МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В.Н.Каразіна
Факультет математики і інформатики
Кафедра теоретичної та прикладної інформатики

Кваліфікаційна робота

магістр

на тему «Методи машинного навчання та аналіз мереж для малих клінічних наборів даних

Виконав: студент 2 курсу, групи МФ-61
спеціальність 122 «Комп'ютерні науки»
освітньо-наукова програма «Інформатика»

Сорокін Г.О.

Керівник: к. ф. м. н. , доцент Леонов О.С.

Рецензент:

Харків – 2026 року

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

PERMAD	Personalized Marker-driven Early Switch to Aflibercept in Metastatic Colorectal Cancer, проспективне багатоцентрове клінічне дослідження, дані якого використано в роботі.
mCRC	metastatic colorectal cancer, метастатичний колоректальний рак.
CAF	cytokines and angiogenic growth factors, цитокіни та ангіогенні фактори росту.
ML	machine learning, машинне навчання.
RF	Random Forest, алгоритм випадкового лісу.
SVM	Support Vector Machine, метод опорних векторів.
k-NN	k-nearest neighbors, метод k-найближчих сусідів.
CV	cross-validation, крос-валідація.
Nested CV	вкладена крос-валідація, схема оцінювання, у якій зовнішній цикл використовується для тестування моделі, а внутрішній для підбору гіперпараметрів.
Stratified Group K-Fold	стратифікована групова схема крос-валідації, у якій зберігається співвідношення класів і водночас усі записи однієї групи залишаються в межах одного фолду.
TDA	topological data analysis, топологічний аналіз даних.
GNN	graph neural network, графова нейронна мережа.
GAT	graph attention network, графова нейронна мережа з механізмом уваги.
DTW	dynamic time warping, метод динамічного вирівнювання часових послідовностей.
RBF	radial basis function, радіальна базисна функція, що використовується для перетворення відстаней у ваги ребер.
ANOVA	analysis of variance, дисперсійний аналіз.
ICC	intraclass correlation coefficient, внутрішньокласовий коефіцієнт кореляції.
Eta2	eta squared, показник частки дисперсії, пов’язаної з певним груповим фактором.
SS2	середнє арифметичне між Sensitivity та Specificity, використане як збалансована метрика якості класифікації.
IQR	interquartile range, міжквартильний інтервал між Q25 та Q75.
Аналітичний конвеєр	впорядкована послідовність етапів обробки даних, побудови моделі, навчання, оцінювання та аналізу результатів.

Fingerprint-фільтрація	процедура обмеження впливу ознак із високою пацієнт-специфічністю, які можуть поводитися як непрямі ідентифікатори пацієнта.
Lens-функція	функція, що проектує складні багатовимірні об’єкти в простір меншої розмірності для подальшого аналізу їхньої структури.
Velocity-lens	lens-функція, що описує інтенсивність локальної зміни протеомного профілю у часовому вікні.
Трансдуктивне навчання	підхід, за якого структура всього графа доступна під час навчання, однак мітки тестових вузлів не використовуються для оновлення параметрів моделі.

ВСТУП

Актуальність теми. Сучасні клінічні дослідження в галузі персоніфікованої медицини дедалі більше орієнтуються на раннє прогнозування ефективності лікування для конкретного пацієнта. Особливо важливою така задача є в онкології, де своєчасне виявлення ознак терапевтичної резистентності може впливати на вибір подальшої схеми лікування. Лонгітюдний моніторинг біомаркерів дозволяє аналізувати не лише окремий стан пацієнта, а й зміну його біологічного профілю в часі.

Основою цієї дипломної роботи є клінічне дослідження PERMAD (Personalized Marker-driven Early Switch to Aflibercept in Metastatic Colorectal Cancer), зареєстроване в базі ClinicalTrials.gov під ідентифікатором NCT02331927 [2]. Дані та первинний машинно-навчальний аналіз цього дослідження були опубліковані у статті Seufferlein та співавторів, присвяченій прогнозуванню резистентності до терапії бевацизумабом у поєднанні зі схемою FOLFOX у пацієнтів із метастатичним колоректальним раком [1]. У межах цієї дипломної роботи зазначена стаття розглядається як базове дослідження, результати та методологія якого використовуються як відправна точка для критичного аналізу, програмного відтворення та подальшої розробки графової моделі.

Водночас побудова надійних моделей машинного навчання для таких задач є складною. Клінічні набори даних часто мають малу кількість пацієнтів, велику кількість ознак і лонгітюдну структуру спостережень. У таких умовах виникають ризики перенавчання, нестабільності метрик і прихованого витоку даних між навчальною та тестовою вибірками. Особливо критичною ця проблема стає тоді, коли один пацієнт має кілька часових зрізів, які помилково розглядаються як незалежні спостереження.

У цій роботі задача прогнозування терапевтичної резистентності розглядається на основі даних PERMAD та опублікованої методології їхнього

машинно-навчального аналізу [1, 2]. Набір даних містить лонгітюдні протеомні профілі пацієнтів, у яких кожні два тижні виконувалося вимірювання цитокінів та ангіогенних факторів росту. Саме лонгітюдна структура цих даних є ключовою для подальшого методологічного аналізу, оскільки кілька часових зрізів можуть належати одному й тому самому пацієнту.

У базовому дослідженні Seufferlein та співавтори застосували класичні алгоритми машинного навчання, зокрема Random Forest, SVM та k-NN, для прогнозування ризику прогресування захворювання протягом 100 днів [1]. Найкращі результати були отримані для Random Forest, що стало підставою для формування скороченої панелі найбільш важливих біомаркерів. Водночас така постановка задачі потребує додаткової методологічної перевірки, оскільки стандартне розбиття лонгітюдних записів може створювати ризик змішування часових зрізів одного пацієнта між навчальною та тестовою вибірками.У зв’язку з цим у роботі особливу увагу приділено критичному аналізу схеми валідації, контролю пацієнт-специфічних ознак і побудові альтернативного графового представлення даних. Запропонований підхід поєднує групову крос-валідацію, відбір інформативних білків, формування часових вікон, використання lens-функцій і побудову графової нейронної мережі для класифікації часових протеомних профілів.

Мета дослідження – підвищення методологічної надійності оцінювання та прогнозування терапевтичної резистентності за малими лонгітюдними протеомними наборами даних шляхом розробки й програмної реалізації графової моделі, що враховує часову структуру спостережень, ризик витоку даних і обмеження класичних табличних підходів.

Для досягнення поставленої мети необхідно вирішити такі завдання дослідження:

1. Виконати програмне відтворення оригінальної методології для перевірки наявності ризиків витоку даних та перенавчання моделей.

2. Спроектувати виправлений базовий конвеєр класичного машинного навчання із застосуванням схеми 5 × 10 Stratified Group K-Fold для повної ізоляції записів пацієнтів.
3. Розробити алгоритм трансформації табличних медичних даних у графові структури за допомогою методів топологічного аналізу даних, метрик подібності, часових вікон та фільтрації ребер.
4. Спроектувати та навчити архітектуру графової нейронної мережі на побудованих топологічних моделях із використанням байєсівської оптимізації гіперпараметрів.
5. Провести порівняльний аналіз результатів розробленої графової моделі та виправлених класичних алгоритмів за метриками Accuracy, Sensitivity, Specificity та збалансованим показником SS2 = (Sensitivity + Specificity) / 2.

Об’єкт дослідження – процес прогнозування терапевтичної відповіді пацієнтів з онкологічними захворюваннями на основі мультипараметричних лонгітюдних протеомних даних.

Предмет дослідження – методи машинного навчання, графового моделювання, топологічного аналізу даних, групової крос-валідації та оптимізації гіперпараметрів для класифікації малих лонгітюдних клінічних наборів даних.

Методи дослідження. Для розв’язання поставлених завдань у роботі використано методи математичної статистики, дисперсійного аналізу, машинного навчання, топологічного аналізу даних, теорії графів і графових нейронних мереж. Для оцінки пацієнт-специфічності протеомних ознак застосовано показники Ratio, ICC та Eta2. Для побудови базових моделей використано алгоритми Random Forest, SVM та k-NN. Для графового моделювання використано часові вікна, DTW-орієнтовану метрику, velocity-lens, RBF-фільтрацію ребер і графову нейронну мережу з механізмом уваги. Підбір гіперпараметрів виконувався за допомогою Optuna, а програмна реалізація

здійснювалася мовою Python із використанням бібліотек scikit-learn, PyTorch, PyTorch Geometric та NetworkX.

Стислий огляд відомих результатів. У базовій статті за даними PERMAD було показано, що класичні алгоритми машинного навчання можуть використовувати лонгітюдні протеомні профілі для прогнозування терапевтичної резистентності [1]. Найкращі результати були отримані для Random Forest, тоді як SVM і k-NN продемонстрували нижчу якість. Також було запропоновано скорочену панель із десяти біомаркерів, яка зберігала значну частину прогностичної здатності повного набору ознак.

Попри це, результати таких моделей потребують обережної інтерпретації. Для малих лонгітюдних клінічних наборів даних стандартна крос-валідація на рівні окремих записів може призводити до змішування часових зрізів одного пацієнта між навчальною та тестовою вибірками. У такій ситуації підсумкові метрики можуть частково відображати здатність моделі розпізнавати індивідуальні профілі пацієнтів, а не лише загальні закономірності прогресування.

Це створює потребу в аналітичному підході, який одночасно враховує часову структуру даних, групову належність записів до пацієнтів і складні зв’язки між протеомними профілями. Одним із таких напрямів є графове моделювання, у якому окремі часові зрізи можуть розглядатися як вузли, а подібність між локальними часовими траєкторіями – як ребра графа. Поєднання такого представлення з графовими нейронними мережами дозволяє перевірити, чи може мережевий підхід забезпечити стійкішу класифікацію порівняно з класичними табличними моделями.

Відомості про одержані результати та їх новизна. Наукова новизна роботи полягає у поєднанні критичного аналізу методології базового дослідження PERMAD із розробкою графового підходу до аналізу малих лонгітюдних протеомних даних. У роботі показано, що оцінювання моделей без групової

ізоляції пацієнтів може бути надто оптимістичним, оскільки часові зрізи одного пацієнта мають спільні індивідуальні патерни.

Для кількісного обґрунтування цього ризику проведено аналіз пацієнт-специфічності білків за показниками Ratio, ICC та Eta2. Це дозволило виділити ознаки, значна частина варіативності яких пояснюється ідентичністю пацієнта. Такий аналіз використано не для вилучення біологічно важливих маркерів із клінічної інтерпретації, а для побудови методологічно обережнішої схеми моделювання.

У роботі розроблено графову модель Velocity-DTW, яка перетворює часові протеомні профілі у граф подібності. На відміну від класичних табличних моделей, цей підхід використовує не лише поточні значення білків, а й локальну часову історію профілю. Для побудови графа застосовано часові вікна, DTW-орієнтовану метрику, velocity-lens, вибір найближчих сусідів і RBF-фільтрацію ребер.

Практичним результатом роботи є програмна реалізація аналітичного конвеєра мовою Python. Він включає підготовку даних, групову крос-валідацію, відбір ознак, побудову графа, навчання графової нейронної мережі, збереження результатів фолдів і аналіз важливості білків.

Теоретичне та практичне значення результатів. Теоретичне значення роботи полягає в уточненні методологічних вимог до застосування машинного навчання на малих лонгітюдних клінічних даних. Показано, що коректне оцінювання таких моделей потребує не лише стандартної крос-валідації, а й групової ізоляції пацієнтів. Також обґрунтовано роль відбору ознак як елемента регуляризації графового представлення.

Практичне значення роботи полягає у створенні програмного інструментарію для аналізу малих протеомних наборів даних із лонгітюдною структурою. Запропонований конвеєр може бути використаний як основа для подальших біоінформатичних досліджень, у яких необхідно уникати витоку

даних між навчальною та тестовою вибірками, контролювати пацієнт-специфічні ознаки та оцінювати стабільність моделей між фолдами.

Області використання. Розроблена методологія може бути використана у наукових дослідженнях з біоінформатики, медичної інформатики та персоніфікованої медицини. Вона є придатною для задач, де необхідно аналізувати невеликі лонгітюдні набори даних із високою розмірністю ознак, зокрема протеомні, метаболомні або інші мультипараметричні клінічні профілі.

Структура роботи. Магістерська робота складається зі вступу, трьох розділів, загальних висновків, списку використаних джерел та додатків. Основний текст роботи містить 15 таблиць та 5 рисунків; з урахуванням додатків робота містить 26 таблиць та 8 рисунків. Список використаних джерел нараховує 30 найменувань.

РОЗДІЛ 1
ОГЛЯД ДАНИХ, АНАЛІЗ БАЗОВОЇ МОДЕЛІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз набору даних PERMAD та загальної методології базового

Набір даних PERMAD, сформований у межах проспективного багатоцентрового клінічного дослідження Personalized Marker-driven Early Switch to Aflibercept in Metastatic Colorectal Cancer, є основним джерелом даних для цієї роботи [2]. Дослідження проводилося на базі клінічних центрів Німеччини та Австрії. Його метою була перевірка гіпотези про те, що динамічний аналіз цитокінів та ангіогенних факторів росту може використовуватися для раннього прогнозування резистентності до першої лінії терапії у пацієнтів із метастатичним колоректальним раком.

У межах дослідження пацієнти отримували терапію бевацизумабом у поєднанні зі схемою mFOLFOX6. Протягом лікування в них кожні два тижні виконували забір плазми крові. Для кожного зразка визначалися кількісні значення 102 протеомних маркерів, зокрема цитокінів та ангіогенних факторів росту. Радіологічне оцінювання прогресування захворювання виконувалося відповідно до критеріїв RECIST 1.1 [3].

Початково до дослідження було залучено 50 пацієнтів. Для фінального CAF-аналізу автори [1] використали дані 41 пацієнта. Загальний масив даних містив 652 часові зрізи, кожен із яких відповідав окремому моменту спостереження за пацієнтом. Основні характеристики набору даних PERMAD наведено в таблиці 1.1.

Методологія базового дослідження ґрунтувалася на перетворенні лонгітюдних протеомних даних у задачу бінарної класифікації. Ключовим елементом такого перетворення було використання 100-денного часового вікна до моменту радіологічного підтвердження прогресування захворювання. Зразки, отримані в межах цього вікна, відносилися до класу прогресування, тоді як

зразки, відібрані раніше, розглядалися як такі, що не належать до класу прогресування в заданому часовому горизонті [1].

Таблиця 1.1 – Основні характеристики набору даних PERMAD

Характеристика	**Значення**
Тип дослідження	Проспективне багатоцентрове клінічне дослідження
Кількість клінічних центрів	15
Загальна кількість пацієнтів	50
Кількість пацієнтів у фінальному CAF-аналізі	41
Кількість часових зрізів	652
Кількість протеомних маркерів	102
Періодичність забору крові	Кожні 2 тижні
Терапевтична схема	Бевацизумаб + mFOLFOX6

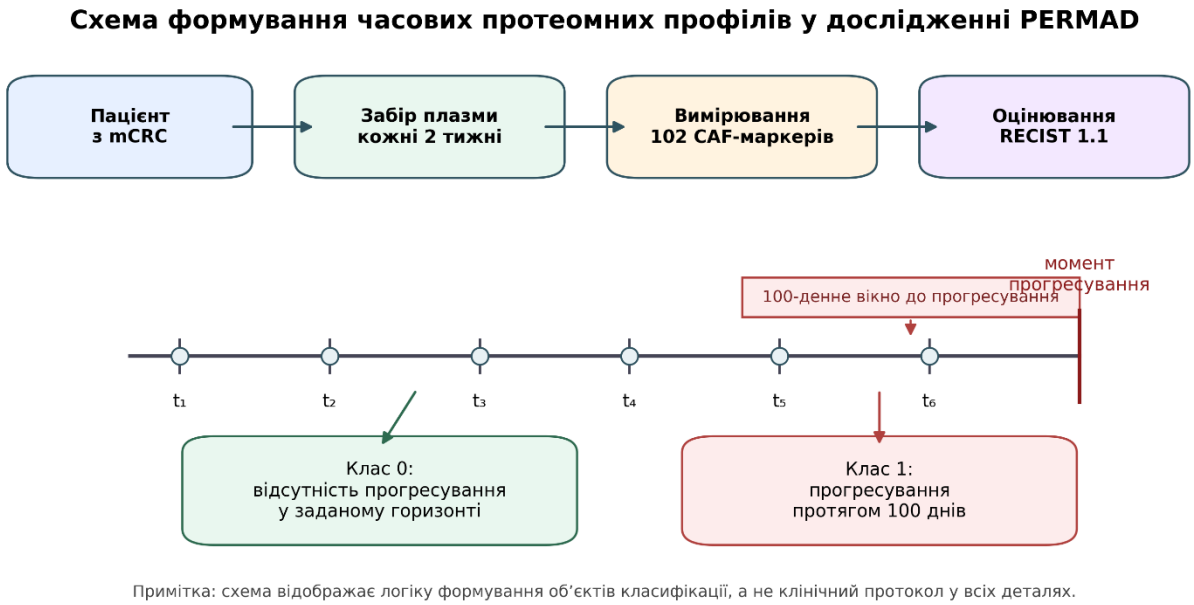


Рисунок 1.1 −Схема формування часових протеомних профілів у дослідженні PERMAD

Такий підхід дозволяє сформувати задачу раннього прогнозування терапевтичної резистентності. Фактично модель має визначити, чи свідчить поточний протеомний профіль пацієнта про наближення прогресування захворювання в межах приблизно трьох місяців. Це є клінічно значущою постановкою задачі, оскільки раннє виявлення неефективності терапії може бути підставою для перегляду лікувальної стратегії.

Для класифікації автори базового дослідження порівнювали три алгоритми машинного навчання: Random Forest, SVM та k-NN [1]. Оцінювання виконувалося за схемою вкладеної крос-валідації 5 × 10. Зовнішній цикл використовувався для оцінювання якості моделей на тестових даних, а внутрішній – для підбору гіперпараметрів на навчальній вибірці.

Найкращі результати в оригінальному дослідженні показав алгоритм Random Forest. На повному наборі зі 102 протеомних маркерів модель досягла тестової Accuracy 80.8%, Sensitivity 71.0%, Specificity 87.4% та SS2 79.2.

Таблиця 1.2 – Тестові метрики моделей в оригінальному дослідженні PERMAD

Модель	**Accuracy**	**Sensitivity**	**Specificity**	**SS2**
RF Full	80.8 [78.2; 85.2]	71.0 [63.8; 78.0]	87.4 [79.8; 91.3]	79.2 [76.9; 82.0]
RF Top-10	76.7 [70.4; 81.4]	70.0 [59.3; 77.5]	81.2 [72.9; 88.2]	75.6 [71.2; 79.7]
SVM	54.6 [49.5; 59.2]	48.0 [40.0; 56.4]	59.0 [51.4; 67.1]	52.7 [47.3; 57.8]
k-NN	59.3 [55.0; 64.8]	39.4 [33.0; 46.1]	73.4 [65.7; 80.0]	56.3 [52.9; 60.9]

На основі Random Forest автори також сформували скорочену панель із десяти найбільш важливих біомаркерів. Така панель мала практичну цінність, оскільки зменшувала кількість потенційно потрібних лабораторних вимірювань і спрощувала подальшу клінічну інтерпретацію моделі. Провідні біомаркери, наведені в оригінальному дослідженні, подано в таблиці 1.3.[Для скороченої Top-10 панелі біомаркерів Random Forest зберіг близький рівень якості: Accuracy

76.7%, Sensitivity 70.0%, Specificity 81.2% та SS2 75.6. Результати SVM і k-NN були нижчими, що вказує на більшу стійкість ансамблевого підходу в умовах малого високовимірного набору даних [1].

Таблиця 1.3 – Top-10 біомаркерів оригінального дослідження PERMAD за важливістю Random Forest

Позиція	**Біомаркер**
1	Antileukoproteinase (ALP)
2	Vascular endothelial growth factor receptor 3 (VEGFR-3)
3	Hepsin
4	Tissue type plasminogen activator (tPA)
5	Insulin-like Growth Factor-Binding Protein 1 (IGFBP-1)
6	Myeloperoxidase (MPO)
7	Ferritin (FRTN)
8	Pepsinogen I (PGI)
9	Immunoglobulin M (IgM)
10	EN-RAGE

Попри високу якість Random Forest у базовому дослідженні, сама структура набору PERMAD створює важливе методологічне питання. Окремі записи в датасеті не є повністю незалежними спостереженнями, оскільки кілька часових зрізів можуть належати одному й тому самому пацієнту. Якщо під час крос-валідації такі зрізи потрапляють одночасно до навчальної та тестової вибірок, модель може частково використовувати індивідуальні особливості пацієнта як непрямий сигнал для класифікації.

Саме це обмеження є ключовою підставою для подальшого критичного аналізу базової методології. У цій роботі результати оригінального дослідження розглядаються не як помилкові, а як такі, що потребують перевірки в суворішій

схемі валідації. Власне відтворення негрупової схеми, оцінювання класичних моделей із груповою ізоляцією пацієнтів і порівняння з графовою моделлю наведено в Розділі 3.

1.2 Теоретичний огляд та обґрунтування вибору базових алгоритмів класифікації

1.2.1 Ансамблеве навчання на основі методу випадкового лісу

Алгоритм випадкового лісу є одним із поширених підходів до класифікації малих високовимірних наборів даних [4]. Його основна ідея полягає в побудові ансамблю дерев рішень, кожне з яких навчається на випадково сформованій підмножині об’єктів і використовує випадкову підмножину ознак під час вибору розбиттів. Підсумкове рішення моделі формується шляхом агрегації результатів окремих дерев.

