МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій
(повне найменування факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)
магістр
(ступінь вищої освіти)

на тему РОЗРОБКА МОДЕЛІ ПРОГНОЗУВАННЯ УСПІШНОСТІ СТУДЕНТІВ ЗА ДОПОМОГОЮ НЕЙРОННИХ МЕРЕЖ
DEVELOPMENT OF A MODEL FOR PREDICTING STUDENT PERFOMANCE USING NEURAL NETWORKS

Виконав(ла): студент(ка) 2 курсу, групи КНТ-214м
Спеціальності 122 Комп'ютерні науки
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Системи штучного інтелекту

СЕМЬОНОВ Н.Р.
(ПРІЗВИЩЕ та ініціали)
Керівник ЗАЙКО Т.А.
(ПРІЗВИЩЕ та ініціали)
Рецензент КОРОТКИЙ О.В.
(ПРІЗВИЩЕ та ініціали)

2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 122 Комп'ютерні науки
(код і найменування)
Освітня програма (спеціалізація) Системи штучного інтелекту
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
_____ Сергій СУББОТІН
"____" _____2025 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

СЕМЬОНОВА Нікіти Романовича
(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка моделі прогнозування успішності студентів за допомогою нейронних мереж.
Development of a Model for Predicting Student Perfomance Using Neural Networks
керівник проєкту (роботи) к.т.н., доцент, ЗАЙКО Тетяна Анатоліївна ,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)
затверджені наказом закладу вищої освіти від " 30 " вересня 2025 року № 447
2. Строк подання студентом проєкту (роботи) 08 грудня 2025 року
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області та постановка задачі. 2. Проєктування моделей аналізу навчальних даних. 3. Матеріали, методи та результати експериментальних досліджень. 4. Розробка програмного забезпечення.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-3 Основна частина	ЗАЙКО Т.А., доцент		
Нормоконтролер	КАЛІНІНА М.В., асистент		

7. Дата видачі завдання “ 30 ” вересня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2-3 тижні	Розділ 1
3	Розробка та удосконалення методів, моделей й алгоритмів	4-5 тижні	Розділ 2
	вирішення задачі.		
4	Розробка архітектури програми.	6 тиждень	Розділ 2
5	Експериментальне дослідження програмного	7-8 тижні	Розділ 3
	забезпечення.		
6	Реалізація та тестування програмного забезпечення.	9 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	10-11 тижні	Додатки
8	Нормоконтроль та рецензування.	12 тиждень	
9	Захист роботи.	12 тиждень	

Студент(ка) _____ Нікіта СЕМЬОНОВ
(підпис) (Імя ПРІЗВИЩЕ)

Керівник проєкту (роботи) _____ ТЕТЯНА ЗАЙКО
(підпис) (Імя ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра: 107 с., 2 табл., 35 рис., 2 дод., 26 джерел.

ПРОГНОЗУВАННЯ УСПІШНОСТІ, РАННЄ ПОПЕРЕДЖЕННЯ, НЕЙРОННІ МЕРЕЖІ, MLP, TCN, ГІБРИДНА МОДЕЛЬ, АНАЛІЗ ОСВІТНІХ ДАНИХ, OULAD.

Об’єкт дослідження – процес обчислень для обробки мультимодальних освітніх даних у навчальних системах.

Предмет дослідження – методи підготовки даних та нейромережеві моделі для задачі класифікації успішності студентів.

Мета роботи – підвищення ефективності прогнозування результатів навчання шляхом розробки та порівняльного аналізу моделей на основі ретельного конструювання ознак та підходу наскрізного навчання.

Матеріали, методи та технічні засоби: мова програмування Python, бібліотеки аналізу та обробки даних Pandas і NumPy, бібліотека машинного навчання Scikit-learn, бібліотека глибокого навчання PyTorch, публічний набір даних OULAD, середа розробки Visual Studio Code, персональний комп’ютер з операційною системою Windows.

Результати. Створено програмний комплекс для аналізу навчальних даних та прогнозування академічної успішності студентів на ранніх етапах навчання.

Висновки. Розроблено нейромережеву систему для оцінювання навчальних результатів на основі освітніх даних.

Галузь використання – системи підтримки прийняття рішень у закладах дистанційної та вищої освіти.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 107 pages, 2 tables, 35 figures, 2 appendixes, 26 sources.

STUDENT PERFORMANCE PREDICTION, EARLY WARNING, NEURAL NETWORKS, MLP, TCN, HYBRID MODEL, EDUCATIONAL DATA MINING, OULAD.

The object of the study is a computational process for handling multimodal educational data in learning systems.

The subject of the study are data preparation methods and neural network models for the task of classifying student performance.

The purpose of the work is to improve the effectiveness of learning outcome prediction by developing and comparatively analyzing models based on careful feature engineering and an end to end learning approach.

Materials, methods, and tools: the Python programming language, the Pandas and NumPy, data analysis and processing libraries, the Scikit-learn machine learning library, the PyTorch deep learning library, the OULAD public dataset, the Visual Studio Code development environment, a personal computer with Microsoft Windows.

Results. A software package has been created for analyzing educational data and predicting students academic success at the early stages of study.

Conclusions. A neural network system has been developed for evaluating educational outcomes based on educational data.

Field of use is a decision support systems in distance and higher education institutions.

ЗМІСТ

С.

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

Adam – Adaptive Moment Estimation;
BCE – Binary Cross-Entropy;
CNN – Convolutional Neural Network;
ECE – Expected Calibration Error;
EDM – Educational Data Mining;
F1-score – Harmonic mean of accuracy and completeness;
GELU – Gaussian Error Linear Unit;
L-BFGS – Limited-memory Broyden–Fletcher–Goldfarb–Shanno;
LSTM – Long Short-Term Memory;
MCC – Matthews Correlation Coefficient;
MLP – Multilayer Perceptron;
MLP-Tower – MLP subnet for tabular data processing;
OULAD – Open University Learning Analytics Dataset;
PR-AUC – Precision-Recall Area Under Curve;
ReLU – Rectified Linear Unit;
RNN – Recurrent Neural Network;
ROC-AUC – Receiver Operating Characteristic Area Under Curve;
TCN – Temporal Convolutional Network;
TCN-Tower – TCN subnet for serial data processing.

ВСТУП

Стрімкий розвиток інформаційних технологій та глобальні виклики останніх років призвели до масового переходу освітнього процесу в онлайн-формат. Платформи дистанційного навчання генерують величезні обсяги даних про активність та успішність студентів. Водночас, однією з ключових проблем онлайн-освіти залишається високий рівень відсіву студентів (dropout rate), що призводить до значних економічних та соціальних втрат як для закладів освіти, так і для самих здобувачів. У цьому контексті надзвичайно актуальною стає задача раннього прогнозування академічної успішності, яка дозволяє своєчасно ідентифікувати студентів групи ризику та вжити превентивних заходів для їх підтримки.

Сучасні підходи до вирішення цієї задачі активно використовують методи машинного навчання та аналізу освітніх даних (Educational Data Mining, EDM). Дослідники застосовують широкий спектр алгоритмів, від класичних (логістична регресія, дерева рішень) до просунутих нейромережевих архітектур.

Тому актуальним **є** наукове обґрунтування вибору оптимальної стратегії обробки мультимодальних освітніх даних (статичних та часових) для підвищення точності та надійності прогностичних моделей в умовах онлайн-навчання.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Підходи до прогнозування на основі табличних даних

Прогнозування академічної успішності студентів за допомогою табличних даних є одним із найбільш досліджених та фундаментальних напрямків у галузі аналізу освітніх даних (Educational Data Mining, EDM) [1]-[2]. Цей підхід часто називають "статичним" або "агрегованим", оскільки він ґрунтується на створенні єдиного "знімка" стану студента в обраний момент часу. Кожен студент представляється у вигляді одного рядка в таблиці, а кожен стовпець відповідає за певну характеристику або ознаку (feature).

Ключовим елементом цієї методології є процес вибору системи інформативних ознак [3]. На відміну від "end-to-end" підходів, де "сирі" дані (наприклад, щоденні логи) подаються безпосередньо в модель, тут дослідник бере на себе відповідальність за попередню обробку та агрегацію. Дослідник формулює гіпотези про те, які саме фактори можуть впливати на успішність, і вручну розраховує ці показники. Ці ознаки, як правило, поділяються на дві великі категорії.

Статичні та демографічні ознаки. Перша категорія включає дані, які є незмінними або майже незмінними протягом усього періоду навчання. Вони збираються, як правило, на етапі реєстрації студента. До них належать:

– демографічні дані: стать, вік, регіон проживання;

– соціально-економічні показники: наприклад, індекс множинної депривації (IMD band), який вказує на рівень добробуту регіону проживання студента;

– академічний бекграунд: найвищий досягнутий рівень освіти (наприклад, A-level, бакалавр тощо), що слугує індикатором попередньої підготовки.

– особливі умови: наявність задекларованої інвалідності;

– параметри реєстрації: кількість обраних кредитів (studied_credits) та кількість попередніх спроб (num_of_prev_attempts) вивчити даний курс.

Ці ознаки є надзвичайно цінними для раннього прогнозування, оскільки вони доступні ще до початку курсу і дозволяють виявити початкові групи ризику [4].

Агреговані динамічні ознаки. Друга, і часто більш прогностично потужна, категорія ознак — це результат агрегації даних, зібраних протягом навчального процесу до певної точки зрізу (cut-off day).

Замість аналізу повної часової послідовності, дослідник розраховує зведені показники, що характеризують поведінку та успішність студента. До показників відносяться:

– показники академічної успішності: це найсильніші предиктори. Замість того, щоб аналізувати кожну оцінку окремо, розраховуються такі показники, як: загальна кількість отриманих балів, середньозважений бал (з урахуванням "ваги" кожного завдання), відсоток зданих робіт від загальної кількості необхідних, максимальний та мінімальний отриманий бал;

– показники залученості до VLE: "сирі" дані про кліки (clickstream data) трансформуються в єдині числа: загальна кількість кліків у системі, кількість унікальних днів активності, загальна кількість переглянутих матеріалів, сумарна кількість повідомлень на форумі;

– показники своєчасності: кількість завдань, зданих із запізненням, середня кількість днів протермінування, відсоток "пізніх" робіт від загальної кількості.

Саме якість цього етапу інженерії ознак напряму впливає на точність фінальної моделі. Багато досліджень, включно з аналізом датасету OULAD, демонструють, що саме агреговані дані про оцінки є значно сильнішими предикторами, ніж дані про поведінку в VLE.

Для моделювання таких табличних даних використовується широкий спектр алгоритмів машинного навчання. До них входять:

– класичні моделі: велика популярність належить логістичній регресії (через її простоту та інтерпретованість), методу опорних векторів (SVM), а також ансамблевим методам, таким як Випадковий Ліс (Random Forest) та

градієнтний бустинг (XGBoost, LightGBM) [5]-[6]. Їхня перевага — висока точність "з коробки" та можливість оцінити "важливість ознак" (feature importance), що є цінним для розуміння, які саме фактори найбільше впливають на відсів;

– нейронні мережі: для пошуку складних, нелінійних залежностей між ознаками ефективно застосовуються нейронні мережі прямого поширення, а саме багатошаровий перцептрон (MLP). Сучасні реалізації MLP (як у PyTorch чи TensorFlow) дозволяють елегантно поєднувати різні типи даних: числові ознаки (наприклад, середній бал) проходять через стандартизацію та подаються напряму у повнозв'язні шари (Linear), тоді як категоріальні ознаки (наприклад, регіон) спочатку перетворюються у вектори занурення (Embeddings), що дозволяє моделі вивчити смислову близькість між категоріями.

Перевагою підходу є те, що у його доведеній роками високій ефективності та гнучкості. Моделі, навчені на якісних, продуманих ознаках, часто показують результати, які важко перевершити [7].

Ключовий недолік — це втрата будь-якої часової інформації. Процес агрегації (усереднення) незворотно знищує дані про динаміку. Наприклад, студент А, який мав середній бал '70' завдяки стабільному навчанню протягом 12 тижнів, для моделі MLP буде абсолютно ідентичним студенту Б, який також має середній бал '70', але отримав його, проваливши всі перші завдання і наздогнавши матеріал в останні тижні. Їхні "знімки" однакові, хоча траєкторії ризику — кардинально різні.

Саме цей фундаментальний недолік ігнорування динаміки є основною мотивацією для розробки та застосування моделей, здатних аналізувати часові послідовності, які будуть детально розглянуті в наступному підпункті.

1.2 Підходи до прогнозування на основі послідовних даних

На противагу статичним підходам, описаним у попередньому підпункті, моделювання на основі послідовних даних ґрунтується на фундаментальному припущенні, що динаміка та траєкторія поведінки студента є ключовими для прогнозування його успішності. Замість того, щоб агрегувати дані у єдиний "знімок", цей підхід аналізує часові ряди студентської активності, намагаючись виявити приховані патерни, тенденції та аномалії [8].

Джерело та представлення даних - основним джерелом даних для цього підходу є логи взаємодії студента з віртуальним навчальним середовищем (VLE), також відомі як "clickstream data". Ці дані фіксують кожну дію студента: кожен клік, перегляд матеріалу, відправку повідомлення на форум чи спробу пройти тест, фіксуючи точну часову мітку.

Для того, щоб ці "сирі" дані можна було подати у нейронну мережу, їх необхідно перетворити у формат послідовностей фіксованої довжини. Це досягається шляхом "дискретизації" часу. Наприклад, вся активність студента групується у часові "кошики" (bins) - по годинах, днях або, що є дуже поширеним, по тижнях.

У результаті, замість одного вектора ознак (як в MLP), кожен студент представляється у вигляді матриці (або тензора) X розмірністю (T×F), де:

– T це кількість часових кроків (наприклад, 12 тижнів);

– F це кількість ознак, що вимірюються на кожному кроці (наприклад, 20 типів активності в VLE, кількість кліків, час сесії тощо).

Таке представлення дозволяє моделі аналізувати не лише що робив студент, але і коли та в якій послідовності він це робив.

Для моделювання послідовностей, обробка часових рядів вимагає спеціалізованих архітектур нейронних мереж, здатних підтримувати "пам'ять" про попередні стани. До них входять:

– рекурентні Нейронні Мережі (RNN), класичний підхід до моделювання послідовностей. RNN мають "цикл" у своїй архітектурі, що дозволяє прихованому стану (hidden state) з попереднього часового кроку впливати на обчислення поточного. Однак, прості RNN страждають від проблем зникаючого та вибухаючого градієнта (vanishing/exploding gradients), що робить їх неефективними для навчання на довгих послідовностях;

– мережі з Довгою Короткочасною Пам'яттю (LSTM) та Gated Recurrent Units (GRU): Ці архітектури були розроблені як вирішення проблем RNN. LSTM та її спрощений варіант GRU використовують складніші рекурентні осередки з механізмами "гейтів" (gates) — вентилів забування, введення та виведення [9]. Ці гейти дозволяють мережі вибірково вирішувати, яку інформацію з минулого зберегти у своєму стані, яку оновити, а яку — "забути". Це робить їх стандартом для багатьох завдань з послідовностями і вони успішно застосовувалися для прогнозування успішності на даних OULAD;

– згорткові Нейронні Мережі (CNN) та Temporal Convolutional Networks (TCN): Хоча CNN традиційно асоціюються з обробкою зображень, їх також можна застосовувати до 1D-послідовностей. Сучасним розвитком цієї ідеї є Часові Згорткові Мережі (TCN);

TCN є потужною альтернативою RNN з низкою ключових переваг:

– паралелізм: На відміну від RNN, які мають обробляти дані крок за кроком, згортки можна обчислювати паралельно, що значно прискорює навчання;

– каузальні згортки (Causal Convolutions): TCN використовують спеціальний тип згортки, який гарантує, що обчислення на кроці t залежить лише від даних на кроках t,t-1,t-2,…,і ніколи не "підглядає у майбутнє". Це критично важливо для задач прогнозування;

– розширені згортки (Dilated Convolutions): TCN використовують згортки з експоненціально зростаючим "розширенням" (dilation). Це дозволяє мережі мати надзвичайно велике "рецептивне поле" (receptive field) - тобто,

вона може враховувати дуже довгу історію минулого, використовуючи при цьому відносно неглибоку архітектуру;

– залишкові з'єднання (Residual Connections): Як і в ResNet, залишкові з'єднання допомагають у навчанні глибоких TCN-мереж.

Дослідження показують, що TCN часто перевершують RNN/LSTM за точністю на багатьох задачах з послідовностями і вже успішно застосовуються для аналізу поведінки студентів та прогнозування груп ризику.

Переваги та недоліки підходу. Головна перевага "end-to-end" підходу з послідовностями полягає в його здатності автоматично вивчати складні часові патерни. Модель потенційно може ідентифікувати такі складні явища, як "вигорання" (burnout) студента (активний початок і різкий спад активності), "прокрастинація" (низька активність і вибух перед дедлайном) або стабільна, методична робота.

Однак, цей підхід має і суттєві недоліки:

– чутливість до "шуму": "Сирі" дані VLE є надзвичайно "шумними". Кількість кліків не завжди корелює з якістю навчання. Студент міг відкрити вкладку з матеріалами і не читати її, або, навпаки, завантажити всі PDF-файли за один день для офлайн-роботи. Моделі послідовностей можуть "заплутатися" у цьому шумі, не знайшовши реального прогнозного сигналу;

– вимоги до даних: Для ефективного навчання TCN або LSTM часто потрібні дуже довгі та деталізовані послідовності. Якщо дані агреговані занадто "грубо" (наприклад, у 12 тижневих блоків), послідовність може виявитися занадто короткою для виявлення складних патернів;

– інтерпретованість: Ці моделі часто є "чорними скриньками", і зрозуміти, чому модель віднесла студента до групи ризику, набагато складніше, ніж проаналізувати важливість ознак у моделі Випадкового Лісу.

Саме поєднання "шумності" вхідних даних VLE та високої прогнозної сили агрегованих оцінок (як було показано в 1.1) мотивує дослідників до розробки гібридних архітектур, які будуть розглянуті далі.

1.3 Гібридні та мультимодальні підходи

Гібридні, або мультимодальні, підходи до прогнозування є логічним розвитком та об'єднанням двох методологій, описаних у попередніх підпунктах. Замість того, щоб робити вибір між потужними, але "пласкими" агрегованими ознаками (1.1) та "шумними", але динамічними послідовностями (1.2), гібридні моделі намагаються використати переваги обох типів даних одночасно.

Обґрунтування та гіпотеза: Основна гіпотеза цього підходу полягає в тому, що різні типи даних (модальності) містять доповнюючу (complementary), а не лише дублюючу інформацію. Успішність студента — це складне явище, яке залежить як від його "статичного" профілю, так і від "динамічної" поведінки. Данні та їх призначення:

– табличні дані (MLP-частина): Надають моделі сильний базовий контекст. Вони відповідають на питання: "Який початковий профіль ризику студента?" (на основі демографії, попередньої освіти) та "Як він справляється з основними контрольними точками?" (на основі агрегованих оцінок);

– послідовні дані (TCN/LSTM-частина): Надають моделі поведінковий контекст у реальному часі. Вони відповідають на питання: "Як студент поводиться між контрольними точками?" (наприклад, його активність у VLE, патерни прокрастинації чи залученості).

Очікується, що модель, яка має доступ до обох джерел, зможе виявити складніші взаємозв'язки. Наприклад, вона може навчитися, що "низька активність у VLE" (динамічна ознака) є критично небезпечною для студентів з низьким рівнем попередньої освіти (статична ознака), але є нормальною для досвідчених студентів, які навчаються офлайн.

Типові архітектури: Реалізація гібридних моделей у глибокому навчанні найчастіше слідує архітектурі "двох веж" (Two-Tower Model), яка об'єднує дані на пізньому етапі (late fusion).

– "вежа" 1: Таблична (MLP-Tower). Спеціалізований набір шарів (наприклад, MLP з Embeddings, як описано в 1.1) обробляє виключно статичні та агреговані табличні дані. На виході ця вежа генерує векторний "зліпок" (embedding) статичного профілю студента;

– "вежа" 2: Послідовна (TCN/RNN-Tower). Паралельна, незалежна гілка (наприклад, TCN або LSTM, як описано в 1.2) обробляє виключно часові послідовності (наприклад, логи VLE). На виході вона генерує векторний зліпок, що узагальнює поведінкову траєкторію студента;

– шар злиття (Fusion Layer): Векторні зліпки з обох веж об'єднуються. Найпростішим методом є конкатенація (просте об'єднання векторів).

Більш просунуті методи можуть використовувати:

– повнозв'язні шари: Конкатенований вектор подається на один або декілька шарів Linear (головний "класифікатор"), які вчаться зважувати важливість ознак з обох модальностей;

– механізми уваги (Attention): Спеціальні шари (наприклад, Squeeze-and-Excitation, Self-Attention) можуть динамічно визначати, якій "вежі" (табличній чи послідовній) слід "довіряти" більше для кожного конкретного студента.

Переваги та виклики: Головна перевага — це потенціал для найвищої точності. Теоретично, така модель має найбільш повну картину про студента, поєднуючи "що" (оцінки) і "як" (поведінка).

Однак, цей підхід несе в собі значні виклики:

– складність архітектури: Модель стає значно складнішою, вимагає більше ресурсів для навчання та її важче налаштовувати (тюнінгувати гіперпараметри);

– ризик "домінування" модальностей: Це ключова проблема, яку часто виявляють дослідники. Якщо одна з модальностей (наприклад, агреговані оцінки) несе в собі значно сильніший і чистіший сигнал, ніж інша (наприклад, "шумні" кліки VLE), нейронна мережа під час навчання може просто "обнулити" ваги для слабшої вежі. В такому випадку гібридна модель

фактично колапсує до моделі однієї модальності (наприклад, стає просто MLP), і вся складність TCN-вежі виявляється непотрібною;

– інтерпретованість: Якщо вже MLP та TCN окремо важко інтерпретувати, то їх комбінація стає ще більшою "чорною скринькою".

Незважаючи на ці виклики, гібридні моделі представляють передовий край досліджень у прогнозуванні успішності, оскільки вони найбільш точно моделюють мультимодальну природу самої проблеми навчання.

1.4 Висновки за розділом 1

Проведений аналіз наукових джерел щодо прогнозування академічної успішності студентів дозволяє виділити три ключові, але конкуруючі методології побудови нейромережевих моделей, які лягли в основу даної дипломної роботи.

Підхід на основі інженерії ознак (Feature Engineering), описаний в 1.1, є найбільш класичним. Він полягає у ручному створенні високоінформативних ознак з подальшим використанням моделей типу MLP. Перевага цього підходу — у високій прогнозній силі продуманих ознак. Недолік – у повній втраті часової динаміки поведінки студента.

Підхід на основі послідовних даних (Sequence Modeling), описаний в 1.2, є більш сучасним "end-to-end" рішенням. Він використовує "сирі" часові ряди (наприклад, логи VLE) і подає їх у спеціалізовані архітектури, такі як TCN або LSTM, для автоматичного пошуку патернів. Перевага — у здатності фіксувати динаміку (наприклад, прокрастинацію. Недолік — у високій чутливості до "шумних" вхідних даних, де, наприклад, кількість кліків не завжди дорівнює якості навчання.

Гібридний (мультимодальний) підхід, описаний в 1.3, намагається об'єднати переваги обох світів, обробляючи табличні та послідовні дані у паралельних "вежах". Перевага — у теоретично найбільш повній картині про студента. Недолік – у складності та ризику "домінування модальностей",

коли модель може навчитися ігнорувати слабку, "шумну" модальність (наприклад, кліки) на користь сильної (наприклад, оцінки).

Таким чином, проведений аналіз виявляє фундаментальне науково-технічне протиріччя: чи є важливішою продумана, агрегована людиною інформація про результати (оцінки), чи "сира", але детальна інформація про процес (кліки)?

Це протиріччя формулює актуальність даної роботи: порівняти три алгоритми (MLP, TCN, Hybrid), та емпірично дослідити, яка зі стратегій моделювання є найбільш ефективною для прогнозування успішності студентів на реальних даних.

2 ПРОЄКТУВАННЯ МОДЕЛЕЙ АНАЛІЗУ НАВЧАЛЬНИХ ДАНИХ

2.1 Обґрунтування вибору засобів розробки та програмного середовища

Ефективність реалізації системи прогнозування успішності студентів значною мірою залежить від правильного вибору інструментарію. Технологічний стек проекту було сформовано з урахуванням вимог до обробки великих масивів даних, необхідності побудови кастомних архітектур нейронних мереж (MLP, TCN, Hybrid) та забезпечення відтворюваності експериментів. До них входять:

а) мова програмування: В якості основної мови реалізації було обрано Python [10]. Цей вибір зумовлений тим, що на сьогодні Python є стандартом у галузі науки про дані (Data Science) та машинного навчання. Ключові фактори, що вплинули на вибір:

1) багата екосистема бібліотек: Наявність оптимізованих інструментів для кожного етапу дослідження — від попередньої обробки даних (Pandas) до глибокого навчання (PyTorch);

2) висока продуктивність розробки: Лаконічний синтаксис дозволяє фокусуватися на логіці алгоритмів, а не на низькорівневому управлінні пам’яттю, що критично важливо на етапі прототипування архітектур;

3) інтеграційні можливості: Python легко інтегрується з високопродуктивними бібліотеками, написаними на C/C++, що нівелює недоліки інтерпретованої природи мови при виконанні важких математичних операцій.

б) інтегроване середовище розробки (IDE). Для написання та налагодження програмного коду було обрано середовище Visual Studio Code (VS Code). Це сучасний, легковажний редактор коду, який завдяки системі розширень перетворюється на повноцінну IDE. Переваги використання VS Code у даній роботі:

1) підтримка Jupyter Notebooks: Можливість поєднувати інтерактивний розвідувальний аналіз даних (EDA) та візуалізацію графіків безпосередньо в середовищі розробки;

2) віддалена розробка (Remote - SSH): Навчання нейронних мереж є ресурсомістким процесом. VS Code дозволяє редагувати код на локальній машині, а виконувати його на віддаленому сервері з потужним GPU через SSH-тунель, що значно прискорює процес експериментів;

3) інструменти налагодження: Вбудований дебагер дозволяє покроково інспектувати розмірності тензорів PyTorch, що є критично важливим для уникнення помилок при проектуванні складних блоків злиття (Fusion) у гібридній моделі.

