко Владилен	0	0	0	1	1	0	1	1	0	0	0	0	1	1	0	1	0	1	1	1	1	1	1	1	1	0	0	1	1	0	1	0	1	1	0	0	1	0	0	0	21	Середній
9-В Заплетнюк Міла	1	0	1	1	1	0	1	1	1	1	0	1	1	1	0	1	1	1	1	1	1	1	1	1	0	0	1	1	1	1	1	0	1	1	1	1	1	1	1	1	33	Відмінник
9-В Теленко Олена	0	0	0	0	0	1	1	1	1	1	0	1	0	1	1	0	1	0	1	0	1	0	0	1	1	0	0	1	1	1	0	1	0	1	0	0	1	0	1	0	20	Середній

Рисунок В.1 – Головна сторніка

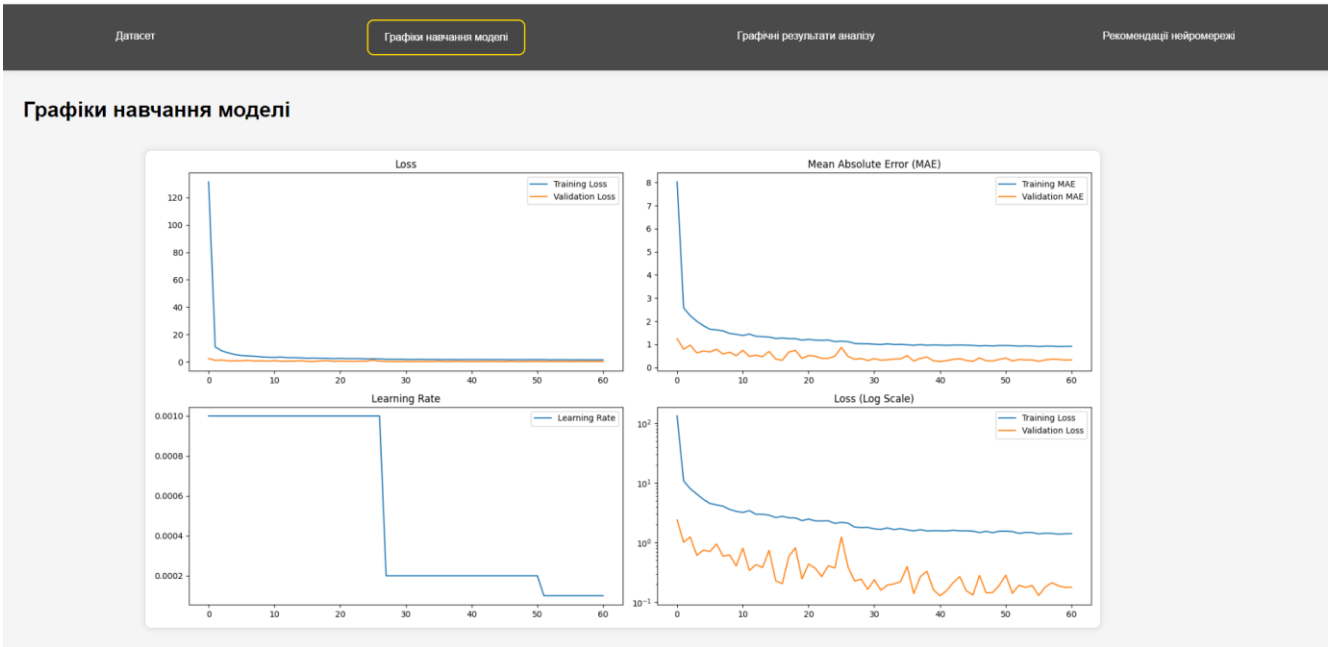


Рисунок В.2 – Сторінка відображення графіків навчання моделі

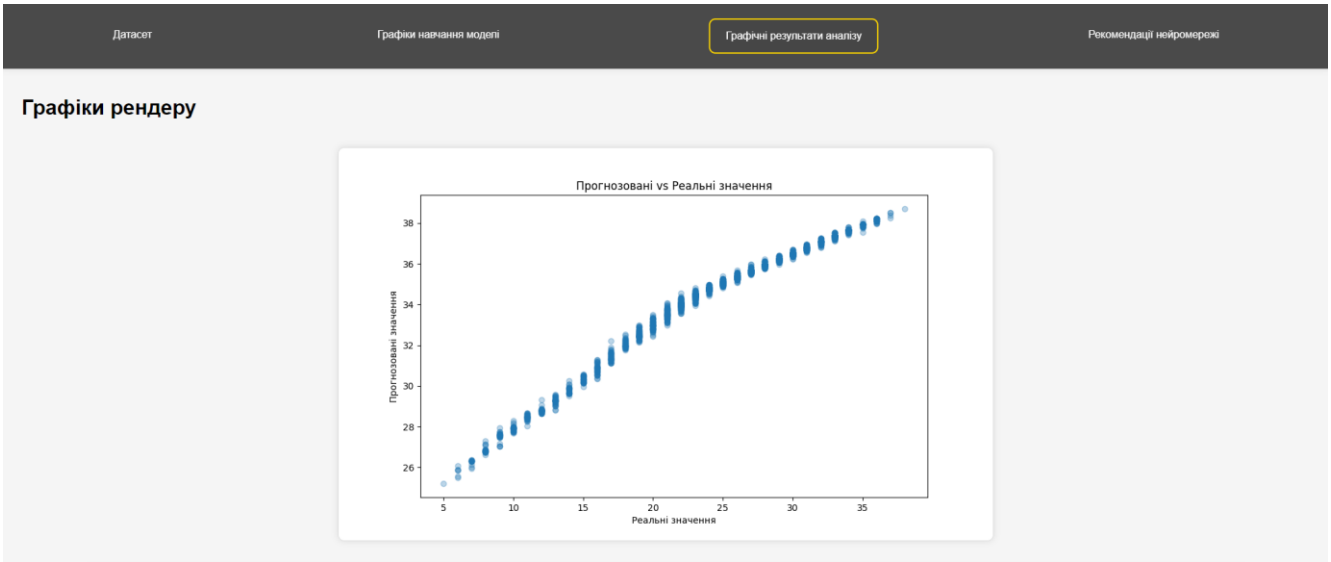


Рисунок В.3 – Сторінка відображення графіків рендеру результатів обчислення моделі

На даній сторінці відображено всі графіки результатів обчислення моделі, щоб ознайомитися із всіма ними, потрібно листати сторінку.