Для клінічних протеомних даних така властивість є важливою, оскільки кількість вимірюваних біомаркерів може бути значною, а кількість пацієнтів залишається обмеженою. У таких умовах окремі ознаки можуть бути шумовими, надлишковими або надто залежними від індивідуального профілю пацієнта. Ансамблевий підхід частково зменшує вплив таких нестабільних ознак за рахунок усереднення рішень багатьох дерев.

Додатковою перевагою Random Forest є його відносна нечутливість до масштабування ознак. На відміну від метричних моделей, випадковий ліс виконує розбиття за пороговими значеннями окремих змінних, тому різниця в абсолютних шкалах вимірювання білків не має такого прямого впливу на результат класифікації. Це робить Random Forest зручним базовим алгоритмом для аналізу протеомних профілів, де концентрації різних маркерів можуть вимірюватися в різних діапазонах.

Ще однією важливою властивістю Random Forest є можливість оцінювання важливості ознак [4]. У базовому дослідженні PERMAD ця властивість використовувалася для формування скороченої панелі біомаркерів [1]. Такий

підхід має практичну цінність, оскільки дозволяє перейти від повного набору з великої кількості білків до меншої групи маркерів, потенційно придатної для подальшої клінічної перевірки.

Водночас інтерпретацію важливості ознак у Random Forest слід виконувати обережно. Висока важливість білка в моделі не завжди означає його самостійну біологічну значущість. Вона може залежати від кореляцій між ознаками, структури вибірки, способу валідації та наявності пацієнт-специфічних патернів. У зв'язку з цим Random Forest у цій роботі розглядається як еталонний класичний алгоритм, результати якого використовуються для порівняння з іншими моделями, зокрема із запропонованим графовим підходом.

1.2.2 Метод опорних векторів та проблема масштабування ознак

Метод опорних векторів спрямований на побудову розділяючої гіперплощини, яка максимально відокремлює об'єкти різних класів у просторі ознак [5]. У лінійному варіанті модель шукає таке розділення, за якого геометричний зазор між класами є найбільшим. Це робить SVM теоретично привабливим інструментом для задач двокласової класифікації.

Однак ефективність SVM суттєво залежить від підготовки простору ознак. Якщо різні білки мають різні масштаби концентрацій або різну дисперсію, модель може надавати непропорційно велику вагу ознакам із більшими абсолютними значеннями. Для протеомних даних це є суттєвим обмеженням, оскільки різні маркери можуть відрізнятися за діапазоном значень і рівнем шуму.

У задачі прогнозування терапевтичної резистентності SVM корисний як базовий алгоритм для перевірки того, чи можна розділити класи в лінійному або близькому до лінійного просторі ознак. Якщо така модель показує нестабільні результати, це може свідчити про складнішу структуру даних, наявність нелінійних залежностей або недостатність статичного табличного представлення.

У даній роботі SVM використовується як один із класичних еталонних підходів який спирається на геометрію простору ознак. Запропонована у даній дипломній роботі графова модель також використовує геометричний підхід до даних із впровадженням відстані та подібності між часовими профілями (але робить це вже в контексті локальної динаміки та структури графа), Тому результати нашої графової моделі цікаво порівняти не лише з ансамблевим Random Forest, а й із SVM.

1.2.3 Метричний підхід на основі алгоритму k-найближчих сусідів

Алгоритм k-найближчих сусідів реалізує просту ідею метричної класифікації: новий об’єкт відноситься до того класу, який переважає серед його найближчих сусідів у просторі ознак [6]. На відміну від Random Forest або SVM, k-NN фактично не будує складної параметричної моделі, а спирається на локальну структуру відстаней між об’єктами.

Цей підхід є корисним для порівняння, оскільки він прямо показує, наскільки інформативною є геометрія початкового простору ознак. Якщо об’єкти одного класу утворюють компактні групи, k-NN може працювати достатньо добре. Якщо ж відстані між об’єктами визначаються шумом, масштабом окремих білків або пацієнт-специфічними ознаками, якість такого методу швидко знижується.

Для високовимірних протеомних даних k-NN має суттєве обмеження, пов’язане з проблемою зниження інформативності відстаней у просторах великої розмірності [19]. У такому просторі найближчі та найдальші сусіди можуть відрізнятися менш виразно, ніж у просторах малої розмірності. Для малих клінічних вибірок це особливо проблемно, оскільки кількість незалежних пацієнтів є обмеженою.

У межах цієї роботи k-NN використовується як найпростіший метричний baseline. Його результати дають змогу оцінити, чи достатньо звичайного порівняння табличних протеомних профілів для прогнозування терапевтичної

резистентності. Порівняння з графовою моделлю є важливим, оскільки в обох випадках використовується поняття близькості між об’єктами, але в запропонованому в цій дипломній роботі графовому підході ця близькість визначається з урахуванням часових вікон і локальної динаміки.

1.2.4 Обґрунтування вибору метрик оцінювання якості

Для оцінювання моделей у роботі використовуються Accuracy, Sensitivity, Specificity та SS2, що відповідає логіці звітування в базовому дослідженні PERMAD [1]. Такий набір метрик дозволяє оцінити не лише загальну частку правильних класифікацій, а й баланс між виявленням випадків прогресування та правильним розпізнаванням випадків без прогресування у заданому часовому вікні.

Accuracy показує загальну частку правильно класифікованих об’єктів. Однак у клінічних задачах ця метрика не завжди є достатньою, оскільки вона може приховувати дисбаланс між класами. Наприклад, модель може демонструвати відносно високу загальну точність, але погано виявляти пацієнтів із ризиком прогресування.

Sensitivity характеризує здатність моделі правильно ідентифікувати випадки прогресування. Для задачі прогнозування терапевтичної резистентності ця метрика має важливе клінічне значення, оскільки пропуск пацієнта з наближенням прогресування може призвести до продовження неефективної терапії.

Specificity, навпаки, відображає здатність моделі правильно розпізнавати випадки без прогресування в заданому часовому горизонті. Висока специфічність важлива для того, щоб уникати помилкових сигналів про неефективність лікування в пацієнтів, які ще мають клінічну користь від поточної терапії.

Метрика SS2 використовується як збалансований показник, що дорівнює середньому між Sensitivity та Specificity. У цій роботі саме SS2 розглядається як

основна узагальнювальна метрика, оскільки вона дозволяє оцінити баланс між двома типами класифікаційних рішень.

$$SS2 = \frac{(Sensitivity + Specificity)}{2} \quad (1.1)$$

Sensitivity – частка правильно визначених випадків прогресування;

Specificity – частка правильно визначених випадків без прогресування.

1.2.5 Методологія оптимізації та вибір програмного інструментарію

Програмна реалізація експериментів виконувалася мовою Python із використанням бібліотек scikit-learn, PyTorch, PyTorch Geometric, NetworkX та Optuna. Такий набір інструментів дозволяє поєднати класичні алгоритми машинного навчання, графове моделювання, навчання графових нейронних мереж і автоматизований підбір гіперпараметрів у межах єдиного аналітичного конвеєра [7–11, 29].

Бібліотека scikit-learn використовувалася для реалізації класичних моделей машинного навчання, схем крос-валідації, попередньої обробки ознак і розрахунку базових метрик [7]. PyTorch і PyTorch Geometric застосовувалися для побудови та навчання графової нейронної мережі [9, 10]. NetworkX використовувався для аналізу структурних характеристик графів [11], а Optuna – для оптимізації гіперпараметрів моделей [8].

У базовому дослідженні для підбору параметрів використовувався сітковий пошук [1]. У цій роботі для власних експериментів застосовано Optuna, що дозволяє виконувати більш гнучкий підбір гіперпараметрів у неперервних і змішаних просторах пошуку [8]. Це особливо важливо для графової моделі, де якість результату залежить не лише від параметрів нейронної мережі, а й від способу формування графа, кількості сусідів, фільтрації ребер і вагових коефіцієнтів.

Водночас оптимізація гіперпараметрів не розглядається як спосіб штучного підлаштування моделі під бажаний результат. У роботі вона використовується як контрольований інструмент пошуку працездатної конфігурації в межах заздалегідь визначеної схеми валідації. Основним критерієм надійності результатів залишається якість на тестових фолдах із груповою ізоляцією пацієнтів.

1.3 Критичний аналіз методології оригінального дослідження

1.3.1 Обмеження подання лонгітюдних даних як незалежних записів

Головною особливістю набору даних PERMAD є його лонгітюдна структура. Один пацієнт представлений не одним спостереженням, а послідовністю часових зрізів, отриманих протягом лікування. Такі записи не можна повністю розглядати як незалежні об’єкти, оскільки вони мають спільне походження, пов’язані з одним організмом і можуть містити стійкі індивідуальні патерни.

У базовому дослідженні кожен часовий зріз використовувався як окремий об’єкт класифікації [1]. Це спрощує постановку задачі й дозволяє застосувати стандартні алгоритми машинного навчання. Водночас така постановка створює ризик того, що модель навчається не лише на загальних закономірностях прогресування, а й на індивідуальних особливостях пацієнта, які повторюються в усіх його часових зрізах.

У задачах машинного навчання така ситуація пов’язана з ризиком витоку даних. Data leakage виникає тоді, коли інформація, недоступна в реальній тестовій ситуації, опосередковано потрапляє до процесу навчання або оцінювання моделі [12, 13]. Для лонгітюдних медичних даних одним із типових джерел такого витоку є некоректне розбиття, за якого записи одного й того самого пацієнта можуть одночасно опинитися в навчальній та тестовій вибірках.

У такій схемі модель може показувати високі метрики не лише тому, що вона виявила загальний біологічний сигнал терапевтичної резистентності, а й

тому, що вона частково розпізнає індивідуальний профіль пацієнта. Це особливо небезпечно для малих клінічних вибірок, де кількість пацієнтів невелика, а кількість часових зрізів може створювати ілюзію значно більшого обсягу даних.

1.3.2 Ризик змішування часових зрізів одного пацієнта між train і test

У стандартній крос-валідації розбиття зазвичай виконується на рівні окремих записів. Якщо кожен часовий зріз пацієнта вважається незалежним об’єктом, то частина його записів може потрапити до навчального фолду, а інша частина – до тестового. Формально модель не бачить тестовий запис під час навчання, однак вона вже могла бачити інші записи того самого пацієнта.

Для задачі прогнозування нового пацієнта така схема є надто оптимістичною. У реальному сценарії модель має оцінювати пацієнта, біологічний профіль якого не був присутній у навчальних даних. Тому методологічно коректнішим підходом є групова крос-валідація, де всі записи одного пацієнта залишаються в межах одного фолду. Саме такий принцип реалізується в GroupKFold або StratifiedGroupKFold [16].

У випадку PERMAD важливо не лише ізолювати пацієнтів, а й зберігати баланс класів між фолдами. Тому групова стратифікована схема є природним вибором: вона намагається одночасно не змішувати групи й підтримувати співвідношення класів. Це не усуває всі проблеми малої вибірки, але зменшує ризик прямого або непрямого витоку інформації між навчальною та тестовою частинами.

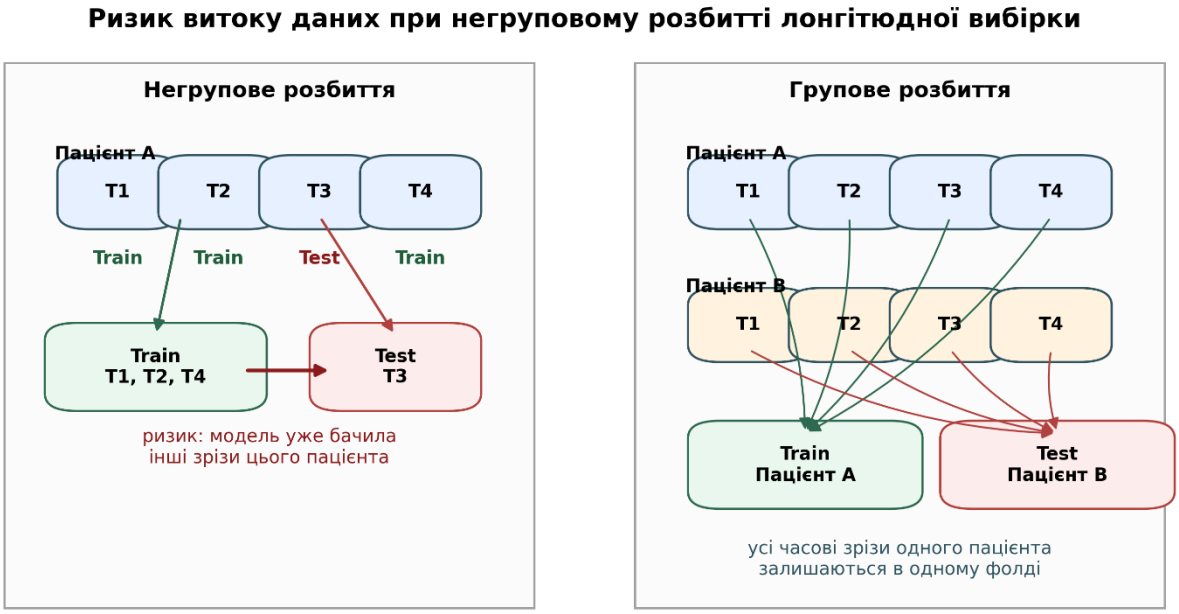


Рисунок 1.2 – Ризик витоку даних при негруповому розбитті лонгітюдної вибірки

1.3.3 Пацієнт-специфічність білків як джерело непрямого витоку

Окремим джерелом методологічного ризику є пацієнт-специфічність окремих білків. Якщо значення певного маркера значною мірою визначається не клінічним станом, а індивідуальними особливостями пацієнта, така ознака може поводитися як непрямий ідентифікатор. У поєднанні з негруповою крос-валідацією це посилює ризик завищення якості моделі.

Для кількісної оцінки цього ефекту в роботі було проведено розвідувальний аналіз пацієнт-специфічності протеомних ознак. Аналіз виконувався після логарифмічного перетворення значень білків, що дозволяло зменшити вплив різниці масштабів. Для оцінювання використовувалися три показники: Ratio, ICC та Eta2.

Показник Ratio відображає співвідношення між міжпацієнтною та внутрішньопацієнтною варіативністю. Високе значення Ratio означає, що середні рівні білка між пацієнтами відрізняються сильніше, ніж значення цього

білка всередині одного пацієнта. У контексті цієї роботи такий білок може бути потенційно пацієнт-специфічним.

Внутрішньокласовий коефіцієнт кореляції ICC використовується для оцінювання частки варіативності, пов'язаної з належністю спостережень до однієї групи [14]. У цій роботі групою є пацієнт. Якщо ICC має високе значення, це означає, що записи одного пацієнта є між собою більш подібними, ніж записи різних пацієнтів.

Eta2 використовується як показник частки загальної дисперсії, що пояснюється певним груповим фактором [15]. У цьому випадку груповим фактором є ідентичність пацієнта. Високе Eta2 означає, що значна частка дисперсії білка пояснюється тим, якому саме пацієнту належить запис.

Основні результати аналізу наведено в таблиці 1.4. Повний програмний код розрахунку Ratio, ICC та Eta2 наведено в додатку А.

Таблиця 1.4 – Оцінка пацієнт-специфічності окремих протеомних маркерів

Protein	**Ratio**	**ICC**	**Eta2**	**Status [Ratio, ICC, Eta]**	**IdentifierScore**
Interleukin-1 beta (IL-1 beta)	2.6768	0.9548	0.9565	+ + +	3
Receptor tyrosine-protein kinase erbB-3 (ErbB3)	2.9236	0.8531	0.8593	+ + +	3
MHC class I chain-related protein A (MICA)	3.9016	0.8306	0.8397	+ + +	3
Serum Amyloid A Protein (SAA)	2.9779	0.6901	0.7107	+ – –	1

Protein	Ratio	ICC	Eta2	Status [Ratio, ICC, Eta]	IdentifierScore
Fatty Acid-Binding Protein, liver (FABP, liver)	2.2927	0.6479	0.6658	+ – –	1
Leucine-rich alpha-2-glycoprotein (LRG1)	2.2070	0.6149	0.6396	+ – –	1
EN-RAGE	4.6635	0.6078	0.6347	+ – –	1
Amphiregulin (AR)	2.7318	0.4861	0.5160	+ – –	1
Platelet-Derived Growth Factor BB (PDGF-BB)	3.4098	0.4420	0.4804	+ – –	1
Matrix Metalloproteinase-9 (MMP-9)	2.6934	0.4418	0.4796	+ – –	1
Neutrophil Activating Peptide 2 (NAP-2)	2.7548	0.4227	0.4624	+ – –	1
Pulmonary and Activation-Regulated Chemokine (PARC)	2.2398	0.3450	0.3890	+ – –	1
Interleukin-12 Subunit p70 (IL-12p70)	inf	0.0000	0.0000	+ – –	1
Interleukin-15 (IL-15)	inf	0.0000	0.0000	+ – –	1

Продовження таблиці 1.4

Примітка. “+” означає перевищення відповідного порогу пацієнт-специфічності: Ratio > 2.0, ICC > 0.8 або Eta2 > 0.8. Таблицю сформовано за результатами власного розрахунку; код наведено в додатку А.

Результати таблиці 1.4 показують, що частина білків має виражену залежність від ідентичності пацієнта. Зокрема, IL-1 beta, ErbB3 та MICA мають високі значення ICC та Eta2, що свідчить про значну частку міжпацієнтної варіативності. Для таких маркерів існує ризик, що в негруповій схемі модель може використовувати їх не лише як клінічні ознаки, а й як непрямі маркери належності запису до конкретного пацієнта.

Важливо підкреслити, що наявність високої пацієнт-специфічності не означає автоматичної біологічної незначущості маркера. Такий білок може мати клінічний сенс. Проте для задачі оцінювання прогностичної моделі він створює додатковий ризик, якщо один і той самий пацієнт представлений і в навчальній, і в тестовій частинах. Тому ці результати використовуються як аргумент на користь групової валідації та контрольованого відбору ознак.

1.3.4 Обмеження звітності метрик у малих вибірках

У базовому дослідженні PERMAD підсумкові метрики подавалися у форматі: «середнє [Q25; Q75]» [1]. Такий формат було збережено і в цій роботі для прямої зіставності результатів. Водночас він має певне обмеження: міжквартильний інтервал зазвичай асоціюється з медіаною, тоді як середнє значення може бути чутливішим до асиметрії розподілу та окремих нестабільних фолдів.

Для великих наборів даних така різниця часто є незначною. Однак у малих клінічних вибірках, де кількість незалежних пацієнтів обмежена, окремі фолди можуть мати помітний вплив на підсумкові оцінки. Тому середнє значення разом із Q25–Q75 слід інтерпретувати не як повний опис розподілу якості моделі, а як компактну форму звітності, придатну для порівняння з оригінальною статтею.

У цій роботі основна увага приділяється не лише абсолютному значенню метрики, а й зміні результатів між різними схемами валідації. Саме порівняння негрупової та групової схем дозволяє оцінити, наскільки результати моделі залежать від способу розбиття лонгітюдних записів.

1.3.5 Обмеження класичного табличного представлення

Класичні моделі машинного навчання працюють із кожним часовим зрізом як з окремим вектором ознак. Такий підхід дозволяє застосувати добре відомі алгоритми, але він слабко враховує послідовну природу даних. У випадку PERMAD кожен пацієнт має серію вимірювань, отриманих із фіксованою періодичністю. Тому важливим може бути не лише абсолютне значення білка в конкретний момент, а й характер зміни профілю в часі.

Random Forest, SVM та k-NN у базовій постановці не використовують локальну часову історію профілю як окрему структуру. Вони аналізують окремі записи в табличному просторі. Це обмежує здатність моделей виявляти закономірності, пов’язані з формою коротких часових траєкторій, темпом зміни протеомного профілю або подібністю між динаміками різних пацієнтів.

Саме це створює підставу для переходу до графового представлення даних. У такому підході окремі часові зрізи можуть бути представлені як вузли, а ребра між ними – як подібність між локальними часовими профілями. Для побудови такого представлення можуть використовуватися ідеї топологічного аналізу даних, зокрема lens-функції та побудова графів подібності [21, 22].

Методи топологічного аналізу даних дають змогу досліджувати форму та структуру даних у просторі ознак, не обмежуючись лише прямою табличною класифікацією [21, 22]. У цій роботі ці ідеї використовуються не як повна класична реалізація Mapper-алгоритму, а як методологічна основа для формування графа подібності між часовими протеомними профілями.

Для порівняння часових вікон також доцільним є використання метрик, які враховують послідовність спостережень. Одним із таких підходів є Dynamic Time Warping, який застосовується для зіставлення часових послідовностей із можливим локальним вирівнюванням [24]. У цій роботі DTW-орієнтована логіка використовується в межах графової моделі, описаної в Розділі 2.

1.3.6 Узагальнення методологічних ризиків базового підходу

Проведений аналіз дозволяє виділити кілька основних методологічних ризиків базового підходу. Перший ризик пов’язаний із тим, що часові зрізи одного пацієнта можуть розглядатися як незалежні об’єкти. Другий ризик полягає в можливому потраплянні записів одного пацієнта одночасно до навчальної та тестової вибірок. Третій ризик пов’язаний із наявністю білків, значна частина варіативності яких визначається ідентичністю пацієнта. Четвертий ризик полягає в обмеженості класичного табличного представлення, яке не використовує локальну часову структуру профілю.

Ці ризики не знецінюють результати оригінального дослідження. Навпаки, вони вказують на те, що PERMAD є цінним набором даних для перевірки більш суворих схем оцінювання та альтернативних способів представлення лонгітюдних протеомних профілів. Саме тому в цій роботі базові результати розглядаються як початкова точка для подальшого методологічного аналізу.

1.4 Постановка задачі власного дослідження

На основі критичного аналізу оригінального дослідження сформульовано задачу власної роботи: розробити й експериментально перевірити підхід, який дозволяє аналізувати малі лонгітюдні протеомні набори даних із урахуванням групової належності записів до пацієнтів, пацієнт-специфічності ознак і локальної часової структури профілів.