в) бібліотеки та фреймворки. Програмна реалізація базується на використанні наступних спеціалізованих бібліотек:

1) PyTorch: Основний фреймворк глибокого навчання. Його було обрано замість TensorFlow через механізм динамічних обчислювальних графів (dynamic computation graphs), який значно спрощує реалізацію нестандартних архітектур, таких як TCN (Temporal Convolutional Network) та механізмів уваги (Attention). Також PyTorch надає зручні абстракції Dataset та DataLoader для ефективного пакетного завантаження даних [11];

2) Pandas та NumPy: Використовувалися для маніпуляцій з табличними даними датасету OULAD. NumPy забезпечує високошвидкісні матричні операції, необхідні для векторизації вхідних даних, а Pandas дозволяє ефективно виконувати агрегацію, групування та фільтрацію часових рядів активності студентів;

3) Scikit-learn: Застосовувалася для попередньої обробки даних (масштабування StandardScaler, кодування категорій) та розрахунку метрик якості моделей (ROC-AUC, F1-score, Confusion Matrix) [12];

4) Matplotlib та Seaborn: Використовувалися для візуалізації результатів, побудови графіків процесу навчання (learning curves) та діаграм калібрування ймовірностей.

г) апаратне забезпечення: Враховуючи обчислювальну складність операцій згортки в моделі TCN та необхідність навчання на великій кількості епох, програмний комплекс було оптимізовано для виконання на графічних процесорах (GPU) з підтримкою технології NVIDIA CUDA. Це дозволило скоротити час навчання моделей з годин до хвилин порівняно з використанням CPU. Для забезпечення відтворюваності результатів (reproducibility) у середовищі було реалізовано фіксацію генераторів псевдовипадкових чисел (seeds) для всіх використовуваних бібліотек.

2.2 Проєктування моделі MLP для табличних даних

Перша з розроблених моделей, багатошаровий перцептрон (MLP), проектувалася з подвійною метою. По-перше, вона слугує потужною базовою моделлю (baseline) для оцінки ефективності прогнозування, спираючись лише на табличні дані. По-друге, вона виконує роль модульної "табличної вежі" (MLP-Tower), яка буде інтегрована у фінальну гібридну архітектуру.

Модель спроектована для роботи з двома різними типами вхідних даних, які були підготовлені на етапі 3.1: 11 числових (Numerical) ознак та 6 категоріальних (Categorical) ознак.

Для ефективної обробки цих гетерогенних даних застосовується комбінований підхід. 11 числових ознак подаються в модель напряму. 6 категоріальних ознак проходять через механізм векторних занурень. Кожна категоріальна ознака отримує власну таблицю занурень (nn.Embedding), яка перетворює числовий індекс категорії (напр., 'Region_A' → 2) у щільний векторний простір. Розмірність (emb_dim) цього простору для кожної ознаки розраховується за поширеною евристикою: min(50,(кількість_категорій ++ 1) //2). Це дозволяє моделі самостійно вивчити семантичну "близькість" між різними категоріями, що є значно ефективнішим за методи, наприклад, One-Hot Encoding.

Архітектура моделі ModelMLP логічно розділена на "вежу" На рисунку 2.1 наведено (MLP_Tower) та "голову" (Head) для забезпечення модульності. MLP_Tower відповідає за обробку вхідних даних та генерацію їх високо-рівневого представлення (зліпка), а Head — за фінальну класифікацію.

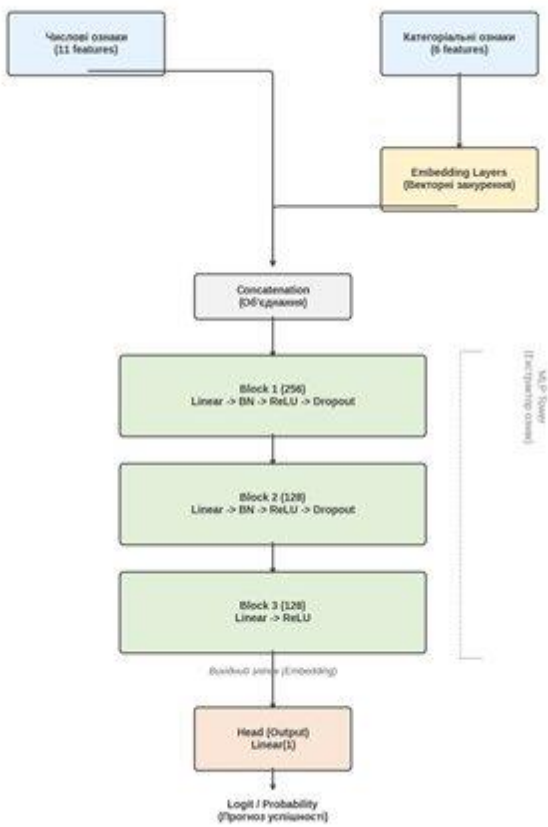

Рисунок 2.1 – Структурна схема архітектури багатошарового перцептрона

"Вежа" (MLP_Tower) приймає на вхід числові (xnum) та категоріальні (xcat) тензори. Її архітектура складається з наступних кроків:

– обробка категорій: Всі 6 категоріальних ознак проходять через свої відповідні шари nn.Embedding;

– конкатенація: Отримані 6 векторів занурень об'єднуються (конкатенуються) між собою та з 11-ма числовими ознаками. Це створює єдиний "плаский" вхідний вектор (mlp_in), що комплексно представляє студента;

– основна мережа: Цей об'єднаний вектор подається на nn.Sequential з трьох повнозв'язних блоків. Загальна концепція такої багатошарової архітектури, де інформація поширюється від вхідного шару через приховані до вихідного, проілюстрована на рисунку 2.1.

Детальна архітектура спроектованих блоків MLP_Tower у програмній реалізації виглядає наступним чином:

а) блок 1 (Вхідний):

1) nn.Linear(mlp_in, 256): Повнозв'язний шар, що проектує вхідний вектор у простір 256 нейронів;

2) nn.BatchNorm1d(256): Шар пакетної нормалізації. Він стабілізує навчання, нормалізуючи активації, що дозволяє використовувати вищі швидкості навчання та прискорює збіжність;

3) nn.ReLU(): Стандартна нелінійна функція активації (Rectified Linear Unit);

4) nn.Dropout(0.2): Шар регуляризації з гіперпараметром p=0.2. На кожному кроці навчання він випадково "виключає" 20% нейронів, запобігаючи перенавчанню та ко-адаптації нейронів [13]-[14].

б) блок 2 (Прихований):

1) nn.Linear(256, 128): Подальше зменшення розмірності до 128 нейронів;

2) nn.BatchNorm1d(128): Пакетна нормалізація;

3) nn.ReLU(): Активація;

4) nn.Dropout(0.2): Регуляризація p=0.2.

в) блок 3 (Вихідний зліпок):

1)-nn.Linear(128, 128): Фінальний повнозв'язний шар, що генерує 128-вимірний вектор;

2) nn.ReLU(): Фінальна активація. Цей 128-вимірний вектор є "зліпком" (embedding) або фінальним представленням студента, яке генерує MLP_Tower;

3) окрема "голова" (Head) слугує універсальним класифікатором. Вона приймає 128-вимірний зліпок від "вежі"(nn.Linear(128, 1)) і виконує єдину дію: nn.Linear(128, 1): Проектує 128-вимірний вектор на єдиний вихідний нейрон. Цей єдиний вихід (logit) представляє собою "сирий" результат моделі перед застосуванням сигмоїдної функції. Він інтерпретується як логарифм шансів успішного завершення курсу. Використання саме логіту, а не ймовірності, є важливою технічною деталлю, оскільки це дозволяє використовувати функцію втрат BCEWithLogitsLoss під час навчання, яка є чисельно більш стабільною, ніж комбінація Sigmoid та BCELoss окремо.

2.3 Проектування моделі TCN для послідовних даних

Друга модель, Часова Згорткова Мережа (TCN), проектувалася для вирішення задачі з іншої, принципово відмінної перспективи. Її мета — слугувати базовою моделлю для прогнозування, спираючись *виключно* на "сирі" часові послідовності, а також виступати як "послідовна вежа" (TCN-Tower) у фінальній гібридній архітектурі.

Ця модель, на відміну від MLP, приймає на вхід єдиний тензор xseq розмірністю (Batch, 21, 12). Цей тензор представляє історію активності студента, підготовлену на етапі 3.1. А саме:

– 1 канал: Це 20 типів активності в VLE (напр., forumng, oucontent, resource) плюс один додатковий бінарний канал-"маска", що позначає, чи була у студента хоч якась активність на даному тижні;

– 12 часових кроків: Це 12 тижнів семестру до точки зрізу (CUT_OFF_WEEK).

Архітектура ModelTCN (і, відповідно, TCN_Tower) базується на архітектурі, запропонованій Бай та ін. та використовує дві ключові концепції, що відрізняють її від звичайних згорток [8].

Перша концепція — каузальні згортки (Causal Convolutions). Це згортки, які гарантують, що обчислення для часового кроку t може залежати лише від даних з минулого (t,t-1,t-2,…) (і ніколи не "підглядає" у майбутнє (t+1). Це є фундаментальною вимогою для прогнозування часових рядів. У програмній реалізації (модуль CausalConv1d) це досягається шляхом додавання асиметричного паддінгу (F.pad) до початку послідовності перед застосуванням згортки.

Друга концепція — розширені згортки (Dilated Convolutions). Щоб охопити довгу історію (всі 12 тижнів), не роблячи мережу надто глибокою, використовуються згортки з експоненціально зростаючим "розширенням" (dilation rate: d=1,2,4,8,…). Це дозволяє фільтрам "перестрибувати" через входи, ефективно збільшуючи своє "рецептивне поле" (receptive field) — тобто, область вхідної послідовності, яка впливає на одне вихідне значення.

На рисунку 2.2 показано, як ця комбінація дозволяє виходу на останньому шарі залежати від усіх входів у послідовності.

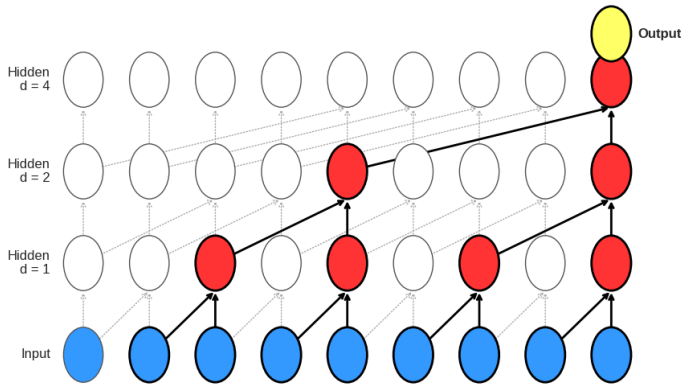

Рисунок 2.2 – Концептуальна схема стеку каузальних та розширених згорток

Сама TCN_Tower побудована не просто зі згорток, а зі спеціалізованих залишкових блоків (TCN_Block). Це необхідно для стабільного навчання глибоких згорткових мереж, оскільки дозволяє градієнтам безперешкодно проходити крізь мережу. Кожен такий блок (TCN_Block) складається з:

– двох шарів CausalConv1d з однаковим розширенням d та розміром ядра k=3;

– після кожної згортки послідовно застосовується nn.BatchNorm1d (для стабілізації), функція активації nn.GELU (більш плавна альтернатива ReLU) та nn.Dropout (з гіперпараметром p=0.1);

– залишкового з'єднання (skip connection), де вхід блоку x додається до виходу h. Якщо кількість каналів на вході та виході блоку не збігається, вхід x пропускається через додаткову згортку nn.Conv1d з ядром 1x1 (self.skip), щоб вирівняти розмірності.

Фінальна архітектура TCN_Tower — це послідовний стек з чотирьох таких TCN_Block, що збільшують як розширення, так і кількість каналів (фільтрів). Далі наведено структуру:

– блок 1: TCN_Block(in_ch=21, out_ch=64, k=3, d=1, drop=0.1);

– блок 2: TCN_Block(in_ch=64, out_ch=64, k=3, d=2, drop=0.1);

– блок 3: TCN_Block(in_ch=64, out_ch=128, k=3, d=4, drop=0.1);

– блок 4: TCN_Block(in_ch=128, out_ch=128, k=3, d=8, drop=0.1).

Після того, як послідовність проходить через усі 4 блоки, ми отримуємо тензор активацій розміром (Batch, 128, 12). Щоб агрегувати всю часову інформацію в єдиний вектор, застосовується шар nn.AdaptiveAvgPool1d(1). Він бере середнє значення по всій осі часу (12 тижнів), створюючи фінальний вектор (Batch, 128).

Цей 128-вимірний зліпок подається на додатковий повнозв'язний шар nn.Linear(128, 128) з активацією nn.ReLU для фінальної обробки. Отриманий вектор є вихідним представленням TCN_Tower і, аналогічно до MLP, подається на ту саму "голову" (Head), що містить nn.Linear(128, 1), для отримання фінального логіту.

2.4 Проектування гібридної мультимодальної моделі

Гібридна модель є кульмінацією даного проектного дослідження. Її архітектура (ModelHybrid) була розроблена з метою перевірки центральної гіпотези роботи: чи призведе одночасне використання статичних/агрегованих даних (з підпункту 2.1) та динамічних часових послідовностей (з підпункту 2.2) до більш точного та стійкого прогнозу, ніж використання будь-якої з цих модальностей окремо.

Цей підхід, відомий у літературі як мультимодальне навчання або "fusion", ґрунтується на припущенні, що два типи даних містять доповнюючу (complementary) інформацію [15]. У контексті даної задачі:

– "таблична" модальність (MLP) надає моделі сильний, надійний базовий сигнал. Вона характеризує академічну успішність (агреговані оцінки, відсоток зданих робіт) та статичний профіль (демографія, попередня освіта). Вона відповідає на питання, який поточний академічний статус та початковий профіль ризику студента;

– "послідовна" модальність (TCN) надає моделі динамічний поведінковий контекст, який неможливо вловити за допомогою простого усереднення. Вона характеризує патерни активності у VLE з плином часу і відповідає на питання, яка траєкторія залученості студента та чи є ознаки прокрастинації, вигорання або, навпаки, стабільної роботи.

Для ефективного поєднання цих двох різнорідних джерел даних була спроектована архітектура ModelHybrid, що базується на концепції "двох веж" (Two-Tower Model). Ця архітектура складається з трьох логічних етапів: паралельне кодування, злиття та класифікація. Розглянуті етапи:

а) етап 1: Паралельне кодування модальностей. На відміну від ModelMLP та ModelTCN, гібридна модель приймає на вхід всі доступні підготовлені дані: числові (xnum), категоріальні (xcat) та послідовні (xseq). На першому етапі кожна модальність обробляється паралельно та незалежно власним "кодером" (вежею), який є екземпляром класу, описаного раніше:

1) "вежа" MLP (self.mlp = MLP_Tower(...)):

вхід: xnum (тензор числових ознак) та xcat (тензор категоріальних ознак);

процес: Дані проходять через шари nn.Embedding та три блоки Linear-BatchNorm-ReLU-Dropout, як детально описано в підпункті 2.1;

вихід: 128-вимірний векторний зліпок z_m (MLP embedding).

Цей вектор є високо-рівневим представленням статичного та академічного профілю студента.

2) "Вежа" TCN (self.tcn = TCN_Tower(...)):

вхід: xseq (тензор послідовності (Batch, 21, 12));

процес: Послідовність проходить через стек із чотирьох TCN_Block з каузальними та розширеними згортками (d=1, 2, 4, 8), як детально описано в підпункті 2.2;

вихід: 128-вимірний векторний зліпок z_t (TCN embedding).

Цей вектор є високо-рівневим представленням поведінкової траєкторії студента.

б) етап 2: Механізм злиття (Fusion Mechanism). Це найважливіша частина гібридної архітектури. Замість простого усереднення або конкатенації зліпків z_m та z_t, було спроектовано більш складний механізм злиття. Його мета — не просто "склеїти" ознаки, а вивчити складні нелінійні взаємозв'язки між ними та динамічно зважити важливість кожної модальності. Цей механізм складається з трьох кроків:

1) конкатенація. Два 128-вимірні вектори z_m та z_t конкатенуються вздовж осі ознак, створюючи єдиний об'єднаний вектор z розмірністю 256. z = torch.cat([zm, zt], dim=1). Цей вектор тепер містить повний набір високо-рівневих ознак з обох "веж", розташованих пліч-о-пліч.

2) Squeeze-and-Excitation (SE Block). До цього 256-вимірного вектора застосовується механізм, подібний до Squeeze-and-Excitation (SE) [16], який є формою канальної уваги (channel attention). Цей блок (self.se) призначений для динамічного зважування важливості кожної з 256

об'єднаних ознак. Інтуїтивно, модель вчиться "звертати більше уваги" на певні ознаки залежно від контексту. Наприклад, якщо TCN-вежа сигналізує про "нульову активність" (сильна ознака z_t), а MLP-вежа показує "високі бали" (сильна ознака z_m), цей блок може навчитися посилювати обидва ці "конфліктуючі" сигнали для прийняття рішення.

Його архітектура (nn.Sequential):

– nn.Linear(256, 64) (Squeeze): Вектор z стискається до 64-вимірного "контекстного" вектора, що узагальнює глобальну інформацію;

– 2nn.ReLU(): Нелінійність;

– nn.Linear(64, 256) (Excitation): Контекстний вектор розширюється назад до 256;

– nn.Sigmoid(): Отримані значення пропускаються через сигмоїду, що генерує 256 "ваг" (значень від 0 до 1).

Ці ваги потім поелементно перемножуються з оригінальним вектором z: z = z * self.se(z). Таким чином, ознаки, визнані моделлю "неважливими" (вага ≈ 0), "приглушуються", а важливі (вага ≈ 1) — проходять далі.

3): Fusion MLP (Блок Взаємодії). Отриманий 256-вимірний відкалібрований вектор ознак z подається на фінальний блок злиття (self.fuse). На відміну від SE-блоку, який лише зважував ознаки, цей блок є двошаровим MLP, призначеним для вивчення вже складних нелінійних взаємодій між відібраними ознаками з обох модальностей даних.

Його архітектура (nn.Sequential):

– nn.Linear(256, 128) → nn.ReLU();

– nn.Linear(128, 128) → nn.ReLU().

На виході цього блоку ми отримуємо фінальний 128-вимірний мультимодальний зліпок. Це остаточне представлення студента, яке увібрало в себе інформацію з обох джерел, пропустивши її через зважування (SE) та нелінійну взаємодію (Fuse).

в) етап 3: Фінальна класифікація. Отриманий 128-вимірний мультимодальний зліпок подається на ту саму "голову" (Head), що

використовується в ModelMLP та ModelTCN. Вона складається з одного шару nn.Linear(128, 1), який генерує фінальний логіт для прогнозування.

Таким чином, спроектована гібридна модель є складною архітектурою, що реалізує підхід "Two-Tower" з просунутим механізмом злиття. Вона не просто комбінує, а глибоко інтегрує статичні та динамічні дані, дозволяючи моделі самій визначати важливість кожної модальності та вивчати їх взаємозв'язки для отримання максимально точного прогнозу.

2.5 Висновки до розділу 2

У даному розділі було виконано детальне проектування трьох нейромережевих архітектур, кожна з яких реалізує один із ключових підходів до прогнозування успішності студентів, проаналізованих у Розділі 1.

Модель MLP (ModelMLP) була спроектована як реалізація "класичного" підходу на основі інженерії ознак. Її архітектура (MLP_Tower) оптимізована для роботи з гетерогенними табличними даними, ефективно поєднуючи числові ознаки та категоріальні занурення (Embeddings) за допомогою стандартних блоків Linear-BatchNorm-Dropout [13]-[14].

Модель TCN (ModelTCN) була спроектована як представник "end-to-end" підходу на основі послідовних даних. Її архітектура (TCN_Tower) є глибокою згортковою мережею, що використовує спеціалізовані каузальні та розширені згортки (dilation rate 1, 2, 4, 8) та залишкові блоки (TCN_Block). Це дозволяє їй аналізувати "сирі" часові ряди активності VLE, зберігаючи їхню динаміку.

Модель Hybrid (ModelHybrid) є найбільш складною спроектованою системою, що реалізує мультимодальний "Two-Tower" підхід. Вона об'єднує MLP_Tower та TCN_Tower як паралельні "кодери" для одночасної обробки обох типів даних. Ключовим проектним рішенням є просунутий механізм злиття (fusion), який не просто конкатенує ознаки, а застосовує механізм

уваги (Squeeze-and-Excitation) для їх зважування та додатковий MLP (self.fuse) для вивчення їх нелінійних взаємодій.

Всі три моделі спроектовані за модульним принципом із застосуванням ідентичного блоку («голови») для класифікації, що забезпечує справедливі умови для їх порівняння. Таким чином, у цьому розділі було представлено "креслення" для трьох конкуруючих моделей.

3 МАТЕРІАЛИ, МЕТОДИ ТА РЕЗУЛЬТАТИ ЕКСПЕРИМЕНТАЛЬНИХ ДОСЛІДЖЕНЬ

3.1 Підготовка та аналіз даних

Програмна реалізація починається з найбільш критичного етапу — підготовки даних. Для дослідження було обрано Open University Learning Analytics Dataset (OULAD), який є публічним, широко визнаним у науковій спільноті та ідеально підходить для цілей даної роботи [17]. Його ключова перевага полягає в мультимодальності: він містить не лише статичні демографічні дані та результати оцінювання, але й деталізовані щоденні логи взаємодії студентів з віртуальним навчальним середовищем (VLE). Це дозволяє реалізувати та порівняти всі три спроектовані підходи (MLP, TCN та гібридний).

Датасет складається з кількох CSV-файлів, що пов'язані між собою ключами (напр., id_student, code_module, code_presentation). Для даного дослідження було використано наступні таблиці:

– studentInfo.csv: Містить демографічну інформацію про студента та його фінальний результат курсу;

– assessments.csv: Описує кожне оцінюване завдання на курсі, його тип та "вагу";

– studentAssessment.csv: Містить результати (оцінки) кожного студента за кожне завдання;

– vle.csv: Довідник активностей, доступних у VLE;

– studentVle.csv: Деталізований "clickstream" — лог-файл, що фіксує кількість кліків кожного студента на кожному ресурсі VLE щодня.

Визначення цільової змінної (Y): Першим кроком є визначення цільової змінної. У вихідній таблиці studentInfo фінальний результат (final_result) має чотири категорії: 'Pass' (Склав), 'Fail' (Не склав), 'Distinction' (Відмінно) та 'Withdrawn' (Покинув курс). Для задачі розробки системи раннього попередження, ці чотири класи були бінаризовані.

Було введено єдину цільову мітку label за наступним правилом:

– успіх (Мітка 1): Присвоюється студентам, які завершили курс з результатом 'Pass' або 'Distinction';

– невдача (Мітка 0): Присвоюється студентам, які 'Fail' або 'Withdrawn'.

Таке групування є логічним, оскільки 'Withdrawn' (покидання курсу) є такою ж формою академічної невдачі, як і 'Fail'. Аналіз розподілу класів (як показано в логах компіляції) виявив значний дисбаланс: на 15385 успішних студентів (Pass + Distinction) припадає 17208 випадків невдач (Fail + Withdrawn). Ця диспропорція була врахована на етапі навчання моделі.

Стратегія валідації: Часовий спліт – для задачі прогнозування успішності студентів використання випадкового перемішування (train_test_split) є методологічно некоректним. Це призводить до "витоку даних з майбутнього", коли модель навчається на прикладах студентів з 2014 року, а потім тестується на студентах з 2013 року, що неможливо в реальному світі.

Тому була застосована сувора часова (time-based) стратегія валідації. Всі дані були розділені на три набори на основі семестру (code_presentation):

– тренувальний набір (Train): Включає всі дані за семестри 2013B та 2013J;

– валідаційний набір (Validation): Включає всі дані за семестр 2014B;

– тестовий набір (Test): Включає всі дані за семестр 2014J.

Такий підхід імітує реальний сценарій: модель навчається на історичних даних (2013 рік), її гіперпараметри та поріг відсічення налаштовуються на "свіжих", але вже завершених даних (2014B), і, нарешті, її фінальна якість оцінюється на абсолютно новій когорті студентів, яку вона ніколи не бачила (2014J).

Визначення точки зрізу (Cut-off): Оскільки мета роботи — раннє прогнозування, було критично важливо визначити момент часу, "стоячи" на якому модель робить прогноз. Для цього було введено гіперпараметр

CUT_OFF_WEEK = 12. Це означає, що для прогнозування фінального результату студента модель може використовувати лише ту інформацію (оцінки, кліки), яка стала доступна до кінця 12-го тижня семестру (тобто до 84-го дня). Вся інформація, що з'явилася пізніше, була відфільтрована з набору ознак (але не з цільової змінної, звісно).

Підготовка даних для MLP (Інженерія ознак): Для моделі MLP (описаної в 2.1) необхідно було перетворити "сирі" дані на єдиний "плаский" вектор ознак для кожного студента. Цей процес складався з двох частин:

– статичні ознаки: З таблиці studentInfo було взято 6 категоріальних (cat_cols) та 2 числові (num_cols) ознаки. Категоріальні (gender, region, highest_education, imd_band, age_band, disability) та числові (studied_credits, num_of_prev_attempts) були очищені від пропусків (заповнені значеннями "Unknown" або 0 відповідно).

– агреговані динамічні ознаки: Це найважливіший крок інженерії ознак. "Сирі" дані про оцінки (studentAssessment) та їх властивості (assessments) були агреговані в потужні прогнозні індикатори за допомогою функції agg_assess.

Процес агрегованих динамічних ознак включав:

– об'єднання таблиць studentAssessment та assessments;

– критичне фільтрування: відбір лише тих записів, де дедлайн завдання (date) був раніше за CUT_OFF_DAY;

– групування даних за кожним студентом;

– розрахунок агрегатів.

До розрахунку агрегатів належать:

– assess_n_total: Загальна кількість завдань, що мали бути здані до 12-го тижня.