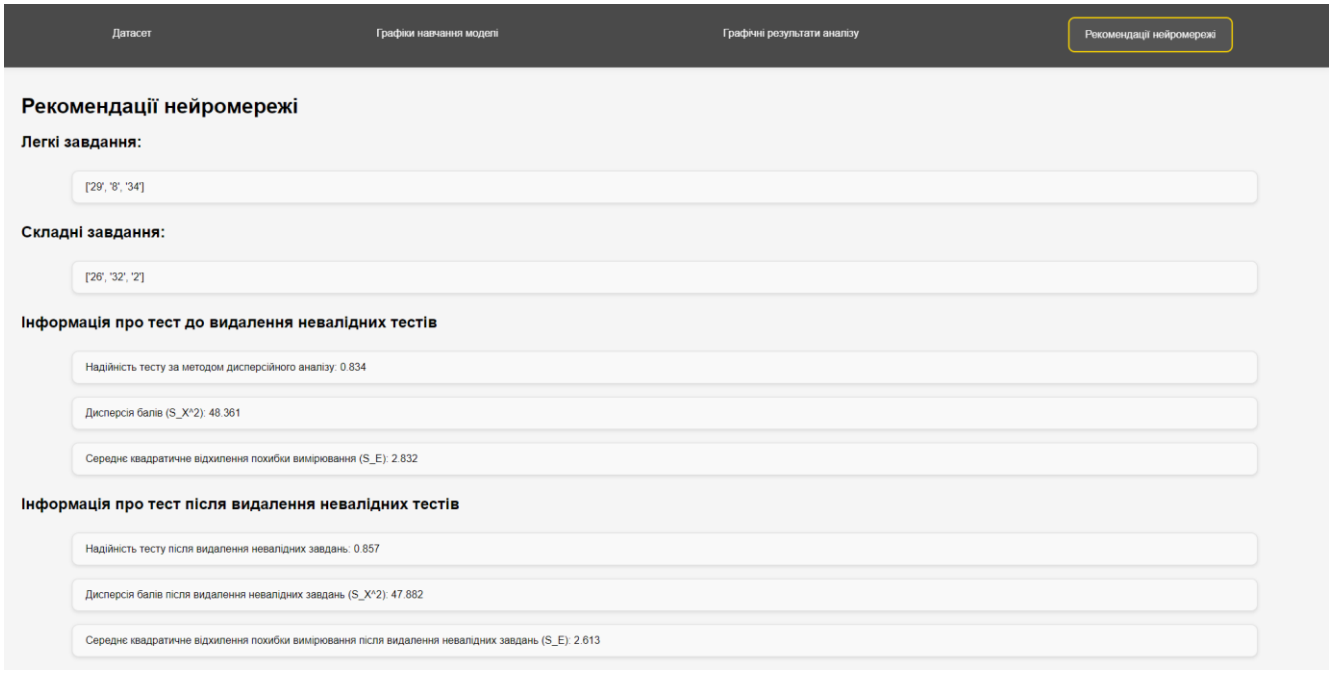


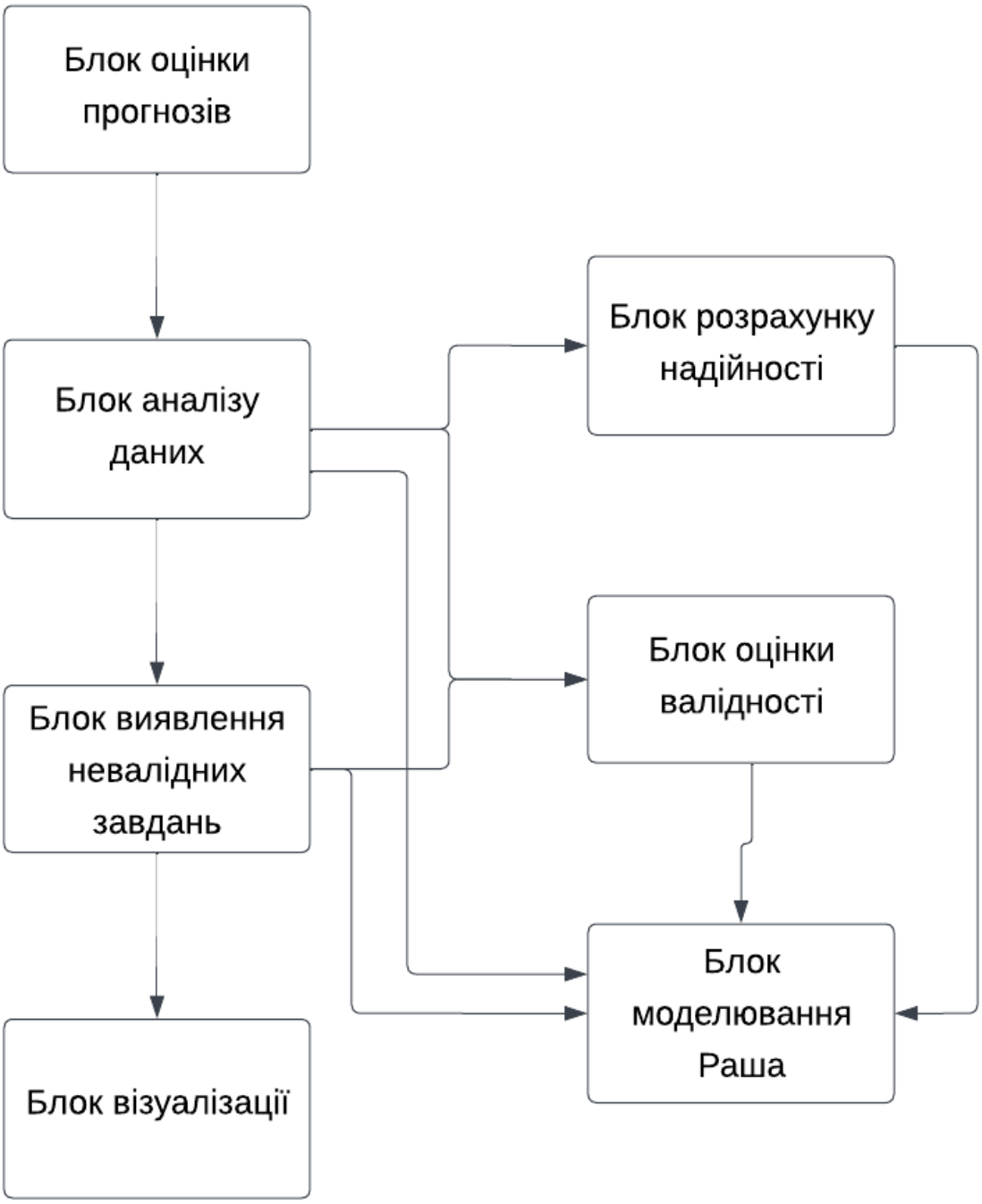
Рисунок В.4 – Сторінка відображення рекомендації нейромережі