Власна методика має відповідати кільком вимогам. По-перше, оцінювання моделей повинно виконуватися з груповою ізоляцією пацієнтів, щоб усі записи одного пацієнта перебували лише в навчальній або лише в тестовій частині. По-друге, необхідно контролювати ознаки з високою пацієнт-специфічністю, які можуть поводитися як непрямі ідентифікатори. По-третє, модель має використовувати не лише окремі значення білків, а й локальну часову історію протеомного профілю.

З огляду на ці вимоги в роботі запропоновано побудову графової моделі Velocity-DTW. У цій моделі часові зрізи пацієнтів перетворюються на вузли графа, а ребра формуються на основі подібності між локальними часовими вікнами. Для класифікації вузлів використовується графова нейронна мережа з механізмом уваги [26]. Такий підхід дозволяє поєднати табличні ознаки, часову динаміку та структуру подібності між спостереженнями.

Важливою частиною власного підходу є аналітичний конвеєр, який включає підготовку даних, групову крос-валідацію, відбір ознак, побудову часових вікон, формування графа, навчання графової нейронної мережі та оцінювання результатів. Конвеєрне представлення експерименту є важливим для відтворюваності наукового аналізу, оскільки дозволяє явно зафіксувати послідовність етапів, вхідні дані, проміжні результати й підсумкові метрики [29].

Для досягнення поставленої мети необхідно вирішити такі підзадачі:

1. Розробити базовий конвеєр машинного навчання в екосистемі Python для відтворення класичних підходів, зокрема Random Forest, SVM та k-NN, із впровадженням групової схеми крос-валідації для ізоляції записів пацієнтів між навчальною та тестовою вибірками.

2. Створити програмний модуль для попередньої обробки багатовимірних зашумлених даних, включаючи логарифмічне перетворення, стандартизацію простору ознак, відбір інформативних маркерів та контроль пацієнт-специфічних ознак.

3. Розробити математичний та програмний апарат для перетворення табличних лонгітюдних клінічних даних у топологічно орієнтовані графові структури з використанням часових вікон, lens-функцій, метрик подібності, k-найближчого сусідства та фільтрації ребер.

4. Спроектувати архітектуру графової нейронної мережі для обробки побудованих графів і класифікації їхніх вузлів відповідно до ризику терапевтичної резистентності.

5. Провести серію обчислювальних експериментів для порівняння якості розробленої графової моделі з класичними алгоритмами машинного навчання в умовах строгої групової валідації, використовуючи показники Accuracy, Sensitivity, Specificity та SS2.

Таблиця 1.5 – Вимоги до власної методики аналізу PERMAD

Методологічна проблема	**Вимога до власного підходу**	**Реалізація в роботі**
Кілька записів одного пацієнта	Ізоляція пацієнтів між train і test	StratifiedGroupKFold
Пацієнт-специфічні білки	Контроль ознак-ідентифікаторів	Ratio, ICC, Eta2, fingerprint-фільтрація
Висока розмірність ознак	Зменшення шуму й відбір маркерів	ANOVA-відбір і вагове підсилення білків
Втрата часової структури	Урахування локальної динаміки	Часові вікна та velocity-lens
Обмеження табличних моделей	Мережеве представлення подібності	Граф подібності та GNN
Ризик нестабільності результатів	Повторна валідація й збереження фолдів	5 × 5 групова CV для графової моделі

Примітка. Програмні модулі, що реалізують основні етапи аналітичного конвеєра, наведено в додатку А.

Таким чином, власна постановка задачі не зводиться лише до повторного навчання іншої моделі на тих самих даних. Основна ідея полягає в перевірці того, чи може більш обережна схема валідації та графове представлення лонгітюдних профілів дати стійкіший результат порівняно з класичними табличними моделями.

Висновки до розділу 1

У першому розділі проведено аналіз набору даних PERMAD, методології базового дослідження та основних алгоритмів машинного навчання, використаних для прогнозування терапевтичної резистентності при метастатичному колоректальному раку. Показано, що PERMAD є цінним клінічним набором даних, оскільки містить регулярні лонгітюдні протеомні профілі пацієнтів, отримані в межах проспективного багатоцентрового дослідження.

Розглянуто результати оригінального дослідження, у якому найкращу якість продемонстрував Random Forest. На повному наборі зі 102 маркерів модель досягла Test SS2 79.2, а на скороченій Top-10 панелі – 75.6. Ці результати свідчать про наявність прогностичного сигналу в даних, однак потребують методологічно обережної інтерпретації.

Проаналізовано ключові обмеження базової методології. Найважливішим із них є ризик витоку даних через відсутність групової ізоляції пацієнтів під час крос-валідації. Оскільки один пацієнт представлений кількома часовими зрізами, негрупове розбиття може призводити до того, що модель частково навчається на індивідуальних особливостях пацієнта.

Для кількісного обґрунтування цього ризику проведено аналіз пацієнт-специфічності білків із використанням Ratio, ICC та Eta2. Результати показали, що частина білків має виражену залежність від ідентичності пацієнта, тому в умовах негрупової валідації вони можуть поводитися як непрямі ідентифікатори.

Також показано, що класичне табличне представлення не повністю використовує лонгітюдну структуру PERMAD. Воно аналізує кожен часовий зріз як окремий вектор ознак і слабко враховує локальну часову динаміку протеомного профілю. Це створює підстави для переходу до графового представлення, у якому вузли відповідають часовим зрізам, а ребра відображають подібність між локальними часовими траєкторіями.

На основі проведеного аналізу сформульовано вимоги до власного підходу: групова ізоляція пацієнтів, контроль пацієнт-специфічних ознак, відбір інформативних білків, урахування часових вікон і побудова графової моделі. Детальна методика реалізації цього підходу розглядається в Розділі 2, а експериментальне порівняння класичних і графових моделей наведено в Розділі 3.

РОЗДІЛ 2
РОЗРОБКА МЕТОДИКИ ГРАФОВОГО МОДЕЛЮВАННЯ ЛОНГІТЮДНИХ КЛІНІЧНИХ ДАНИХ

2.1 Загальна структура аналітичного конвеєра дослідження

2.1.1 Призначення та логіка аналітичного конвеєра

У другому розділі описано методику побудови графової моделі для аналізу малих лонгітюдних клінічних наборів даних. Основна ідея полягає у переході від стандартного табличного представлення протеомних профілів до графової структури. У такому графі окремі часові зрізи пацієнтів розглядаються як вершини, а зв’язки між ними формуються на основі подібності локальних динамічних характеристик.

У межах даної роботи термін “аналітичний конвеєр” використовується для позначення впорядкованої послідовності етапів обробки даних, побудови ознакового простору, формування графової структури, навчання моделей та оцінювання їхньої якості. У цьому значенні термін відповідає поширеному в машинному навчанні поняттю “machine learning pipeline” [29].

Потреба у такому конвеєрі зумовлена структурою набору даних PERMAD. Кожен пацієнт має декілька послідовних вимірювань, виконаних із фіксованою періодичністю протягом лікування. Тому окремі записи не можна розглядати як повністю незалежні спостереження. Крім того, кількість протеомних маркерів є значною відносно кількості пацієнтів, що створює ризик перенавчання та нестабільності результатів. Тому методика дослідження має одночасно враховувати часову структуру даних, групову належність записів до пацієнтів і високу розмірність простору ознак.

Загальну структуру аналітичного конвеєра наведено на рисунку 2.1.

Рисунок 2.1 – Загальна структура аналітичного конвеєра дослідження

2.1.2 Основні етапи побудови графової моделі

Початковим етапом є підготовка даних PERMAD для графового аналізу. На цьому етапі формується матриця протеомних ознак, зберігаються ідентифікатори пацієнтів, часові змінні, цільові мітки та список назв біомаркерів. Ці дані надалі використовуються для побудови часових вікон і формування графа подібності.

Після підготовки даних виконується групове розбиття вибірки. Одиницею розбиття є пацієнт, а не окремий часовий зріз. Це необхідно для того, щоб часові точки одного пацієнта не потрапляли одночасно до навчальної та тестової частин. Така схема зменшує ризик прихованого витоку даних і дозволяє оцінити здатність моделі узагальнювати закономірності на нових пацієнтів [12, 13, 16].

Наступним етапом є відбір ознак. Він використовується для зменшення розмірності протеомного простору та обмеження впливу пацієнт-специфічних маркерів. Після цього з відібраних ознак формуються часові вікна, які описують локальну історію зміни протеомного профілю пацієнта.

На основі часових вікон обчислюється матриця відстаней між вузлами. Далі для кожного вузла визначаються найближчі сусіди, слабкі зв’язки

відсікаються за допомогою RBF-фільтрації, і формується зважений граф. Побудований граф передається на вхід графовій нейронній мережі, яка виконує класифікацію вузлів.

Завершальним етапом є оцінювання якості моделі. Для цього використовуються метрики Accuracy, Sensitivity, Specificity, F1 та SS2. Окремо зберігаються параметри кожного фолду, характеристики графа, важливість білків і візуалізація графової структури. Це дозволяє аналізувати не лише середню якість моделі, а й поведінку окремих фолдів.

Основні етапи методики узагальнено в таблиці 2.1.

Таблиця 2.1 – Основні етапи аналітичного конвеєра графової моделі

Етап	**Вхідні дані**	**Основна операція**	**Результат**
Підготовка даних	Таблиці PERMAD	Очищення, об’єднання когорт, формування ознак	Початкова матриця протеомних профілів
Формування мітки	Часові змінні	Визначення 100-денного вікна до прогресування	Бінарна цільова мітка
Групова валідація	Ідентифікатори пацієнтів	Stratified Group K-Fold	Ізольовані train/test фолди
Відбір ознак	Протеомні маркери	Fingerprint-фільтрація, ANOVA-відбір	Компактний набір білків
Часове представлення	Відібрані білки	Формування вікон із п’яти точок	Ознаки вузлів і часові вектори
Побудова графа	Часові вікна	DTW, velocity-lens, kNN, RBF-фільтрація	Зважений граф подібності

Етап	Вхідні дані	Основна операція	Результат
Навчання GNN	Граф і ознаки вузлів	GATConv, регуляризація, оптимізація	Прогноз класу вузлів
Оцінювання	Прогнози моделі	Accuracy, Sensitivity, Specificity, SS2	Підсумкові метрики та рейтинги білків

Продовження таблиці 2.1

2.1.3 Обчислювальна реалізація та використання тензорних операцій

Окремою особливістю програмної реалізації є використання тензорних обчислень. Значна частина операцій у графовій частині реалізована через бібліотеку PyTorch [9]. Це стосується логарифмічного перетворення, стандартизації, формування часових вікон, обчислення матриць відстаней, розрахунку DTW-подібності, побудови ознак вузлів та навчання графової нейронної мережі.

Використання тензорів має практичне значення, оскільки побудова графа потребує багаторазового обчислення попарних відстаней між часовими протеомними профілями. Для набору PERMAD кількість часових зрізів становить сотні записів, а кожен вузол формується на основі багатовимірного часового вікна. У таких умовах реалізація через вкладені цикли Python призвела б до суттєвого зростання часу виконання, особливо під час багаторазового підбору гіперпараметрів.

У коді пристрій обчислень визначається автоматично: якщо доступна графічна карта з підтримкою CUDA, основні тензорні операції виконуються на GPU; якщо така карта недоступна, розрахунки виконуються на CPU. При цьому використання PyTorch залишається доцільним і на CPU, оскільки векторизовані тензорні операції зазвичай виконуються ефективніше, ніж еквівалентні обчислення через явні цикли Python [9].

Найбільш ресурсоємними етапами є формування матриці відстаней між вузлами та обчислення DTW-подібності для часових вікон. Саме тому ці частини реалізовано через пакетні операції над тензорами. Це дає змогу виконувати експерименти з вкладеною крос-валідацією та оптимізацією гіперпараметрів у прийнятний час.

Використання GPU не змінює математичну постановку задачі. Воно впливає лише на спосіб виконання обчислень. Усі основні етапи аналітичного конвеєра залишаються однаковими незалежно від того, чи виконуються вони на CPU або GPU.

2.2 Попередня обробка даних PERMAD

2.2.1 Формування початкової матриці протеомних ознак

Початкові дані дослідження PERMAD були представлені у вигляді двох Excel-таблиць, що відповідали двом когортам пацієнтів [1]. Кожна таблиця містила службові клінічні поля, ідентифікатори зразків, часові характеристики та кількісні значення протеомних маркерів. На етапі попередньої обробки з цих таблиць було сформовано єдину матрицю ознак, у якій рядки відповідають протеомним маркерам, а стовпці відповідають окремим часовим зрізам пацієнтів.

Перед об’єднанням когорт виконувалося вилучення пацієнтів, які вибули з дослідження або не мали коректної структури спостережень. Також видалялися порожні службові рядки, що не відповідали реальним зразкам. Після цього з кожної когорти відокремлювалися метадані та кількісні значення білків. Отримані матриці двох когорт об’єднувалися в єдину таблицю, а назви стовпців очищувалися від зайвих пробілів і приводилися до однорідного формату.

Оскільки у вхідних таблицях частина значень могла містити службові символи, перед подальшими розрахунками виконувалося очищення таких записів. Після видалення службових символів усі значення протеомних маркерів перетворювалися до числового типу. Це було необхідно для подальшого

застосування статистичних методів, обчислення відстаней між профілями та навчання моделей машинного навчання.

У результаті цього етапу було сформовано початкову матрицю протеомних ознак, яка містила кількісні значення маркерів для збережених часових зрізів пацієнтів. Додатково для кожного зразка зберігалися ідентифікатор пацієнта, номер когорти та часові змінні, необхідні для побудови лонгітюдної структури даних.

2.2.2 Формування часових змінних і цільової мітки

Для аналізу терапевтичної резистентності важливим є не лише сам факт наявності протеомного профілю, а й його положення відносно моменту радіологічного підтвердження прогресування захворювання. Тому в процесі попередньої обробки для кожного зразка розраховувалися часові змінні, які описують його відстань до контрольної точки прогресування.

Основною часовою змінною є час до прогресування, обчислений на основі дати забору крові, дати контрольного обстеження та відстані до комп’ютерної томографії. Для кожного пацієнта визначався референтний зразок, пов’язаний із моментом прогресування, після чого інші часові зрізи розміщувалися відносно цієї точки. Такий підхід дозволяє порівнювати профілі різних пацієнтів не за календарними датами, а за їхнім положенням у клінічній траєкторії.

Цільова мітка для графового аналізу формувалася за правилом 100-денного вікна до прогресування. Якщо часовий зріз знаходився ближче ніж за 100 днів до моменту прогресування, він відносився до класу прогресування. Інші часові зрізи відносилися до класу відсутності прогресування в заданому часовому вікні. Така логіка відповідає постановці задачі в оригінальному дослідженні PERMAD [1].

Важливо, що така постановка задачі зберігає лонгітюдний характер даних. Один пацієнт може мати кілька часових зрізів, які належать до різних ділянок клінічної траєкторії. Саме тому подальше розбиття даних на навчальні та тестові

частини має виконуватися з урахуванням ідентичності пацієнта, а не лише на рівні окремих записів.

2.2.3 Підготовка даних для графового аналізу

Для графової частини роботи було сформовано файл PERMAD_graph_raw.pkl. Він містить матрицю протеомних ознак, цільові мітки, ідентифікатори пацієнтів, номер когорти, часові змінні та список назв біомаркерів. Саме цей файл використовується як вхід для побудови графової моделі. Перелік основних програмних модулів наведено в додатку А.

На відміну від варіанта даних, який використовувався для відтворення класичних моделей, у графовому наборі збережено долікувальні часові точки. Це пов’язано з тим, що вершина графа формується не лише на основі поточного вимірювання, а й на основі фіксованого вікна попередніх вимірювань того самого пацієнта. Якщо вилучити ранні точки до побудови таких вікон, частина динамічної інформації буде втрачена.

У графовому наборі з первинної матриці видалялися лише окремі браковані зразки, позначені як 10_8 та 21_7. Інші часові точки зберігалися, оскільки вони можуть бути потрібними для побудови локальної часової історії пацієнта. Така логіка відповідає задачі графового моделювання, де важливо не лише класифікувати окремий зріз, а й врахувати його місце у послідовності попередніх вимірювань.

Після завантаження підготовленого графового набору в основному коді застосовується логарифмічне перетворення значень за допомогою функції log1p. Це перетворення виконується вже на етапі моделювання, а не під час формування PERMAD_graph_raw.pkl. Його призначення полягає у зменшенні впливу великих амплітудних відмінностей між окремими білками перед стандартизацією та побудовою матриці відстаней.

2.2.4 Контроль структури підготовлених даних

Після формування PERMAD_graph_raw.pkl виконувалася перевірка структури підготовленого набору. На цьому етапі контролювалася відповідність кількості ознак кількості назв протеомних маркерів, коректність числового типу значень, наявність ідентифікаторів пацієнтів, а також узгодженість часових змінних і цільових міток.

Окрему увагу було приділено збереженню групової структури даних. Для кожного запису зберігався ідентифікатор пацієнта, який надалі використовувався у схемі Stratified Group K-Fold [16]. Це дозволило забезпечити коректне розбиття даних у подальших експериментах і уникнути змішування часових зрізів одного пацієнта між навчальною та тестовою частинами вибірки.

Результатом попередньої обробки для графової частини роботи став серіалізований об’єкт PERMAD_graph_raw.pkl. Такий формат спрощує подальші експерименти, оскільки всі необхідні компоненти для побудови графа зберігаються в одному об’єкті.

2.3 Групова схема валідації та ізоляція пацієнтів

2.3.1 Необхідність групового розбиття лонгітюдних даних

Однією з ключових методологічних вимог даної роботи є коректне розділення даних на навчальну та тестову частини. У звичайних задачах машинного навчання окремі записи часто розглядаються як незалежні спостереження. Проте у випадку набору PERMAD така умова не виконується, оскільки один пацієнт має кілька часових зрізів, отриманих у різні моменти лікування.

Якщо виконувати випадкове розбиття на рівні окремих записів, часові зрізи одного пацієнта можуть одночасно потрапити і до навчальної, і до тестової вибірки. У такій ситуації модель частково бачить індивідуальні особливості пацієнта під час навчання, а потім тестується на інших записах того самого

пацієнта. Це створює ризик прихованого витоку даних і може призводити до завищення оцінок якості моделі [12, 13].

Для уникнення цієї проблеми у роботі використовується групова схема крос-валідації. У ній одиницею розбиття є не окремий часовий зріз, а пацієнт. Це означає, що всі записи одного пацієнта потрапляють лише в одну частину розбиття: або до навчальної вибірки, або до тестової. Такий підхід дозволяє оцінювати здатність моделі узагальнювати закономірності на нових пацієнтів, а не лише впізнавати вже частково представлені у навчанні індивідуальні профілі.

2.3.2 Використання Stratified Group K-Fold

У роботі застосовується схема Stratified Group K-Fold. Вона поєднує дві вимоги: збереження приблизного співвідношення класів у фолдах і повну ізоляцію груп між навчальною та тестовою частинами [16]. У даному випадку групою є ідентифікатор пацієнта, а класом є мітка прогресування або відсутності прогресування у межах 100-денного вікна.

Стратифікація потрібна через те, що класи у клінічних задачах можуть бути представлені нерівномірно. Якщо не контролювати співвідношення класів у фолдах, окремі тестові частини можуть отримати надто мало прикладів одного з класів. Це зробить оцінки чутливості, специфічності та SS2 нестабільними. Груповий компонент, своєю чергою, запобігає потраплянню записів одного пацієнта до різних частин розбиття.

У програмній реалізації зовнішня крос-валідація використовується для оцінювання якості моделі на тестових пацієнтах. Внутрішня крос-валідація використовується для підбору гіперпараметрів. Така вкладена схема дозволяє відокремити процес налаштування моделі від фінального оцінювання. Це важливо, оскільки параметри моделі не повинні підбиратися з урахуванням тестових пацієнтів [17, 18].

У фінальній графовій моделі використовується п’ять фолдів зовнішньої крос-валідації з кількома повторами за різних значень генератора випадкових

чисел. Такий підхід дозволяє отримати не одну випадкову оцінку якості, а набір результатів для різних розбиттів пацієнтів. Далі результати усереднюються, а розкид між фолдами використовується для оцінки стабільності моделі.

2.3.3 Вкладена схема підбору гіперпараметрів

Підбір гіперпараметрів виконується лише на навчальній частині кожного зовнішнього фолду. Для цього навчальні пацієнти додатково розбиваються на внутрішні фолди за тією самою логікою групової ізоляції. На кожній ітерації оптимізації модель навчається на внутрішній навчальній частині та перевіряється на внутрішній валідаційній частині.

Такий підхід дозволяє оцінити якість конкретного набору параметрів без використання зовнішньої тестової вибірки. Після завершення підбору найкращі параметри застосовуються для навчання моделі на всій навчальній частині зовнішнього фолду. Лише після цього модель оцінюється на зовнішній тестовій частині.

Для оптимізації гіперпараметрів використовується Optuna [8]. Цей інструмент дозволяє автоматизувати пошук у заданому просторі параметрів і зменшити кількість ручних експериментів. У межах даної роботи оптимізуються параметри графової нейронної мережі, параметри побудови графа та параметри регуляризації.

Основним критерієм підбору є показник SS2 на внутрішній валідації. Цей показник обрано тому, що він одночасно враховує чутливість і специфічність. Для клінічної задачі це важливо, оскільки надмірна орієнтація лише на Accuracy може приховувати перекіс моделі в бік одного з класів.