– assess_submitted: Кількість фактично зданих завдань.

– assess_frac_sub: Частка зданих робіт (submitted / total).

– assess_weighted_avg: Середньозважений бал. Розраховується як (score * weight) / sum(weight). Важливо, що пропущені роботи (score=NaN) враховувалися як 0, що реалістично "штрафувало" студента за невиконання.

– assess_late_cnt: Кількість робіт, зданих із запізненням.

– assess_mean_late: Середня кількість днів запізнення.

– assess_max/min: Максимальний та мінімальний отриманий бал.

Ці агрегати, разом зі статичними даними, сформували 11 числових та 6 категоріальних ознак, що стали входом для MLP_Tower.

Підготовка даних для TCN (Формування послідовностей): Для моделі TCN (описаної в 2.2) потрібен був зовсім інший підхід. Метою було перетворити "сирий" clickstream (studentVle) у тривимірний тензор (N_students, N_features, N_timesteps). Етапи для TCN:

– фільтрація: Так само, як і для MLP, були відібрані лише ті кліки, що сталися до CUT_OFF_DAY;

– дискретизація часу: Безперервний час (стовпець date) був дискретизований шляхом цілочисельного ділення на 7. Це перетворило щоденні логи у 12-тижневі "кошики" (timesteps), що відповідає CUT_OFF_WEEK = 12;

– визначення ознак (каналів): Було виділено 20 унікальних типів активності (activity_type) з таблиці vle (напр., forumng, oucontent, resource);

– створення матриці: Було створено нульовий 3D-тензор X_dense розмірністю (Кількість студентів, 20 типів активності, 12 тижнів);

– заповнення матриці: Використовуючи високоефективну функцію numpy.add.at, сума кліків (sum_click) кожного студента була атомарно додана у відповідну комірку (id_student, act_idx, week);

– логарифмічне перетворення: Дані про кліки мають сильний "правий хвіст" (багато нулів і поодинокі екстремальні значення). Для стабілізації даних та зменшення впливу викидів було застосовано логарифмічне перетворення np.log1p (логарифм від 1+x);

– додавання маски: До 20 каналів активності було додано 21-й канал – бінарну "маску", що позначає, чи була у студента хоч якась активність на даному тижні. Це дозволяє моделі розрізняти "нуль кліків, тому що студент не заходив" і "нуль кліків, тому що активність цього типу була відсутня". У результаті було отримано фінальний тензор X_seq розмірністю (N_students, 21, 12), що став входом для TCN_Tower;

– фіналізація: Уникнення витоку даних при масштабуванні. На останньому етапі всі три матриці (Xnum_all, Xcat_all, Xseq_all) та вектор y_all були розділені на тренувальний, валідаційний та тестовий набори відповідно до часових масок, визначених у 3.1.2. Розмірності отриманих наборів (напр., MLP num: (13529, 11)) точно відповідають логам компіляції.

Для запобігання витоку даних (data leakage) під час нормалізації та кодування було вжито ключових заходів:

– масштабування StandardScaler: Об'єкт scaler був навчений (.fit()) виключно на тренувальних числових даних (Xnum_tr). Після цього той самий навчений scaler (з середнім та дисперсією тренувального набору) був застосований (.transform()) до валідаційного та тестового наборів;

– кодування категорій: Словники (cat_maps) для 6 категоріальних ознак були створені виключно на унікальних значеннях з тренувального набору. Будь-які нові категорії, що раптово з'явилися у валідаційному чи тестовому наборі, були прирівняні до "невідомого" токена (<UNK>).

Ці кроки гарантують, що жодна інформація зі статистики валідаційного чи тестового наборів не "просочилася" у процес навчання, що робить фінальну оцінку на тестовому наборі об'єктивною. Нарешті, отримані NumPy-масиви були загорнуті у кастомний Dataset та DataLoader фреймворку PyTorch для подачі у моделі під час навчання.

3.2 Програмна реалізація та процес навчання

Обґрунтування вибору мови програмування: Для реалізації програмного забезпечення прогнозування успішності студентів було обрано мову програмування Python. Цей вибір зумовлений сукупністю факторів, що роблять Python стандартом у галузі науки про дані (Data Science) та машинного навчання. До причин вибору входять:

– екосистема бібліотек: Python надає доступ до найпотужніших інструментів для обробки даних. Бібліотека Pandas забезпечує ефективну роботу з табличними даними (DataFrames), що є критично важливим для етапу інженерії ознак. Бібліотека NumPy дозволяє виконувати високопродуктивні векторні та матричні обчислення, які лежать в основі роботи нейронних мереж;

– підтримка глибокого навчання: Саме Python є основною мовою для провідних фреймворків глибокого навчання, таких як PyTorch та TensorFlow. Використання PyTorch у даній роботі дозволило спроектувати нестандартні архітектури (гібридну модель та TCN) завдяки гнучкому механізму динамічних обчислювальних графів [18];

– продуктивність: Хоча Python є інтерпретованою мовою, основні обчислювальні операції у використовуваних бібліотеках реалізовані на низькорівневих мовах C/C++ та оптимізовані для паралельних обчислень. Це дозволяє поєднати зручність високорівневого синтаксису з продуктивністю компільованого коду;

–спільнота та документація: Велика спільнота розробників та наявність вичерпної документації значно прискорюють процес вирішення технічних проблем, що виникають у ході виконання досліджень.

Альтернативні варіанти, такі як мова R (орієнтована переважно на статистичний аналіз, а не на Deep Learning) або C++ (яка потребує значно більших часових витрат на розробку прототипів), були відхилені як менш ефективні для досягнення мети даної роботи [19]-[21].

Обґрунтування вибору середовища розробки: В якості інтегрованого середовища розробки (IDE) було обрано Visual Studio Code (VS Code). Цей інструмент від Microsoft поєднує в собі легковажність текстового редактора з потужним функціоналом повноцінної IDE.

Ключові переваги VS Code для даного дослідження:

– інтеграція з Jupyter Notebooks: VS Code має вбудовану підтримку інтерактивних блокнотів (.ipynb). Це дозволяє проводити розвідувальний аналіз даних (EDA), будувати графіки та тестувати окремі фрагменти коду в одному інтерфейсі з основним модульним кодом (.py).

– віддалена розробка (Remote - SSH): Оскільки навчання нейронних мереж є ресурсомістким процесом, VS Code дозволяє писати код на локальному комп'ютері, а виконувати його на віддаленому сервері з потужним GPU через SSH-тунель, що забезпечує зручність роботи.

– налагодження (Debugging): Вбудований дебагер дозволяє покроково перевіряти стан тензорів та змінних під час виконання скрипту, що є критично важливим для пошуку помилок у складних архітектурах нейромереж.

– розширюваність: Завдяки розширенню "Python" середовище автоматично підтримує віртуальні оточення (venv/conda), форматування коду (black/flake8) та автодоповнення (IntelliSense), що підвищує якість коду.

– порівняно з PyCharm (який є більш вимогливим до ресурсів системи) та Jupyter Lab (який менш зручний для написання модульного коду та великих проектів), VS Code забезпечує оптимальний баланс функціональності.

Архітектура програмного модуля та процес навчання: Середовище виконання та відтворюваність Для прискорення обчислювальних процесів, зокрема операцій згортки в моделі TCN, навчання проводилося з використанням апаратного прискорення на графічному процесорі (GPU) на базі архітектури NVIDIA CUDA. У програмному коді це реалізовано через

ініціалізацію глобального об'єкта DEVICE, який автоматично визначає наявність сумісного обладнання.

Для забезпечення повної відтворюваності результатів дослідження, що є обов'язковою вимогою наукової роботи, на початку виконання алгоритму виконується фіксація генераторів псевдовипадкових чисел («зерна» або seed). Встановлення константи SEED = 42 гарантує ідентичну ініціалізацію вагових коефіцієнтів нейронних мереж та порядок перемішування даних при кожному запуску програми.

Конвеєр подачі даних (Data Pipeline): для ефективного завантаження підготовлених масивів у модель розроблено кастомний клас OuladDS, що наслідує базовий клас torch.utils.data.Dataset. Цей компонент відповідає за конвертацію даних у формат тензорів PyTorch безпосередньо перед подачею в мережу. На основі цього класу створено об'єкти DataLoader, які забезпечують:

– пакетування (Batching): Формування пакетів даних (batch size = 512) для стабілізації градієнтів та завантаження GPU;

– рандомізацію (Shuffling): Перемішування тренувальних даних на початку кожної епохи для покращення узагальнюючої здатності моделі;

– паралелізм: Асинхронне підвантаження даних (num_workers=2) для усунення простоїв обчислювальних потужностей.

Реалізація процедури навчання: Ключовим елементом програмного комплексу є уніфікована функція train_model, що реалізує повний цикл навчання та валідації моделей. Процес побудовано з урахуванням сучасних практик глибокого навчання:

– оптимізація та планування: Використано адаптивний оптимізатор Adam, який забезпечує швидку збіжність завдяки індивідуальному налаштуванню кроку навчання для кожного параметра [22]. Додатково застосовано планувальник CosineAnnealingLR для плавного зменшення швидкості навчання за косинусоїдальним законом, що дозволяє точніше знайти локальний мінімум функції втрат;

– балансування класів: Для вирішення проблеми незбалансованості вибірки застосовано метод зважування функції втрат BCEWithLogitsLoss. Параметр pos_weight (розрахований як 1.059) збільшує штраф за помилки на менш представленому позитивному класі, змушуючи модель приділяти йому належну увагу;

– етапи обчислень: Кожна епоха чітко розділена на фазу навчання (model.train()), де активні шари Dropout та оновлюються ваги, та фазу оцінювання (model.eval()), де градієнти не обчислюються [23].

Механізм вибору найкращої моделі: замість фіксованої кількості ітерацій, реалізовано стратегію ранньої зупинки (Early Stopping). Система моніторить метрику PR-AUC (площа під кривою точність-повнота) на валідаційному наборі даних. Обробка навчання:

– якщо метрика покращується, стан моделі (ваги) зберігається в оперативну пам'ять;

– якщо покращення відсутнє протягом заданої кількості епох (patience), процес навчання примусово зупиняється. Це дозволяє отримати модель на піку її продуктивності та запобігти ефекту перенавчання.

Пост-обробка прогнозів: Отримані «сирі» виходи моделі (логіти) проходять два етапи оптимізації на валідаційних даних:

– калібрування ймовірностей: Застосування методу температурного масштабування (Temperature Scaling) для узгодження прогнозованих ймовірностей з реальною частотою подій. Параметр температури T оптимізується алгоритмом L-BFGS.

– оптимізація порогу: Автоматичний підбір порогу класифікації, який максимізує метрику F1-macro, що є критично важливим для незбалансованих даних.

Таким чином, розроблений програмний модуль є комплексним рішенням, що охоплює всі етапи від завантаження даних до отримання фінальних каліброваних прогнозів.

3.3 Аналіз та порівняння експериментальних результатів

Після успішного завершення процесів підготовки даних (3.1) та навчання моделей (3.2), було проведено глибокий, багатоаспектний аналіз отриманих результатів. Цей аналіз має на меті не лише формально порівняти фінальні метрики, але й глибоко зрозуміти поведінку кожної з трьох спроектованих моделей (MLP, TCN, Hybrid), діагностувати їхні сильні та слабкі сторони та валідувати зроблені проектні рішення.

Аналіз проводився у чотири послідовні етапи:

– аналіз динаміки навчання: Оцінка того, як моделі навчалися, чи не перенавчалися вони, та як швидко збігалися до оптимального рішення;

– аналіз фінальних метрик: Ключове порівняння точності, повноти та загальної ефективності моделей на "сліпому" тестовому наборі;

– глибокий аналіз якості прогнозу: Дослідження калібрування моделей та характеру їхніх помилок для розуміння, *чому* вони працюють (або не працюють);

– аналіз стабільності: Перевірка надійності отриманих висновків за допомогою крос-валідації, щоб виключити фактор випадковості;

– аналіз динаміки навчання. Первинний аналіз логів тренувального процесу дозволяє оцінити, наскільки ефективно кожна модель засвоювала інформацію з тренувального набору (2013B, 2013J) та узагальнювала її на валідаційному (2014B).

Логи тренувального процесу:

– ModelMLP: Процес навчання був швидким та ефективним. Модель продемонструвала швидке зростання якості, досягнувши свого найкращого результату на валідаційному наборі PR-AUC = 0.8454 вже на 7-й епосі. Після цього її продуктивність стабілізувалася на високому плато. Механізм EarlyStopping (з patience=8) коректно зупинив навчання на 15-й епосі, запобігши марному витрачанню ресурсів та потенційному перенавчанню. Це

свідчить про те, що інженерні ознаки, подані в MLP, містили сильний та чіткий сигнал;

– ModelTCN: Логи цієї моделі демонструють повну відсутність навчання. Найкращий результат PR-AUC = 0.4303 було досягнуто на 1-й епосі, після чого метрика не покращувалася жодного разу протягом 9 епох. Цей показник є еквівалентом випадкового вгадування для даної задачі. Це свідчить про те, що оптимізатор Adam не зміг знайти жодного корисного градієнта для мінімізації помилки, а "сирі" дані VLE у даному форматі не містили явного прогнозного сигналу;

– ModelHybrid: Динаміка навчання гібридної моделі виявилася дуже схожою на чисту MLP. Найкращий результат PR-AUC = 0.8433 було досягнуто на 5-й епосі, що також свідчить про дуже швидку збіжність. Це є першим індикатором того, що гібридна модель, ймовірно, навчилася здебільшого покладатися на свою MLP-гілку.

Графік зниження помилки на валідації: Візуалізація процесу навчання на рисунку 3.1 надає наочне підтвердження висновків з логів. Тут в якості метрики помилки використано Brier score loss, який одночасно оцінює і точність, і калібрування.

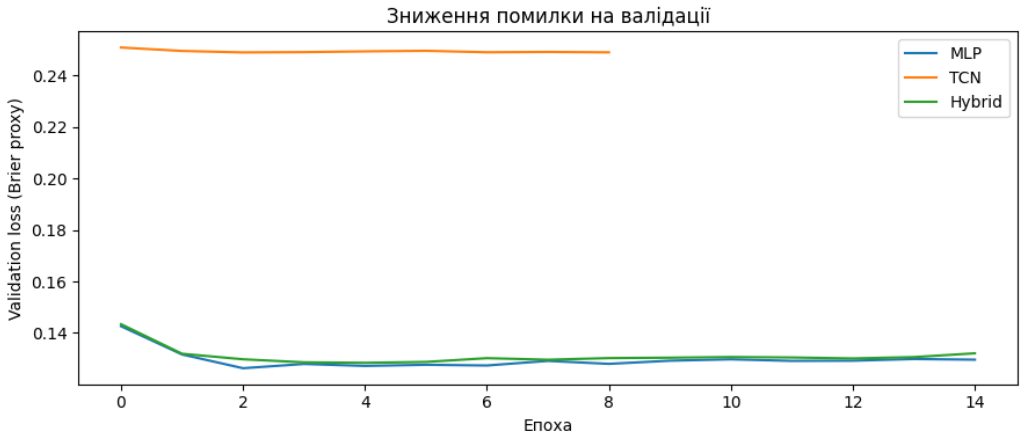

Рисунок 3.1 – Зниження помилки на валідації для моделей MLP, TCN та Hybrid

Цей графік чітко ілюструє три різні сценарії:

– крива TCN (помаранчева) є практично ідеально пласкою і знаходиться на аномально високому рівні помилки (Brier ≈ 0.25, що для бінарної задачі є еквівалентом повного провалу). Це візуальний доказ того, що модель не навчається;

– криві MLP (синя) та Hybrid (зелена) стартують з значно нижчої помилки (Brier ≈ 0.14) і миттєво збігаються до стабільного плато (Brief ≈ 0.13). Важливо, що криві не починають зростати, що свідчить про ефективну регуляризацію (Dropout, BatchNorm) та коректну роботу механізму EarlyStopping, який зупинив процес до початку перенавчання;

– аналіз фінальних метрик на тестовому наборі. Після того, як найкращі версії кожної моделі були відібрані (за val_PR-AUC), калібровані (fit_temperature) та оптимізовані за порогом (tune_threshold) на валідаційному наборі, була проведена їхня фінальна, "сліпа" оцінка на тестовому наборі (2014J). Результати представлено в таблиці 3.1.

Таблиця 3.1 – Підсумкові метрики моделей на тестовому наборі

Model	PR_AUC	ROC_AUC	F1_macro	MCC	Brier	ECE	thr_used
Hybrid	0.8317	0.8671	0.7797	0.5635	0.1542	0.0564	0.55
MLP	0.8295	0.8651	0.7771	0.5570	0.1546	0.0472	0.55
TCN	0.5011	0.5152	0.3264	0.0000	0.2498	0.0166	0.55

Аналіз цієї таблиці дає ключові відповіді на питання дослідження:

– провал TCN: Модель TCN, що працювала на "сирих" послідовностях, показала повну неспроможність. Її ROC_AUC = 0.5152 (де 0.5 — випадкове вгадування) та MCC = 0.0000 (де 0 — повна відсутність кореляції) однозначно вказують на те, що модель не має жодної прогнозної сили;

– успіх MLP та Hybrid: Обидві моделі, що використовували інженерні ознаки (табличні дані), показали високі та практично значущі результати (напр., ROC_AUC > 0.86, F1-macro > 0.77);

– порівняння MLP та Hybrid: Гібридна модель (Hybrid) показала мінімальну, статистично незначущу перевагу над чистою MLP за всіма основними метриками (наприклад, PR_AUC 0.8317 проти 0.8295). Це свідчить про те, що додавання "сирих" послідовних даних до гібриду не дало значного покращення, хоча й дещо покращило результат;

– калібрування: Цікаво, що чиста MLP виявилася краще каліброваною (нижчий ECE — Expected Calibration Error, 0.0472 проти 0.0564). Це означає, що її прогнози-ймовірності є більш "чесними" та надійними. Аномально низький ECE у TCN (0.0166) є артефактом — оскільки модель завжди прогнозує ≈ 0.5, вона не є "надмірно впевненою", що формально робить її "каліброваною", але водночас абсолютно марною;

– поріг: Оптимальний поріг для обох успішних моделей (thr_used) склав 0.55, що близько до стандарту, але його оптимізація на валідаційному наборі була важливим кроком;

Діаграма на рисунку 3.2 візуально підкреслює висновки: домінування MLP (синій) та Hybrid (зелений) над TCN (помаранчевий) за всіма ключовими показниками (PR_AUC, F1, MCC, ROC_AUC).

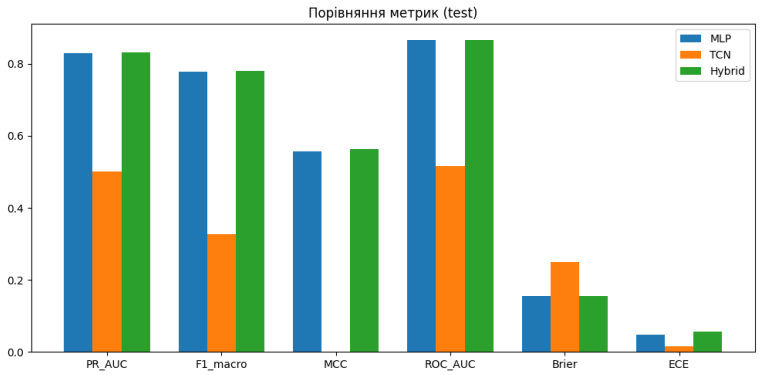

Рисунок 3.2 – Порівняльна діаграма метрик моделей на тестовому наборі

Глибокий аналіз якості моделей та помилок. Самі по собі метрики не пояснюють, чому TCN провалилася, а MLP та Hybrid — ні. Для цього було проведено аналіз калібрувальних кривих та розподілу помилок.

Діагностика провалу TCN:

Графіки для моделі TCN дають вичерпне пояснення її результатів.

На рисунку 3.3 синя лінія (Predicted P(y=1)) є майже ідеально горизонтальною на рівні p=0.5. Це означає, що модель TCN не змогла диференціювати студентів. Вона видає однаковий прогноз ("шанси 50/50") для всіх студентів, незалежно від їхньої реальної успішності (помаранчева лінія Actual).

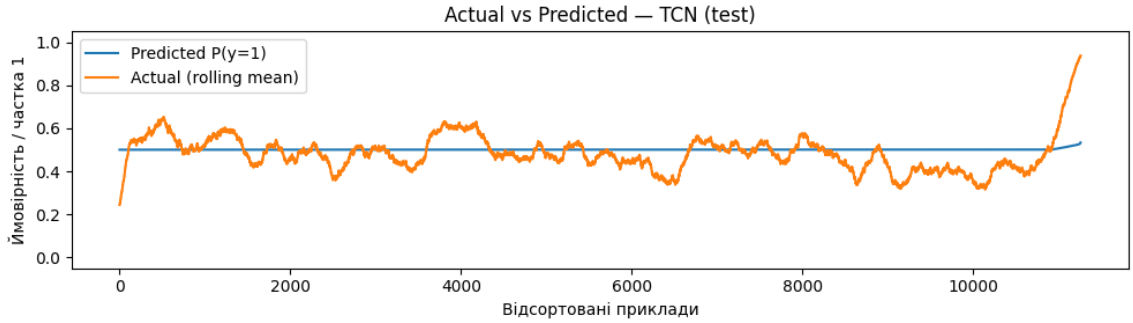

Рисунок 3.3 – Калібрувальний графік (Actual vs Predicted) для TCN на тестовому наборі

Рисунок 3.4 підтверджує це. Оскільки модель завжди прогнозує p ≈ 0.5:

– для студентів з невдачею (y=0), помилка e = 0.5 – 0 = +0.5

– для успішних студентів (y=1), помилка e = 0.5 – 1 = -0.5

Графік показує лише ці два стовпчики, що є візуалізацією повної відсутності прогнозу.

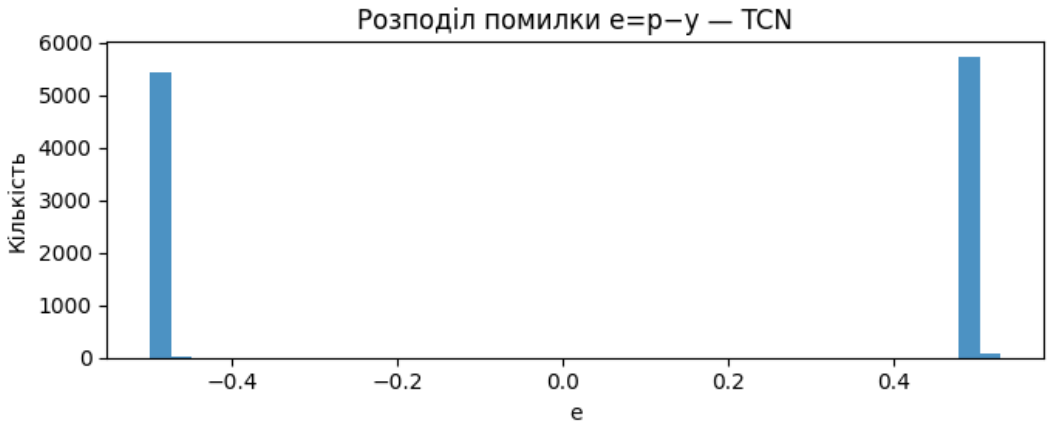

Рисунок 3.4 – Розподіл помилки e = p - y для TCN

Рисунок 3.5 є фінальним доказом. Графік практично порожній, оскільки модель майже ніколи не робить прогнозів, що сильно відхиляються від 0.5, і, таким чином, робить дуже мало помилок FP/FN (за порогом 0.55), але при цьому має MCC = 0.

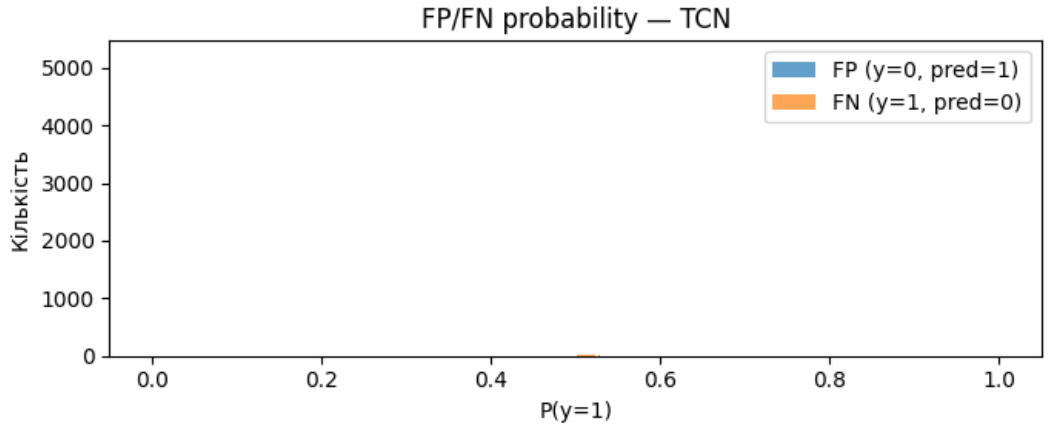

Рисунок 3.5 – Розподіл ймовірностей для помилок FP/FN (TCN)

Висновок: "сирі" дані про кліки у форматі (21, 12) виявилися надто "шумними", розрідженими або неінформативними для TCN.

Діагностика успіху MLP та Hybrid:

На противагу TCN, моделі MLP та Hybrid демонструють чудову якість прогнозування.

На рисунках 3.6 та 3.7 синя лінія (прогноз) дуже близько слідує за помаранчевою (реальна частка успішних студентів). Це свідчить про високу якість калібрування (завдяки TemperatureScaler): коли модель прогнозує p=0.8, частка успішних студентів у цій групі дійсно становить близько 80%.