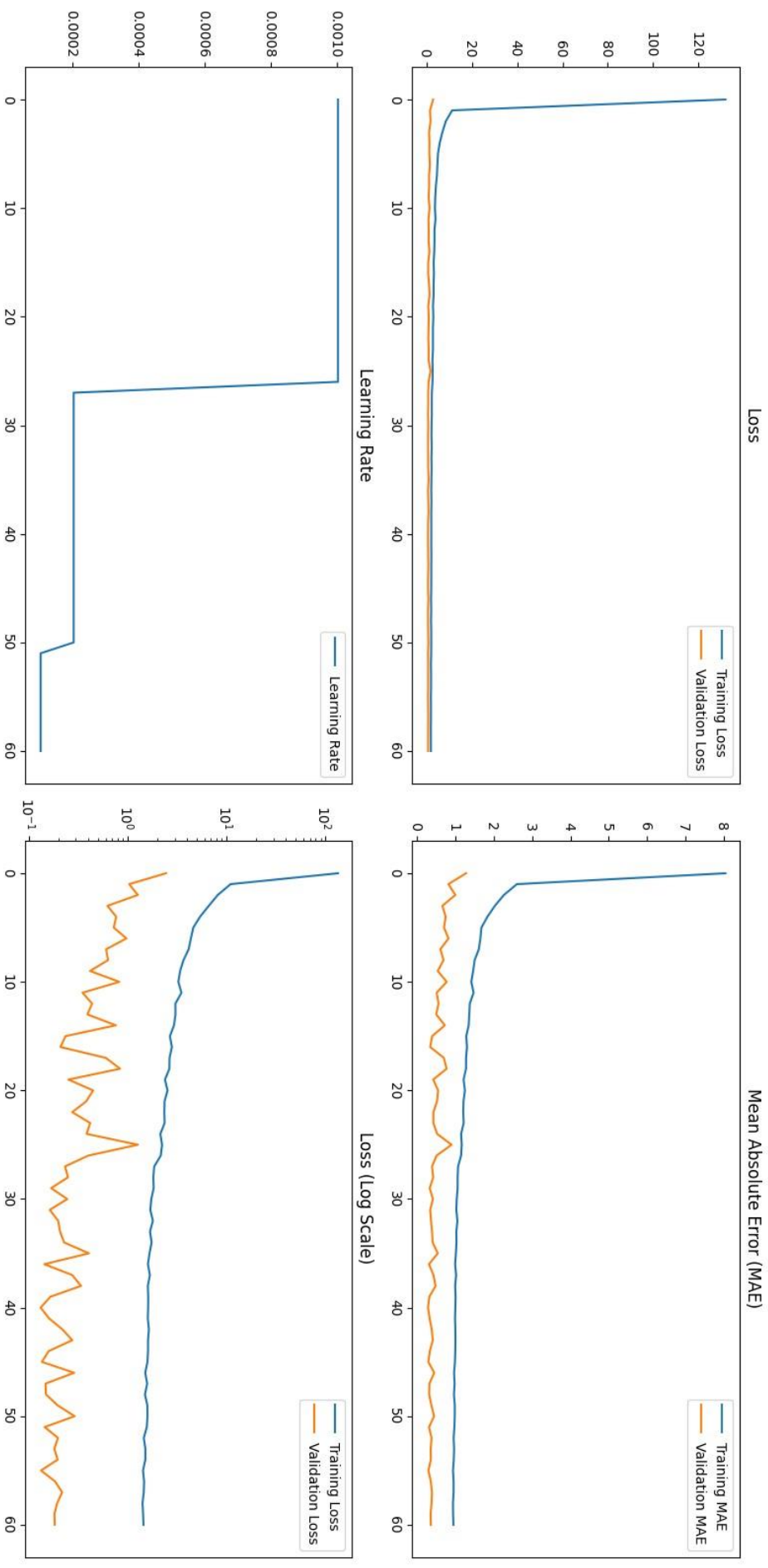
ДОДАТОК Г
ГРАФІЧНА ЧАСТИНА

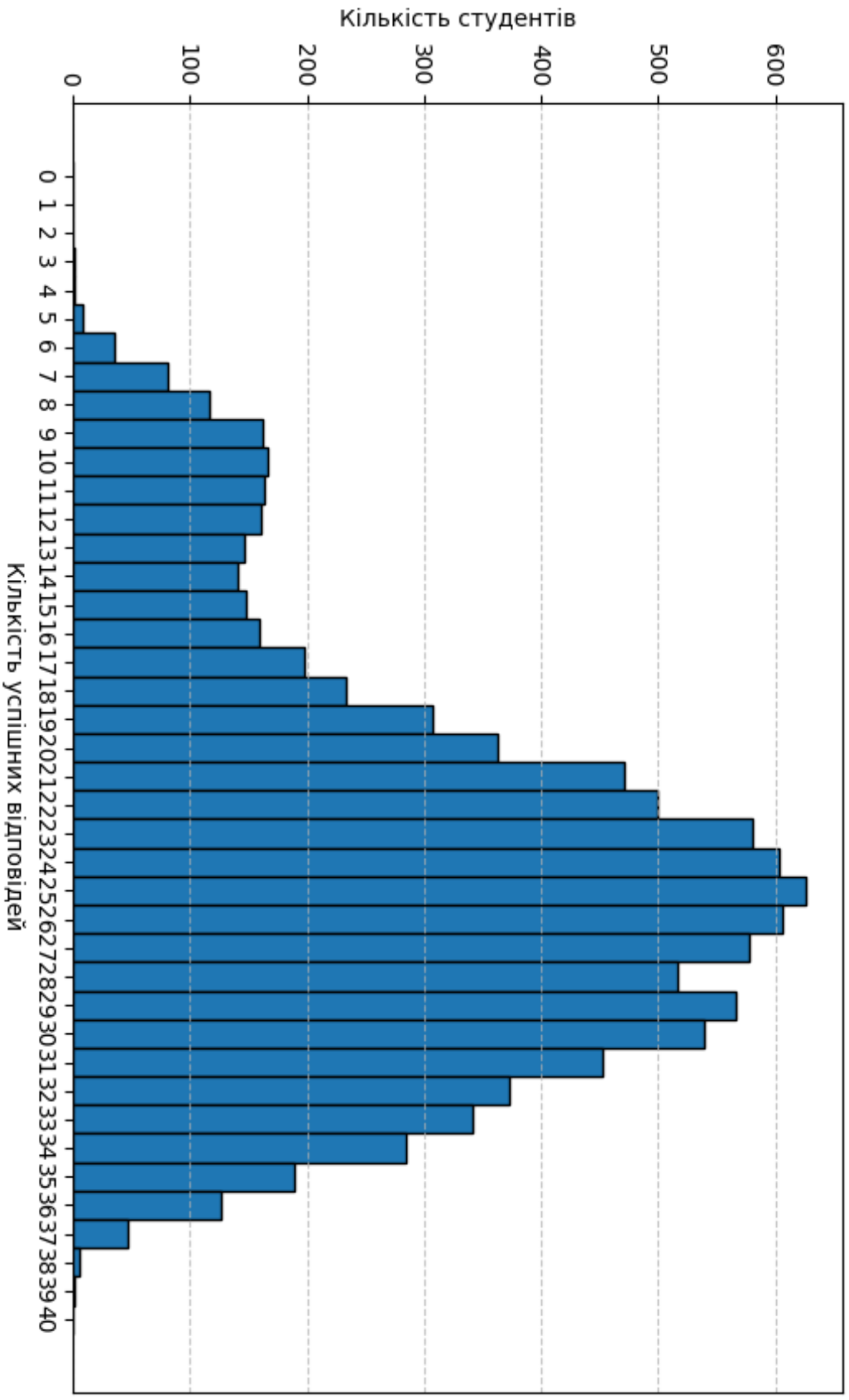
ПРОГРАМНИЙ МОДУЛЬ ОЦІНЮВАННЯ ЯКОСТІ ТЕСТОВИХ ЗАВДАНЬ В ОСВІТНІХ ВИМІРЮВАННЯХ

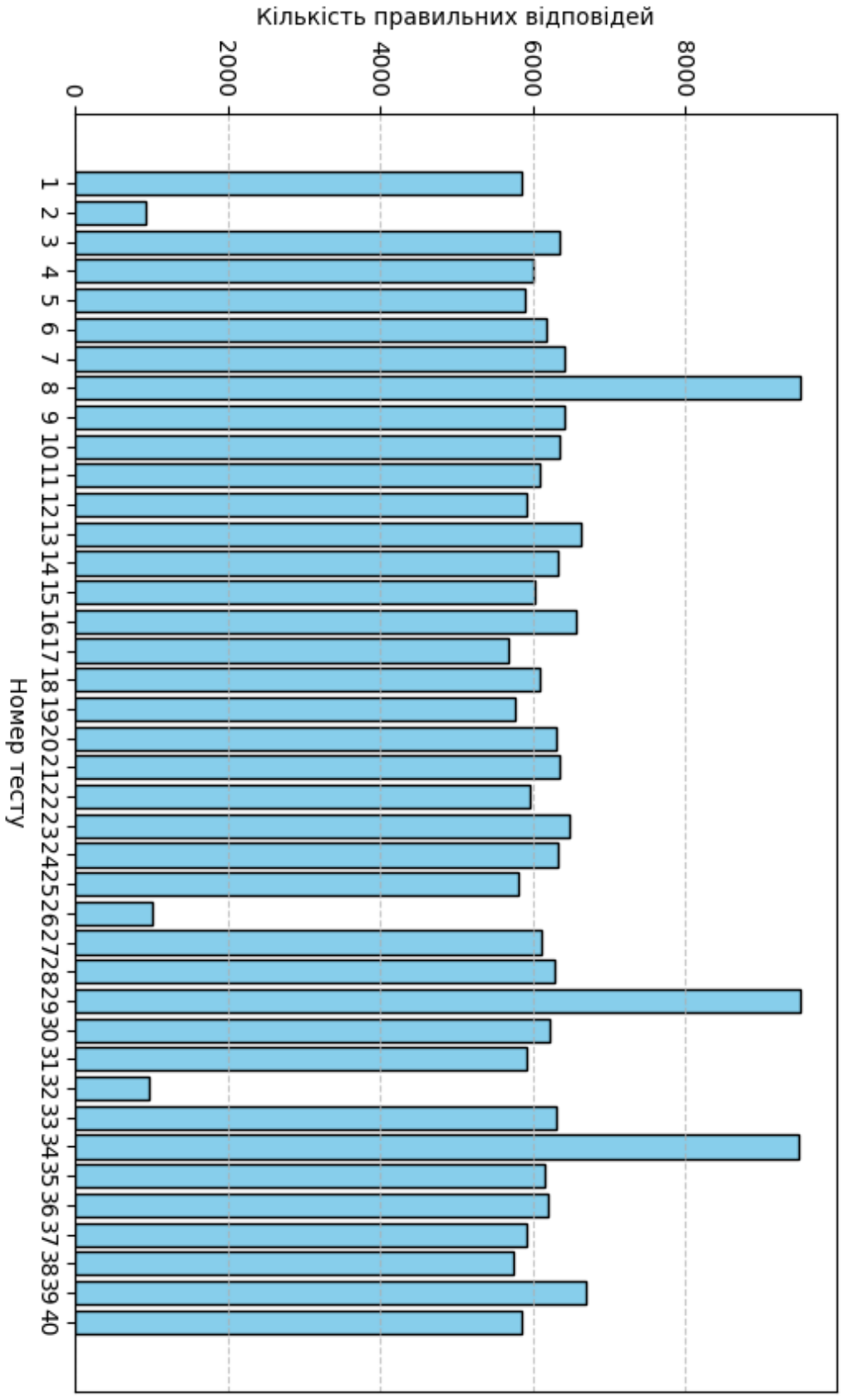
Виконав: студент 4 курсу, групи КН-22мс
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Христянчук В. В
(прізвище та ініціали)
Керівник: асистент каф. КН
Шевчук О. Ф.
(прізвище та ініціали)
«07» 06 2024 р.

Вінниця ВНТУ – 2024 рік


Блок оцінки прогнозів
Блок аналізу даних
Блок розрахунку надійності
Блок виявлення невалідних завдань
Блок оцінки валідності
Блок моделювання Раша
Блок візуалізації


Loss
Training Loss
Validation Loss
Mean Absolute Error (MAE)
Training MAE
Validation MAE
Learning Rate
Learning Rate
Loss (Log Scale)
Training Loss
Validation Loss


Розподіл кількості успішних відповідей студентів
Кількість студентів
0
100
200
300
400
500
600
Кількість успішних відповідей
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40


Кількість студентів, що правильно відповіли на конкретний тест
Кількість правильних відповідей
0
2000
4000
6000
8000
Номер тесту
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40