2.3.4 Збереження результатів фолдів і відновлення обчислень

Оскільки навчання графової моделі та підбір гіперпараметрів потребують значного часу, у програмній реалізації передбачено проміжне збереження результатів після завершення кожного фолду. Для кожного фолду зберігаються

метрики навчальної та тестової вибірок, найкращі знайдені параметри, характеристики графа, важливість білків, файл графа та службова інформація про вершини.

Такий підхід має практичне значення. Якщо виконання програми переривається, вже завершені фолди не потрібно рахувати повторно. При повторному запуску код перевіряє наявність збереженого результату для відповідного фолду та переходить до наступного. Це особливо важливо для багаторазової крос-валідації, де повний експеримент складається з великої кількості незалежних запусків.

У межах даної роботи відновлення реалізовано саме на рівні завершених фолдів. Такий варіант є компромісом між надійністю та швидкістю: він не створює зайвих операцій запису після кожної ітерації оптимізації, але дозволяє не втрачати вже завершені частини експерименту. Фрагменти програмної реалізації збереження результатів наведено в додатку А.

2.4 Відбір ознак, контроль специфічності та оцінка важливості білків

2.4.1 Роль відбору ознак у графовому представленні

Відбір ознак у даній роботі використовується як проміжний етап між попередньою обробкою даних і побудовою графової структури. Його основна мета полягає у зменшенні розмірності протеомного простору перед формуванням часових вікон, матриці відстаней і ребер графа [20]. Це важливо, оскільки у графовій моделі ознаки впливають не лише на класифікацію, а й на саму структуру графа.

На відміну від класичних табличних моделей, де відбір ознак безпосередньо визначає набір вхідних змінних класифікатора, у графовому підході він має ширше значення. Відібрані білки використовуються для формування ознак вузлів, розрахунку динамічних характеристик часових профілів і визначення подібності між різними спостереженнями. Тому

нестабільні або надмірно індивідуалізовані маркери можуть впливати не тільки на результат навчання, а й на топологію побудованого графа.

Методологічне обґрунтування необхідності контролю пацієнт-специфічних маркерів наведено у Розділі 1. У цьому підрозділі розглядається вже не сама критика наявності таких ознак, а практичне врахування зробленого аналізу в програмній реалізації графової моделі.

2.4.2 Fingerprint-фільтрація пацієнт-специфічних ознак

Першим етапом відбору є контроль ознак, які мають надмірно виражену залежність від ідентичності пацієнта. У коді цей етап реалізовано як fingerprint-фільтрацію. Її призначення полягає у зменшенні впливу білків, які можуть поводитися як непрямі ідентифікатори пацієнта.

Необхідність такого кроку випливає з результатів аналізу, наведеного у Розділі 1. Там було показано, що частина білків демонструє високу залежність від пацієнта за показниками ICC та Eta2 [14, 15]. Це не означає, що такі білки не мають біологічного значення, однак у задачі машинного навчання вони можуть створювати методологічний ризик: модель здатна частково навчатися розпізнавати індивідуальні патерни пацієнтів, а не загальні ознаки терапевтичної резистентності.

У графовій моделі цей ризик є особливо важливим, оскільки ознаки використовуються не лише для класифікації, а й для побудови ребер. Якщо пацієнт-специфічний маркер сильно впливає на матрицю відстаней, він може змінити саму структуру графа. Тому fingerprint-фільтрація застосовується до етапу формування часових вікон і побудови графа.

Водночас fingerprint-фільтрація не розглядається як клінічне вилучення певних білків із біологічного аналізу. Її роль є методологічною: зменшити ризик прихованого витоку даних і стабілізувати графове представлення в умовах малої лонгітюдної вибірки.

2.4.3 ANOVA-відбір інформативних білків

Після контролю пацієнт-специфічності виконується відбір білків, які мають найбільшу здатність розділяти два класи. Для цього використовується ANOVA-подібний дисперсійний критерій. Він оцінює співвідношення міжкласової та внутрішньокласової варіативності для кожної ознаки [20].

Якщо білок має високу міжкласову варіативність і відносно нижчу внутрішньокласову варіативність, він краще підходить для розділення профілів прогресування та відсутності прогресування. Після обчислення такого критерію білки ранжуються, і до подальшої побудови графа потрапляє обмежена кількість найінформативніших ознак.

У фінальній графовій моделі використовується 15 протеомних маркерів. Таке обмеження є компромісом між збереженням біологічно релевантної інформації та зменшенням ризику перенавчання. Крім того, менша кількість ознак знижує розмірність часових вікон і спрощує обчислення матриці відстаней.

ANOVA-відбір у цій роботі не трактується як доказ остаточної клінічної значущості відібраних білків. Його основне призначення полягає у формуванні компактного й достатньо інформативного ознакового простору для побудови графової моделі.

2.4.4 Вагове підсилення відібраних ознак

Після відбору білків у програмній реалізації застосовується вагове підсилення ознак. Його мета полягає в тому, щоб маркери з вищою міжкласовою інформативністю мали більший вплив на обчислення відстаней між часовими профілями. Оскільки саме відстані визначають подальше формування ребер, цей етап впливає на геометрію майбутнього графа.

Ваги ознак нормалізуються відносно максимального значення серед відібраних білків. Для менш виражених, але потенційно корисних маркерів задається нижня межа ваги. Це дозволяє уникнути ситуації, коли кілька

найсильніших білків повністю домінують у просторі відстаней, а решта відібраних ознак фактично втрачає вплив.

Вагове підсилення не змінює кількість ознак і не створює нових змінних. Воно лише змінює внесок окремих білків у простір відстаней, у якому далі будуються часові вектори, velocity-lens і граф подібності.

2.4.5 Відстеження важливості білків у фолдах

Після навчання моделі для кожного фолду виконується оцінка важливості білків. Це потрібно для того, щоб зрозуміти, які маркери найчастіше впливають на класифікаційне рішення графової моделі. У роботі важливість білків розглядається як інтерпретаційний інструмент, а не як самостійний критерій оптимізації моделі.

Оцінка важливості виконується на рівні фолдів. Для кожного фолду зберігається список білків і числові значення їхнього внеску. Після завершення всіх фолдів ці значення агрегуються, що дозволяє побудувати фінальний рейтинг маркерів за середньою важливістю та розкидом між фолдами.

Такий підхід дає змогу відрізнити білки, які стабільно з’являються у різних фолдах, від маркерів, що мають високу важливість лише в окремих розбиттях. Це особливо важливо для малого клінічного набору даних, де результат одного фолду може бути залежним від конкретного складу пацієнтів у навчальній і тестовій частинах.

У фінальному експерименті всі 15 відібраних білків були присутні в усіх фолдах. Найвищу середню важливість продемонстрували IGFBP-1, Epiregulin, VEGF, Galectin-3, EN-RAGE, SAP та 6Ckine. Цей результат використовується у Розділі 3 для інтерпретації моделі та порівняння з результатами класичного Random Forest.

2.4.6 Значення відбору ознак для подальшої побудови графа

Відбір ознак впливає на всі наступні етапи графового моделювання. По-перше, він визначає склад часового вікна, з якого формується ознаковий вектор вузла. По-друге, він впливає на розрахунок velocity-lens, оскільки ця функція відображає інтенсивність змін саме у просторі відібраних білків. По-третє, він змінює матрицю відстаней між вузлами, а отже, і структуру ребер у графі.

Тому відбір ознак у даній роботі не є допоміжною косметичною процедурою. Він є одним із елементів побудови графового представлення, від якого залежить форма простору подібності між часовими протеомними профілями. Саме після цього етапу дані можуть бути перетворені у часові вікна та використані для побудови графа.

Разом із тим отриманий список білків не слід трактувати як універсальну біологічну сигнатуру. У межах даної роботи він використовується як експериментальна ознакова основа для перевірки графового підходу до класифікації лонгітюдних протеомних профілів.

2.5 Формування часових вікон і ознак вузлів графа

2.5.1 Перехід від окремого вимірювання до часового представлення

Після попередньої обробки даних і відбору ознак кожен часовий зріз пацієнта перетворюється на вузол майбутнього графа. При цьому вузол не описується лише поточним значенням протеомного профілю. Для збереження локальної динаміки лікування використовується фіксоване часове вікно, яке включає поточне вимірювання та кілька попередніх вимірювань того самого пацієнта.

Такий підхід дозволяє перейти від статичного представлення запису до динамічного. У класичній табличній постановці кожен зразок розглядається окремо, без явного врахування попередніх точок траєкторії. У графовій моделі, навпаки, кожна вершина містить інформацію про локальну історію зміни протеомного профілю. Це важливо для задачі прогнозування терапевтичної

резистентності, оскільки клінічно значущим може бути не тільки абсолютне значення маркера, а й напрям або інтенсивність його зміни в часі.

У роботі використовується фіксований розмір часового вікна, що дорівнює п’яти вимірюванням. Для кожного часового зрізу формується послідовність із п’яти точок одного пацієнта, які передують поточному зрізу або збігаються з ним. Якщо для ранніх вимірювань пацієнта доступно менше п’яти попередніх точок, перше доступне вимірювання дублюється для заповнення вікна. Це дозволяє зберегти однакову розмірність вузлів для всіх спостережень.

$$V\_i = (x\_i(t-4), x\_i(t-3), x\_i(t-2), x\_i(t-1), x\_i(t)) \quad (2.1)$$

$V\_i -$ часове вікно для вузла i;

$x\_i(t) -$ протеомний профіль пацієнта в момент часу t.

2.5.2 Стандартизація ознак перед формуванням вікон

Перед формуванням часових вікон протеомні ознаки стандартизуються. Стандартизація виконується на основі навчальної частини відповідного фолду, щоб тестові пацієнти не впливали на параметри перетворення. Це важливо для збереження коректності експериментальної схеми, оскільки середні значення та стандартні відхилення ознак не повинні оцінюватися з використанням тестової вибірки.

Після стандартизації відібрані ознаки можуть додатково зважуватися відповідно до їхньої інформативності. Як було зазначено у попередньому підрозділі, таке зважування впливає на геометрію простору, у якому надалі обчислюються подібності між часовими профілями. У результаті маркери з вищою міжкласовою інформативністю отримують більший внесок у формування відстаней, але менш виражені ознаки не вилучаються повністю.

На цьому етапі формуються два близькі, але не тотожні представлення. Перше використовується для обчислення відстаней між вузлами та побудови графа. Друге використовується як вхідний ознаковий вектор для графової

нейронної мережі. Такий поділ потрібний, оскільки для побудови графа важливо порівнювати часові послідовності, тоді як для GNN потрібно мати компактне ознакове представлення.

2.5.3 Формування вектора для розрахунку відстаней

Для обчислення подібності між вузлами використовується представлення часового вікна у вигляді єдиного вектора. Усі значення відібраних білків у межах вікна послідовно об’єднуються в один довгий вектор. Якщо використовується 15 білків і вікно з п’яти часових точок, то таке представлення містить 75 числових значень.

Цей вектор використовується для обчислення відстаней між часовими профілями. Саме на його основі формується матриця подібності, яка далі визначає потенційні зв’язки між вузлами графа. Такий підхід дозволяє порівнювати не лише поточні значення білків, а й коротку локальну траєкторію їхньої зміни.

За своєю логікою це представлення близьке до підходів часових вкладень, де стан об’єкта описується не однією точкою, а послідовністю попередніх значень [23]. У даній роботі цей принцип використовується не для повної реконструкції динамічної системи, а як практичний спосіб включити часовий контекст у графове представлення клінічних даних.

2.5.4 Формування ознак вузла для графової нейронної мережі

Окремо формується вектор ознак, який безпосередньо подається на вхід графової нейронної мережі. Він складається з трьох частин.

Перша частина містить значення відібраних білків у першій точці часового вікна. Вона описує початковий стан локального фрагмента траєкторії. Друга частина містить значення в останній точці вікна, тобто поточний стан профілю. Третя частина описує різницю між поточним і початковим станом. Таким чином,

вузол містить інформацію про початковий рівень, поточний рівень і локальну зміну протеомного профілю.

Таке представлення є компактнішим, ніж повне розгортання всіх часових точок у вікні, але все одно зберігає важливу динамічну інформацію. Воно також краще підходить для подачі на вхід графової нейронної мережі, оскільки дозволяє моделі працювати з ознаками вузла, не перевантажуючи її надмірно довгими векторами.

2.5.5 Velocity-lens як характеристика локальної динаміки

Для додаткового опису часового вікна використовується velocity-lens. У загальному розумінні lens-функція є способом відобразити складні багатовимірні об’єкти в простір меншої розмірності, щоб далі аналізувати їхню структуру [21, 22]. У цій роботі така ідея використовується для побудови одновимірної характеристики локальної динаміки протеомного профілю.

Velocity-lens описує інтенсивність зміни профілю всередині часового вікна. Для кожного вузла обчислюється середня величина зміни між сусідніми точками вікна. Якщо профіль змінюється різко, значення velocity-lens буде вищим. Якщо профіль стабільний, значення буде нижчим.

$$lens\_i \ = \ mean(||x\_i(t\_k) \ - \ x\_i(t\_\{k-1\})||), k \ = \ 2,\ldots,5 \quad (2.2)$$

$$lens_i \ - \ \text{значення } velocity - lens \text{ для вузла } i;$$

$$x_i(t_k) \text{ та } x_i(t_{k-1}) \ - \ \text{сусідні часові точки у вікні.}$$

Використання velocity-lens має практичний сенс для даної клінічної задачі. Якщо терапевтична резистентність проявляється не лише у значеннях окремих білків, а й у зміні їхньої динаміки, то інтенсивність таких змін може бути важливою характеристикою вузла. Саме тому velocity-lens використовується як один із компонентів подальшого розрахунку подібності між вузлами.

У цій роботі velocity-lens не розглядається як самостійна модель прогнозування. Його роль полягає у формуванні додаткового одновимірного представлення, яке допомагає структурувати простір часових профілів перед побудовою графа. Далі ця інформація поєднується з метрикою відстані між часовими вікнами.

2.5.6 Значення часового представлення для подальшої побудови графа

Формування часових вікон є проміжним етапом між відбором ознак і побудовою графа. Саме на цьому етапі табличні протеомні дані перетворюються у набір вузлів, кожен із яких містить локальну часову інформацію. Без цього перетворення граф будувався б лише за статичними зрізами, що суттєво обмежило б можливість аналізу лонгітюдної структури даних.

Отримані часові представлення використовуються двома способами. По-перше, вони є основою для обчислення матриці відстаней між вузлами. По-друге, вони формують ознаки, які передаються до графової нейронної мережі. Тому якість цього етапу безпосередньо впливає і на структуру графа, і на результат класифікації.

Схему переходу від часового вікна до графа подібності наведено на рисунку 2.2.

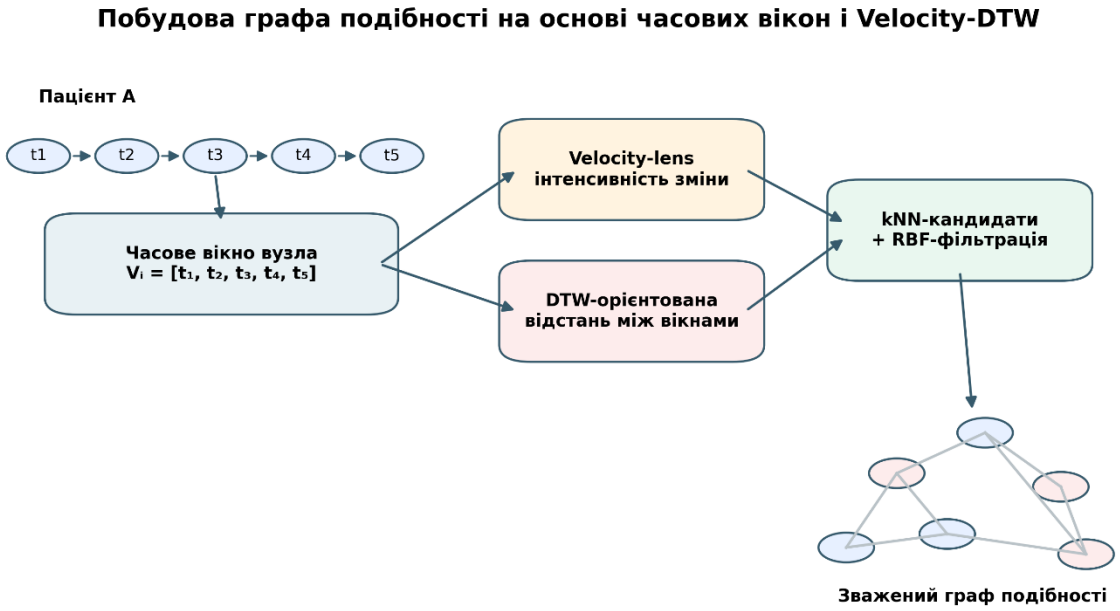


Рисунок 2.2 – Побудова графа подібності на основі часових вікон і Velocity-DTW

2.6 Побудова графа подібності між часовими профілями

2.6.1 Топологічна ідея графового представлення

Графове представлення використовується для того, щоб перейти від аналізу окремих ізольованих записів до аналізу структури подібності між ними. У побудованому графі кожна вершина відповідає часовому зрізу пацієнта, описаному через локальне часове вікно. Ребра між вершинами відображають подібність відповідних часових протеомних профілів.

Такий підхід пов’язаний з ідеями топологічного аналізу даних. У класичних TDA-підходах важливою є не лише абсолютна позиція об’єктів у просторі ознак, а й форма, зв’язність і локальна структура множини даних [21]. Метод Mapper, зокрема, використовує lens-функцію та покриття простору для побудови графового опису високовимірних даних [22].

У даній роботі не реалізується повний класичний Mapper-алгоритм. Натомість використовується близька ідея: часові протеомні профілі

проєктуються через velocity-lens і поєднуються з метрикою подібності для формування зваженого графа. Такий варіант є прикладним компромісом між TDA-ідеєю структурного представлення даних і практичною задачею класифікації вузлів графовою нейронною мережею. Повна реалізація Mapper-алгоритму передбачає стратифікацію даних за lens-функцією та подальшу кластеризацію об'єктів у сформованих шарах. Для малої клінічної вибірки така кластеризація може бути нестабільною, тому в цій роботі її роль частково замінено локальним пошуком k-NN-кандидатів, на основі яких формуються зважені ребра графа.

2.6.2 Метрика відстані між часовими вікнами

Для порівняння часових вікон використовується DTW-орієнтована логіка. Dynamic Time Warping є методом порівняння часових послідовностей, який дозволяє враховувати локальні відмінності у формі траєкторії [24]. У цій роботі DTW застосовується не для клінічного вирівнювання нерівномірних часових шкал, а для порівняння локальної форми коротких протеомних вікон.

Базова відстань між двома вузлами обчислюється на основі їхніх часових векторів. Далі вона поєднується з різницею значень velocity-lens. Це дозволяє врахувати як загальну схожість часових профілів, так і схожість інтенсивності локальної зміни.

$$D\_ij = w\_corr \cdot D\_base(i,j) + (1 - w\_corr) \cdot |lens\_i - lens\_j| \quad (2.3)$$

D_{ij} – підсумкова відстань між вузлами i та j;

$D_{base(i,j)}$ – базовою відстанню між часовими вікнами;

$lens_i$ та $lens_j$ – значеннями $velocity - lens$;

w_{corr} – вага базової відстані.

2.6.3 Вибір найближчих сусідів

Після обчислення матриці відстаней для кожного вузла визначається набір найближчих сусідів. Такий підхід дозволяє сформувати локальні кандидатні ребра, не створюючи повний граф між усіма парами вузлів. Повний граф був би надто щільним і погано інтерпретованим, особливо для малого клінічного набору даних.

Параметр k_neighbors визначає кількість найближчих сусідів, які розглядаються для кожного вузла. Якщо цей параметр занадто малий, граф може виявитися незв'язним або надмірно розрідженим. Якщо він занадто великий, граф стає щільним, а локальна структура подібності розмивається. Тому вибір k_neighbors є одним із важливих гіперпараметрів побудови графа.

2.6.4 RBF-фільтрація та зважування ребер

Для кожного кандидатного ребра обчислюється вага на основі RBF-подібного перетворення відстані. Чим меншою є відстань між двома вузлами, тим більшою буде вага відповідного ребра. Навпаки, вузли з більшою відстанню отримують нижчу вагу зв'язку.

$$W_{ij} = \exp\left(-\frac{D_{ij}^2}{(sigma_i \cdot sigma_j)}\right) \quad (2.4)$$

W_{ij} – вага ребра між вузлами i та j;

D_{ij} – відстань між вузлами;

$sigma_i$ та $sigma_j$ – локальний масштаб відстаней для відповідних вузлів.

Параметр rbf_min_weight використовується як поріг відсікання слабких ребер. Якщо вага ребра є нижчою за заданий поріг, таке ребро не включається до фінального графа. Це дозволяє зменшити кількість слабких або випадкових зв'язків і зробити структуру графа більш локальною.

Додатково у формулі ваги використовується параметр локальної щільності kde_bandwidth. Він впливає на те, наскільки жорстко відстані перетворюються у ваги ребер. У практичному сенсі цей параметр допомагає контролювати щільність графа: за одних значень граф може ставати надто густим, за інших значень надмірно розрідженим. Тому підбір цього параметра є важливим для отримання придатної графової структури.

2.6.5 Регуляризація міжкласових ребер у навчальній частині графа

У фінальній реалізації використовується параметр class_penalty, який не змінює етап вибору кандидатних сусідів. Він застосовується лише після формування ребер і впливає на ваги вже сформованих міжкласових ребер у навчальній частині графа.

Його призначення полягає у зменшенні внеску міжкласових зв’язків між навчальними вузлами. Такий параметр не створює нових міток, не використовується для тестових вузлів як джерело навчальної інформації і не змінює саму множину кандидатних сусідів. Він лише послаблює ваги частини train-train ребер, якщо ці ребра поєднують вузли різних класів.