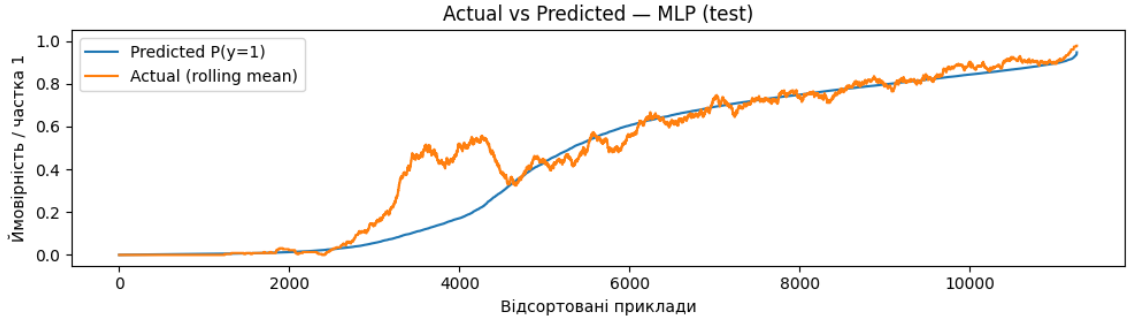

Рисунок 3.6 – Калібрувальний графік (Actual vs Predicted) для MLP на тестовому наборі

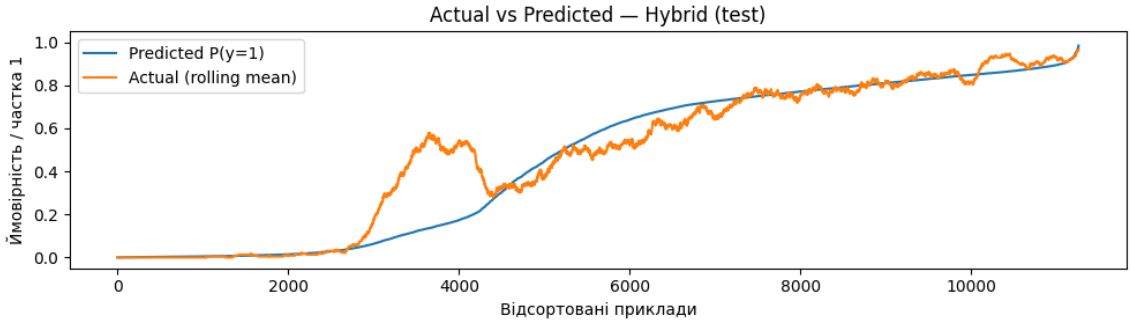

Рисунок 3.7 – Калібрувальний графік (Actual vs Predicted) для Hybrid на тестовому наборі

Спільний калібрувальний графік (рисунок 3.8) ще раз ілюструє різке протиставлення: лінії MLP та Hybrid (синя, зелена) слідують за реальною часткою (червоний пунктир), тоді як лінія TCN (помаранчева) залишається пласкою.

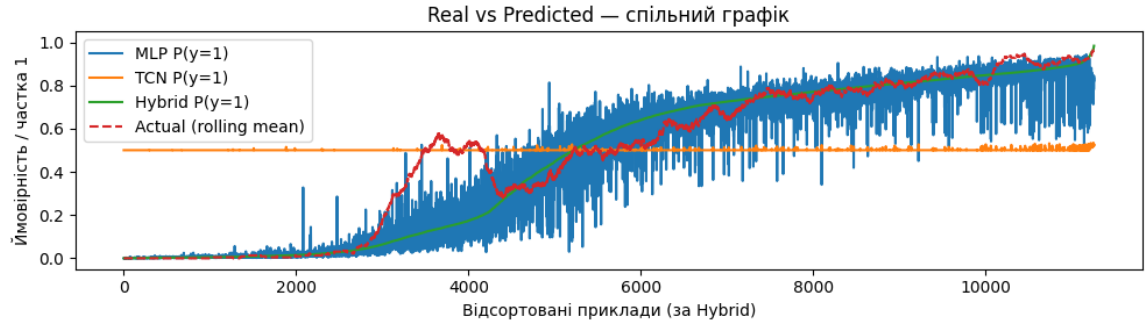

Рисунок 3.8 – Спільний калібрувальний графік (Actual vs Predicted)

Щодо "дивності" синьої лінії (MLP): вона виглядає більш "шумною" або "нервовою", ніж зелена (Hybrid). Це може бути пов'язано з тим, що порядок сортування прикладів на осі X визначався за прогнозами гібридної моделі (base_order = np.argsort(RES["Hybrid"]["probs"]["test"])). Оскільки MLP та Hybrid дають схожі, але не ідентичні прогнози, при сортуванні за Hybrid-ом прогнози MLP можуть здаватися менш "гладкими". Це не є ознакою проблеми, а лише артефактом візуалізації.

Аналіз типів помилок:

Розподіл помилок e = p – y для успішних моделей (рисунок 3.9 для MLP, рисунок 3.10 для Hybrid) показує здоровий розподіл, сконцентрований навколо нуля, з "хвостами" для складних випадків. Це кардинально відрізняється від двопікового розподілу TCN.

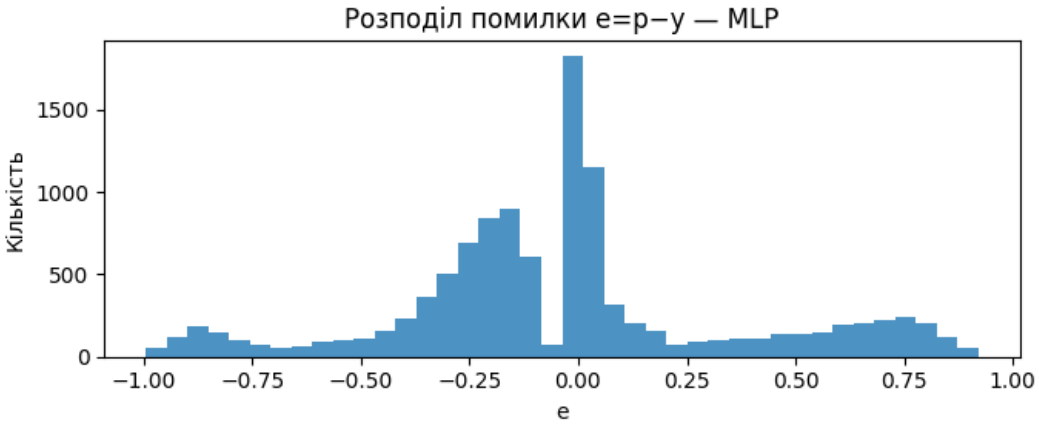

Рисунок 3.9 – Розподіл помилки e = p – y для MLP

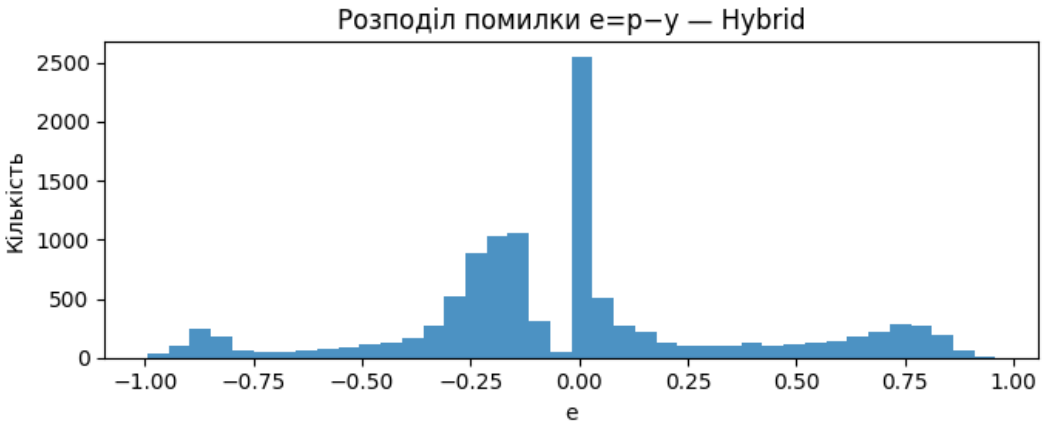

Рисунок 3.10 – Розподіл помилки e = p – y для Hybrid

Більш глибокий аналіз типів помилок (False Positives та False Negatives) на рисунку 3.11 (для Hybrid) та рисунку 3.12 (для MLP, що є практично ідентичним) показує, що моделі добре "розділяють" класи:

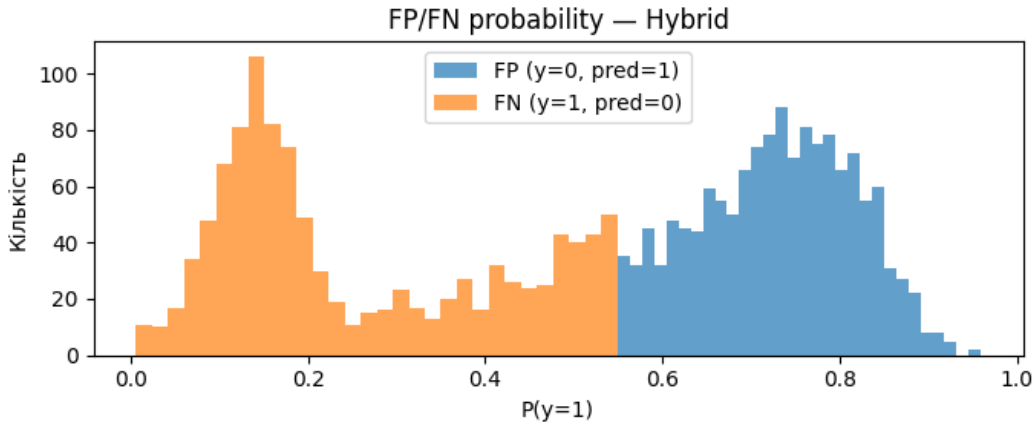

Рисунок 3.11 – Розподіл ймовірностей для помилок FP/FN (Hybrid)

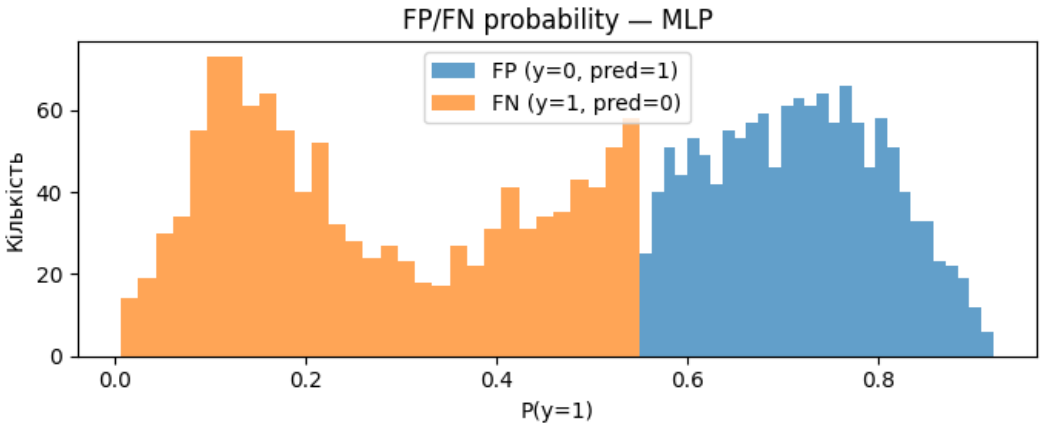

Рисунок 3.12 – Розподіл ймовірностей для помилок FP/FN (MLP)

– FN (False Negatives), помаранчевий: Це успішні студенти (y=1), яким модель помилково спрогнозувала невдачу. Їхні прогнозовані ймовірності p здебільшого низькі (< 0.5);

– FP (False Positives), синій: Це студенти з невдачею (y=0), яким модель помилково спрогнозувала успіх. Їхні прогнозовані p здебільшого високі (> 0.5).

Область перекриття (приблизно $0.4 < p < 0.6$) представляє найбільш "складних" студентів, яких важко класифікувати.

Аналіз стабільності моделей (Крос-валідація). Фінальний експеримент мав на меті перевірити, чи була незначна перевага Hybrid над MLP на тестовому наборі (2014J) статистично значущою, чи лише випадковістю. Для цього було проведено крос-валідацію (GroupKFold) на 3-х фолдах. Цей метод є набагато надійнішим за єдиний спліт, оскільки він усереднює результати по кількох "зрізах" даних. Критично важливо, що фолди формувалися не випадково, а за групами семестрів (напр., 2013B був однією групою). Це найнадійніший спосіб оцінки стабільності моделі в часі.

Результати крос-валідації (таблиця 3.2 та рисунок 3.13) виявились показовими.

Таблиця 3.2 – Результати 3-фолдної крос-валідації

Model	PR_AUC (mean±std)	ROC_AUC (mean±std)
Hybrid	0.8337 ± 0.0095	0.8710 ± 0.0359
MLP	0.8395 ± 0.0085	0.8875 ± 0.0103
TCN	0.4589 ± 0.0359	0.4955 ± 0.0078

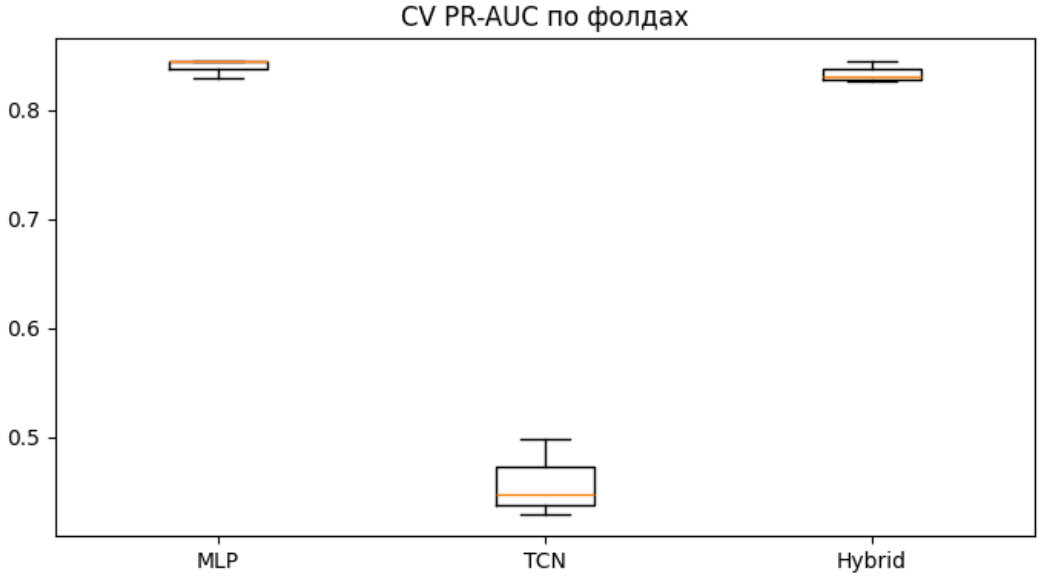

Рисунок 3.13 – Бокс-плот PR-AUC за результатами 3-фолдної крос-валідації

Вони привели до основного висновку роботи:

– вони остаточно підтвердили повний провал TCN (середній PR_AUC ≈ 0.46);

– вони показали протилежний до тестового набору результат: в середньому (mean) простіша модель MLP (PR_AUC 0.8395) показала кращий результат, ніж складніша Hybrid (PR_AUC 0.8337);

– MLP також показала менший стандартний розкид (std 0.0085 проти 0.0095), що свідчить про її вищу стабільність на різних семестрах. Як видно з

"вусів" на бокс-плоті (рисунок 3.13), діапазон якості MLP є вужчим, ніж у Hybrid.

Незначна перевага гібридної моделі на єдиному тестовому наборі (2014J) була, найімовірніше, статистичною випадковістю. Більш надійний тест крос-валідації довів, що додавання "шумної" TCN-гілки не покращує, а в середньому навіть дещо погіршує та дестабілізує прогноз порівняно з чистою, надійною MLP-моделлю, що працює на якісних інженерних ознаках.

3.4 Висновки до розділу 3

У даному розділі було детально розглянуто процес програмної реалізації спроєктованих нейромережевих моделей прогнозування успішності студентів, а також виконано ґрунтовний аналіз результатів експериментальних досліджень. Основну увагу було зосереджено на підготовці даних, організації процесу навчання моделей та порівняльній оцінці їх ефективності.

Підготовка даних ґрунтувалася на використанні відкритого набору даних Open University Learning Analytics Dataset. У процесі формування навчальної вибірки було застосовано часовий поділ даних, що дозволило уникнути витоку інформації між тренувальним і тестовим наборами. Для забезпечення можливості раннього прогнозування успішності студентів було встановлено точку зрізу на дванадцятому тижні навчання. Реалізація підготовки ознак передбачала використання двох принципово різних підходів: побудову агрегованих інженерних ознак на основі результатів оцінювання для моделі багатошарового перцептрона та формування часових послідовностей взаємодії студентів із віртуальним навчальним середовищем для темпоральної згорткової мережі. Особливу увагу було приділено коректності процедур масштабування числових ознак і кодування категоріальних змінних, які виконувалися виключно на тренувальних даних з метою запобігання витоку інформації.

Процес навчання моделей було уніфіковано для всіх архітектур шляхом використання спільної функції навчання. Навчання здійснювалося із застосуванням оптимізатора Adam та планувальника швидкості навчання CosineAnnealingLR, що забезпечувало стабільну збіжність. Для компенсації дисбалансу класів використовувалося зважування функції втрат. Контроль перенавчання реалізовано за допомогою механізму ранньої зупинки з вибором найкращої моделі на основі значення показника PR-AUC на валідаційному наборі. Після завершення навчання проводилася додаткова постобробка результатів, яка включала калібрування прогнозованих ймовірностей методом температурного масштабування та оптимізацію порогу класифікації, обидві процедури виконувалися на валідаційних даних.

Аналіз отриманих результатів дозволив зробити однозначні висновки щодо ефективності досліджуваних підходів. Темпоральна згорткова мережа, яка працювала безпосередньо з необробленими часовими послідовностями взаємодії студентів у віртуальному навчальному середовищі, продемонструвала вкрай низьку якість прогнозування. Значення основних метрик якості та результати діагностичних графіків свідчать про те, що у межах поставленої задачі такі дані виявилися надто зашумленими та недостатньо інформативними для побудови надійної моделі.

Натомість модель багатошарового перцептрона, побудована на основі ретельно спроєктованих інженерних ознак, зокрема агрегованих показників успішності студентів, показала високу ефективність, хорошу каліброваність прогнозів та швидку збіжність у процесі навчання. Отримані результати підтверджують вирішальну роль якісної інженерії ознак у задачах прогнозування на освітніх даних [24].

Гібридна модель, яка поєднувала виходи багатошарового перцептрона та темпоральної згорткової мережі, продемонструвала лише незначну перевагу над моделлю MLP на окремому тестовому наборі, однак за результатами крос-валідації поступалася їй за стабільністю та середніми значеннями метрик. Це дозволяє зробити висновок, що включення

слабкоінформативної темпоральної гілки не забезпечило статистично значущого покращення якості прогнозування і, ймовірно, негативно вплинуло на стабільність моделі.

Таким чином, експериментальна частина роботи переконливо продемонструвала перевагу підходу, заснованого на інженерії ознак та використанні багатошарового перцептрона, над end-to-end підходом із застосуванням темпоральних згорткових мереж для даної задачі та набору даних. Гібридизація моделей у запропонованій постановці не виправдала себе через низьку інформативність одного з джерел даних, що підкреслює важливість попереднього аналізу якості вхідної інформації при побудові складних мультимодальних моделей [25]-[26].

4 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Загальна архітектура та ініціалізація

4.1.1 Архітектура програмного забезпечення

Програмне забезпечення розроблено з використанням мови Python та стандартних бібліотек для роботи з табличними даними, машинного навчання і глибоких нейронних мереж.

Основна структура включає такі компоненти:

а) завантаження та перевірка даних – модуль відповідає за коректне завантаження CSV-файлів з директорії data/, перевірку їх наявності та базову очистку;

б) обробка та підготовка даних – побудова базової таблиці студентів, формування числових та категоріальних ознак, а також послідовностей подій у системі VLE;

в) моделі прогнозування – реалізація трьох основних моделей:

1) MLP (Multilayer Perceptron) для табличних даних;

2) TCN (Temporal Convolutional Network) для послідовностей подій VLE;

3) Hybrid – гібридна модель, що об’єднує MLP та TCN з механізмом SE-гейтингу.

г) навчання та крос-валідація – навчання моделей з використанням батчів, ранньої зупинки, підбір оптимального порога та калібрування ймовірностей;

ґ) оцінка та візуалізація – обчислення метрик, побудова графіків розподілу ймовірностей та порівняння прогнозів;

д) збереження результатів та моделей – збереження навченої моделі, прогнозів та параметрів калібрування у відповідних директоріях.

Всі компоненти реалізовані у вигляді модулів і класів, що забезпечує повторне використання коду, простоту налагодження та масштабування.

4.1.2 Конфігурація та ініціалізація

Для уніфікації параметрів програми створено клас CFG, який містить всі необхідні налаштування:

– параметри навчання: кількість епох, розмір батчу, швидкість навчання (lr), терплячість для ранньої зупинки;

– параметри даних: шлях до директорії з даними, назви файлів, обмеження по тижнях (cut_off_week);

– крос-валідація: кількість фолдів (cv_folds) та епох для навчання на фолдах;

– системні параметри: кількість потоків для DataLoader (num_workers), пристрій для обчислень (CPU/GPU).

Метод __post_init__ автоматично створює словник файлів, якщо він не переданий користувачем. Також клас містить властивість cut_off_day, що перераховує тижні у дні для відсічення подій студентів після певного часу.

Використання такого класу дозволяє централізовано змінювати параметри і забезпечує повторюваність експериментів.

4.2 Завантаження та підготовка даних

4.2.1 Завантаження та перевірка даних

Модуль load_oulad(cfg: CFG) відповідає за завантаження семи CSV-файлів з даними студентів та курсів:

– courses.csv – інформація про курси;

– assessments.csv – інформація про завдання та їх вагу;

– studentAssessment.csv – оцінки студентів;

– vle.csv – події у віртуальному навчальному середовищі;

– studentVle.csv – взаємодія студентів з ресурсами VLE;

– studentInfo.csv – основні демографічні та академічні дані студентів;

– studentRegistration.csv – статуси реєстрацій.

Перед завантаженням відбувається перевірка наявності всіх файлів функцією assert_files_exist(), яка генерує повідомлення про відсутні дані та зупиняє виконання, якщо файл не знайдено.

4.2.2 Підготовка ознак та формування таблиці

а) підготовка міток

Функція prepare_labels_and_base() формує бінарні мітки (label) для прогнозування успішності студентів:

1) "Pass" та "Distinction" кодуються як 1;

2) "Fail" та "Withdrawn" – як 0.

Категоріальні ознаки (gender, region, highest_education, imd_band, age_band, disability) заповнюються значенням "Unknown" у разі пропусків.

Числові ознаки (studied_credits, num_of_prev_attempts) заповнюються нулями, якщо дані відсутні.

б) функціональні ознаки оцінок

Для кожного студента створюються агреговані ознаки за завданнями:

1) кількість завдань (assess_n_total);

2) кількість зданих завдань (assess_submitted);

3) частка зданих завдань (assess_frac_sub);

4) сума ваг (assess_weight_sum);

5) середньозважена оцінка (assess_weighted_avg);

6) кількість запізнень (assess_late_cnt);

7) середнє запізнення (assess_mean_late);

8) мінімальна та максимальна оцінка (assess_min, assess_max).

Також обчислюються нормовані показники для порівняння між курсами.

в) побудова послідовностей VLE

Послідовності подій студентів у системі VLE формуються у тривимірний масив x_seq з розмірами (N_students, N_activities + 1, T_weeks), де останній канал – маска активності.

1) act_types – усі типи активностей;

2) act_idx – індекс активності у послідовності;

3) логарифмічне перетворення log1p застосовується до числа кліків для зменшення впливу великих значень.

Це дозволяє подавати послідовності у TCN.

4.2.3 Категоріальна та числова обробка

Для категоріальних ознак будується словник відповідності унікальних значень числовим індексам (<UNK> для невідомих).

Для числових ознак використовується StandardScaler, який нормалізує ознаки до нульового середнього та одиничної дисперсії, фітуючись лише на тренувальній вибірці.

4.3 Реалізація та навчання моделей

4.3.1 Реалізація моделей

MLP: MLP модуль складається з вежі для табличних ознак (MLP_Tower) та голови (Head). Категоріальні ознаки кодуються через Embedding, числові подаються безпосередньо.

Структура MLP:

– вхідний шар → 256 нейронів → BatchNorm → ReLU → Dropout;

– 128 нейронів → BatchNorm → ReLU → Dropout;

– вихідний шар → 128 → ReLU;

– голова: лінійний шар 128 → 1.

TCN: TCN модуль використовує каскад TCN_Block з каскадними каузальними свертками. Кожен блок:

– дві каузальні свертки (CausalConv1d);

– BatchNorm + GELU + Dropout;

– скіп-з’єднання для стабільності градієнтів;

– фінально, AdaptiveAvgPool1d зводить послідовність до вектору фіксованої довжини 128.

Hybrid: Гібридна модель об’єднує MLP та TCN. Векторні представлення моделей конкатенуються і пропускаються через SE-гейтинг (Squeeze-and-Excitation) для вагового балансування ознак, після чого виконується остаточне злиття через fuse і вихідна голова [16].

4.3.2 Навчання та оцінка

Навчання реалізовано у функції train_model(), яка включає:

– передачу батчів через DataLoader;

– використання BCEWithLogitsLoss;

– адаптивне оновлення градієнтів через Adam;

– Scheduler CosineAnnealingLR для плавного зменшення learning rate;

– ранню зупинку по показнику PR-AUC на валідації;

– після навчання виконується калібрування температури (fit_temperature) та підбір оптимального порога (tune_threshold) для дискретизації прогнозу;

– метрики оцінки: PR-AUC, ROC-AUC, F1-macro, Matthews Correlation Coefficient, Brier Score, Expected Calibration Error (ECE);

Це дозволяє оцінити як точність, так і каліброваність ймовірностей.

4.3.3 Крос-валідація

а) Крос-валідація

Для перевірки стабільності моделей використовується GroupKFold по code_presentation, що дозволяє уникнути витоку даних між різними презентаціями курсу.

Результати зберігаються у DataFrame, із середнім та стандартним відхиленням по фолдах. Побудова boxplot дозволяє візуально порівняти розкид показника PR-AUC між моделями.

б) Збереження результатів та прогнозів

1) моделі зберігаються у директорії saved_models/;

2) метрики та параметри калібрування у results_oulad/;

3) прогнози студентів на тестовій вибірці – у форматі long_df, включно з Ensemble (середнє по моделях).

Це забезпечує повторне використання моделей та перевірку результатів без повторного навчання.

4.4 Робота програми, візуалізація та інтерфейс користувача

Візуалізація результатів: Програмне забезпечення дозволяє будувати такі графіки:

– зміна Brier Loss по епохах – для спостереження конвергенції моделей;

– Actual vs Predicted – відсортовані за прогнозом ймовірності студенти, накладений скользячий середній реальних результатів;

– Joint Pred – порівняння всіх моделей на одній шкалі;

– Error Distribution – гістограми помилок і розподілу FP/FN для аналізу помилкових прогнозів.

Структура коду проекту: На рисунку 4.1 можна побачити структуру проекту.

Структура коду проекту:

На рисунку 4.1 можна побачити структуру проекту.

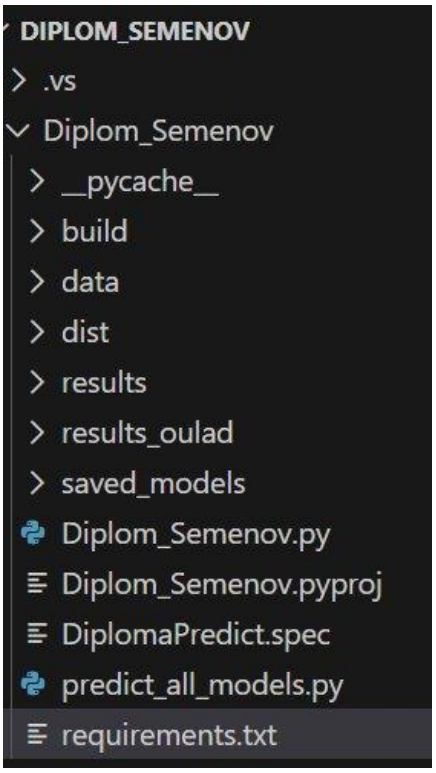

Рисунок 4.1 – Головний інтерфейс застосунку

До структури проєкту входить:

– data - OULAD дані у форматі csv;

– results - папка збереження результатів прогнозування;

– results_oulad – папка с результом навчання: калібрування, метрики, прогноз на тесті;

–saved_models – збережені моделі після навчання за даними;

– Diplom_Semenov.py – основний код тренування;

– predict_all_models.py – основний код прогнозу;

– requirements.txt – бібліотеки, потрібні проекту;

Інтерфейс користувача: додаток містить GUI на PyQt6, що дозволяє:

– вибирати студентів (конкретні, діапазон, випадкові);

–вибирати моделі для прогнозу та побудови графіків;

– виконувати прогноз у реальному часі та зберігати результати;

–візуалізувати ймовірності успішності та ensemble прогноз.

GUI інтегрується з модулем тренування і повністю використовує підготовлені числові, категоріальні та послідовні ознаки.

Робота програми:

Рисунок 4.2 показує основний інтерфейс застосонку.

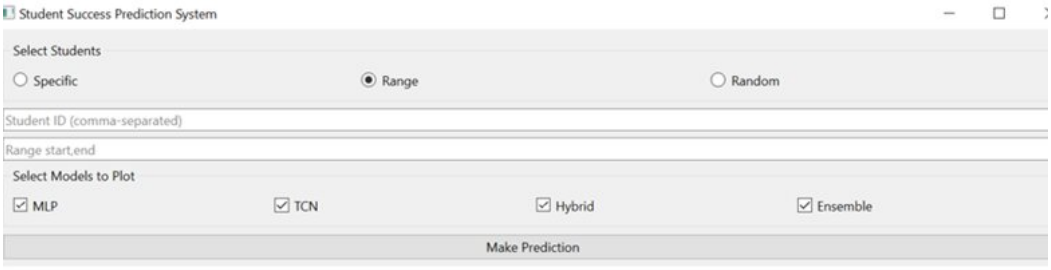

Рисунок 4.2 – Головний інтерфейс застосунку

Інтерфейс застосунку має опції:

а) Select Students:

1) Specific – поле student ID (1 або n учнів через кому);

2) Range – поле Range (start index, end index – по номеру строки в таблиці);

3) Random – випадковим чином вибирає студентів для прогнозу.

б) Select models to plot (що зберігати на графік):

1) обрати усі моделі (покаже на графіке всі моделі);

2) обрати конкретні моделі (покаже тільки їх на графіках);

3) обрати 0 моделей, збереження графіків не буде.

в) Натиснути на кнопку Make Prediction

г) Отримати результати:

1) у программі користувача таблиця та повідомлення;

2) у папці results результати, а саме csv файл прогнозу, та png. Для png за вибраними моделеми, для вибраного діапазону.

Запуск програми:

Рисунок 4.3 показує, як працює при виборі довільної вибірки. Random – нажимаємо Make Prediction

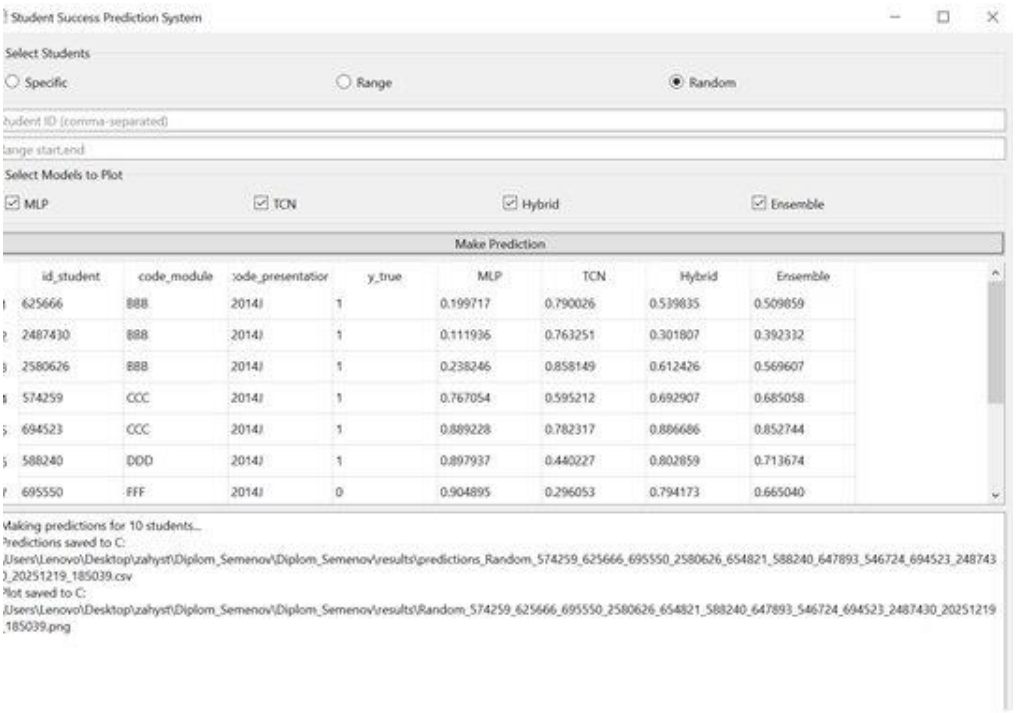

Рисунок 4.3 – Вибір 3

Рисунок 4.4 показує опрацювання вибірки за срізом. Range – вводимо start index, end index. Наприклад: 10, 30

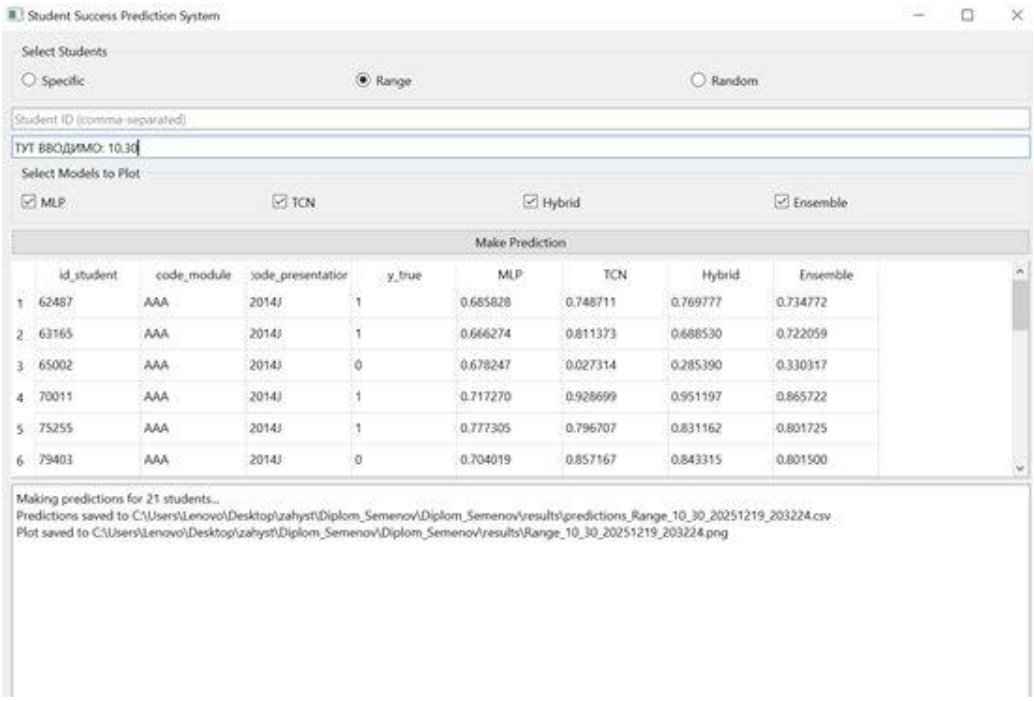

Рисунок 4.4 – Вибір 2

Рисунок 4.5 показує опрацювання 1-n студентів за id_student.

Specific – вводимо конкретний ряд , через кому, індексів студентів. Наприклад: 62487, 63165

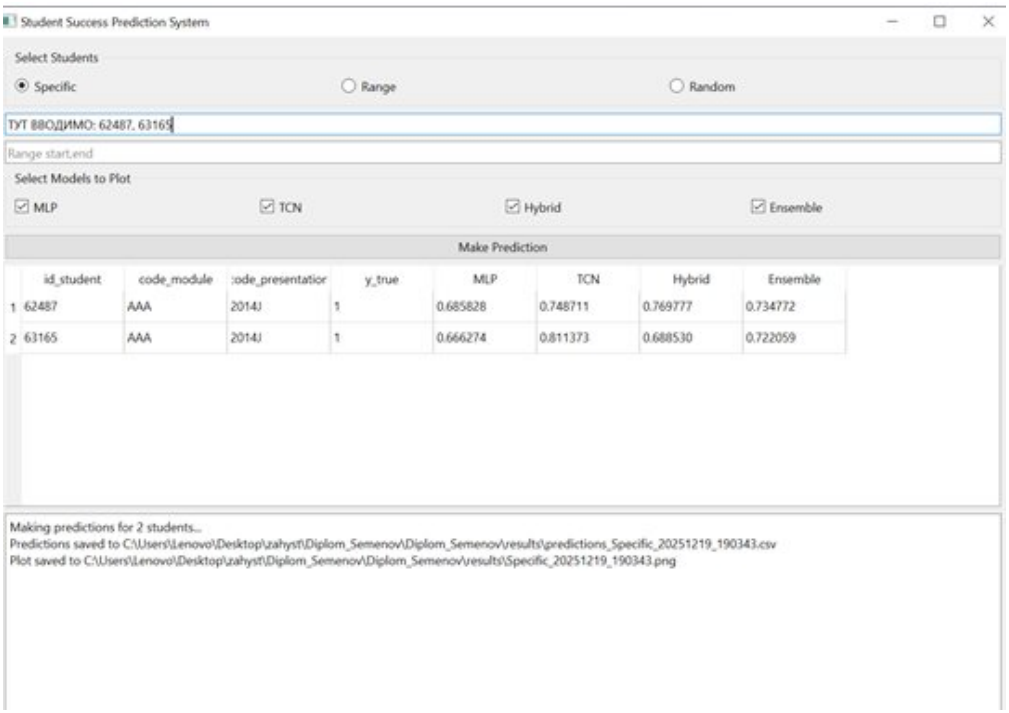

Рисунок 4.5 – Вибір 1

4.5 Висновки за розділом 4

У розділі розглядається розробка програмного забезпечення для прогнозування успішності студентів, яке реалізовано з використанням мови програмування Python та сучасних бібліотек аналізу даних і машинного навчання. Програмне забезпечення має модульну архітектуру, що забезпечує зручність супроводу, розширення та повторного використання коду. Загальна структура системи складається з кількох логічно пов’язаних компонентів, які разом забезпечують повний цикл обробки навчальних даних та отримання прогнозних результатів.

Архітектура програмного забезпечення передбачає наявність модуля завантаження та перевірки даних, який відповідає за коректне зчитування CSV-файлів з набору даних OULAD, перевірку їх наявності та цілісності.

Також реалізовано модуль обробки та підготовки даних, у межах якого здійснюється формування базової таблиці студентів, створення числових і категоріальних ознак, а також побудова часових послідовностей взаємодії студентів з навчальними ресурсами. Окремий модуль прогнозування включає реалізацію кількох моделей машинного навчання, зокрема моделі MLP, призначеної для роботи з табличними даними, моделі TCN, яка використовується для аналізу послідовних даних активності у віртуальному навчальному середовищі, а також гібридної моделі Hybrid, що поєднує виходи MLP та TCN з використанням механізму SE-гейтингу. Крім того, у складі системи реалізовано модуль навчання та оцінювання, який забезпечує процес оптимізації параметрів моделей, проведення крос-валідації та оцінку якості прогнозування.

Для централізованого керування параметрами програмного забезпечення використовується конфігураційний клас CFG, у якому зосереджено основні налаштування системи. До них належать параметри навчання моделей, зокрема кількість епох, розмір батчу та швидкість навчання, параметри доступу до даних, що включають шляхи до файлів і часові обмеження обробки інформації, а також параметри крос-валідації та системні налаштування, пов’язані з використанням обчислювальних ресурсів. Застосування такого підходу забезпечує відтворюваність експериментів і дозволяє змінювати конфігурацію системи без необхідності редагування основного програмного коду.

Завантаження даних реалізовано у вигляді окремого модуля, який працює з набором CSV-файлів, що містять інформацію про курси, студентів, результати оцінювання та активність у віртуальному навчальному середовищі. Перед початком обробки даних виконується перевірка наявності всіх необхідних файлів, що дозволяє уникнути помилок на етапі виконання програми та забезпечує коректність подальших обчислень.

Підготовка даних включає формування цільових міток успішності студентів, при цьому результати навчання кодуються у бінарному вигляді.

Категоріальні ознаки попередньо очищуються від пропущених значень та перетворюються у числовий формат, тоді як числові ознаки нормалізуються з використанням стандартного масштабування. Окрема увага приділяється формуванню функціональних ознак на основі результатів оцінювання, які дозволяють відобразити рівень навчальної активності та успішності студентів протягом проходження курсу.

Для аналізу послідовної поведінки студентів формується набір часових ознак, що описують взаємодію з ресурсами віртуального навчального середовища у розрізі тижнів навчання. Отримані послідовності подаються у вигляді багатовимірних масивів, що забезпечує можливість ефективного використання згорткових нейронних мереж для аналізу часових рядів.

Реалізація моделей прогнозування включає побудову нейронних мереж типу MLP, TCN та гібридної моделі, яка поєднує їхні переваги.

Навчання моделей здійснюється з використанням сучасних методів оптимізації, механізму ранньої зупинки та адаптивної зміни швидкості навчання. Для оцінювання якості прогнозування застосовується набір метрик, що дозволяє комплексно оцінити точність, стабільність та надійність отриманих результатів.

Завершальним етапом роботи програмного забезпечення є візуалізація результатів та взаємодія користувача з системою через графічний інтерфейс. Реалізований інтерфейс дозволяє обирати моделі, виконувати прогнозування для окремих студентів або їх груп, а також аналізувати отримані результати у наочному вигляді, що робить програмне забезпечення придатним для практичного використання у навчальному процесі.

ВИСНОВКИ

У даній магістерській дипломній роботі було вирішено актуальну науково прикладну задачу розробки та порівняльного аналізу нейромережевих моделей для раннього прогнозування академічної успішності студентів в умовах онлайн навчання. Основною метою роботи було створення ефективних прогнозних моделей та емпіричне дослідження результативності різних стратегій моделювання навчальних даних із порівнянням підходу на основі інженерії ознак та підходу прямого навчання моделей на часових послідовностях без попередньої агрегації, а також оцінка доцільності їх спільного використання в межах гібридної архітектури.

У процесі виконання роботи було проведено ґрунтовний аналіз сучасних наукових публікацій у галузі освітньої аналітики та прогнозування навчальних результатів, що дозволило систематизувати існуючі підходи та виділити основні напрями досліджень, пов’язані з використанням табличних даних, послідовних логів активності та їх комбінування. На основі цього аналізу було спроєктовано та реалізовано три нейромережеві моделі, кожна з яких відповідає окремій стратегії обробки даних, а саме багатошаровий перцептрон для роботи з гетерогенними табличними ознаками, часову згорткову нейронну мережу для аналізу послідовностей активності у віртуальному навчальному середовищі та гібридну модель, що поєднує обидва типи представлень даних у межах спільної архітектури.

Для експериментального дослідження було використано публічний набір даних OULAD, для якого реалізовано методологічно коректну схему підготовки даних із часовим поділом на навчальну, валідаційну та тестову вибірки за семестрами навчання. З метою забезпечення раннього прогнозування було визначено точку зрізу на дванадцятому тижні курсу. Підготовка ознак здійснювалася з використанням двох окремих конвеєрів, орієнтованих на специфіку табличних та послідовних даних, що дозволило забезпечити коректність експериментального порівняння моделей.

У межах роботи було реалізовано програмний комплекс мовою Python із використанням бібліотеки PyTorch, який включає нейромережеві моделі, модулі підготовки та подачі даних, а також уніфікований механізм навчання та оцінювання моделей. Процес навчання супроводжувався використанням сучасних методів оптимізації, компенсації дисбалансу класів, механізму ранньої зупинки, калібрування ймовірностей та оптимізації порогів прийняття рішень, що забезпечило високу надійність отриманих результатів.

Проведений експериментальний аналіз показав істотну різницю між ефективністю розглянутих підходів. Модель, побудована на основі інженерії ознак, продемонструвала високу точність прогнозування та стабільність результатів, що підтвердило вирішальну роль якісної підготовки даних у задачах освітньої аналітики. Водночас використання сирих послідовних логів активності без попередньої агрегації виявилося малоефективним через високий рівень зашумленості та розрідженості таких даних. Застосування гібридної архітектури не забезпечило статистично значущого покращення результатів, що свідчить про обмежену інформативність послідовної модальності у даній постановці задачі.

Отримані результати мають як наукову, так і практичну цінність, оскільки демонструють доцільність використання агрегованих показників успішності для побудови систем раннього виявлення студентів групи ризику. Робота також підтверджує важливість ретельного аналізу даних та інженерії ознак навіть за умов застосування сучасних методів глибокого навчання. Запропонований підхід може бути використаний закладами освіти для підтримки прийняття рішень та подальшого розвитку інтелектуальних навчальних систем.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Baker, R. S. Educational data mining and learning analytics. Potentials and possibilities for online education / R. S. Baker, P. S. Inventado // Handbook of Educational Data Mining. – 2016. – P. 83–98. – [Electronic resource]. – Mode of access: https://scispace.com/pdf/educational-data-mining-and-learning-analytics-2kepggrl5r.pdf

2. Romero, C. Educational data mining: a review of the state-of-the-art / C. Romero, S. Ventura // IEEE Transactions on Systems, Man, and Cybernetics. – 2010. – Vol. 40, No. 6. – P. 601–618. – Mode of access: https://www.researchgate.net/publication/224160756

3. Субботін, С. О. Нейронні мережі: теорія та практика : навчальний посібник / С. О. Субботін. – Житомир : Вид-во О. О. Євенок, 2020. – 184 с. – Режим доступу: https://eir.zp.edu.ua/server/api/core/bitstreams/2abb401b-9ee6-4afc-a92a-2de5c332d12f/content

4. Niu, K. A hybrid model for predicting academic performance of engineering undergraduates / K. Niu, B. Jia, Y. Zhou, G. Lu. – 2022. – Mode of access: https://www.researchgate.net/publication/362393994

5. Dass, S. Predicting student dropout in self-paced MOOC course using random forest model / S. Dass, K. Gary, J. Cunningham // Information. – 2021. – Vol. 12, No. 11. – Art. 476. – Mode of access: https://doi.org/10.3390/info12110476

6. Garg, A. Machine learning-based model to predict student success in higher education / A. Garg [et al.] // ICIDSSD 2022. – 2023. – Mode of access: https://doi.org/10.4108/eai.24-3-2022.2318766

7. Глибовець, М. М. Штучний інтелект : підручник / М. М. Глибовець, О. В. Олецький. – Київ : КМ-Академія, 2002. – 371 с. – Режим доступу: https://pdf.lib.vntu.edu.ua/books/2020/Glybovec_2002_366.pdf

8. Bai, S. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling / S. Bai, J. Z. Kolter, V. Koltun. – 2018. –

[Electronic resource]. – Mode of access: https://arxiv.org/abs/1803.01271

9. Vives, L. Prediction of students' academic performance in the programming fundamentals course using long short-term memory neural networks / L. Vives, I. Cabezas, J. C. Vives, N. G. Reyes, J. Aquino, J. B. Cóndor // IEEE Access. – 2024. – Vol. 12. – P. 5882–5898. – Mode of access: https://doi.org/10.1109/ACCESS.2024.3350169

10. The Python standard library [Electronic resource]. – 2025. – Mode of access: https://docs.python.org/3/library/index.html

11. Paszke, A. PyTorch: an imperative style, high-performance deep learning library / A. Paszke [et al.] // NeurIPS 2019. – Mode of access: https://arxiv.org/abs/1912.01703

12. Scikit-learn: machine learning in Python [Electronic resource]. – 2024. – Mode of access: https://scikit-learn.org/stable/user_guide.html

13. Jha, K. OULAD MOOC dropout and result prediction using ensemble, deep learning and regression techniques / K. Jha, I. Ghergulescu [et al.] // CSEDU 2019. – P. 154–164. – Mode of access: https://www.scitepress.org/Papers/2019/77679/77679.pdf

14. Srivastava, N. Dropout: a simple way to prevent neural networks from overfitting / N. Srivastava [et al.] // Journal of Machine Learning Research. – 2014. – Vol. 15, No. 1. – P. 1929–1958. – Mode of access: https://www.researchgate.net/publication/286794765

15. Yan, L. Student engagement assessment using multimodal deep learning / L. Yan, X. Wu, Y. Wang // PLoS One. – 2025. – Vol. 20, No. 6. – Mode of access: https://doi.org/10.1371/journal.pone.0325377

16. Hu, J. Squeeze-and-excitation networks / J. Hu, L. Shen, G. Sun [et al.] // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. – 2017. – Mode of access: https://arxiv.org/abs/1709.01507

17. Kuzilek, J. Open university learning analytics dataset / J. Kuzilek, M. Hlosta, Z. Zdrahal // Scientific Data. – 2017. – Vol. 4, No. 1. – Art. 170171. – Mode of access: https://www.nature.com/articles/sdata2017171

18. PyTorch documentation [Electronic resource]. – 2024. – Mode of access: https://pytorch.org/docs/stable/index.html

19. Goodfellow, I. Deep learning / I. Goodfellow, Y. Bengio, A. Courville. – Cambridge : MIT Press, 2016. – 800 p. – [Electronic resource]. – Mode of access: https://pzs.dstu.dp.ua/DataMining/bibl/Deep%20Learning.pdf

20. Chollet, F. Deep learning with Python / F. Chollet. – Shelter Island : Manning Publications, 2017. – 384 p.

21. Gao, Y. Predicting high-risk students using learning behavior / Y. Gao, Y. Liu, Z. Jin [et al.] // Mathematics. – 2022. – [Electronic resource]. – Mode of access: https://doi.org/10.3390/math10142483

22. Kingma, D. P. Adam: a method for stochastic optimization / D. P. Kingma, J. Ba // International Conference on Learning Representations. – 2015. – Mode of access: https://arxiv.org/abs/1412.6980

23. Aulck, L. Predicting student dropout in higher education / L. Aulck, N. Velagapudi, J. Blumenstock, J. West. – 2016. – [Electronic resource]. – Mode of access: https://arxiv.org/abs/1606.06364

24. Довбиш, А. С. Інтелектуальні інформаційні технології в електронному навчанні / А. С. Довбиш [та ін.] – Суми : Сумський державний університет, 2013. – 177 с. – Режим доступу: http://essuir.sumdu.edu.ua/handle/123456789/33429

25. Вакалюк, Т. А. Прогнозне моделювання аналітики успішності студентів з використанням алгоритмів машинного навчання / Т. А. Вакалюк, В. М. Янчук, Д. С. Морозов [та ін.] // Вчені записки ТНУ імені В. І. Вернадського. Серія: Технічні науки. – 2023. – Т. 34(73), № 5. – С. 108–116. – Режим доступу: https://eprints.zu.edu.ua/38273/

26. Al-Shabandar, R. Machine learning approaches to predict learning outcomes in massive open online courses / R. Al-Shabandar [et al.]. – [Electronic resource]. – Mode of access: https://scispace.com/pdf/machine-learning-approaches-to-predict-learning-outcomes-in-529z38ngco.pdf