Цей етап слід розглядати як евристичну регуляризацію структури графа. Його використання пов’язане з високим рівнем шуму у клінічних даних і складністю побудови стабільної графової структури на малій вибірці. У практичному сенсі class_penalty зменшує внесок потенційно суперечливих зв’язків між навчальними вузлами різних класів.

Водночас цей параметр має важливе методологічне обмеження. Його не можна застосовувати до тестових міток, оскільки це призвело б до витоку інформації. Тому у фінальній реалізації він працює лише з навчальною частиною графа, де мітки відомі в межах процесу навчання.

2.6.6 Структурні характеристики графа

Після побудови графа обчислюються його структурні характеристики. До них належать кількість вузлів, кількість ребер, середній ступінь вузла, максимальний ступінь, коефіцієнт кластеризації, кількість ізольованих вузлів, homophily, modularity та adjusted Rand index [25].

Ці показники використовуються для аналізу якості побудованої графової структури. Наприклад, надто велика кількість ізольованих вузлів означає, що граф погано передає локальну подібність між профілями. Надто висока щільність ребер, навпаки, може свідчити про втрату локальної структури й перетворення графа на майже повнозв'язну мережу.

Структурні характеристики графа не є основною цільовою функцією дослідження. Основним критерієм залишається якість класифікації на тестових фолдах. Проте ці показники допомагають інтерпретувати, чому окремі конфігурації параметрів працюють краще або гірше.

2.7 Навчання графової нейронної мережі

2.7.1 Загальна архітектура моделі

Для класифікації вузлів побудованого графа використовується графова нейронна мережа. На вхід моделі подаються ознаки вузлів, індекси ребер і ваги ребер. Архітектура моделі поєднує графовий attention-шар, skip-з'єднання, dropout-регуляризацію та фінальний лінійний класифікатор.

Модель має трансдуктивний характер. Це означає, що структура графа формується для всіх вузлів, включаючи тестові, однак мітки тестових вузлів не використовуються під час навчання. Навчання виконується лише на вузлах навчальної частини відповідного фолду, а тестові вузли використовуються тільки для фінального оцінювання якості. Такий підхід є типовим для частини графових задач, де структура графа доступна повністю, але мітки відомі лише для навчальних вузлів [27, 28].

Основні компоненти архітектури наведено в таблиці 2.2.

Таблиця 2.2 – Основні компоненти графової нейронної мережі

Компонент	**Призначення**
Ознаки вузлів	Подання локального стану та зміни протеомного профілю
Edge index	Опис структури ребер графа
Edge weight	Вага подібності між вузлами
GATConv	Агрегація інформації від сусідніх вузлів із механізмом уваги
Skip-з'єднання	Збереження частини початкового подання вузла
Dropout	Регуляризація моделі та зменшення перенавчання
Лінійний класифікатор	Перетворення прихованого подання у прогноз класу

2.7.2 Використання механізму графової уваги

Основним шаром нейронної мережі є GATConv, тобто шар графової мережі з механізмом уваги [10, 26]. На відміну від простого усереднення ознак сусідів, механізм уваги дозволяє моделі надавати різну вагу різним сусіднім вузлам. Це важливо для клінічних даних, оскільки не всі зв'язки між протеомними профілями мають однакову інформативність.

У межах даної роботи використання Graph Attention Network має практичне призначення: модель може навчитися сильніше враховувати ті сусідні вузли, які краще підтримують класифікаційне рішення, і слабше враховувати менш релевантні зв'язки. Такий підхід краще відповідає природі побудованого графа, де ребра відображають ступінь подібності, але не гарантують повну клінічну еквівалентність профілів.

Замість окремого рисунка архітектуру GNN подано таблицею 2.2, оскільки повна схема несе той самий зміст і не перевантажує розділ. Ідея використання

attention у графових нейронних мережах ґрунтується на підході Graph Attention Networks [26].

2.7.3 Регуляризація та запобігання перенавчанню

Через малий розмір клінічного набору даних модель має високий ризик перенавчання. Для зменшення цього ризику в архітектурі використовується кілька механізмів регуляризації. По-перше, застосовується dropout, який випадково вимикає частину активацій під час навчання. По-друге, використовується L1-регуляризація, яка обмежує надмірне зростання ваг моделі. По-третє, навчання зупиняється достроково, якщо якість на валідаційній частині перестає покращуватися.

Також важливу роль відіграє відбір ознак. Як показано в Розділі 3, запуск графової моделі без ANOVA-відбору, fingerprint-фільтрації та вагового підсилення білків призводить до різкого зростання навчальних метрик при нижчій тестовій якості. Це підтверджує, що контроль ознакового простору є не декоративним етапом, а важливою частиною регуляризації всієї графової моделі.

2.7.4 Підбір гіперпараметрів

Підбір гіперпараметрів виконується за допомогою Optuna [8]. Оптимізуються як параметри нейронної мережі, так і параметри побудови графа. До них належать розмір прихованого шару, швидкість навчання, коефіцієнти регуляризації, dropout, кількість найближчих сусідів, поріг RBF-фільтрації, вага базової відстані w_corr та параметр kde_bandwidth.

Основним критерієм оптимізації є SS2 на внутрішній валідації. Такий вибір дозволяє уникнути ситуації, коли модель оптимізується лише під загальну точність і погано працює для одного з класів. Для клінічної задачі баланс між чутливістю та специфічністю є важливішим за ізольоване значення Accuracy.

Після завершення оптимізації найкращі параметри використовуються для навчання моделі на всій навчальній частині зовнішнього фолду. Потім модель

оцінюється на тестових пацієнтах відповідного зовнішнього фолду. Така схема дозволяє розділити етап підбору параметрів і етап фінального оцінювання.

2.7.5 Збереження результатів і візуалізація графів

Після завершення кожного фолду зберігаються метрики, параметри, характеристики графа, важливість білків, список вузлів і візуалізація графової структури. Це дозволяє надалі аналізувати не лише підсумкові середні метрики, а й поведінку окремих фолдів.

Візуалізація графів використовується як допоміжний інструмент. Вона дозволяє оцінити, чи має побудований граф зрозумілу локальну структуру, чи не є він надто щільним, чи не містить великої кількості ізольованих вузлів. Однак остаточна якість моделі визначається не візуальною формою графа, а результатами на тестових фолдах.

Повні приклади графів окремих фолдів наведено в додатку Б. У Розділі 3 використано один репрезентативний граф як ілюстрацію фінальної моделі.

Висновки до Розділу 2

У другому розділі було описано методику побудови графової моделі для аналізу малих лонгітюдних клінічних наборів даних. Основну увагу приділено переходу від табличного представлення протеомних профілів до графової структури, у якій вузли відповідають часовим зрізам пацієнтів, а ребра відображають подібність між локальними протеомними траєкторіями.

Було розглянуто попередню обробку даних PERMAD, формування часових змінних і цільової мітки, а також підготовку даних для графового аналізу. Окремо обґрунтовано необхідність групової схеми крос-валідації, яка забезпечує ізоляцію пацієнтів між навчальною та тестовою вибірками. Це є ключовою умовою коректного оцінювання моделей на лонгітюдних клінічних даних.

Для зменшення розмірності протеомного простору було описано процедуру відбору ознак. Відібрані білки використовуються для формування

часових вікон, розрахунку динамічних характеристик профілів і побудови матриці відстаней. Такий підхід дозволяє враховувати не лише поточний стан профілю, а й локальну історію його зміни.

Побудова графа виконувалася на основі поєднання DTW-орієнтованої метрики, velocity-lens, вибору найближчих сусідів і RBF-фільтрації ребер. Додатково використовується м’яке послаблення ваг міжкласових ребер у навчальній частині графа, яке не впливає на вибір кандидатних сусідів і не використовує мітки тестових вузлів.

Для класифікації вузлів побудованого графа використано графову нейронну мережу на основі механізму уваги. Архітектура моделі поєднує графовий attention-шар, skip-з’єднання, dropout-регуляризацію та фінальний лінійний класифікатор. Підбір гіперпараметрів виконувався за допомогою Optuna в межах внутрішньої групової крос-валідації.

Таким чином, у Розділі 2 сформовано повну методику графового моделювання лонгітюдних протеомних даних: від підготовки початкових вимірювань до навчання графової нейронної мережі та збереження результатів по кожному фолду. Описана методика створює основу для експериментального порівняння графової моделі з класичними алгоритмами машинного навчання, що виконується у наступному розділі.

РОЗДІЛ 3
ЕКСПЕРИМЕНТАЛЬНА ОЦІНКА РОЗРОБЛЕНОЇ ГРАФОВОЇ МОДЕЛІ ТА ПОРІВНЯННЯ З КЛАСИЧНИМИ ПІДХОДАМИ

3.1 Загальна схема експериментального порівняння

У третьому розділі наведено результати експериментального порівняння класичних алгоритмів машинного навчання та розробленої графової моделі Velocity-DTW. Основна мета цього етапу полягає не лише в отриманні найвищого значення метрик, а й у перевірці методологічної стійкості різних підходів до аналізу малого лонгітюдного клінічного набору даних.

Порівняння виконувалося у чотирьох основних режимах. Перший режим відповідає результатам оригінального дослідження PERMAD, у якому було використано класичні алгоритми Random Forest, SVM та k-NN [1]. Другий режим відтворює близьку до оригінальної негрупову схему оцінювання, де розбиття виконується на рівні окремих часових зрізів. Третій режим використовує групову ізоляцію пацієнтів для класичних моделей. Четвертий режим відповідає запропонованій графовій моделі Velocity-DTW із груповою схемою валідації.

Окремо було виконано контрольний графовий експеримент без ANOVA-відбору, fingerprint-фільтрації та вагового підсилення білків. Його призначення полягає в тому, щоб оцінити, чи справді відбір ознак і контроль пацієнт-специфічних маркерів виконують роль регуляризації, а не є лише допоміжним технічним етапом.

Загальну схему експериментального порівняння наведено в таблиці 3.1.

У всіх режимах для порівняння використовувалися метрики Accuracy, Sensitivity, Specificity та SS2. Основною узагальнювальною метрикою є SS2, оскільки вона враховує баланс між правильним виявленням випадків прогресування та правильним розпізнаванням випадків без прогресування. Такий підхід відповідає логіці звітування в оригінальному дослідженні PERMAD [1].

Таблиця 3.1 – Основні експериментальні режими дослідження

Підхід	**Моделі**	**Схема валідації**	**Призначення**
Оригінальна стаття	RF, RF Top-10, SVM, k-NN	5 × 10 nested CV без групування пацієнтів	Початковий орієнтир для порівняння
Власне відтворення без групування	RF, RF Top-10, SVM, k-NN	5 × 10 StratifiedKFold	Перевірка відтворюваності негрупової схеми
Виправлена класична схема	RF, RF Top-10, SVM, k-NN	5 × 10 StratifiedGroupKFold	Оцінка після ізоляції пацієнтів
Графова модель	Velocity-DTW GNN	5 × 5 StratifiedGroupKFold	Перевірка графового представлення даних
Контрольний графовий прогін	Velocity-DTW GNN без відбору ознак	5 × 5 StratifiedGroupKFold	Перевірка ролі ANOVA, fingerprint і ваг білків

Для класичних моделей використовувалися алгоритми Random Forest, SVM та k-NN [4–6]. Для графової моделі використовувалося представлення часових зрізів як вузлів графа, а класифікація виконувалася за допомогою графової нейронної мережі з механізмом уваги [10, 26]. Підбір гіперпараметрів графової моделі виконувався за допомогою Optuna [8].

3.2 Відтворення негрупової схеми класичних моделей

Першим етапом експериментальної перевірки було відтворення схеми, близької до оригінального дослідження PERMAD. Мета цього етапу полягала в тому, щоб перевірити, чи здатна незалежна програмна реалізація отримати рівень якості, близький до опублікованих результатів. Для цього було використано негрупову схему розбиття, у якій окремі часові зрізи розглядаються як самостійні об’єкти.

У такій постановці відтворено головну особливість базового підходу: модель працює з табличним представленням протеомних профілів і не враховує групову належність записів до пацієнтів. Саме тому цей експеримент не розглядається як фінально коректна оцінка узагальнювальної здатності, а використовується як контрольна точка для порівняння з оригінальною статтею [1].

Тестові результати відтворення негрупової схеми наведено в таблиці 3.2.

Таблиця 3.2 – Тестові результати відтворення негрупової схеми класичних моделей

Модель	**Accuracy**	**Sensitivity**	**Specificity**	**SS2**
RF Full	79.7 [76.6; 82.8]	83.3 [79.2; 90.4]	77.1 [71.4; 82.9]	80.5 [77.9; 83.5]
RF Top-10	75.9 [72.4; 80.7]	82.6 [73.9; 91.3]	71.4 [66.4; 77.1]	77.1 [73.5; 81.2]
SVM	67.2 [63.8; 69.5]	48.9 [41.7; 54.2]	80.0 [74.3; 82.9]	64.6 [59.6; 67.0]
k-NN	58.6 [55.2; 62.7]	41.3 [34.8; 50.0]	70.0 [65.7; 77.1]	56.0 [52.6; 60.1]

Отримані результати показують, що Random Forest у негруповій схемі відтворює високий рівень якості, близький до оригінального дослідження. Для повного набору ознак Test SS2 становить 80.5, а для скороченої Top-10 моделі — 77.1. Це узгоджується з висновком оригінальної статті про те, що Random Forest є найсильнішою серед класичних моделей у цій постановці [1, 4].

Водночас цей результат не можна трактувати як остаточне підтвердження високої клінічної узагальнювальної здатності моделі. Як було показано в Розділі 1, негрупова схема може змішувати часові зрізи одного пацієнта між навчальною та тестовою частинами. У такому разі тестові метрики можуть частково відображати не здатність моделі прогнозувати нових пацієнтів, а здатність розпізнавати індивідуальні патерни пацієнтів, присутніх у навчанні [12, 13].

Результати SVM і k-NN залишаються нижчими за Random Forest. Це очікувано для малого високовимірного набору даних, де метричні та геометричні моделі можуть бути чутливими до масштабування, шуму та структури простору ознак [5, 6, 19]. У межах даної роботи ці моделі використовуються як базові орієнтири для подальшого порівняння з груповими й графовими підходами.

3.3 Результати класичних моделей із груповою ізоляцією пацієнтів

Другим етапом було оцінювання класичних моделей у груповій схемі крос-валідації. У цьому режимі всі записи одного пацієнта залишаються в межах одного фолду. Така схема є методологічно суворішою, оскільки модель тестується на пацієнтах, яких вона не бачила під час навчання [16–18].

Результати класичних моделей із груповою ізоляцією пацієнтів наведено в таблиці 3.3.

Таблиця 3.3 – Тестові результати класичних моделей із груповою ізоляцією пацієнтів

Модель	**Accuracy**	**Sensitivity**	**Specificity**	**SS2**
RF Full	57.8 [49.5; 61.6]	32.6 [19.1; 50.9]	74.3 [61.5; 83.3]	53.9 [48.6; 57.8]
RF Top-10	61.2 [53.0; 67.4]	59.9 [33.0; 68.7]	66.7 [53.5; 84.7]	60.6 [53.2; 66.4]
SVM	55.1 [48.3; 64.1]	64.8 [48.2; 82.1]	49.9 [32.4; 71.4]	57.3 [52.4; 63.6]
k-NN	46.1 [40.8; 54.5]	51.2 [40.0; 62.8]	45.7 [33.8; 56.5]	46.3 [41.4; 54.1]

Примітка. Таблицю сформовано за результатами власного експерименту з груповою ізоляцією пацієнтів.

Після переходу до групової валідації якість класичних моделей суттєво знизилася. Найбільше це помітно для RF Full: Test SS2 зменшився з 80.5 у негруповій схемі до 53.9 у груповій. Це свідчить про те, що висока якість Random Forest у негруповому режимі значною мірою залежить від способу розбиття даних.

Модель RF Top-10 у груповій схемі показала кращий результат, ніж RF Full: Test SS2 становив 60.6. Це можна пояснити тим, що скорочення простору ознак частково зменшує шум і ризик перенавчання. Для малого набору даних надмірна кількість ознак може не покращувати, а навпаки погіршувати узагальнювальну здатність моделі [20, 30].

Результати SVM і k-NN також залишаються нижчими за негрупову схему. SVM показав Test SS2 57.3, а k-NN — 46.3. Такі значення підтверджують, що задача прогнозування терапевтичної резистентності на рівні нових пацієнтів є значно складнішою, ніж оцінювання окремих часових зрізів без групової ізоляції.

Отже, групова перевірка підтвердила головну методологічну тезу цієї роботи: для лонгітюдних клінічних даних оцінювання без ізоляції пацієнтів може давати надто оптимістичні результати. Саме тому подальше порівняння графової моделі виконується з орієнтацією не на негрупові метрики, а на групову схему як суворіший і ближчий до реального клінічного застосування режим оцінювання.

3.4 Результати основної графової моделі Velocity-DTW

Третім експериментальним режимом була фінальна графова модель Velocity-DTW. На відміну від класичних моделей, вона використовує не лише поточні значення білків, а й локальну часову структуру протеомного профілю. Кожен вузол графа формується на основі часового вікна, а ребра між вузлами відображають подібність між локальними протеомними траєкторіями.

Фінальна модель використовує ANOVA-відбір ознак, fingerprint-фільтрацію, вагове підсилення білків, DTW-орієнтовану метрику, velocity-lens,

RBF-фільтрацію ребер і графову нейронну мережу з механізмом уваги [20–22, 24, 26]. Оцінювання виконувалося в груповій схемі крос-валідації. Підсумкові результати основного прогону за 25 графами наведено в таблиці 3.4.

Основна графова модель показала Test SS2 63.9. Це вище за результати класичних моделей у груповій схемі, зокрема RF Top-10 із Test SS2 60.6, SVM із Test SS2 57.3 та RF Full із Test SS2 53.9. Отже, графове представлення дало помірне, але помітне покращення порівняно з класичними моделями в суворішій схемі валідації.

Таблиця 3.4 – Результати основної графової моделі Velocity-DTW

Фаза	**Accuracy**	**Sensitivity**	**Specificity**	**SS2**
Training	79.5 [78.7; 80.0]	80.8 [79.1; 81.7]	78.7 [77.5; 80.1]	79.7 [78.9; 80.3]
Test	63.6 [58.1; 67.5]	63.2 [52.7; 75.9]	64.6 [56.1; 75.3]	63.9 [61.4; 67.8]

Примітка. Таблицю сформовано за результатами основного графового прогону з 25 фолдів/графів.

Водночас між навчальною та тестовою фазами зберігається розрив. Train SS2 становить 79.7, тоді як Test SS2 — 63.9. Це свідчить про помірне перенавчання, що є очікуваним для малої клінічної вибірки з високою розмірністю ознак. Важливо, що розрив не досягає рівня контрольного графового прогону без відбору ознак, розглянутого в підрозділі 3.5.

Приклад побудованого графа основної моделі наведено на рисунку 3.1. Повні приклади графів окремих фолдів можна винести в додаток Б, якщо потрібно показати варіативність структури між фолдами.

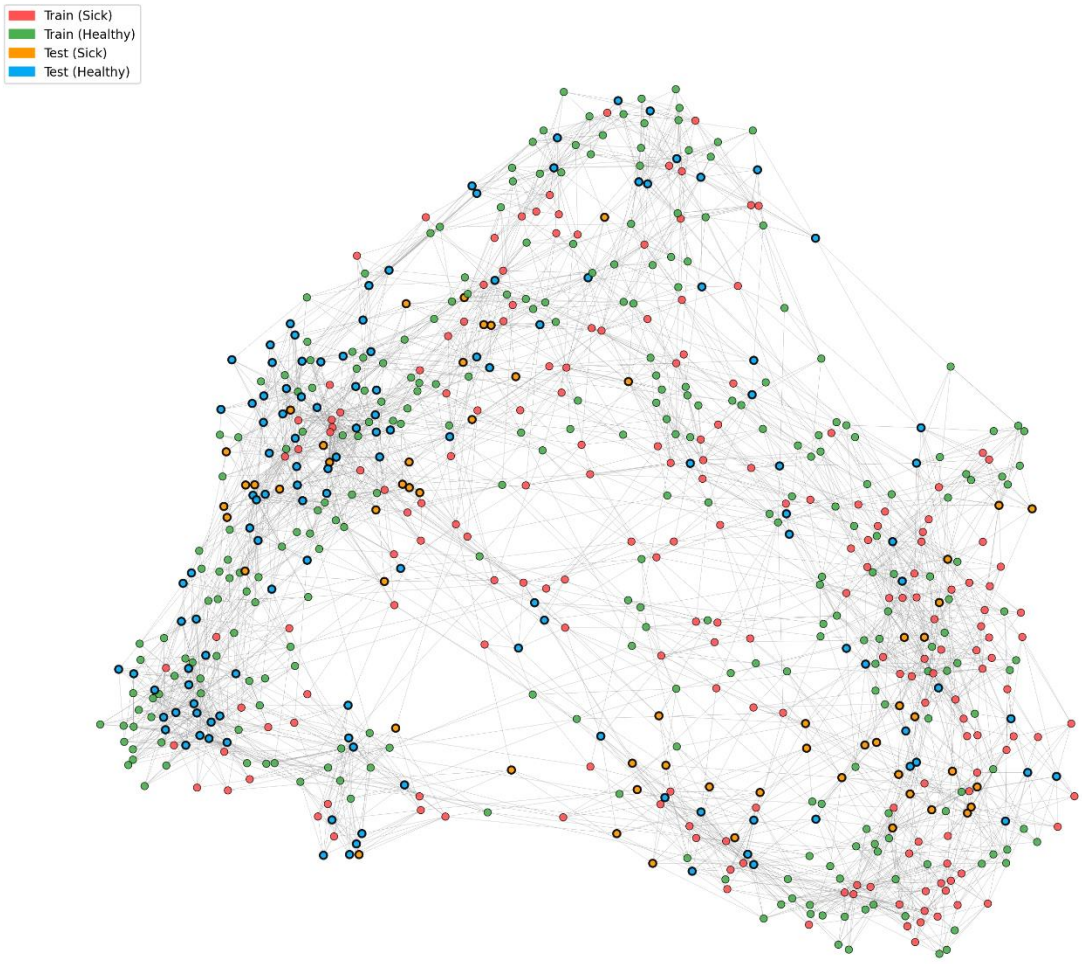


Рисунок 3.1 – Приклад графа подібності часових протеомних профілів у фінальній моделі Velocity-DTW

3.5 Контрольний експеримент без відбору ознак

Для перевірки ролі відбору ознак було виконано контрольний графовий експеримент без ANOVA-відбору, fingerprint-фільтрації та вагового підсилення білків. Мета цього експерименту полягала в тому, щоб оцінити, чи справді попередній відбір ознак покращує узагальнювальну здатність графової моделі, чи лише обмежує простір пошуку.