ДОДАТОК А

Текст програми

А.1 Текст файлу Diplom_Semenov.py

```
# -*- coding: utf-8 -*-
"""
OULAD: MLP (tabular), TCN (VLE sequence), Hybrid (fusion + SE gating)
- Cutoff: first 12 weeks
- Split by code_presentation
- Calibration: Temperature scaling (fit on val)
- Threshold tuning: macro-F1 on val
- Saves metrics + predictions to results_oulad/
"""

import os
import re
import gc
import json
import random
from dataclasses import dataclass
from pathlib import Path

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.metrics import (
    average_precision_score, roc_auc_score,
    f1_score, matthews_corrcoef, brier_score_loss
)
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GroupKFold

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader

# --------------------------- CONFIG ---------------------------

@dataclass
class CFG:
    seed: int = 42

    # шлях:
    BASE_DIR = Path(__file__).resolve().parent
    data_dir: Path = BASE_DIR / "data"

    files: dict = None

    cut_off_week: int = 12
    batch: int = 512

    epochs_mlp: int = 50
    epochs_tcn: int = 50
    epochs_hyb: int = 60

    lr_mlp: float = 1e-3
    lr_tcn: float = 1e-3
    lr_hyb: float = 8e-4

    patience_mlp: int = 8
    patience_tcn: int = 8
    patience_hyb: int = 10
```

```
            cv_folds: int = 3
            cv_epochs: int = 20

            num_workers: int = 0

            def __post_init__(self):
                if self.files is None:
                    self.files = {
                        "courses": "courses.csv",
                        "assessments": "assessments.csv",
                        "studentAssessment": "studentAssessment.csv",
                        "vle": "vle.csv",
                        "studentVle": "studentVle.csv",
                        "studentInfo": "studentInfo.csv",
                        "studentRegistration": "studentRegistration.csv",
                    }

            @property
            def cut_off_day(self) -> int:
                return self.cut_off_week * 7

        CFG = CFG()

        DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        PIN = (DEVICE.type == "cuda")

        # --------------------------- UTILS ---------------------------

        def seed_everything(seed: int):
            random.seed(seed)
            np.random.seed(seed)
            torch.manual_seed(seed)
            if torch.cuda.is_available():
                torch.cuda.manual_seed_all(seed)

        def assert_files_exist(data_dir: Path, files: dict):
            for name, fn in files.items():
                p = data_dir / fn
                if not p.exists():
                    raise FileNotFoundError(f"Не знайдено {fn} у
{data_dir.resolve()}")

        def parse_present(code: str):
            m = re.match(r"(\d{4})([A-Z])", str(code))
            return (int(m.group(1)), m.group(2)) if m else (9999, "Z")

        def make_split(studentInfo: pd.DataFrame):
            presentations =
sorted(studentInfo["code_presentation"].unique().tolist())
            presentations_sorted = sorted(presentations, key=parse_present)

            predef = {"train": ["2013B", "2013J"], "val": ["2014B"], "test":
["2014J"]}
            ok = all(p in presentations for p in sum(predef.values(), []))
            if ok:
                split = predef
            else:
                split = {
                    "train": presentations_sorted[:-2],
                    "val": [presentations_sorted[-2]],
                    "test": [presentations_sorted[-1]],
                }
            return split
```

```
        def groupby_apply_compat(gb, func):
            # Pandas 2.x: include_groups=False
            try:
                out = gb.apply(func,
include_groups=False).reset_index(drop=True)
            except TypeError:
                out = gb.apply(func).reset_index(drop=True)
            return out

        def emb_dim(card: int) -> int:
            return min(50, (card + 1) // 2)

        def sigmoid_np(x: np.ndarray) -> np.ndarray:
            return 1.0 / (1.0 + np.exp(-x))

        # --------------------------- LOAD DATA ---------------------------

        def load_oulad(cfg: CFG):
            assert_files_exist(cfg.data_dir, cfg.files)

            courses = pd.read_csv(cfg.data_dir / cfg.files["courses"])
            assessments = pd.read_csv(cfg.data_dir / cfg.files["assessments"])
            studentAssessment = pd.read_csv(cfg.data_dir /
cfg.files["studentAssessment"])
            vle = pd.read_csv(cfg.data_dir / cfg.files["vle"])
            studentVle = pd.read_csv(cfg.data_dir / cfg.files["studentVle"])
            studentInfo = pd.read_csv(cfg.data_dir / cfg.files["studentInfo"])
            studentRegistration = pd.read_csv(cfg.data_dir /
cfg.files["studentRegistration"])

            return dict(
                courses=courses,
                assessments=assessments,
                studentAssessment=studentAssessment,
                vle=vle,
                studentVle=studentVle,
                studentInfo=studentInfo,
                studentRegistration=studentRegistration
            )

        # --------------------------- FEATURE ENGINEERING --------------------
-------

        def prepare_labels_and_base(studentInfo: pd.DataFrame):
            label_map = {"Pass": 1, "Distinction": 1, "Fail": 0, "Withdrawn":
0}
            studentInfo = studentInfo.copy()
            studentInfo["label"] =
studentInfo["final_result"].map(label_map).astype("int8")

            cat_cols = ["gender", "region", "highest_education", "imd_band",
"age_band", "disability"]
            num_cols = ["studied_credits", "num_of_prev_attempts"]

            for c in cat_cols:
                if c not in studentInfo.columns:
                    studentInfo[c] = "Unknown"
            studentInfo[cat_cols] = studentInfo[cat_cols].fillna("Unknown")

            for c in num_cols:
                if c not in studentInfo.columns:
                    studentInfo[c] = 0
            studentInfo[num_cols] = studentInfo[num_cols].fillna(0)
```

```
            base = studentInfo[["id_student", "code_module",
"code_presentation", "label"] + cat_cols + num_cols].copy()
            return studentInfo, base, cat_cols, num_cols

        def build_assessment_features(assessments: pd.DataFrame,
studentAssessment: pd.DataFrame, cut_off_day: int):
            sa = studentAssessment.merge(
                assessments[["id_assessment", "code_module",
"code_presentation", "assessment_type", "date", "weight"]],
                on="id_assessment", how="left"
            )
            sa_cut = sa[sa["date"] <= cut_off_day].copy()

            def agg_assess(df: pd.DataFrame):
                n_total = df["id_assessment"].nunique()

                submitted = df["score"].notna().sum()
                frac_sub = submitted / n_total if n_total > 0 else 0.0

                sum_w = df["weight"].sum() if n_total > 0 else 0.0
                wavg = (df["score"].fillna(0) * df["weight"].fillna(0)).sum()
/ sum_w if sum_w > 0 else 0.0

                late = ((df["date_submitted"] - df["date"]) >
0).fillna(False).sum()
                mean_late = ((df["date_submitted"] -
df["date"]).clip(lower=0)).fillna(0).mean() if submitted > 0 else 0.0

                max_score = df["score"].max() if submitted > 0 else 0.0
                min_score = df["score"].min() if submitted > 0 else 0.0

                return pd.Series(dict(
                    assess_n_total=n_total,
                    assess_submitted=submitted,
                    assess_frac_sub=frac_sub,
                    assess_weight_sum=sum_w,
                    assess_weighted_avg=wavg,
                    assess_late_cnt=late,
                    assess_mean_late=mean_late,
                    assess_max=max_score,
                    assess_min=min_score
                ))

            gb = sa_cut.groupby(["id_student", "code_module",
"code_presentation"], as_index=False)
            ass_feats = groupby_apply_compat(gb, agg_assess)

            ass_sum_by_course = (
                assessments[assessments["date"] <= cut_off_day]
                .groupby(["code_module", "code_presentation"])["weight"]
                .sum().rename("course_weight_sum").reset_index()
            )

            ass_feats = ass_feats.merge(ass_sum_by_course, on=["code_module",
"code_presentation"], how="left")
            ass_feats["course_weight_sum"] =
ass_feats["course_weight_sum"].replace(0, np.nan)

            ass_feats["assess_weight_sum_norm"] =
ass_feats["assess_weight_sum"] / ass_feats["course_weight_sum"]
            ass_feats["assess_weighted_avg_norm"] =
ass_feats["assess_weighted_avg"]

            return ass_feats
```

```
        def merge_tabular(base: pd.DataFrame, ass_feats: pd.DataFrame):
            key_cols = ["id_student", "code_module", "code_presentation"]
            tab = base.merge(ass_feats, on=key_cols, how="left")

            must_cols = [
                "assess_n_total", "assess_submitted", "assess_frac_sub",
"assess_weight_sum",
                "assess_weight_sum_norm", "assess_weighted_avg",
"assess_weighted_avg_norm",
                "assess_late_cnt", "assess_mean_late", "assess_max",
"assess_min"
            ]
            for c in must_cols:
                if c not in tab.columns:
                    tab[c] = 0.0

            tab = tab.fillna({
                "assess_n_total": 0, "assess_submitted": 0, "assess_frac_sub":
0.0,
                "assess_weight_sum": 0.0, "assess_weight_sum_norm": 0.0,
                "assess_weighted_avg": 0.0, "assess_weighted_avg_norm": 0.0,
                "assess_late_cnt": 0.0, "assess_mean_late": 0.0,
                "assess_max": 0.0, "assess_min": 0.0
            })
            return tab

        def build_vle_sequence(vle: pd.DataFrame, studentVle: pd.DataFrame,
keys_tab: pd.DataFrame, cut_off_day: int, cut_off_week: int):
            sv = studentVle[studentVle["date"] <= cut_off_day].copy()
            sv = sv.merge(
                vle[["code_module", "code_presentation", "id_site",
"activity_type"]],
                on=["code_module", "code_presentation", "id_site"],
                how="left"
            )
            sv["week"] = (sv["date"] // 7).clip(lower=0, upper=cut_off_week -
1)

            # тільки ключі, що є в tab
            sv = sv.merge(keys_tab, on=["id_student", "code_module",
"code_presentation"], how="inner")

            act_types =
sorted(vle["activity_type"].dropna().unique().tolist())
            act_to_idx = {a: i for i, a in enumerate(act_types)}
            n_act = len(act_types)
            seq_t = cut_off_week

            sv = sv.dropna(subset=["activity_type"])
            sv["act_idx"] = sv["activity_type"].map(act_to_idx).astype(int)

            keys_tab = keys_tab.copy()
            keys_tab["_rid"] = np.arange(len(keys_tab))
            sv = sv.merge(keys_tab, on=["id_student", "code_module",
"code_presentation"], how="left")

            n = len(keys_tab)
            x_dense = np.zeros((n, n_act, seq_t), dtype=np.float32)

            np.add.at(
                x_dense,
                (
                    sv["_rid"].values.astype(np.int64),
```

```
                    sv["act_idx"].values.astype(np.int64),
                    sv["week"].values.astype(np.int64)
                ),
                sv["sum_click"].values.astype(np.float32)
            )

            x_dense = np.log1p(x_dense)
            mask = (x_dense.sum(axis=1) > 0).astype(np.float32)[:, None, :]
# (N,1,T)
            x_seq = np.concatenate([x_dense, mask], axis=1)
# (N, n_act+1, T)

            return x_seq, act_types

        def build_cat_maps(tab_joint: pd.DataFrame, cat_cols: list,
mask_train: np.ndarray):
            def build_map(series_train):
                uniq = ["<UNK>"] +
sorted(pd.Series(series_train).astype(str).unique().tolist())
                return {v: i for i, v in enumerate(uniq)}

            def encode(series, m):
                return series.astype(str).map(lambda v: m.get(v,
0)).values.astype(np.int64)

            cat_maps = {}
            cards = []
            for c in cat_cols:
                m = build_map(tab_joint.loc[mask_train, c])
                cat_maps[c] = m
                cards.append(len(m))

            xcat_all = np.stack([encode(tab_joint[c], cat_maps[c]) for c in
cat_cols], axis=1)
            return xcat_all, cat_maps, cards

        def build_num_scaled(tab_joint: pd.DataFrame, num_cols: list,
mask_train: np.ndarray):
            scaler = StandardScaler()
            x = tab_joint[num_cols].values.astype(np.float32)
            scaler.fit(x[mask_train])
            return scaler.transform(x), scaler

        # ---------------------------- DATASET ----------------------------

        class OuladDS(Dataset):
            def __init__(self, Xnum, Xcat, Xseq, y):
                self.Xnum = Xnum
                self.Xcat = Xcat
                self.Xseq = Xseq
                self.y = y

            def __len__(self):
                return len(self.y)

            def __getitem__(self, i):
                return (
                    torch.tensor(self.Xnum[i], dtype=torch.float32),
                    torch.tensor(self.Xcat[i], dtype=torch.long),
                    torch.tensor(self.Xseq[i], dtype=torch.float32),
                    torch.tensor(self.y[i], dtype=torch.float32)
                )

        def make_loaders(Xnum_tr, Xcat_tr, Xseq_tr, y_tr,
```

```
                        Xnum_va, Xcat_va, Xseq_va, y_va,
                        Xnum_te, Xcat_te, Xseq_te, y_te,
                        cfg: CFG):
            dl_tr = DataLoader(OuladDS(Xnum_tr, Xcat_tr, Xseq_tr, y_tr),
                               batch_size=cfg.batch, shuffle=True,
                               num_workers=cfg.num_workers, pin_memory=PIN)
            dl_va = DataLoader(OuladDS(Xnum_va, Xcat_va, Xseq_va, y_va),
                               batch_size=cfg.batch, shuffle=False,
                               num_workers=cfg.num_workers, pin_memory=PIN)
            dl_te = DataLoader(OuladDS(Xnum_te, Xcat_te, Xseq_te, y_te),
                               batch_size=cfg.batch, shuffle=False,
                               num_workers=cfg.num_workers, pin_memory=PIN)
            return dl_tr, dl_va, dl_te

        # -------------------------- MODELS --------------------------

        class MLP_Tower(nn.Module):
            def __init__(self, num_dim, cat_cards, out_emb=128):
                super().__init__()
                self.embs = nn.ModuleList([nn.Embedding(c, emb_dim(c)) for c
in cat_cards])
                mlp_in = num_dim + sum(emb_dim(c) for c in cat_cards)

                self.net = nn.Sequential(
                    nn.Linear(mlp_in, 256), nn.BatchNorm1d(256), nn.ReLU(),
nn.Dropout(0.2),
                    nn.Linear(256, 128), nn.BatchNorm1d(128), nn.ReLU(),
nn.Dropout(0.2),
                    nn.Linear(128, out_emb), nn.ReLU()
                )

            def forward(self, xnum, xcat):
                em = [emb(xcat[:, i]) for i, emb in enumerate(self.embs)]
                em = torch.cat(em, dim=1) if em else torch.zeros(xnum.size(0),
0, device=xnum.device)
                x = torch.cat([xnum, em], dim=1)
                return self.net(x)

        class CausalConv1d(nn.Module):
            def __init__(self, in_ch, out_ch, k, d):
                super().__init__()
                self.pad = (k - 1) * d
                self.conv = nn.Conv1d(in_ch, out_ch, kernel_size=k,
dilation=d)

            def forward(self, x):
                x = F.pad(x, (self.pad, 0))
                return self.conv(x)

        class TCN_Block(nn.Module):
            def __init__(self, ch_in, ch_out, k=3, d=1, drop=0.1):
                super().__init__()
                self.c1 = CausalConv1d(ch_in, ch_out, k, d)
                self.c2 = CausalConv1d(ch_out, ch_out, k, d)
                self.bn1 = nn.BatchNorm1d(ch_out)
                self.bn2 = nn.BatchNorm1d(ch_out)
                self.drop = nn.Dropout(drop)
                self.skip = nn.Conv1d(ch_in, ch_out, 1) if ch_in != ch_out
else nn.Identity()

            def forward(self, x):
                h = self.drop(F.gelu(self.bn1(self.c1(x))))
                h = self.drop(F.gelu(self.bn2(self.c2(h))))
                return h + self.skip(x)
```

```
class TCN_Tower(nn.Module):
    def __init__(self, in_ch, out_emb=128, drop=0.1):
        super().__init__()
        self.b1 = TCN_Block(in_ch, 64, k=3, d=1, drop=drop)
        self.b2 = TCN_Block(64, 64, k=3, d=2, drop=drop)
        self.b3 = TCN_Block(64, 128, k=3, d=4, drop=drop)
        self.b4 = TCN_Block(128, 128, k=3, d=8, drop=drop)
        self.pool = nn.AdaptiveAvgPool1d(1)
        self.fc = nn.Linear(128, out_emb)

    def forward(self, xseq):
        h = self.b4(self.b3(self.b2(self.b1(xseq))))
        h = self.pool(h).squeeze(-1)
        return F.relu(self.fc(h))

class Head(nn.Module):
    def __init__(self, in_dim):
        super().__init__()
        self.fc = nn.Linear(in_dim, 1)

    def forward(self, z):
        return self.fc(z).squeeze(1)

class ModelMLP(nn.Module):
    def __init__(self, num_dim, cat_cards):
        super().__init__()
        self.tower = MLP_Tower(num_dim, cat_cards, out_emb=128)
        self.head = Head(128)

    def forward(self, xnum, xcat, xseq=None):
        z = self.tower(xnum, xcat)
        return self.head(z), z

class ModelTCN(nn.Module):
    def __init__(self, in_ch):
        super().__init__()
        self.tower = TCN_Tower(in_ch, out_emb=128)
        self.head = Head(128)

    def forward(self, xnum=None, xcat=None, xseq=None):
        z = self.tower(xseq)
        return self.head(z), z

class ModelHybrid(nn.Module):
    def __init__(self, num_dim, cat_cards, in_ch):
        super().__init__()
        self.mlp = MLP_Tower(num_dim, cat_cards, out_emb=128)
        self.tcn = TCN_Tower(in_ch, out_emb=128)
        self.se = nn.Sequential(
            nn.Linear(256, 64), nn.ReLU(),
            nn.Linear(64, 256), nn.Sigmoid()
        )
        self.fuse = nn.Sequential(
            nn.Linear(256, 128), nn.ReLU(),
            nn.Linear(128, 128), nn.ReLU()
        )
        self.head = Head(128)

    def forward(self, xnum, xcat, xseq):
        zm = self.mlp(xnum, xcat)
        zt = self.tcn(xseq)
        z = torch.cat([zm, zt], dim=1)
        z = z * self.se(z)
```

```
                z = self.fuse(z)
                return self.head(z), z

        def count_params(model):
            return sum(p.numel() for p in model.parameters() if
p.requires_grad)

        # -------------------------- METRICS / CALIBRATION ------------------
---------

        def compute_metrics(y_true, p):
            y_pred = (p >= 0.5).astype(int)
            return dict(
                PR_AUC=average_precision_score(y_true, p),
                ROC_AUC=roc_auc_score(y_true, p),
                F1_macro=f1_score(y_true, y_pred, average="macro"),
                MCC=matthews_corrcoef(y_true, y_pred),
                Brier=brier_score_loss(y_true, p),
            )

        def expected_calibration_error(y_true, p, M=15):
            bins = np.linspace(0, 1, M + 1)
            ind = np.digitize(p, bins) - 1
            ece = 0.0
            for m in range(M):
                mask = ind == m
                if mask.sum() == 0:
                    continue
                ece += mask.mean() * abs(p[mask].mean() - y_true[mask].mean())
            return float(ece)

        class TemperatureScaler(nn.Module):
            def __init__(self):
                super().__init__()
                self.t = nn.Parameter(torch.ones(1))

            def forward(self, logits):
                return logits / self.t.clamp_min(1e-3)

        def fit_temperature(logits_val, y_val):
            ts = TemperatureScaler().to(DEVICE)
            yv = torch.tensor(y_val, dtype=torch.float32, device=DEVICE)
            lv = torch.tensor(logits_val, dtype=torch.float32, device=DEVICE)

            opt = torch.optim.LBFGS(ts.parameters(), lr=0.1, max_iter=50)
            loss_fn = nn.BCEWithLogitsLoss()

            def closure():
                opt.zero_grad()
                loss = loss_fn(ts(lv), yv)
                loss.backward()
                return loss

            opt.step(closure)
            return float(ts.t.detach().cpu().item())

        def tune_threshold(y_val, p_val):
            thr = np.linspace(0.05, 0.95, 19)
            scores = [f1_score(y_val, (p_val >= t).astype(int),
average="macro") for t in thr]
            return float(thr[int(np.argmax(scores))])

        # -------------------------- TRAINING --------------------------
```

```
        def forward_logits(model_name: str, model: nn.Module, xb_num, xb_cat,
xb_seq):
            if model_name == "MLP":
                logits, _ = model(xb_num, xb_cat, None)
            elif model_name == "TCN":
                logits, _ = model(None, None, xb_seq)
            else:
                logits, _ = model(xb_num, xb_cat, xb_seq)
            return logits

        def train_model(model_name, model, dl_tr, dl_va, dl_te,
                        epochs=50, lr=1e-3, patience=8, pos_weight=None):

            model = model.to(DEVICE)
            print(f"{model_name} params: {count_params(model)}")

            opt = torch.optim.Adam(model.parameters(), lr=lr)
            sched = torch.optim.lr_scheduler.CosineAnnealingLR(opt,
T_max=epochs)

            if pos_weight is not None:
                loss_fn =
nn.BCEWithLogitsLoss(pos_weight=torch.tensor([pos_weight], device=DEVICE))
            else:
                loss_fn = nn.BCEWithLogitsLoss()

            hist = {"train_loss": [], "val_brier": [], "val_pr": []}
            best = {"val_pr": -1, "state": None, "epoch": -1}
            wait = 0

            for ep in range(1, epochs + 1):
                # train
                model.train()
                tr_loss = 0.0
                for xb_num, xb_cat, xb_seq, yb in dl_tr:
                    xb_num = xb_num.to(DEVICE)
                    xb_cat = xb_cat.to(DEVICE)
                    xb_seq = xb_seq.to(DEVICE)
                    yb = yb.to(DEVICE)

                    opt.zero_grad()
                    logits = forward_logits(model_name, model, xb_num, xb_cat,
xb_seq)
                    loss = loss_fn(logits, yb)
                    loss.backward()
                    opt.step()

                    tr_loss += float(loss.item()) * len(yb)

                tr_loss /= len(dl_tr.dataset)

                # val
                model.eval()
                val_logits, val_y = [], []
                with torch.no_grad():
                    for xb_num, xb_cat, xb_seq, yb in dl_va:
                        xb_num = xb_num.to(DEVICE)
                        xb_cat = xb_cat.to(DEVICE)
                        xb_seq = xb_seq.to(DEVICE)

                        l = forward_logits(model_name, model, xb_num, xb_cat,
xb_seq)
                        val_logits.append(l.detach().cpu().numpy())
                        val_y.append(yb.numpy())
```

```
                val_logits = np.concatenate(val_logits)
                val_y = np.concatenate(val_y)
                val_p = sigmoid_np(val_logits)
                val_pr = average_precision_score(val_y, val_p)
                val_brier = brier_score_loss(val_y, val_p)

                hist["train_loss"].append(tr_loss)
                hist["val_brier"].append(val_brier)
                hist["val_pr"].append(val_pr)

                sched.step()
                print(f"[{model_name}] ep {ep:02d} | train_loss={tr_loss:.4f}
| val_PR-AUC={val_pr:.4f} | val_Brier={val_brier:.4f}")

                if val_pr > best["val_pr"] + 1e-6:
                    best["val_pr"] = val_pr
                    best["epoch"] = ep
                    best["state"] = {k: v.detach().cpu().clone() for k, v in
model.state_dict().items()}
                    wait = 0
                else:
                    wait += 1
                    if wait >= patience:
                        print(f"Early stop at ep {ep}. Best ep {best['epoch']}
PR-AUC {best['val_pr']:.4f}")
                        break

            # restore best
            model.load_state_dict({k: v.to(DEVICE) for k, v in
best["state"].items()})
            model.eval()

            def get_logits(dloader):
                out, yy = [], []
                with torch.no_grad():
                    for xb_num, xb_cat, xb_seq, yb in dloader:
                        xb_num = xb_num.to(DEVICE)
                        xb_cat = xb_cat.to(DEVICE)
                        xb_seq = xb_seq.to(DEVICE)
                        l = forward_logits(model_name, model, xb_num, xb_cat,
xb_seq)
                        out.append(l.detach().cpu().numpy())
                        yy.append(yb.numpy())
                return np.concatenate(out), np.concatenate(yy)

            log_tr, y_tr = get_logits(dl_tr)
            log_va, y_va = get_logits(dl_va)
            log_te, y_te = get_logits(dl_te)

            # temperature calibration (fit on val)
            Tcal = fit_temperature(log_va, y_va)

            p_tr = sigmoid_np(log_tr / Tcal)
            p_va = sigmoid_np(log_va / Tcal)
            p_te = sigmoid_np(log_te / Tcal)

            thr = tune_threshold(y_va, p_va)

            return dict(
                model=model,
                T=Tcal,
                thr=thr,
                val_best_pr=best["val_pr"],
```

```
            logits={"train": log_tr, "val": log_va, "test": log_te},
            probs={"train": p_tr, "val": p_va, "test": p_te},
            y={"train": y_tr, "val": y_va, "test": y_te},
            history=hist
        )

    # -------------------------- PLOTS --------------------------

    def plot_val_brier(RES):
        plt.figure(figsize=(9, 4))
        for name in ["MLP", "TCN", "Hybrid"]:
            plt.plot(RES[name]["history"]["val_brier"], label=name)
        plt.xlabel("Епоха")
        plt.ylabel("Validation Brier")
        plt.title("Зниження помилки на валідації (Brier)")
        plt.legend()
        plt.tight_layout()
        plt.show()

    def line_actual_vs_pred(y, p, window=301, title=""):
        idx = np.argsort(p)
        p_sorted = p[idx]
        y_sorted = y[idx]

        w = min(window, max(5, (len(y) // 100) * 2 + 1))
        w += (w % 2 == 0)
        kern = np.ones(w) / w
        y_smooth = np.convolve(np.pad(y_sorted, (w // 2, w // 2),
mode="edge"), kern, mode="valid")

        plt.figure(figsize=(10, 3))
        plt.plot(p_sorted, label="Predicted P(y=1)")
        plt.plot(y_smooth, label="Actual (rolling mean)")
        plt.ylim(-0.05, 1.05)
        plt.xlabel("Відсортовані приклади")
        plt.ylabel("Ймовірність / частка 1")
        plt.title(title)
        plt.legend()
        plt.tight_layout()
        plt.show()

    def plot_joint_pred(RES, base="Hybrid"):
        base_order = np.argsort(RES[base]["probs"]["test"])
        w = max(5, (len(base_order) // 100) * 2 + 1)
        w += (w % 2 == 0)
        kern = np.ones(w) / w

        y_sorted = RES[base]["y"]["test"][base_order]
        y_smooth = np.convolve(np.pad(y_sorted, (w // 2, w // 2),
mode="edge"), kern, mode="valid")

        plt.figure(figsize=(10, 3))
        for name in ["MLP", "TCN", "Hybrid"]:
            p_sorted = RES[name]["probs"]["test"][base_order]
            plt.plot(p_sorted, label=f"{name} P(y=1)")
        plt.plot(y_smooth, label="Actual (rolling mean)", linestyle="--")
        plt.ylim(-0.05, 1.05)
        plt.xlabel(f"Відсортовані приклади (за {base})")
        plt.ylabel("Ймовірність / частка 1")
        plt.title("Real vs Predicted — спільний графік")
        plt.legend()
        plt.tight_layout()
        plt.show()
```

```
        def error_distribution_plot(name, y, p, thr):
            err = p - y
            plt.figure(figsize=(7, 3))
            plt.hist(err, bins=40, alpha=0.8)
            plt.title(f"Розподіл помилки e=p-y — {name}")
            plt.xlabel("e")
            plt.ylabel("Кількість")
            plt.tight_layout()
            plt.show()

            yhat = (p >= thr).astype(int)
            fp = p[(y == 0) & (yhat == 1)]
            fn = p[(y == 1) & (yhat == 0)]

            plt.figure(figsize=(7, 3))
            plt.hist(fp, bins=30, alpha=0.7, label="FP (y=0, pred=1)")
            plt.hist(fn, bins=30, alpha=0.7, label="FN (y=1, pred=0)")
            plt.xlabel("P(y=1)")
            plt.ylabel("Кількість")
            plt.title(f"FP/FN probability — {name}")
            plt.legend()
            plt.tight_layout()
            plt.show()

        # --------------------------- CV ---------------------------

        def dataloader_from_idx(Xnum, Xcat, Xseq, y, idx_tr, idx_va,
batch=512, num_workers=0):
            dl_tr = DataLoader(OuladDS(Xnum[idx_tr], Xcat[idx_tr],
Xseq[idx_tr], y[idx_tr]),
                               batch_size=batch, shuffle=True, num_workers=0,
pin_memory=PIN)
            dl_va = DataLoader(OuladDS(Xnum[idx_va], Xcat[idx_va],
Xseq[idx_va], y[idx_va]),
                               batch_size=batch, shuffle=False, num_workers=0,
pin_memory=PIN)
            return dl_tr, dl_va

        def cv_run(model_name, make_model, Xnum, Xcat, Xseq, y, groups,
folds=3, epochs=20, pos_weight=None):
            gkf = GroupKFold(n_splits=folds)
            scores = []
            for fold, (itr, iva) in enumerate(gkf.split(Xnum, y,
groups=groups), 1):
                dltr, dlva = dataloader_from_idx(Xnum, Xcat, Xseq, y, itr,
iva, batch=CFG.batch, num_workers=CFG.num_workers)
                m = make_model()
                # для CV "test" не потрібен, але train_model очікує dl_te,
підсунемо dlva як заглушку
                res = train_model(model_name, m, dltr, dlva, dlva,
epochs=epochs, lr=1e-3, patience=5, pos_weight=pos_weight)

                p = res["probs"]["val"]
                yv = res["y"]["val"]
                scores.append(dict(
                    fold=fold,
                    PR_AUC=average_precision_score(yv, p),
                    ROC_AUC=roc_auc_score(yv, p)
                ))
                gc.collect()
                torch.cuda.empty_cache()
            return pd.DataFrame(scores)

        # --------------------------- MAIN ---------------------------
```

```
        def main():
            seed_everything(CFG.seed)

            def load_or_train(name, info, dl_tr, dl_va, dl_te, pos_weight):
              if info["path"].exists():
                print(f"{name}: loading saved model and predicting")
                model = info["class"](*info["args"]).to(DEVICE)
                model.load_state_dict(torch.load(info["path"],
map_location=DEVICE))
                model.eval()
                return {"model": model}
              else:
                print(f"{name}: training")
                res = train_model(name, info["class"](*info["args"]), dl_tr,
dl_va, dl_te,
                                  epochs=info["epochs"], lr=info["lr"],
patience=info["patience"],
                                  pos_weight=pos_weight)
                torch.save(res["model"].state_dict(), info["path"])
                return res

            # --------- model save/load paths ---------
            model_dir = Path("saved_models")
            model_dir.mkdir(exist_ok=True)

            mlp_path = model_dir / "mlp.pth"
            tcn_path = model_dir / "tcn.pth"
            hyb_path = model_dir / "hybrid.pth"

            model_infos = {
            "MLP": {"class": ModelMLP, "path": mlp_path, "args":
(Xnum_tr.shape[1], cat_cards),
                    "epochs": CFG.epochs_mlp, "lr": CFG.lr_mlp, "patience":
CFG.patience_mlp},
            "TCN": {"class": ModelTCN, "path": tcn_path, "args":
(Xseq_tr.shape[1],),
                    "epochs": CFG.epochs_tcn, "lr": CFG.lr_tcn, "patience":
CFG.patience_tcn},
            "Hybrid": {"class": ModelHybrid, "path": hyb_path, "args":
(Xnum_tr.shape[1], cat_cards, Xseq_tr.shape[1]),
                       "epochs": CFG.epochs_hyb, "lr": CFG.lr_hyb, "patience":
CFG.patience_hyb},
        }


            data = load_oulad(CFG)
            assessments = data["assessments"]
            studentAssessment = data["studentAssessment"]
            vle = data["vle"]
            studentVle = data["studentVle"]
            studentInfo = data["studentInfo"]

            # labels + base
            studentInfo, base, cat_cols, num_cols =
prepare_labels_and_base(studentInfo)
            print("Label distribution:\n",
studentInfo["final_result"].value_counts(dropna=False))

            SPLIT = make_split(studentInfo)
            print("SPLIT:", SPLIT)

            # assessment features
```

```
            ass_feats = build_assessment_features(assessments,
studentAssessment, CFG.cut_off_day)
            tab = merge_tabular(base, ass_feats)

            # numeric cols for MLP
            mlp_num_cols = [
                "studied_credits", "num_of_prev_attempts",
                "assess_n_total", "assess_submitted", "assess_frac_sub",
                "assess_weight_sum_norm", "assess_weighted_avg_norm",
                "assess_late_cnt", "assess_mean_late", "assess_max",
"assess_min"
            ]

            # sequence (VLE)
            key_cols = ["id_student", "code_module", "code_presentation"]
            keys_tab = tab[key_cols].drop_duplicates()

            X_seq, act_types = build_vle_sequence(vle, studentVle, keys_tab,
CFG.cut_off_day, CFG.cut_off_week)

            # joint table aligned with X_seq order
            keys_tab2 = keys_tab.copy()
            keys_tab2["_rid"] = np.arange(len(keys_tab2))
            tab_joint = keys_tab2.merge(tab, on=key_cols,
how="left").sort_values("_rid")

            y_all = tab_joint["label"].values.astype(np.int64)

            def mask_by_split(df, split_names):
                return df["code_presentation"].isin(split_names).values

            mask_train = mask_by_split(tab_joint, SPLIT["train"])
            mask_val = mask_by_split(tab_joint, SPLIT["val"])
            mask_test = mask_by_split(tab_joint, SPLIT["test"])

            # categorical encoding (fit on train)
            Xcat_all, cat_maps, cat_cards = build_cat_maps(tab_joint,
cat_cols, mask_train)

            # numeric scaling (fit on train)
            Xnum_all, scaler = build_num_scaled(tab_joint, mlp_num_cols,
mask_train)

            # split
            def take(a, m): return a[m]

            Xnum_tr, Xnum_va, Xnum_te = take(Xnum_all, mask_train),
take(Xnum_all, mask_val), take(Xnum_all, mask_test)
            Xcat_tr, Xcat_va, Xcat_te = take(Xcat_all, mask_train),
take(Xcat_all, mask_val), take(Xcat_all, mask_test)
            Xseq_tr, Xseq_va, Xseq_te = take(X_seq, mask_train), take(X_seq,
mask_val), take(X_seq, mask_test)
            y_tr, y_va, y_te = y_all[mask_train], y_all[mask_val],
y_all[mask_test]

            print("Shapes:",
                  "\n  MLP num:", Xnum_tr.shape, Xnum_va.shape, Xnum_te.shape,
                  "\n  MLP cat:", Xcat_tr.shape, Xcat_va.shape, Xcat_te.shape,
                  "\n  TCN seq:", Xseq_tr.shape, Xseq_va.shape, Xseq_te.shape,
                  "\n  y:", y_tr.shape, y_va.shape, y_te.shape)

            # loaders
            dl_tr, dl_va, dl_te = make_loaders(
                Xnum_tr, Xcat_tr, Xseq_tr, y_tr,
```

```
            Xnum_va, Xcat_va, Xseq_va, y_va,
            Xnum_te, Xcat_te, Xseq_te, y_te,
            CFG
        )

        # pos_weight
        pos = y_tr.sum()
        neg = len(y_tr) - pos
        pos_weight = float(neg / max(pos, 1))
        print("pos_weight:", round(pos_weight, 3))

        # models
        num_dim = Xnum_tr.shape[1]
        in_ch = Xseq_tr.shape[1]

        RES = {}
        for name, info in model_infos.items():
           RES[name] = load_or_train(name, info, dl_tr, dl_va, dl_te,
pos_weight)


        # evaluation summary
        def eval_block(name, pack):
            yte = pack["y"]["test"]
            pte = pack["probs"]["test"]
            m = compute_metrics(yte, pte)
            m["ECE"] = expected_calibration_error(yte, pte)
            m["thr_used"] = pack["thr"]
            yhat_thr = (pte >= pack["thr"]).astype(int)
            m["F1_macro_thr"] = f1_score(yte, yhat_thr, average="macro")
            m["MCC_thr"] = matthews_corrcoef(yte, yhat_thr)
            m["Model"] = name
            return m

        summary = pd.DataFrame([eval_block(k, v) for k, v in
RES.items()]).set_index("Model").sort_index()
        print("\n=== ПІДСУМКОВІ МЕТРИКИ (test) ===\n", summary.round(4))

        # plots
        plot_val_brier(RES)
        for name in ["MLP", "TCN", "Hybrid"]:
            y = RES[name]["y"]["test"]
            p = RES[name]["probs"]["test"]
            line_actual_vs_pred(y, p, title=f"Actual vs Predicted — {name}
(test)")

        plot_joint_pred(RES, base="Hybrid")

        for name in ["MLP", "TCN", "Hybrid"]:
            error_distribution_plot(name, RES[name]["y"]["test"],
RES[name]["probs"]["test"], RES[name]["thr"])

        # CV on train+val
        mask_trval = mask_train | mask_val
        groups_trval = tab_joint.loc[mask_trval,
"code_presentation"].values

        Xnum_tv = Xnum_all[mask_trval]
        Xcat_tv = Xcat_all[mask_trval]
        Xseq_tv = X_seq[mask_trval]
        y_tv = y_all[mask_trval]

        cv_mlp = cv_run("MLP", lambda: ModelMLP(num_dim, cat_cards),
                        Xnum_tv, Xcat_tv, Xseq_tv, y_tv, groups_trval,
```

```
                            folds=CFG.cv_folds, epochs=CFG.cv_epochs,
pos_weight=pos_weight)
            cv_tcn = cv_run("TCN", lambda: ModelTCN(in_ch),
                            Xnum_tv, Xcat_tv, Xseq_tv, y_tv, groups_trval,
                            folds=CFG.cv_folds, epochs=CFG.cv_epochs,
pos_weight=pos_weight)
            cv_hyb = cv_run("Hybrid", lambda: ModelHybrid(num_dim, cat_cards,
in_ch),
                            Xnum_tv, Xcat_tv, Xseq_tv, y_tv, groups_trval,
                            folds=CFG.cv_folds, epochs=CFG.cv_epochs,
pos_weight=pos_weight)

            cv_mlp["Model"] = "MLP"
            cv_tcn["Model"] = "TCN"
            cv_hyb["Model"] = "Hybrid"
            cv_all = pd.concat([cv_mlp, cv_tcn, cv_hyb], ignore_index=True)

            cv_summary = cv_all.groupby("Model")[["PR_AUC",
"ROC_AUC"]].agg(["mean", "std"]).round(4)
            print("\n=== Крос-валідаційна стабільність (train+val, GroupKFold
by presentation) ===")
            print(cv_summary)

            plt.figure(figsize=(7, 4))
            plt.boxplot([cv_mlp["PR_AUC"].values, cv_tcn["PR_AUC"].values,
cv_hyb["PR_AUC"].values],
                        labels=["MLP", "TCN", "Hybrid"])
            plt.title("CV PR-AUC по фолдах")
            plt.tight_layout()
            plt.show()

            # save results
            out_dir = Path("results_oulad")
            out_dir.mkdir(exist_ok=True)

            summary.round(6).to_csv(out_dir / "metrics_summary_test.csv")
            json.dump(
                {name: {"T": RES[name]["T"], "thr": RES[name]["thr"],
"val_best_pr": RES[name]["val_best_pr"]}
                 for name in RES},
                open(out_dir / "calibration_and_thresholds.json", "w",
encoding="utf-8"),
                indent=2
            )

            long_df = pd.concat(
                [pd.DataFrame({"model": n, "y_true": RES[n]["y"]["test"], "p":
RES[n]["probs"]["test"]})
                 for n in RES],
                ignore_index=True
            )
            long_df.to_csv(out_dir / "test_predictions_long.csv", index=False)

            print(f"\nSaved to: {out_dir.resolve()}")

        if __name__ == "__main__":
            main()
```