Результати контрольного прогону наведено в таблиці 3.5.

Контрольний експеримент показав різке зростання навчальних метрик: Train SS2 становив 98.6. Водночас Test SS2 знизився до 58.7. Такий результат є

типовою ознакою перенавчання: модель добре пристосовується до навчальних пацієнтів, але гірше переносить знайдені закономірності на тестові фолди.

Таблиця 3.5 – Результати контрольної графової моделі без відбору ознак

Фаза	**Accuracy**	**Sensitivity**	**Specificity**	**SS2**
Training	98.4 [97.9; 98.9]	99.4 [99.0; 100.0]	97.8 [97.4; 98.7]	98.6 [98.2; 98.9]
Test	56.8 [51.1; 61.3]	62.3 [54.2; 72.9]	55.1 [41.5; 67.0]	58.7 [56.0; 61.9]

Це підтверджує, що відбір ознак у фінальній моделі виконує роль регуляризації. ANOVA-відбір, fingerprint-фільтрація та вагове підсилення білків не просто скорочують кількість ознак, а допомагають сформувати більш стійкий простір подібності між часовими профілями. Без цього графова структура стає більш чутливою до шуму та індивідуальних особливостей окремих пацієнтів.

Таким чином, контрольний експеримент обґрунтовує використання ознакового обмеження у фінальній графовій моделі. Просте збільшення кількості білків не покращує тестову якість, а навпаки збільшує розрив між навчанням і тестуванням.

3.6 Порівняння основних експериментальних режимів

Для узагальнення результатів у таблиці 3.6 наведено порівняння Test SS2 для основних експериментальних режимів. Ця таблиця дозволяє стисло зіставити оригінальне дослідження, негрупове відтворення, групову схему класичних моделей і графові експерименти.

За таблицею 3.6 видно, що найвищі метрики отримано в оригінальному дослідженні та в негруповому відтворенні. Однак ці результати належать до менш суворої схеми оцінювання. Після переходу до групової ізоляції пацієнтів якість класичних моделей суттєво знижується.

Таблиця 3.6 – Порівняння Test SS2 для основних експериментальних режимів

Підхід	**Модель**	**Test SS2**
Оригінальна стаття	RF Full	79.2 [76.9; 82.0]
	RF Top-10	75.6 [71.2; 79.7]
Відтворення без групування	RF Full	80.5 [77.9; 83.5]
	RF Top-10	77.1 [73.5; 81.2]
Виправлена класична схема	RF Full	53.9 [48.6; 57.8]
	RF Top-10	60.6 [53.2; 66.4]
	SVM	57.3 [52.4; 63.6]
	k-NN	46.3 [41.4; 54.1]
Графова модель	Velocity-DTW	63.9 [61.4; 67.8]
Контрольний графовий прогін	Velocity-DTW без відбору ознак	58.7 [56.0; 61.9]

Основна графова модель Velocity-DTW не досягає рівня негрупових результатів Random Forest. Проте вона перевищує класичні моделі у груповій схемі. Це означає, що графовий підхід не усуває всіх обмежень малої клінічної вибірки, але дозволяє частково покращити тестову якість у більш коректному режимі оцінювання.

3.7 Аналіз важливості білків у графовій моделі

Окремим результатом графової моделі є рейтинг важливості білків. Він формується шляхом агрегації внеску білків за всіма фолдами основного експерименту. Такий рейтинг не є клінічно верифікованою біомаркерною панеллю, але дозволяє оцінити, які маркери найчастіше впливали на рішення графової моделі.

Підсумковий рейтинг білків основного графового прогону наведено в таблиці 3.7.

Таблиця 3.7 – Підсумковий рейтинг білків основної графової моделі

Позиція	**Білок**	**Mean Importance**	**Std Importance**	**Folds Present**
1	Insulin-like Growth Factor-Binding Protein 1 (IGFBP-1)	0.0365	0.0377	25
2	Epiregulin (EPR)	0.0359	0.0319	25
3	Vascular Endothelial Growth Factor (VEGF)	0.0290	0.0267	25
4	Galectin-3	0.0272	0.0252	25
5	EN-RAGE	0.0263	0.0285	25
6	Serum Amyloid P-Component (SAP)	0.0243	0.0279	25
7	6Ckine	0.0212	0.0279	25
8	Tetranectin	0.0160	0.0185	25
9	Immunoglobulin A (IgA)	0.0149	0.0201	25
10	Urokinase-type plasminogen activator receptor (uPAR)	0.0127	0.0163	25
11	Leucine-rich alpha-2-glycoprotein (LRG1)	0.0081	0.0137	25
12	Neuropilin-1	0.0073	0.0114	25
13	Antileukoproteinase (ALP)	0.0072	0.0091	25
14	Mast/stem cell growth factor receptor (SCFR)	0.0070	0.0100	25
15	Haptoglobin	0.0065	0.0101	25

Порівняння рейтингів показує частковий збіг між класичним Random Forest і графовою моделлю. Зокрема, IGFBP-1 та VEGF стабільно з'являються серед важливих маркерів у різних підходах. ALP, який є першим у рейтингах Random Forest, у графовій моделі також присутній, але займає нижчу позицію. Це означає, що графова модель не просто повторює важливість білків із RF, а перебудовує її відповідно до часової та мережевої структури даних.

У контрольному прогоні без відбору ознак серед важливих білків також з'явилися IGFBP-1, VEGF, Epiregulin, Neuropilin-1 та Galectin-3. Однак цей прогін супроводжується значним перенавчанням, тому його рейтинг не слід розглядати як надійніший або біологічно сильніший. Навпаки, він показує, що ширший простір ознак може виводити на перші позиції окремі маркери, але не забезпечує кращої тестової якості.

Таблиця 3.8 – Top-15 білків у контрольному графовому прогоні без відбору ознак

Позиція	**Білок**	**Mean Importance**	**Std Importance**	**Folds Present**
1	IGFBP-1	0.0307	0.0288	25
2	VEGF	0.0268	0.0330	25
3	MIP-3 beta	0.0146	0.0156	25
4	Complement C3 (C3)	0.0140	0.0154	25
5	IL-1ra	0.0130	0.0174	25
6	MSP	0.0130	0.0171	25
7	Epiregulin (EPR)	0.0126	0.0156	25
8	Adiponectin	0.0124	0.0181	25
9	Neuropilin-1	0.0119	0.0120	25
10	Galectin-3	0.0118	0.0200	25

11	BAFF	0.0115	0.0177	25
12	Cystatin-B	0.0110	0.0101	25
13	PARC	0.0107	0.0135	25
14	Fibulin-1C	0.0104	0.0114	25
15	Pepsinogen I (PGI)	0.0102	0.0132	25

Продрвження таблиці 3.8

Важливо, що показники Std Importance у багатьох білків є близькими до їхнього середнього внеску або навіть перевищують його. Це свідчить про нестабільність важливості між фолдами. Тому рейтинги білків у цій роботі мають інтерпретуватися як експериментальна характеристика моделей, а не як готова клінічна біомаркерна панель.

3.8 Обговорення отриманих результатів

Порівняння експериментальних режимів дозволяє зробити кілька основних висновків. По-перше, негрупова схема добре відтворює високі результати Random Forest, близькі до оригінального дослідження PERMAD. Це означає, що базовий машинно-навчальний підхід справді знаходить сигнал у даних. Однак характер цього сигналу потребує обережної інтерпретації.

По-друге, після переходу до групової ізоляції пацієнтів якість класичних моделей різко знижується. Це підтверджує ризик, описаний у Розділі 1: якщо часові зрізи одного пацієнта потрапляють у різні частини розбиття, модель може частково спиратися на індивідуальні пацієнт-специфічні патерни [12, 13]. Групова схема усуває цей ефект, але одночасно робить задачу суттєво складнішою.

По-третє, графова модель Velocity-DTW показує кращий результат, ніж класичні моделі в груповій схемі. Її Test SS2 становить 63.9, тоді як найкращий класичний результат у груповій схемі — 60.6 для RF Top-10. Це покращення не є радикальним, однак воно демонструє, що використання часових вікон і

графового представлення може бути корисним для аналізу лонгітюдних протеомних профілів.

По-четверте, контрольний графовий експеримент без відбору ознак показав значне перенавчання. Train SS2 98.6 при Test SS2 58.7 свідчить, що розширення ознакового простору без регуляризації не покращує узагальнювальну здатність. Це підтверджує роль ANOVA-відбору, fingerprint-фільтрації та вагового підсилення білків як важливих елементів фінальної методики.

Отримані результати також показують, що графова структура сама по собі не є гарантією високої якості. Якість моделі залежить від способу формування вузлів, метрики подібності, фільтрації ребер, кількості сусідів і регуляризації нейронної мережі. На малих клінічних даних навіть невеликі зміни цих компонентів можуть суттєво впливати на форму графа та тестові метрики.

Таким чином, основний результат роботи полягає не в тому, що графова модель повністю замінює класичні методи, а в тому, що вона дає помірне покращення в методологічно суворішій схемі оцінювання. Це важливий результат для малих лонгітюдних наборів даних, де головною проблемою є не лише досягнення високої точності, а й уникнення завищених оцінок через витік даних.

3.9 Обмеження експериментального дослідження

Отримані результати мають кілька обмежень. Перше обмеження пов’язане з розміром набору PERMAD. Незважаючи на наявність 652 часових зрізів, кількість незалежних пацієнтів у фінальному аналізі становить лише 41 [1]. Саме кількість пацієнтів, а не лише кількість записів, визначає реальну складність задачі узагальнення.

Друге обмеження пов’язане з високою розмірністю протеомного простору. Навіть після відбору ознак модель працює з даними, у яких кількість біологічних

змінних є значною відносно кількості пацієнтів. Це створює ризик перенавчання та нестабільності важливості білків між фолдами [20, 30].

Третє обмеження стосується трансдуктивного характеру графової моделі. Структура графа формується для всіх вузлів відповідного фолду, включаючи тестові вузли. При цьому тестові мітки не використовуються для навчання, тому прямого витоку міток немає. Однак така постановка відрізняється від повністю індуктивного клінічного сценарію, де новий пацієнт з'являється після навчання моделі й має бути доданий до графа окремо [27, 28].

Четверте обмеження пов'язане з нестабільністю графової структури. Побудова графа залежить від гіперпараметрів, метрики, кількості сусідів, RBF-фільтрації та складу фолду. Тому різні фолди можуть мати відмінну візуальну структуру, щільність ребер і локальні кластери. Це природно для малих клінічних даних, але потребує обережності при інтерпретації окремих графів.

П'яте обмеження полягає в тому, що отримані рейтинги білків не можна безпосередньо переносити в клінічну практику. Вони є результатом експериментальної моделі на одному наборі даних і потребують перевірки на незалежних вибірках. Тому біомаркерні результати слід розглядати як гіпотези для подальшої перевірки, а не як готову діагностичну панель.

Висновки до Розділу 3

У третьому розділі проведено експериментальне порівняння класичних алгоритмів машинного навчання та розробленої графової моделі Velocity-DTW. Спочатку було відтворено негрупову схему, близьку до оригінального дослідження PERMAD. У цьому режимі Random Forest показав високий результат: Test SS2 становив 80.5 для повного набору ознак і 77.1 для Top-10 моделі.

Після переходу до групової ізоляції пацієнтів якість класичних моделей суттєво знизилася. Найкращим серед класичних підходів у груповій схемі став

RF Top-10 із Test SS2 60.6. Це підтвердило, що негрупова схема може давати надто оптимістичні оцінки для лонгітюдних клінічних даних.

Основна графова модель Velocity-DTW показала Test SS2 63.9, що перевищує результати класичних моделей у груповій схемі. Це свідчить про практичну доцільність використання графового представлення часових протеомних профілів, хоча отримане покращення є помірним.

Контрольний графовий експеримент без відбору ознак показав значне перенавчання: Train SS2 становив 98.6, тоді як Test SS2 знизився до 58.7. Це підтвердило, що ANOVA-відбір, fingerprint-фільтрація та вагове підсилення білків є важливими елементами регуляризації графової моделі.

Аналіз важливості білків показав, що у фінальній графовій моделі найвищий внесок мали IGFBP-1, Epiregulin, VEGF, Galectin-3, EN-RAGE, SAP та 6Ckine. Частина цих білків перетинається з маркерами, важливими для класичного Random Forest, однак повного збігу рейтингів немає. Це пояснюється тим, що графова модель оцінює білки не лише як статичні ознаки, а як частину динамічного представлення часових профілів.

Загалом результати Розділу 3 показують, що графовий підхід не усуває повністю обмеження малої клінічної вибірки, але дозволяє отримати кращу тестову якість у методологічно суворішій груповій схемі оцінювання. Це підтверджує доцільність подальшого розвитку графових моделей для аналізу малих лонгітюдних біомедичних наборів даних.

ВИСНОВКИ

У дипломній роботі розглянуто задачу прогнозування терапевтичної резистентності у пацієнтів із метастатичним колоректальним раком на основі малих лонгітюдних протеомних наборів даних. Основну увагу приділено методологічній надійності машинного навчання в умовах малої кількості пацієнтів, високої розмірності простору ознак і наявності кількох часових зрізів для одного пацієнта.

У ході роботи було виконано критичний аналіз базового дослідження PERMAD, у якому для прогнозування резистентності до терапії бевацизумабом у поєднанні зі схемою FOLFOX застосовувалися класичні алгоритми машинного навчання. Було показано, що основним методологічним ризиком такого підходу є використання негрупової схеми крос-валідації, за якої різні часові зрізи одного пацієнта можуть одночасно потрапляти до навчальної та тестової вибірок. Для лонгітюдних клінічних даних це створює ризик прихованого витоку даних і може призводити до завищення оцінок якості.

Для кількісного обґрунтування цього ризику було проаналізовано пацієнт-специфічність протеомних маркерів. За допомогою показників Ratio, ICC та Eta2 виявлено групу білків, значна частина дисперсії яких пояснюється ідентичністю пацієнта. Це підтвердило, що частина ознак може поводитися як непрямий індивідуальний “відбиток” пацієнта, а не як узагальнений маркер терапевтичної відповіді. Такий результат обґрунтовує необхідність групової ізоляції пацієнтів під час оцінювання моделей.

Було виконано програмне відтворення класичної негрупової схеми машинного навчання, близької до базового дослідження. Отримані результати підтвердили відтворюваність високих метрик Random Forest: для повного набору ознак Test SS2 становив 80.5, а для RF Top-10 — 77.1. Це показало, що незалежна реалізація здатна відтворити рівень якості, близький до опублікованих результатів, якщо використовувати негрупове розбиття часових зрізів.

Після переходу до групової схеми крос-валідації якість класичних моделей суттєво знизилася. Для RF Full Test SS2 становив 53.9, для RF Top-10 — 60.6, для SVM — 57.3, для k-NN — 46.3. Це підтвердило, що висока якість негрупових моделей не повністю відображає здатність алгоритмів узагальнювати закономірності на нових пацієнтів. Отже, для оцінювання моделей на таких даних групова ізоляція пацієнтів є не додатковою опцією, а базовою методологічною вимогою.

У межах роботи було розроблено методику перетворення табличних лонгітюдних протеомних даних у графове представлення. Для цього було виконано попередню обробку даних PERMAD, формування часових змінних і цільової мітки, відбір ознак, побудову часових вікон, розрахунок матриці відстаней між локальними протеомними траєкторіями та формування графа подібності. У графовій моделі кожен вузол відповідає часовому зрізу пацієнта, а ребра відображають подібність між часовими профілями.

Для класифікації вузлів побудованого графа було реалізовано графову нейронну мережу Velocity-DTW, яка використовує ознаки вузлів, ваги ребер і структуру графового оточення. Модель навчалася в умовах групової крос-валідації, що забезпечувало ізоляцію пацієнтів між навчальною та тестовою частинами. Підбір гіперпараметрів виконувався за допомогою Optuna, а оцінювання проводилося за метриками Accuracy, Sensitivity, Specificity та SS2.

Фінальна графова модель показала Test Accuracy 63.6, Sensitivity 63.2, Specificity 64.6 та SS2 63.9. На навчальній частині SS2 становив 79.7, що свідчить про помірне перенавчання, очікуване для малої клінічної вибірки з високою розмірністю ознак. Водночас тестовий результат графової моделі перевищив результати виправлених класичних моделей у груповій схемі, зокрема RF Top-10 із Test SS2 60.6. Це підтвердило, що графове представлення лонгітюдних протеомних профілів може дати помірне покращення якості класифікації в суворішій схемі оцінювання.

Для перевірки ролі відбору ознак було проведено контрольний графовий експеримент без ANOVA-відбору, fingerprint-фільтрації та вагового підсилення білків. У цьому режимі модель досягла Train SS2 98.6, але Test SS2 знизився до 58.7. Такий результат показав, що просте розширення простору ознак не покращує узагальнювальну здатність моделі, а навпаки підсилює перенавчання. Отже, відбір ознак у фінальній графовій моделі є важливим елементом регуляризації, а не допоміжною технічною процедурою.

Аналіз важливості білків показав частковий збіг між класичними та графовими підходами. У графовій моделі найбільшу середню важливість показали IGFBP-1, Epiregulin, VEGF, Galectin-3, EN-RAGE, SAP та 6Ckine. Частина цих маркерів, зокрема IGFBP-1 і VEGF, також з'являлася серед важливих ознак у класичних моделях, що свідчить про певну стабільність біологічного сигналу. Водночас отримані рейтинги не слід трактувати як готову клінічну біомаркерну панель, оскільки вони залежать від малої вибірки, структури фолдів і параметрів моделі.

Практичним результатом роботи є реалізований аналітичний конвеєр мовою Python, який включає підготовку даних, групову крос-валідацію, контроль пацієнт-специфічних ознак, побудову графа подібності, навчання графової нейронної мережі, збереження результатів фолдів і аналіз важливості білків. Такий конвеєр може бути використаний як основа для подальших досліджень малих лонгітюдних клінічних наборів даних, де важливо уникати витоку інформації між навчальною та тестовою вибірками.

Основне обмеження отриманих результатів пов'язане з малим розміром набору PERMAD, високим рівнем шуму клінічних вимірювань і нестабільністю графової структури між окремими фолдами. Крім того, графова модель має трансдуктивний характер, оскільки структура графа формується для всіх вузлів відповідного фолду. Це не порушує навчання за тестовими мітками, але потребує обережної інтерпретації при перенесенні підходу в реальні клінічні сценарії.

Таким чином, поставлена мета роботи досягнута: було виконано критичну перевірку класичних підходів, реалізовано виправлену групову схему оцінювання, розроблено графову модель для аналізу лонгітюдних протеомних профілів і проведено її експериментальне порівняння з класичними алгоритмами машинного навчання. Отримані результати показали, що графове представлення може покращити якість класифікації порівняно з класичними моделями в умовах групової ізоляції пацієнтів, однак це покращення є помірним і потребує подальшої перевірки на більших незалежних клінічних наборах даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] Seufferlein T., Lausser L., Stein A., Arnold D., Prager G., Kasper-Virchow S., Niedermeier M., Müller L., Kubicka S., König A., Büchner-Steudel P., Wille K., Berger A. W., Kestler A. M. R., Kraus J. M., Werle S. D., Perkhofer L., Ettrich T. J., Kestler H. A. Prediction of resistance to bevacizumab plus FOLFOX in metastatic colorectal cancer — Results of the prospective multicenter PERMAD trial. PLOS ONE. 2024. Vol. 19, No. 6. Article e0304324. DOI: 10.1371/journal.pone.0304324.

[2] ClinicalTrials.gov. Personalized Marker-driven Early Switch to Aflibercept in Metastatic Colorectal Cancer (PERMAD). Identifier: NCT02331927. URL: https://clinicaltrials.gov/study/NCT02331927 (date of access: 17.04.2026).

[3] Eisenhauer E. A., Therasse P., Bogaerts J., Schwartz L. H., Sargent D., Ford R., Dancey J., Arbuck S., Gwyther S., Mooney M., Rubinstein L., Shankar L., Dodd L., Kaplan R., Lacombe D., Verweij J. New response evaluation criteria in solid tumours: revised RECIST guideline version 1.1. European Journal of Cancer. 2009. Vol. 45, No. 2. P. 228–247. DOI: 10.1016/j.ejca.2008.10.026.

[4] Breiman L. Random Forests. Machine Learning. 2001. Vol. 45. P. 5–32. DOI: 10.1023/A:1010933404324.

[5] Cortes C., Vapnik V. Support-vector networks. Machine Learning. 1995. Vol. 20. P. 273–297. DOI: 10.1007/BF00994018.

[6] Cover T., Hart P. Nearest neighbor pattern classification. IEEE Transactions on Information Theory. 1967. Vol. 13, No. 1. P. 21–27. DOI: 10.1109/TIT.1967.1053964.

[7] Pedregosa F., Varoquaux G., Gramfort A., Michel V., Thirion B., Grisel O., Blondel M., Prettenhofer P., Weiss R., Dubourg V., Vanderplas J., Passos A., Cournapeau D., Brucher M., Perrot M., Duchesnay E. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research. 2011. Vol. 12. P. 2825–2830.

[8] Akiba T., Sano S., Yanase T., Ohta T., Koyama M. Optuna: A Next-generation Hyperparameter Optimization Framework. Proceedings of the 25th ACM SIGKDD

International Conference on Knowledge Discovery and Data Mining. 2019. P. 2623–2631. DOI: 10.1145/3292500.3330701.

[9] Paszke A., Gross S., Massa F., Lerer A., Bradbury J., Chanan G., Killeen T., Lin Z., Gimelshein N., Antiga L., Desmaison A., Kopf A., Yang E., DeVito Z., Raison M., Tejani A., Chilamkurthy S., Steiner B., Fang L., Bai J., Chintala S. PyTorch: An Imperative Style, High-Performance Deep Learning Library. Advances in Neural Information Processing Systems. 2019. Vol. 32.

[10] Fey M., Lenssen J. E. Fast Graph Representation Learning with PyTorch Geometric. ICLR Workshop on Representation Learning on Graphs and Manifolds. 2019.

[11] Hagberg A. A., Schult D. A., Swart P. J. Exploring Network Structure, Dynamics, and Function using NetworkX. Proceedings of the 7th Python in Science Conference. 2008. P. 11–15.

[12] Kapoor S., Narayanan A. Leakage and the reproducibility crisis in machine-learning-based science. Patterns. 2023. Vol. 4, No. 9. Article 100804. DOI: 10.1016/j.patter.2023.100804.

[13] Rosenblatt M., Tejavibulya L., Jiang R., Noble S., Scheinost D. Data leakage inflates prediction performance in connectome-based machine learning models. Nature Communications. 2024. Vol. 15. Article 1829. DOI: 10.1038/s41467-024-46150-w.

[14] Koo T. K., Li M. Y. A Guideline of Selecting and Reporting Intraclass Correlation Coefficients for Reliability Research. Journal of Chiropractic Medicine. 2016. Vol. 15, No. 2. P. 155–163. DOI: 10.1016/j.jcm.2016.02.012.

[15] Richardson J. T. E. Eta squared and partial eta squared as measures of effect size in educational research. Educational Research Review. 2011. Vol. 6, No. 2. P. 135–147. DOI: 10.1016/j.edurev.2010.12.001.

[16] Scikit-learn documentation. StratifiedGroupKFold. URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedGroupKFold.html (date of access: 17.04.2026).

[17] Kohavi R. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. Proceedings of the 14th International Joint Conference on Artificial Intelligence. 1995. Vol. 2. P. 1137–1143.

[18] Stone M. Cross-validatory choice and assessment of statistical predictions. Journal of the Royal Statistical Society: Series B. 1974. Vol. 36, No. 2. P. 111–147. DOI: 10.1111/j.2517-6161.1974.tb00994.x.

[19] Bellman R. Adaptive Control Processes: A Guided Tour. Princeton: Princeton University Press, 1961. 255 p.

[20] Guyon I., Elisseeff A. An Introduction to Variable and Feature Selection. Journal of Machine Learning Research. 2003. Vol. 3. P. 1157–1182.

[21] Carlsson G. Topology and data. Bulletin of the American Mathematical Society. 2009. Vol. 46, No. 2. P. 255–308. DOI: 10.1090/S0273-0979-09-01249-X.

[22] Singh G., Mémoli F., Carlsson G. Topological Methods for the Analysis of High Dimensional Data Sets and 3D Object Recognition. Eurographics Symposium on Point-Based Graphics. 2007. P. 91–100. DOI: 10.2312/SPBG/SPBG07/091-100.

[23] Takens F. Detecting Strange Attractors in Turbulence. Dynamical Systems and Turbulence, Warwick 1980 / ed. by D. A. Rand, L.-S. Young. Lecture Notes in Mathematics. Vol. 898. Berlin, Heidelberg: Springer, 1981. P. 366–381. DOI: 10.1007/BFb0091924.

[24] Sakoe H., Chiba S. Dynamic programming algorithm optimization for spoken word recognition. IEEE Transactions on Acoustics, Speech, and Signal Processing. 1978. Vol. 26, No. 1. P. 43–49. DOI: 10.1109/TASSP.1978.1163055.

[25] Newman M. E. J. Modularity and community structure in networks. Proceedings of the National Academy of Sciences. 2006. Vol. 103, No. 23. P. 8577–8582. DOI: 10.1073/pnas.0601602103.

[26] Veličković P., Cucurull G., Casanova A., Romero A., Liò P., Bengio Y. Graph Attention Networks. International Conference on Learning Representations. 2018. URL: https://openreview.net/forum?id=rJXMpikCZ

(date of access: 17.04.2026).

[27] Kipf T. N., Welling M. Semi-Supervised Classification with Graph Convolutional Networks. International Conference on Learning Representations. 2017. URL: https://openreview.net/forum?id=SJU4ayYgl

(date of access: 17.04.2026).

[28] Hamilton W. L. Graph Representation Learning. Synthesis Lectures on Artificial Intelligence and Machine Learning. 2020. Vol. 14, No. 3. P. 1–159. DOI: 10.2200/S01045ED1V01Y202009AIM046.

[29] Ludäscher B., Altintas I., Berkley C., Higgins D., Jaeger E., Jones M., Lee E. A., Tao J., Zhao Y. Scientific workflow management and the Kepler system. Concurrency and Computation: Practice and Experience. 2006. Vol. 18, No. 10. P. 1039–1065. DOI: 10.1002/cpe.994.

[30] Hastie T., Tibshirani R., Friedman J. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. 2nd ed. New York: Springer, 2009. 745 p.

ДОДАТКИ

ДОДАТОК А − ПРОГРАМНІ МОДУЛІ ЕКСПЕРИМЕНТАЛЬНОГО ДОСЛІДЖЕННЯ

У додатку А наведено програмний код основних модулів, використаних для підготовки даних, відтворення класичних моделей, оцінювання пацієнт-специфічності білків та побудови графової моделі Velocity-DTW.

Таблиця А.1 – Перелік програмних модулів

№	Файл	Призначення
1	preprocess_article.py	Попередня обробка даних для класичних моделей
2	preprocess_graph.py	Попередня обробка даних для графової моделі
3	check_specificity.py	Розрахунок Ratio, ICC та Eta2 для білків
4	process_article.py	Відтворення негрупової схеми класичних моделей
5	process_article_grouped.py	Оцінювання класичних моделей із груповою ізоляцією пацієнтів
6	process_tda.py	Побудова графа Velocity-DTW та навчання графової нейронної мережі

Примітка. Назви файлів у таблиці мають відповідати фактичним назвам програмних модулів, використаних у роботі.

Лістинг А.1 – Файл preprocess_article.py

```
      import pandas as pd
      import numpy as np
      import pickle
      import os

      def preprocess_permad():
          header_df = pd.read_excel("PERMAD_cohort_1.xlsx", sheet_name=0, nrows=0)
          all_cols = header_df.columns.tolist()
          featurenames = [str(col).strip() for col in all_cols if 'Unnamed' not in
str(col)]

          if len(featurenames) > 102:
               featurenames = featurenames[-102:]

          raw_1 = pd.read_excel("PERMAD_cohort_1.xlsx", sheet_name=0, header=7)
          raw_2 = pd.read_excel("PERMAD_cohort_2.xlsx", sheet_name=0, header=7)

          dropouts_1_ids = ["106-001", "110-004", "112-001", "204-001"]
          dropouts_2_ids = ["107-010", "109-004", "112-002", "113-002", "113-005", "201-
010"]

          def get_root_id(val):
              if pd.isna(val): return ""
              return str(val).split(" ")[0]

          ids_to_drop_1   =   raw_1[raw_1['ID'].isin(dropouts_1_ids)]['Myriad   ID   +
Samples'].apply(get_root_id).tolist()
          ids_to_drop_2   =   raw_2[raw_2['ID'].isin(dropouts_2_ids)]['Myriad   ID   +
Samples'].apply(get_root_id).tolist()

          mask_drop_1            =            raw_1['Myriad            ID            +
Samples'].apply(get_root_id).isin(ids_to_drop_1)
          mask_drop_2            =            raw_2['Myriad            ID            +
Samples'].apply(get_root_id).isin(ids_to_drop_2)

          mask_na_1 = raw_1.iloc[:, 0].isna()
          mask_na_2 = raw_2.iloc[:, 0].isna()

          clean_1 = raw_1[~mask_drop_1 & ~mask_na_1].copy()
          clean_2 = raw_2[~mask_drop_2 & ~mask_na_2].copy()

          meta_cols_count = 5
          dat_1 = clean_1.iloc[:, meta_cols_count:].T
          dat_2 = clean_2.iloc[:, meta_cols_count:].T

          data_matrix = pd.concat([dat_1, dat_2], axis=1)

          if len(featurenames) == data_matrix.shape[0]:
              data_matrix.index = featurenames
          else:
              data_matrix.index      =      [f"Feature_{i+1}"      for      i      in
range(data_matrix.shape[0])]

          sample_names_1 = clean_1['Myriad ID + Samples']
          sample_names_2 = clean_2['Myriad ID + Samples']
          combined_names = pd.concat([sample_names_1, sample_names_2])
```

```
          cleaned_colnames    =    combined_names.astype(str).str.replace("    ",    "_",
n=1).str.replace(" ", "", regex=False)
          data_matrix.columns = cleaned_colnames

          data_matrix = data_matrix.replace({'<': '', '>': '', ' ': ''}, regex=True)
          data_matrix = data_matrix.apply(pd.to_numeric)

          cohort    =    np.concatenate([np.repeat(1,    dat_1.shape[1]),    np.repeat(2,
dat_2.shape[1])])
          sample_id = combined_names.apply(lambda x: int(str(x).split(' ')[0])).values

          time_deltact = pd.concat([clean_1['Abstand CT in Tagen'], clean_2['Abstand CT
in Tagen']]).values.astype(float)

          indices_26 = np.where(sample_id == 26)[0]
          if len(indices_26) > 0:
              time_deltact[indices_26[-1]] = 15

          trtct = (~np.isnan(time_deltact)).astype(int) * 2

          _, unique_indices = np.unique(sample_id, return_index=True)
          seen = set()
          first_occurrences = []
          for idx, sid in enumerate(sample_id):
              if sid not in seen:
                  first_occurrences.append(idx)
                  seen.add(sid)

          trtct[first_occurrences] = 1

          btonlyone = [9, 10, 12, 15, 32, 33, 35, 36, 38, 39, 40, 46, 49, 50]
          for idx in first_occurrences:
              sid = sample_id[idx]
              if sid not in btonlyone and (idx + 1) < len(sample_id) and sample_id[idx+1]
== sid:
                   trtct[idx + 1] = 1

          df_temp = pd.DataFrame({'sid': sample_id, 'idx': range(len(sample_id))})
          last_occurrences = df_temp.groupby('sid')['idx'].max().values
          trtct[last_occurrences] = 3

          dat_normalized = data_matrix.copy()

          for sid in np.unique(sample_id):
              mask_sid = (sample_id == sid)
              mask_bt = mask_sid & (trtct == 1)

              if not mask_bt.any():
                  continue

              baseline_mean = data_matrix.loc[:, mask_bt].mean(axis=1)
              dat_normalized.loc[:,        mask_sid]        =        data_matrix.loc[:,
mask_sid].sub(baseline_mean, axis=0)

          time_1    =    pd.to_datetime(clean_1['Datum    Abnahme'],    format="%d/%m/%Y",
dayfirst=True, errors='coerce')
          time_2    =    pd.to_datetime(clean_2['Datum    Abnahme'],    format="%d/%m/%Y",
dayfirst=True, errors='coerce')
```

```
            if time_1.isna().any():
                time_1    =    time_1.fillna(pd.to_datetime(clean_1['Datum    Abnahme'],
errors='coerce'))
            if time_2.isna().any():
                time_2    =    time_2.fillna(pd.to_datetime(clean_2['Datum    Abnahme'],
errors='coerce'))

            time_date = pd.concat([time_1, time_2]).reset_index(drop=True)

            time_bp = np.zeros(len(sample_id))
            time_ct_arr = np.zeros(len(sample_id))
            time_id = np.zeros(len(sample_id))

            for sid in np.unique(sample_id):
                idx = np.where(sample_id == sid)[0]
                idx_3 = idx[trtct[idx] == 3]

                if len(idx_3) == 0:
                    continue

                ref_date = time_date.iloc[idx_3[0]]
                ref_delta = time_deltact[idx_3[0]]
                current_dates = time_date.iloc[idx]

                diffs = (current_dates - ref_date).dt.days
                time_bp[idx] = diffs - ref_delta

                current_deltas = time_deltact[idx]
                time_ct_arr[idx] = diffs - ref_delta + np.nan_to_num(current_deltas)

                seq = np.arange(-46, -46 + len(idx))
                time_id[idx] = np.abs(seq)[::-1]

            labs = (time_bp > -100).astype(int) + 1

            cols_to_remove = ["10_8", "21_7"]
            invalid_mask = dat_normalized.columns.isin(cols_to_remove)
            remove_mask = invalid_mask | (trtct == 1)
            keep_mask = ~remove_mask

            PERMAD = {
                "data": dat_normalized.loc[:, keep_mask],
                "labs": labs[keep_mask],
                "sample_id": sample_id[keep_mask],
                "cohort": cohort[keep_mask],
                "time": time_bp[keep_mask],
                "time_ct": time_ct_arr[keep_mask],
                "time_id": time_id[keep_mask],
                "features": data_matrix.index.tolist(),
                "name": "PERMAD"
            }

            os.makedirs("data", exist_ok=True)
            with open("data/PERMAD.pkl", "wb") as f:
                pickle.dump(PERMAD, f)

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

Лістинг А.2 – Файл preprocess_graph.py

```
      import pandas as pd
      import numpy as np
      import pickle
      import os

      def preprocess_permad():
          # Витягування назв ознак (біомаркерів)
          header_df = pd.read_excel("PERMAD_cohort_1.xlsx", sheet_name=0, nrows=0)
          all_cols = header_df.columns.tolist()
          featurenames = [str(col).strip() for col in all_cols if 'Unnamed' not in
str(col)]

          if len(featurenames) > 102:
               featurenames = featurenames[-102:]

          # Завантаження сирих даних
          raw_1 = pd.read_excel("PERMAD_cohort_1.xlsx", sheet_name=0, header=7)
          raw_2 = pd.read_excel("PERMAD_cohort_2.xlsx", sheet_name=0, header=7)

          # Видалення проблемних пацієнтів
          dropouts_1_ids = ["106-001", "110-004", "112-001", "204-001"]
          dropouts_2_ids = ["107-010", "109-004", "112-002", "113-002", "113-005", "201-
010"]

          def get_root_id(val):
              if pd.isna(val): return ""
              return str(val).split(" ")[0]

          ids_to_drop_1   =   raw_1[raw_1['ID'].isin(dropouts_1_ids)]['Myriad   ID   +
Samples'].apply(get_root_id).tolist()
          ids_to_drop_2   =   raw_2[raw_2['ID'].isin(dropouts_2_ids)]['Myriad   ID   +
Samples'].apply(get_root_id).tolist()

          mask_drop_1            =            raw_1['Myriad            ID            +
Samples'].apply(get_root_id).isin(ids_to_drop_1)
          mask_drop_2            =            raw_2['Myriad            ID            +
Samples'].apply(get_root_id).isin(ids_to_drop_2)

          mask_na_1 = raw_1.iloc[:, 0].isna()
          mask_na_2 = raw_2.iloc[:, 0].isna()

          clean_1 = raw_1[~mask_drop_1 & ~mask_na_1].copy()
          clean_2 = raw_2[~mask_drop_2 & ~mask_na_2].copy()

          meta_cols_count = 5
          dat_1 = clean_1.iloc[:, meta_cols_count:].T
          dat_2 = clean_2.iloc[:, meta_cols_count:].T

          data_matrix = pd.concat([dat_1, dat_2], axis=1)

          if len(featurenames) == data_matrix.shape[0]:
              data_matrix.index = featurenames
          else:
              data_matrix.index      =      [f"Feature_{i+1}"      for      i      in
range(data_matrix.shape[0])]

          sample_names_1 = clean_1['Myriad ID + Samples']
```

```
            sample_names_2 = clean_2['Myriad ID + Samples']
            combined_names = pd.concat([sample_names_1, sample_names_2])

            cleaned_colnames    =    combined_names.astype(str).str.replace("    ",    "_",
n=1).str.replace(" ", "", regex=False)
            data_matrix.columns = cleaned_colnames

            data_matrix = data_matrix.replace({'<': '', '>': '', ' ': ''}, regex=True)
            data_matrix = data_matrix.apply(pd.to_numeric)

            cohort    =    np.concatenate([np.repeat(1,    dat_1.shape[1]),    np.repeat(2,
dat_2.shape[1])])
            sample_id = combined_names.apply(lambda x: int(str(x).split(' ')[0])).values

            time_deltact = pd.concat([clean_1['Abstand CT in Tagen'], clean_2['Abstand CT
in Tagen']]).values.astype(float)

            indices_26 = np.where(sample_id == 26)[0]
            if len(indices_26) > 0:
                time_deltact[indices_26[-1]] = 15

            trtct = (~np.isnan(time_deltact)).astype(int) * 2

            _, unique_indices = np.unique(sample_id, return_index=True)
            seen = set()
            first_occurrences = []
            for idx, sid in enumerate(sample_id):
                if sid not in seen:
                    first_occurrences.append(idx)
                    seen.add(sid)

            trtct[first_occurrences] = 1

            btonlyone = [9, 10, 12, 15, 32, 33, 35, 36, 38, 39, 40, 46, 49, 50]
            for idx in first_occurrences:
                sid = sample_id[idx]
                if sid not in btonlyone and (idx + 1) < len(sample_id) and sample_id[idx+1]
== sid:
                     trtct[idx + 1] = 1

            df_temp = pd.DataFrame({'sid': sample_id, 'idx': range(len(sample_id))})
            last_occurrences = df_temp.groupby('sid')['idx'].max().values
            trtct[last_occurrences] = 3

            # Розрахунок часових міток
            time_1    =    pd.to_datetime(clean_1['Datum    Abnahme'],    format="%d/%m/%Y",
dayfirst=True)
            time_2    =    pd.to_datetime(clean_2['Datum    Abnahme'],    format="%d/%m/%Y",
dayfirst=True)
            time_date = pd.concat([time_1, time_2]).reset_index(drop=True)

            time_bp = np.zeros(len(sample_id))
            time_ct_arr = np.zeros(len(sample_id))
            time_id = np.zeros(len(sample_id))

            for sid in np.unique(sample_id):
                idx = np.where(sample_id == sid)[0]
                idx_3 = idx[trtct[idx] == 3]
```

```
            if len(idx_3) == 0:
                continue

            ref_date = time_date.iloc[idx_3[0]]
            ref_delta = time_deltact[idx_3[0]]
            current_dates = time_date.iloc[idx]

            diffs = (current_dates - ref_date).dt.days
            time_bp[idx] = diffs - ref_delta

            current_deltas = time_deltact[idx]
            time_ct_arr[idx] = diffs - ref_delta + np.nan_to_num(current_deltas)

            seq = np.arange(-46, -46 + len(idx))
            time_id[idx] = np.abs(seq)[::-1]

        # Формування міток
        labs = (time_bp > -100).astype(int)

        # Видалення лише бракованих зразків (збереження долікувальних trtct == 1)
        cols_to_remove = ["10_8", "21_7"]
        invalid_mask = data_matrix.columns.isin(cols_to_remove)
        keep_mask = ~invalid_mask

        PERMAD = {
            "data": data_matrix.loc[:, keep_mask],
            "labs": labs[keep_mask],
            "sample_id": sample_id[keep_mask],
            "cohort": cohort[keep_mask],
            "time": time_bp[keep_mask],
            "time_ct": time_ct_arr[keep_mask],
            "time_id": time_id[keep_mask],
            "features": data_matrix.index.tolist(),
            "name": "PERMAD_Graph_Raw"
        }

        os.makedirs("data", exist_ok=True)
        with open("data/PERMAD_graph_raw.pkl", "wb") as f:
            pickle.dump(PERMAD, f)