А.2 Текст файлу predict_all_models.py

```
import sys
from pathlib import Path

# ------------------ Базова директорія (py / exe) ------------------
if getattr(sys, 'frozen', False):
    BASE_DIR = Path(sys.executable).parent
else:
    BASE_DIR = Path(__file__).parent

# ------------------ Директорії ------------------
results_dir = BASE_DIR / "results"
results_dir.mkdir(exist_ok=True)

saved_models_dir = BASE_DIR / "saved_models"
data_dir = BASE_DIR / "data"
results_oulad_dir = BASE_DIR / "results_oulad"
results_oulad_dir.mkdir(exist_ok=True)

import numpy as np
import pandas as pd
import torch
from PyQt6 import QtWidgets
import matplotlib.pyplot as plt
from datetime import datetime

from Diplom_Semenov import (
    CFG, DEVICE, PIN, load_oulad, prepare_labels_and_base,
    merge_tabular, build_assessment_features, build_vle_sequence,
      build_cat_maps,  build_num_scaled,  OuladDS,  ModelMLP,  ModelTCN,
ModelHybrid,
    sigmoid_np, make_split
)
from torch.utils.data import DataLoader

# ------------------ Підготовка директорії для результатів -----------
-------
results_dir = BASE_DIR / "results"
results_dir.mkdir(exist_ok=True)

# ------------------ Завантаження даних ------------------
```

```
        data = load_oulad(CFG)
        assessments = data["assessments"]
        studentAssessment = data["studentAssessment"]
        vle = data["vle"]
        studentVle = data["studentVle"]
        studentInfo = data["studentInfo"]

        # ------------------ Підготовка бази та міток ------------------
        studentInfo,         base,         cat_cols,         num_cols         =
prepare_labels_and_base(studentInfo)
        SPLIT = make_split(studentInfo)
        ass_feats  =  build_assessment_features(assessments,  studentAssessment,
CFG.cut_off_day)
        tab = merge_tabular(base, ass_feats)

        # ------------------ Числові колонки для MLP ------------------
        mlp_num_cols = [
            "studied_credits", "num_of_prev_attempts",
            "assess_n_total", "assess_submitted", "assess_frac_sub",
            "assess_weight_sum_norm", "assess_weighted_avg_norm",
            "assess_late_cnt", "assess_mean_late", "assess_max", "assess_min"
        ]

        # ------------------ Послідовності VLE ------------------
        key_cols = ["id_student", "code_module", "code_presentation"]
        keys_tab = tab[key_cols].drop_duplicates()
        X_seq,  act_types  =  build_vle_sequence(vle,  studentVle,  keys_tab,
CFG.cut_off_day, CFG.cut_off_week)

        # ------------------ Об'єднана таблиця ------------------
        keys_tab2 = keys_tab.copy()
        keys_tab2["_rid"] = np.arange(len(keys_tab2))
        tab_joint          =          keys_tab2.merge(tab,          on=key_cols,
how="left").sort_values("_rid")
        y_all  =  tab_joint["label"].fillna(0).values.astype(np.int64)   #  якщо
немає y_true, ставимо 0
        mask_test = tab_joint["code_presentation"].isin(SPLIT["test"]).values

        # ------------------ Категоріальні та числові дані ------------------
        Xcat_all,  cat_maps,  cat_cards  =  build_cat_maps(tab_joint,  cat_cols,
mask_train=~mask_test)
        Xnum_all,    scaler    =    build_num_scaled(tab_joint,    mlp_num_cols,
mask_train=~mask_test)
```

```
        # ------------------ Завантаження моделей ------------------
        model_dir = BASE_DIR / "saved_models"
        mlp = ModelMLP(Xnum_all.shape[1], cat_cards).to(DEVICE)
        mlp.load_state_dict(torch.load(model_dir / "mlp.pth",
map_location=DEVICE))
        mlp.eval()
        tcn = ModelTCN(X_seq.shape[1]).to(DEVICE)
        tcn.load_state_dict(torch.load(model_dir / "tcn.pth",
map_location=DEVICE))
        tcn.eval()
        hyb = ModelHybrid(Xnum_all.shape[1], cat_cards,
X_seq.shape[1]).to(DEVICE)
        hyb.load_state_dict(torch.load(model_dir / "hybrid.pth",
map_location=DEVICE))
        hyb.eval()

        # ------------------ Функції прогнозу ------------------
        def predict(model, Xnum, Xcat, Xseq, model_name, T=1.0):
            # Створення датасету та DataLoader
            dataset = OuladDS(Xnum, Xcat, Xseq, np.zeros(len(Xnum)))
              dl = DataLoader(dataset, batch_size=CFG.batch, shuffle=False,
pin_memory=PIN)
            out = []
            with torch.no_grad():
                for xb_num, xb_cat, xb_seq, _ in dl:
                    xb_num = xb_num.to(DEVICE)
                    xb_cat = xb_cat.to(DEVICE)
                    xb_seq = xb_seq.to(DEVICE)
                    if model_name == "MLP":
                        logits, _ = model(xb_num, xb_cat, None)
                    elif model_name == "TCN":
                        logits, _ = model(None, None, xb_seq)
                    else:
                        logits, _ = model(xb_num, xb_cat, xb_seq)
                    out.append(logits.cpu().numpy())
            return sigmoid_np(np.concatenate(out) / T)

        def save_predictions(ids, module, presentation, y_true, p_mlp, p_tcn,
p_hyb, filename):
            df_pred = pd.DataFrame({
                "id_student": ids,
                "code_module": module,
```

```
                "code_presentation": presentation,
                "y_true": y_true,
                "MLP": p_mlp,
                "TCN": p_tcn,
                "Hybrid": p_hyb
            })
                    df_pred["Ensemble"]     =     df_pred[["MLP",     "TCN",
"Hybrid"]].mean(axis=1)
            path = results_dir / filename
            df_pred.to_csv(path, index=False)
            return df_pred, path

        def           plot_predictions(df_pred,           models_to_plot=None,
title="Predictions"):
            if models_to_plot is None or len(models_to_plot)==0:
                return None
            plt.figure(figsize=(10,6))
                      for      m,      color      in      zip(models_to_plot,
['skyblue','orange','green','red']):
                plt.plot(df_pred[m].values, label=m, color=color)
            plt.ylabel("Probability")
            plt.title(title)
            plt.legend()
            plt.tight_layout()
                    fig_path     =     results_dir     /     f"{title.replace('
','_')}_{datetime.now().strftime('%Y%m%d_%H%M%S')}.png"
            plt.savefig(fig_path)
            plt.close()
            return fig_path

        # ------------------ GUI ------------------
        class DiplomaGUI(QtWidgets.QWidget):
            def __init__(self):
                super().__init__()

                layout = QtWidgets.QVBoxLayout(self)
                # поле
                self.hidden_input = QtWidgets.QLineEdit()
                self.hidden_input.setVisible(False)
                layout.addWidget(self.hidden_input)

                self.setWindowTitle("Student Success Prediction System")
                self.resize(1000, 750)
```

```
                # Вибір студентів
                group_sel = QtWidgets.QGroupBox("Select Students")
                layout_sel = QtWidgets.QHBoxLayout(group_sel)
                self.radio_specific = QtWidgets.QRadioButton("Specific")
                self.radio_range = QtWidgets.QRadioButton("Range")
                self.radio_random = QtWidgets.QRadioButton("Random")
                self.radio_range.setChecked(True)
                layout_sel.addWidget(self.radio_specific)
                layout_sel.addWidget(self.radio_range)
                layout_sel.addWidget(self.radio_random)
                layout.addWidget(group_sel)

                # Поля вводу
                self.input_specific = QtWidgets.QLineEdit()
                   self.input_specific.setPlaceholderText("Student  ID  (comma-
separated)")
                self.input_range = QtWidgets.QLineEdit()
                self.input_range.setPlaceholderText("Range start,end")
                layout.addWidget(self.input_specific)
                layout.addWidget(self.input_range)

                # Вибір моделей
                group_model = QtWidgets.QGroupBox("Select Models to Plot")
                layout_model = QtWidgets.QHBoxLayout(group_model)
                self.chk_mlp = QtWidgets.QCheckBox("MLP")
                self.chk_tcn = QtWidgets.QCheckBox("TCN")
                self.chk_hyb = QtWidgets.QCheckBox("Hybrid")
                self.chk_ens = QtWidgets.QCheckBox("Ensemble")
                                                    for          chk          in
[self.chk_mlp,self.chk_tcn,self.chk_hyb,self.chk_ens]:
                    chk.setChecked(True)
                    layout_model.addWidget(chk)
                layout.addWidget(group_model)

                # Кнопка прогнозу
                self.btn_predict = QtWidgets.QPushButton("Make Prediction")
                layout.addWidget(self.btn_predict)

                # Таблиця для прогнозів
                self.table = QtWidgets.QTableWidget()
                layout.addWidget(self.table)
```

```
        # Лог
        self.text_log = QtWidgets.QTextEdit()
        self.text_log.setReadOnly(True)
        layout.addWidget(self.text_log)

        self.btn_predict.clicked.connect(self.make_prediction)

    def log(self, msg):
        self.text_log.append(msg)
        QtWidgets.QApplication.processEvents()

    def make_prediction(self):
        # --- Очищення перед новим прогнозом ---
        self.table.clearContents()
        self.table.setRowCount(0)
        self.text_log.clear()

        mask_sel = pd.Series(False, index=tab_joint.index)
        title_type = "Selection"

        if self.radio_specific.isChecked():
            try:
                                    ids   =   [int(i.strip())   for   i   in
self.input_specific.text().split(",")]
                mask_sel = tab_joint["id_student"].isin(ids)
                title_type = "Specific"
            except:
                self.log("Invalid input for Specific.")
                return

        elif self.radio_range.isChecked():
            try:
                            start,   end   =   [int(i.strip())   for   i   in
self.input_range.text().split(",")]
                test_indices = np.where(mask_test)[0]
                mask_sel = pd.Series(False, index=tab_joint.index)
                mask_sel.iloc[test_indices[start:end+1]] = True
                title_type = f"Range_{start}_{end}"
            except:
                self.log("Invalid input for Range.")
                return

        else:  # Random 10
```

```
                                                        rand_ids        =
np.random.choice(tab_joint["id_student"][mask_test], size=10, replace=False)
                    mask_sel = tab_joint["id_student"].isin(rand_ids)
                    title_type = f"Random_{'_'.join(map(str,rand_ids))}"

                # --- Фінальний маск ---
                if self.radio_specific.isChecked():
                  mask = mask_sel  # Specific НЕ обмежується test
                else:
                  mask = mask_test & mask_sel  # Range / Random тільки test

                if mask.sum() == 0:
                  self.log("No students found for this selection.")
                  return

                Xnum_sel = Xnum_all[mask]
                Xcat_sel = Xcat_all[mask]
                Xseq_sel = X_seq[mask]
                y_sel = y_all[mask]

                # Якщо немає даних, створюємо нулі
                if len(Xnum_sel) == 0:
                    Xnum_sel = np.zeros((mask.sum(), Xnum_all.shape[1]))
                if len(Xcat_sel) == 0:
                    Xcat_sel = np.zeros((mask.sum(), Xcat_all.shape[1]))
                if len(Xseq_sel) == 0:
                    Xseq_sel = np.zeros((mask.sum(), X_seq.shape[1]))

                self.log(f"Making predictions for {mask.sum()} students...")

                p_mlp = predict(mlp, Xnum_sel, Xcat_sel, Xseq_sel, "MLP")
                p_tcn = predict(tcn, Xnum_sel, Xcat_sel, Xseq_sel, "TCN")
                p_hyb = predict(hyb, Xnum_sel, Xcat_sel, Xseq_sel, "Hybrid")

                                                          filename      =
f"predictions_{title_type}_{datetime.now().strftime('%Y%m%d_%H%M%S')}.csv"
                df_pred, path = save_predictions(
                    tab_joint["id_student"][mask],
                    tab_joint["code_module"][mask],
                    tab_joint["code_presentation"][mask],
                    y_sel, p_mlp, p_tcn, p_hyb, filename
                )
                self.log(f"Predictions saved to {path}")
```

```
                # Моделі для графіку
                models_to_plot = []
                if self.chk_mlp.isChecked(): models_to_plot.append("MLP")
                if self.chk_tcn.isChecked(): models_to_plot.append("TCN")
                if self.chk_hyb.isChecked(): models_to_plot.append("Hybrid")
                if self.chk_ens.isChecked(): models_to_plot.append("Ensemble")

                if len(models_to_plot) > 0:
                        fig_path = plot_predictions(df_pred, models_to_plot,
title_type)
                    if fig_path:
                        self.log(f"Plot saved to {fig_path}")

                # Заповнюємо таблицю
                self.table.setRowCount(len(df_pred))
                self.table.setColumnCount(len(df_pred.columns))
                self.table.setHorizontalHeaderLabels(df_pred.columns)

                df_show = df_pred.reset_index(drop=True)

                for i in range(len(df_show)):
                  for j, col in enumerate(df_show.columns):
                    val = df_show.iloc[i, j]
                    if isinstance(val, (np.floating, float)):
                       val = f"{val:.6f}"
                    else:
                       val = str(val)
                    self.table.setItem(i, j, QtWidgets.QTableWidgetItem(val))

                # --- Очистка полів вводу ---
                self.input_specific.clear()
                self.input_range.clear()

        if __name__ == "__main__":
            app = QtWidgets.QApplication(sys.argv)
            gui = DiplomaGUI()
            gui.show()
            sys.exit(app.exec())
```

ДОДАТОК Б

Слайди презентації

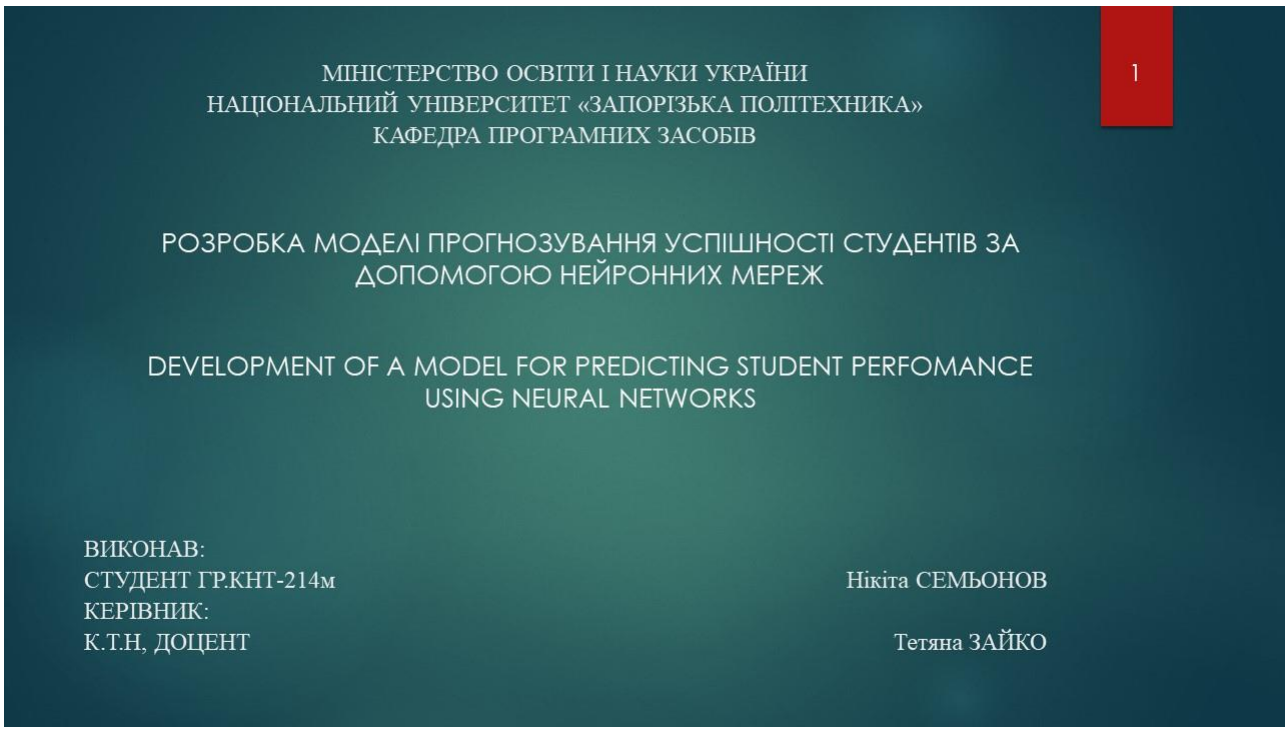

Рисунок В.1 – Титульний слайд

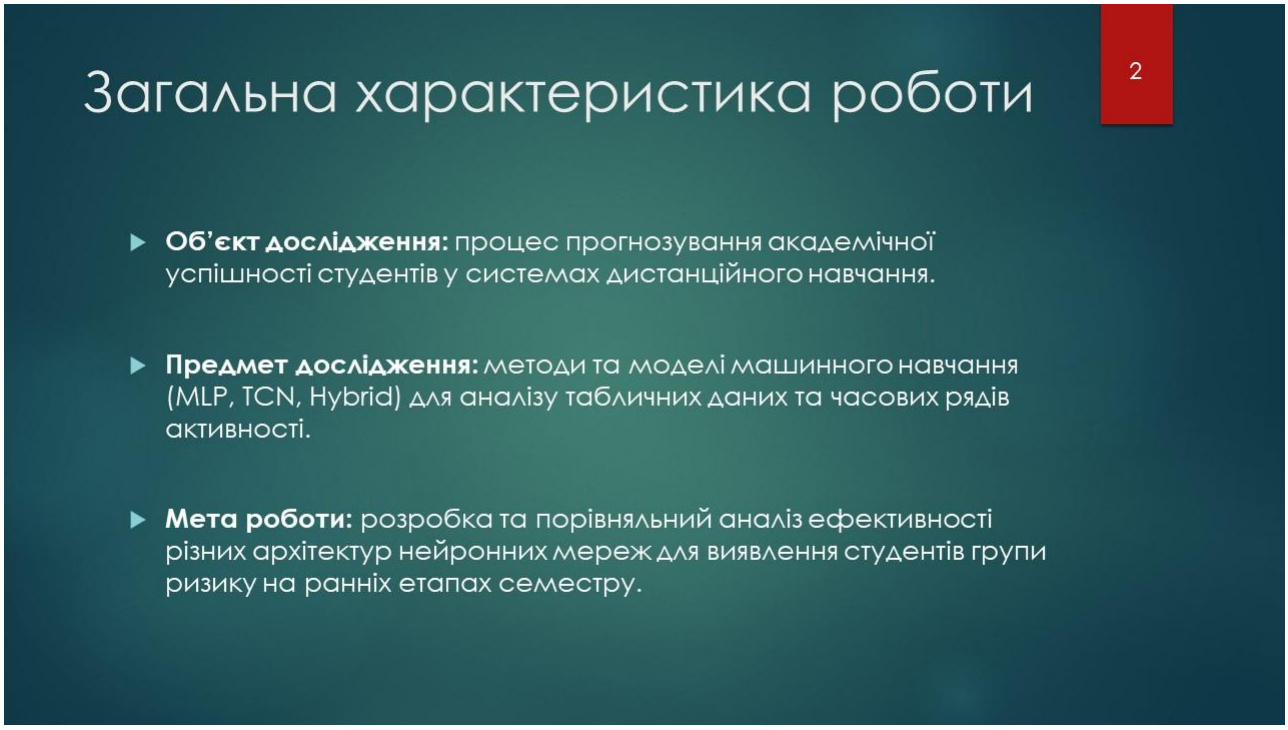

Рисунок В.2 – Об'єкт роботи, предмет та мета

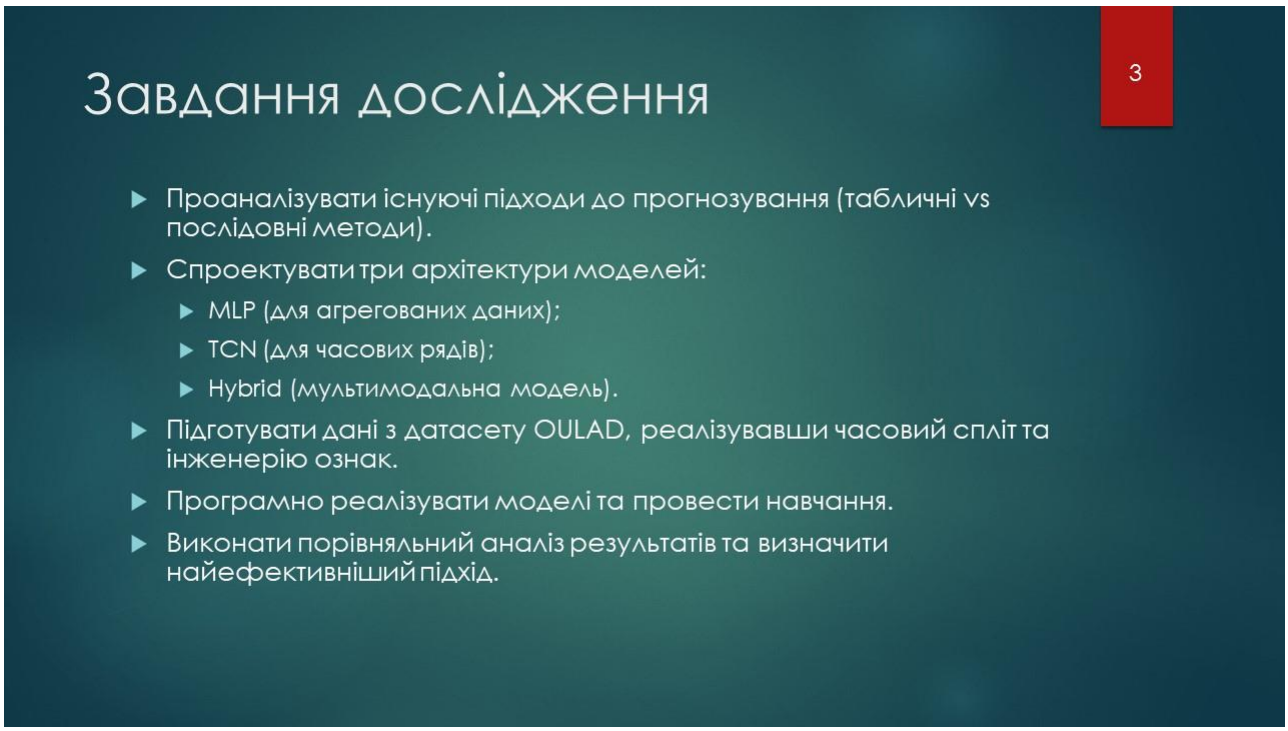

Рисунок В.3 – Завдання дослідження

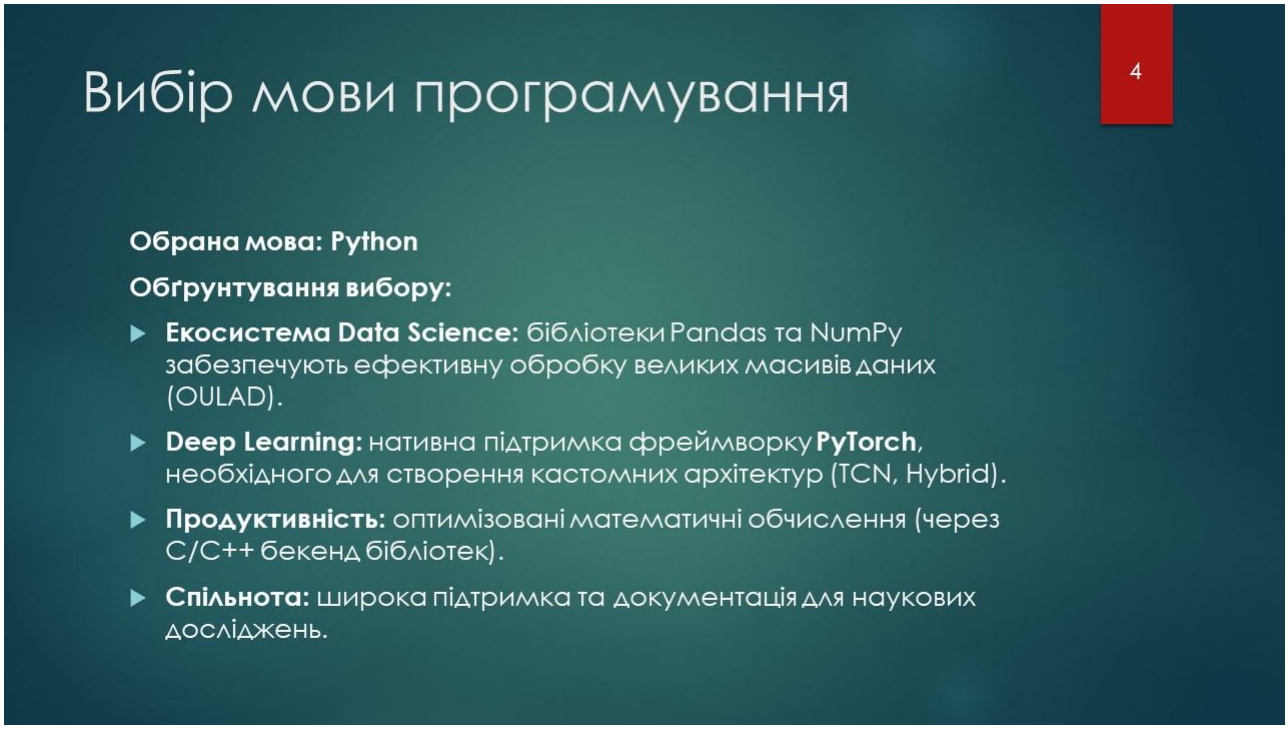

Рисунок В.4 – Вибір мови програмування

Вибір середовища розробки

5

Обране середовище: Visual Studio Code (VS Code)

Переваги для даного проекту:

- **Інтеграція з Jupyter:** можливість поєднувати розвідувальний аналіз (EDA) та модульний код в одному вікні.
- **Remote-SSH:** зручне навчання моделей на віддаленому сервері з GPU.
- **Налагодження:** вбудований дебагер для перевірки розмірності тензорів PyTorch.
- **Легковажність:** швидка робота порівняно з "важкими" IDE (як PyCharm).

Рисунок В.5 – Вибір середовища програмування

Підготовка даних

6

Датасет: Open University Learning Analytics Dataset (OULAD).

Стратегія валідації: часовий спліт (Time-based split):

- Train: 2013B, 2013J
- Val: 2014B
- Test: 2014J **точка зрізу (Cut-off):** 12-й тиждень семестру.

два потоки даних:

1. **Табличні (для MLP):** демографія + Агреговані оцінки (середній бал, % зданих).
2. **Послідовні (для TCN):** логи кліків VLE → Тензор (Batch, 21 канал, 12 тижнів).

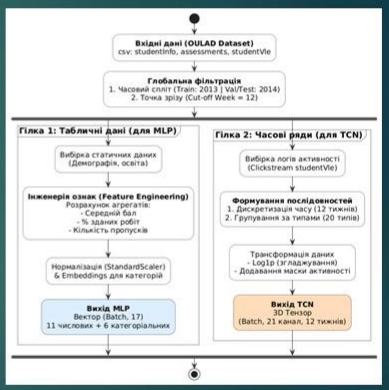

Рисунок В.6 – Підготовка данних

Архітектура моделі MLP

7

Призначення: Baseline на інженерних ознаках. **Вхід:** 11 числових ознак + 6 категоріальних (через Embeddings).

Архітектура:

- Embeddings замість One-Hot Encoding.
- Стек з 3-х блоків: Linear -> BatchNorm -> ReLU -> Dropout
- Вихід: Логіт (ймовірність успіху). **Особливість:** Ефективно працює з агрегованою статистикою успішності.

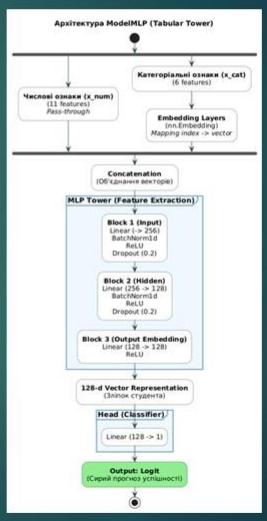

Рисунок В.7 – Архітектура моделі MLP

Архітектура моделі TCN

8

Призначення: Аналіз "сирих" часових рядів (End-to-End). **Вхід:** Тензор активності (21 тип активності × 12 тижнів). **Ключові компоненти:**

- **Causal Convolutions:** запобігають "підгляданню" в майбутнє.
- **Dilated Convolutions:** d=1,2,4,8. Дозволяють охопити весь семестр.
- **Residual Blocks:** для стабільного навчання глибокої мережі.

Рисунок В.8 – Архітектура моделі TCN

9

Гібридна архітектура

Концепція: Мультимодальна модель ("Two-Tower"). **Склад:**

1. **MLP-Tower:** обробляє статичний профіль студента.
2. **TCN-Tower:** обробляє динаміку активності.
3. Блок злиття (Fusion):
   - **Squeeze-and-Excitation (SE):** механізм уваги для зважування важливості модальностей.
   - **Interaction MLP:** вивчає нелінійні зв'язки між оцінками та поведінкою.

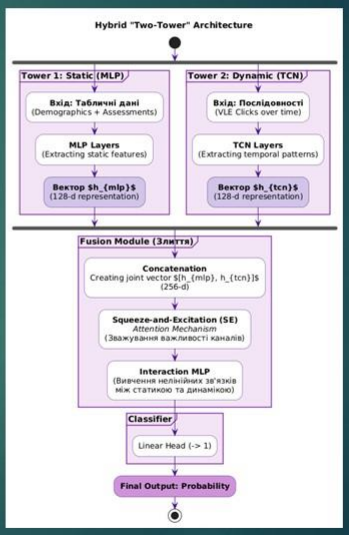

Рисунок В.9 – Мультимодальна модель

10

Функціонування програмного забезпечення – тренування

Аналіз процесу навчання:

- **MLP та Hybrid (Синя/Зелена лінії):** швидка збіжність, стабільне навчання, ефективна робота Early Stopping.
- **TCN (Помаранчева лінія):** відсутність навчання (пряма лінія). Модель не змогла знайти патерни в шумних даних кліків.

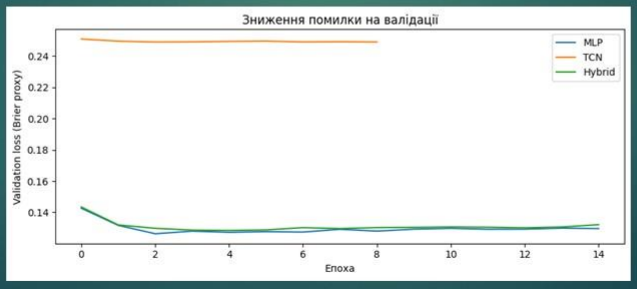

Рисунок В.10 – Тренування

11

Функціонування програмного забезпечення – результати

Ключові метрики (Test Set 2014J):

Модель	ROC-AUC	F1-macro	Висновок
MLP	**0.8651**	**0.7771**	**Висока точність**
Hybrid	0.8671	0.7797	Незначна перевага
TCN	0.5152	0.3264	Провал (рандом)

Рисунок В.11 – Результати моделей

12

ПРИКЛАД РОБОТИ З ПРОГРАМОЮ

ІНТЕРФЕЙС ВЗАЄМОДІЇ КОРИСТУВАЧА З ПРОГРАМНИМ КОМПЛЕКСОМ

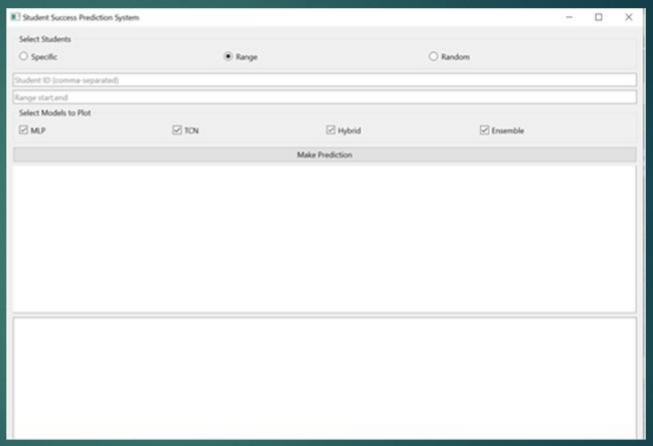

Рисунок В.12 – Інтерфейс програми

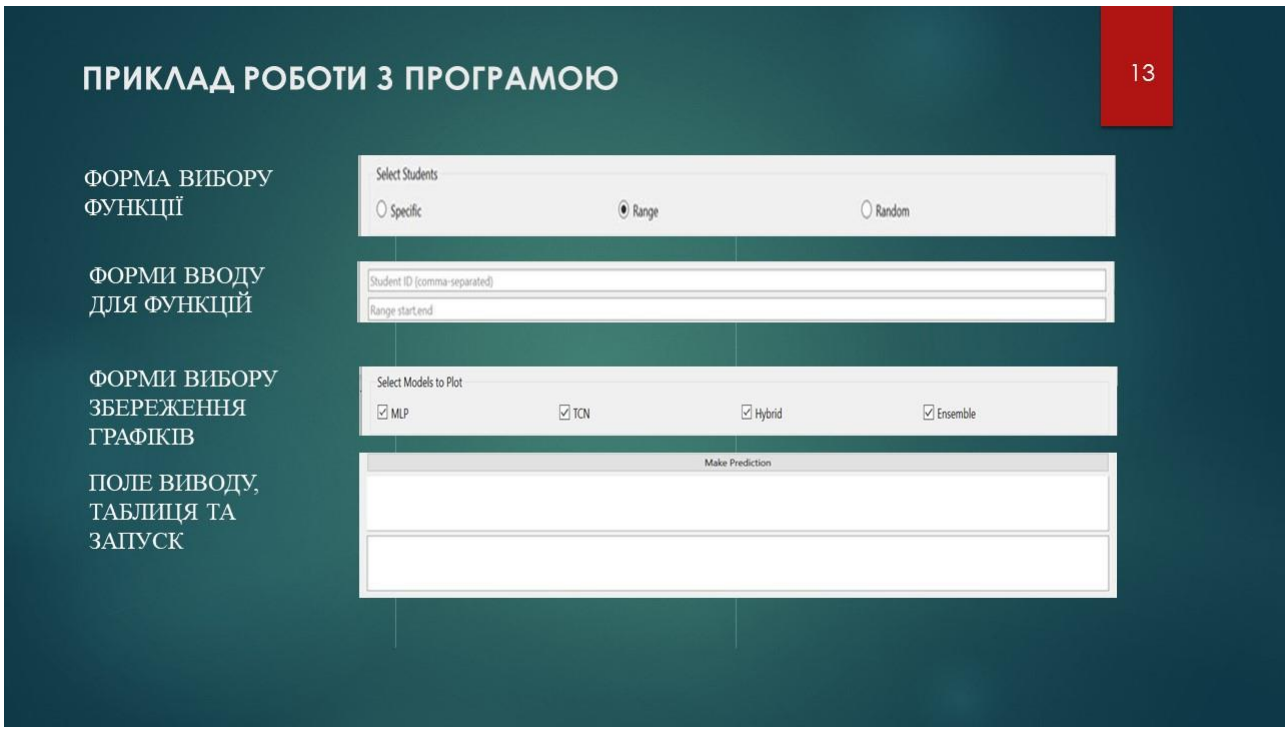

Рисунок В.13 – Взаємодія з програмою

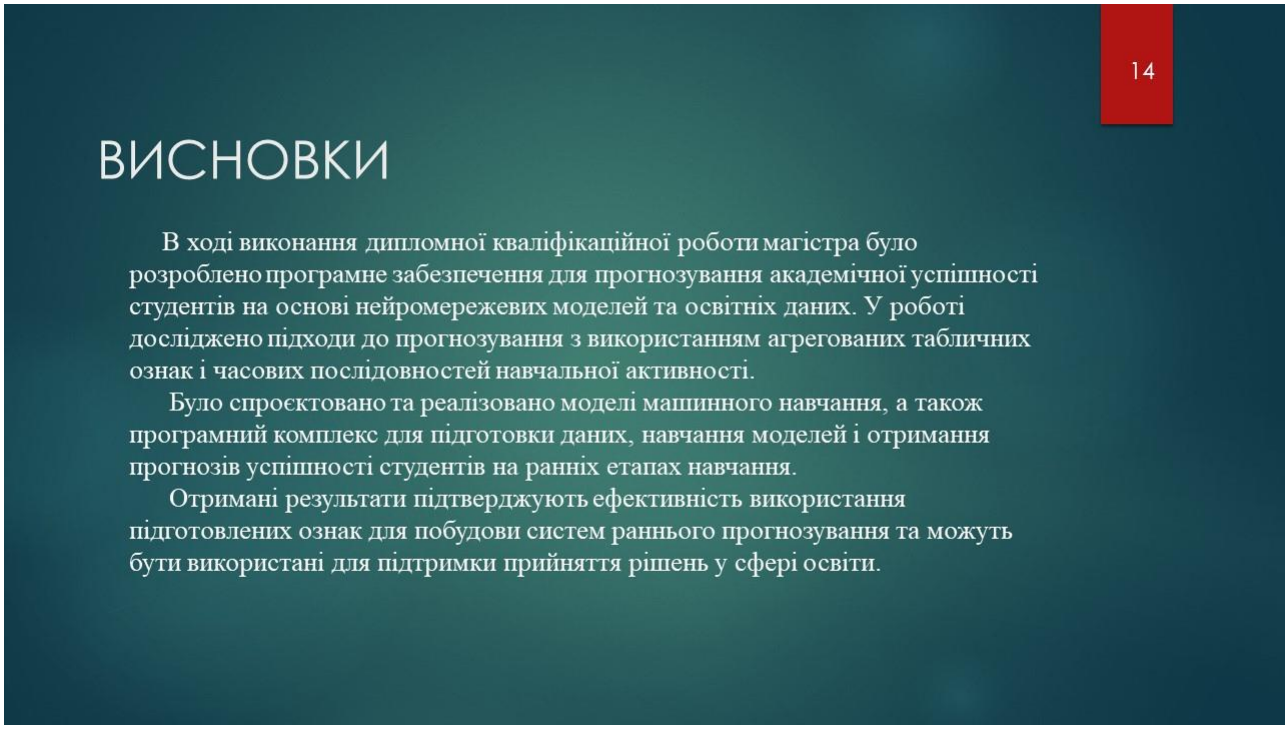

Рисунок В.14 – Висновки