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

Лістинг А.3 – Файл check_specificity.py

```
import pandas as pd
import numpy as np
import pickle
import os
import warnings

def main():
    target_proteins = ["ALL"]
    file_path = "PERMAD.pkl"

    if not os.path.exists(file_path):
        if os.path.exists("data/PERMAD.pkl"):
            file_path = "data/PERMAD.pkl"
        else:
            print(f"[!] Помилка: Файл {file_path} не знайдено!")
            return

    with open(file_path, "rb") as f:
        PERMAD = pickle.load(f)

    features = list(PERMAD['features'])
    df = pd.DataFrame(PERMAD['data'].T, columns=features)
    df['Patient_ID'] = PERMAD['sample_id']

    if "ALL" in [str(p).upper() for p in target_proteins]:
        target_proteins = features

    valid_proteins = [p for p in target_proteins if p in df.columns]
    if not valid_proteins:
        return

    # Логарифмічне перетворення для стабілізації розподілу
    df_subset = df[valid_proteins + ['Patient_ID']].copy()
    df_subset[valid_proteins] = np.log2(df_subset[valid_proteins].astype(float) +
1.0)

    # ==========================================================================
    # БАЗОВІ ОБЧИСЛЕННЯ (Спільні для всіх методів)
    # ==========================================================================
    N = len(df_subset)
    k = df_subset['Patient_ID'].nunique()

    counts = df_subset.groupby('Patient_ID')[valid_proteins].count()
    n0 = (N - (counts**2).sum() / N) / (k - 1)

    grand_mean = df_subset[valid_proteins].mean()
    patient_means = df_subset.groupby('Patient_ID')[valid_proteins].mean()
    patient_sds = df_subset.groupby('Patient_ID')[valid_proteins].std(ddof=1)

    group_means_transformed                                                    =
df_subset.groupby('Patient_ID')[valid_proteins].transform('mean')

    # ==========================================================================
    # 1. МЕТОД RATIO (SD_inter / SD_intra)
    # ==========================================================================
    inter_sd = patient_means.std(ddof=1)
    intra_sd = patient_sds.mean()
```

```
            # ========================================================================
            # 2. МЕТОД ICC (Intraclass Correlation Coefficient, ANOVA)
            # ========================================================================
            ss_within = ((df_subset[valid_proteins] - group_means_transformed)**2).sum()
            ss_between = ((patient_means - grand_mean)**2).multiply(counts, axis=0).sum()

            ms_within = ss_within / (N - k)
            ms_between = ss_between / (k - 1)

            var_inter = np.maximum(0.0, (ms_between - ms_within) / n0)
            var_intra = np.maximum(0.0, ms_within)

            with np.errstate(divide='ignore', invalid='ignore'):
                icc = np.where((var_inter + var_intra) > 0, var_inter / (var_inter +
var_intra), 0.0)
            icc = np.clip(icc, 0.0, 1.0)

            pooled_intra_sd = np.sqrt(ss_within / (N - k))

            with np.errstate(divide='ignore', invalid='ignore'):
                ratio_pooled = np.where(pooled_intra_sd > 0, inter_sd / pooled_intra_sd,
np.inf)

            # ========================================================================
            # 3. МЕТОД ETA-SQUARED (η²)
            # ========================================================================
            sst = ((df_subset[valid_proteins] - grand_mean)**2).sum()

            with np.errstate(divide='ignore', invalid='ignore'):
                eta_squared = np.where(sst > 0, ss_between / sst, 0.0)
            eta_squared = np.clip(eta_squared, 0.0, 1.0)

            # ========================================================================
            # ФОРМУВАННЯ СТАТУСУ ТА РЕЗУЛЬТАТІВ
            # ========================================================================
            # Пороги чутливості для визначення "Ідентифікатора"
            flag_ratio = ratio_pooled > 2.0
            flag_icc = icc > 0.8
            flag_eta2 = eta_squared > 0.8

            # Створення строкового ідентифікатора (наприклад: "+++", "+--")
            status_str = (
                np.where(flag_ratio, '+', '-') +
                np.where(flag_icc, '+', '-') +
                np.where(flag_eta2, '+', '-')
            )

            # Збірка підсумкового DataFrame
            results_df = pd.DataFrame({
                'Protein': valid_proteins,
                'Ratio': ratio_pooled,
                'ICC': icc,
                'Eta2': eta_squared,
                'Status_[Ratio,ICC,Eta]': status_str
            })
```

```
            # Сортування: спочатку за кількістю спрацьованих критеріїв, потім за Eta2 та
Ratio
            results_df['IdentifierScore'] = (
                flag_ratio.astype(int) +
                flag_icc.astype(int) +
                flag_eta2.astype(int)
                )

            results_df = results_df.sort_values(
                by=['IdentifierScore', 'Eta2', 'ICC', 'Ratio'],
                ascending=[False, False, False, False]
                ).reset_index(drop=True)

            # ======================================================================
            # ВИВІД ТА ЗБЕРЕЖЕННЯ
            # ======================================================================
            pd.set_option('display.max_rows', 50)
            pd.set_option('display.max_columns', None)
            pd.set_option('display.width', 1000)
            pd.set_option('display.float_format', lambda x: f'{x:.4f}')

            print("\n" + "="*95)
            print(results_df.head(30).to_string(index=False))

            if len(results_df) > 30:
                print(f"\n... та ще {len(results_df) - 30} білків.")

            os.makedirs("results_specificity", exist_ok=True)
            results_df.to_pickle("results_specificity/protein_all_metrics.pkl")
            results_df.to_csv("results_specificity/protein_all_metrics.csv", index=False)

            print("="*95)
            print("[INFO] Легенда статусу [Ratio, ICC, Eta2]:")
            print("      '+' означає, що за відповідною метрикою білок має ознаки пацієнт-
специфічного маркера.")
            print("         '-' означає, що за відповідною метрикою не виявлено високої
пацієнт-специфічності.")
            print("       Пороги: Ratio > 2.0 | ICC > 0.8 | Eta2 > 0.8")

        if __name__ == "__main__":
            warnings.filterwarnings("ignore")
            main()
```

Лістинг А.4 – Файл process _article.py

```
try:
    from sklearnex import patch_sklearn
    patch_sklearn()
except ImportError:
    pass

import pandas as pd
import numpy as np
import pickle
import os
import optuna
import warnings
from functools import partial
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import confusion_matrix
from scipy.stats import rankdata
from joblib import Parallel, delayed

# Загальні налаштування бібліотек
warnings.filterwarnings("ignore")
optuna.logging.set_verbosity(optuna.logging.INFO)


def calculate_metrics(y_true, y_pred):
    cm = confusion_matrix(y_true, y_pred, labels=[1, 2])
    if cm.shape != (2, 2):
        return {"Acc": 0, "Sens1": 0, "Sens2": 0, "SS2": 0}
    tn, fp, fn, tp = cm.ravel()
    acc = (tp + tn) / (tp + tn + fp + fn) if (tp + tn + fp + fn) > 0 else 0
    sens1 = tn / (tn + fp) if (tn + fp) > 0 else 0
    sens2 = tp / (tp + fn) if (tp + fn) > 0 else 0
    ss2 = (sens1 + sens2) / 2
    return {"Acc": acc, "Sens1": sens1, "Sens2": sens2, "SS2": ss2}


def generate_cv_runs(y, ntimes=5, nfold=10, seed=54321):
    runs = []
    for i in range(ntimes):
        kf = StratifiedKFold(n_splits=nfold, shuffle=True, random_state=seed + i)
        folds = [test_idx for _, test_idx in kf.split(np.zeros(len(y)), y)]
        runs.append(folds)
    return runs


def objective(trial, X_train, y_train, model_type):
    inner_folds = generate_cv_runs(y_train, ntimes=5, nfold=10, seed=123)
    val_metrics_list = []

    if model_type == "KNN":
        model                                                                    =
KNeighborsClassifier(n_neighbors=trial.suggest_categorical('n_neighbors', [1, 3, 5, 7]))
    elif model_type == "SVM":
        model = LinearSVC(
            C=trial.suggest_categorical('C', [0.01, 0.1, 1.0, 10.0, 100.0]),
```

```
                    random_state=123,
                    max_iter=10000,
                    tol=1e-3,
                    dual=False
                )
            elif model_type in ["RF", "RF_top10"]:
                wt = trial.suggest_float('classwt', 0.05, 0.95, step=0.05)
                model = RandomForestClassifier(
                    n_estimators=trial.suggest_int('n_estimators', 25, 100, step=25),
                    min_samples_leaf=trial.suggest_int('min_samples_leaf', 1, 5),
                    max_features='sqrt',
                    class_weight={1: wt, 2: 1.0 - wt},
                    random_state=123,
                    n_jobs=1
                )

            for run in inner_folds:
                for val_idx in run:
                    mask = np.ones(len(y_train), dtype=bool)
                    mask[val_idx] = False
                    model.fit(X_train[mask], y_train[mask])
                    val_metrics_list.append(calculate_metrics(y_train[val_idx],
model.predict(X_train[val_idx])))

            avg_metrics = {k: np.mean([m[k] for m in val_metrics_list]) for k in
val_metrics_list[0].keys()}
            for m_name, m_val in avg_metrics.items():
                trial.set_user_attr(m_name, m_val)
            return avg_metrics['SS2']


        def optimize_and_train_fold(r_idx, f_idx, test_idx, X, y, model_type, n_trials,
temp_dir):
            temp_file = os.path.join(temp_dir, f"run_{r_idx}_fold_{f_idx}.pkl")
            if os.path.exists(temp_file):
                with open(temp_file, "rb") as f:
                    return pickle.load(f)

            mask_train = np.ones(len(y), dtype=bool)
            mask_train[test_idx] = False
            X_train, y_train, X_test, y_test = X[mask_train], y[mask_train], X[test_idx],
y[test_idx]

            study = optuna.create_study(direction="maximize",
sampler=optuna.samplers.TPESampler(seed=123 + r_idx * 100 + f_idx))
            study.optimize(partial(objective, X_train=X_train, y_train=y_train,
model_type=model_type), n_trials=n_trials)

            best_metrics_inner = {k: study.best_trial.user_attrs[k] for k in ["Acc",
"Sens1", "Sens2", "SS2"]}
            bp = study.best_params.copy()

            if model_type == "KNN":
                model = KNeighborsClassifier(n_neighbors=bp['n_neighbors'])
            elif model_type == "SVM":
                model = LinearSVC(C=bp['C'], random_state=123, max_iter=10000, tol=1e-3,
dual=False)
            elif "RF" in model_type:
```

```
            model      =      RandomForestClassifier(n_estimators=bp['n_estimators'],
min_samples_leaf=bp['min_samples_leaf'],
                                                  max_features='sqrt',      class_weight={1:
bp['classwt'], 2: 1.0 - bp['classwt']},
                                                  random_state=123, n_jobs=-1)

          model.fit(X_train, y_train)
          res = {
              'run': r_idx,
              'fold': f_idx,
              'metrics_train': best_metrics_inner,
              'metrics_train_final':                          calculate_metrics(y_train,
model.predict(X_train)),
              'metrics_test': calculate_metrics(y_test, model.predict(X_test)),
              'params': bp
          }
          if "RF" in model_type:
              res['feature_importances'] = model.feature_importances_
          with open(temp_file, "wb") as f:
              pickle.dump(res, f)
          return res


      def run_simulation(data_obj, model_type, n_trials):
          X, y = data_obj['data'].T.values, np.array(data_obj['labs']).astype(int)
          temp_dir = os.path.join("results_article_baseline", f"temp_{model_type}")
          os.makedirs(temp_dir, exist_ok=True)

          final_path = f"results_article_baseline/RESULT_{model_type}.pkl"
          if os.path.exists(final_path):
              with open(final_path, "rb") as f:
                  return pickle.load(f)

          outer_folds = generate_cv_runs(y, ntimes=5, nfold=10)
          tasks = [(r_idx, f_idx, tidx) for r_idx, run in enumerate(outer_folds) for
f_idx, tidx in enumerate(run)]

          results = Parallel(n_jobs=-1, backend='loky', verbose=10)(
              delayed(optimize_and_train_fold)(*t, X, y, model_type, n_trials, temp_dir)
for t in tasks
          )

          with open(final_path, "wb") as f:
              pickle.dump(results, f)
          return results


      def main():
          if not os.path.exists("data/PERMAD.pkl"):
              return
          with open("data/PERMAD.pkl", "rb") as f:
              PERMAD = pickle.load(f)
          os.makedirs("results_article_baseline", exist_ok=True)

          for m_key, trials in [("KNN", 4), ("SVM", 5), ("RF", 60)]:
              path = f"results_article_baseline/RESULT_{m_key}.pkl"
              if not os.path.exists(path):
                  res = run_simulation(PERMAD, m_key, n_trials=trials)
                  with open(path, "wb") as f:
```

```
                    pickle.dump(res, f)

        res_rf_path = "results_article_baseline/RESULT_RF.pkl"
        if os.path.exists(res_rf_path):
            res_rf = pickle.load(open(res_rf_path, "rb"))
            all_imp = [r['feature_importances'] for r in res_rf if
'feature_importances' in r]
            if len(all_imp) == 0:
                return
            mean_ranks = np.mean([rankdata(imp) for imp in all_imp], axis=0)

            ranking_df = pd.DataFrame({
                'Biomarker': PERMAD['features'],
                'Mean_Rank': mean_ranks,
                'Mean_Gini': np.mean(all_imp, axis=0)
            }).sort_values('Mean_Rank', ascending=False)


ranking_df.to_pickle("results_article_baseline/protein_importance_statistics.pkl")
            ranking_df.to_csv("results_article_baseline/FULL_PROTEIN_RANKING.csv",
index=False)

            top10_idx = ranking_df.index[:10].tolist()
            PERMAD_top10 = PERMAD.copy()
            PERMAD_top10['data'], PERMAD_top10['features'] =
PERMAD['data'].iloc[top10_idx, :], np.array(PERMAD['features'])[top10_idx]

            if not os.path.exists("results_article_baseline/RESULT_RF_top10.pkl"):
                res_rf10 = run_simulation(PERMAD_top10, "RF_top10", n_trials=40)
                with open("results_article_baseline/RESULT_RF_top10.pkl", "wb") as f:
                    pickle.dump(res_rf10, f)


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

Лістинг А.5 – Файл process_article_grouped.py

```
import numpy as np
import pandas as pd
import pickle
import os
import optuna
import warnings
from functools import partial
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import StratifiedGroupKFold
from sklearn.metrics import confusion_matrix
from scipy.stats import rankdata
from joblib import Parallel, delayed

# ======================================================================
# ГЛОБАЛЬНІ НАЛАШТУВАННЯ
# ======================================================================
SAVE_DIR = "results_article_grouped"
DATA_PATH = "data/PERMAD.pkl"
FINGERPRINT_PATH = os.path.join("results_specificity", "protein_all_metrics.csv")

warnings.filterwarnings("ignore")
optuna.logging.set_verbosity(optuna.logging.INFO)
os.makedirs(SAVE_DIR, exist_ok=True)

# ======================================================================
# МЕТРИКИ
# ======================================================================


def calculate_metrics(y_true, y_pred):
    cm = confusion_matrix(y_true, y_pred, labels=[1, 2])
    if cm.shape != (2, 2):
        return {"Acc": 0, "Sens1": 0, "Sens2": 0, "SS2": 0}
    tn, fp, fn, tp = cm.ravel()
    acc = (tp + tn) / (tp + tn + fp + fn) if (tp + tn + fp + fn) > 0 else 0
    sens1 = tn / (tn + fp) if (tn + fp) > 0 else 0
    sens2 = tp / (tp + fn) if (tp + fn) > 0 else 0
    return {"Acc": acc, "Sens1": sens1, "Sens2": sens2, "SS2": (sens1 + sens2) /
2}

# ======================================================================
# ЯДРО ОБЧИСЛЕНЬ
# ======================================================================


def generate_group_cv_runs(X, y, groups, ntimes=5, nfold=10, seed=54321):
    runs = []
    for i in range(ntimes):
        cv = StratifiedGroupKFold(n_splits=nfold, shuffle=True, random_state=seed
+ i)
        folds = [test_idx for _, test_idx in cv.split(X, y, groups)]
        runs.append(folds)
    return runs
```

```
      def build_model(trial, model_type):
          if model_type == "KNN":
              return Pipeline([
                  ('scaler', StandardScaler()),
                  ('model',
KNeighborsClassifier(n_neighbors=trial.suggest_categorical('n_neighbors',  [1,  3,  5,
7])))
              ])
          elif model_type == "SVM":
              return Pipeline([
                  ('scaler', StandardScaler()),
                  ('model', LinearSVC(
                      C=trial.suggest_categorical('C', [0.01, 0.1, 1.0, 10.0, 100.0]),
                      random_state=123,
                      max_iter=10000,
                      tol=1e-3,
                      dual=False
                  ))
              ])
          elif "RF" in model_type:
              wt = trial.suggest_float('classwt', 0.05, 0.95, step=0.05)
              return RandomForestClassifier(
                  n_estimators=trial.suggest_int('n_estimators', 25, 100, step=25),
                  min_samples_leaf=trial.suggest_int('min_samples_leaf', 1, 5),
                  max_features='sqrt',
                  class_weight={1: wt, 2: 1.0 - wt},
                  random_state=123,
                  n_jobs=1
              )


      def build_final_model(bp, model_type):
          if model_type == "KNN":
              return Pipeline([
                  ('scaler', StandardScaler()),
                  ('model', KNeighborsClassifier(n_neighbors=bp['n_neighbors']))
              ])
          elif model_type == "SVM":
              return Pipeline([
                  ('scaler', StandardScaler()),
                  ('model',  LinearSVC(C=bp['C'],  random_state=123,  max_iter=10000,
tol=1e-3, dual=False))
              ])
          elif "RF" in model_type:
              return RandomForestClassifier(
                  n_estimators=bp['n_estimators'],
                  min_samples_leaf=bp['min_samples_leaf'],
                  max_features='sqrt',
                  class_weight={1: bp['classwt'], 2: 1.0 - bp['classwt']},
                  random_state=123,
                  n_jobs=-1
              )


      def objective(trial, X_train, y_train, g_train, model_type):
          inner_folds  =  generate_group_cv_runs(X_train,  y_train,  g_train,  ntimes=5,
nfold=10, seed=123)
```

```
            val_metrics_list = []
            model = build_model(trial, model_type)

            for run in inner_folds:
                for val_idx in run:
                    mask = np.ones(len(y_train), dtype=bool)
                    mask[val_idx] = False
                    model.fit(X_train[mask], y_train[mask])
                    val_metrics_list.append(calculate_metrics(y_train[val_idx],
model.predict(X_train[val_idx])))

            avg_metrics = {k: np.mean([m[k] for m in val_metrics_list]) for k in
val_metrics_list[0].keys()}
            for m_name, m_val in avg_metrics.items():
                trial.set_user_attr(m_name, m_val)
            return avg_metrics['SS2']


        def optimize_and_train_fold(r_idx, f_idx, test_idx, X, y, groups, model_type,
n_trials, temp_dir):
            temp_file = os.path.join(temp_dir, f"run_{r_idx}_fold_{f_idx}.pkl")
            if os.path.exists(temp_file):
                with open(temp_file, "rb") as f:
                    return pickle.load(f)

            mask_train = np.ones(len(y), dtype=bool)
            mask_train[test_idx] = False
            X_train, y_train, g_train = X[mask_train], y[mask_train], groups[mask_train]
            X_test, y_test = X[test_idx], y[test_idx]

            study = optuna.create_study(direction="maximize",
sampler=optuna.samplers.TPESampler(seed=123 + r_idx * 100 + f_idx))
            study.optimize(partial(objective, X_train=X_train, y_train=y_train,
g_train=g_train, model_type=model_type), n_trials=n_trials)

            best_metrics_inner = {k: study.best_trial.user_attrs[k] for k in ["Acc",
"Sens1", "Sens2", "SS2"]}
            bp = study.best_params.copy()
            model = build_final_model(bp, model_type)

            model.fit(X_train, y_train)
            res = {
                'run': r_idx,
                'fold': f_idx,
                'metrics_train': best_metrics_inner,
                'metrics_train_final': calculate_metrics(y_train,
model.predict(X_train)),
                'metrics_test': calculate_metrics(y_test, model.predict(X_test)),
                'params': bp
            }
            if "RF" in model_type:
                res['feature_importances'] = model.feature_importances_

            with open(temp_file, "wb") as f:
                pickle.dump(res, f)
            return res

        # ======================================================================
        # СИМУЛЯЦІЯ
```

```
    # ======================================================================


    def run_simulation(data_obj, model_type, n_trials):
        X, y = data_obj['data'].T.values, np.array(data_obj['labs']).astype(int)
        groups = np.array(data_obj['sample_id'])

        final_path = os.path.join(SAVE_DIR, f"RESULT_{model_type}.pkl")
        if os.path.exists(final_path):
            with open(final_path, "rb") as f:
                return pickle.load(f)

        temp_dir = os.path.join(SAVE_DIR, f"temp_{model_type}")
        os.makedirs(temp_dir, exist_ok=True)

        outer_folds = generate_group_cv_runs(X, y, groups, ntimes=5, nfold=10,
seed=54321)
        tasks = [(r_idx, f_idx, tidx) for r_idx, run in enumerate(outer_folds) for
f_idx, tidx in enumerate(run)]

        n_cores = max(1, os.cpu_count() - 2)
        results = Parallel(n_jobs=n_cores, backend='loky', verbose=10)(
            delayed(optimize_and_train_fold)(*t, X, y, groups, model_type, n_trials,
temp_dir) for t in tasks
        )

        with open(final_path, "wb") as f:
            pickle.dump(results, f)
        return results

    # ======================================================================
    # MAIN
    # ======================================================================


    def main():
        if not os.path.exists(DATA_PATH):
            return
        with open(DATA_PATH, "rb") as f:
            PERMAD = pickle.load(f)

        # Фільтрація за Eta-squared
        if os.path.exists(FINGERPRINT_PATH):
            m_df = pd.read_csv(FINGERPRINT_PATH)
            safe = m_df[m_df['Eta2'] <= 0.8]['Protein'].tolist()
            feats = list(PERMAD['features'])
            idx = [feats.index(p) for p in safe if p in feats]
            PERMAD['data'] = PERMAD['data'].iloc[idx, :]
            PERMAD['features'] = np.array([feats[i] for i in idx])

        # Основні моделі
        for m_key, tr in [("KNN", 4), ("SVM", 5), ("RF", 60)]:
            run_simulation(PERMAD, m_key, n_trials=tr)

        # Аналіз важливості та Top-10
        rf_res_path = os.path.join(SAVE_DIR, "RESULT_RF.pkl")
        if os.path.exists(rf_res_path):
            with open(rf_res_path, "rb") as f:
                res_rf = pickle.load(f)
```

```
            all_imp = [r['feature_importances'] for r in res_rf if
'feature_importances' in r]
            if len(all_imp) == 0:
                return

            mean_ranks = np.mean([rankdata(imp) for imp in all_imp], axis=0)
            ranking_df = pd.DataFrame({
                'Biomarker': PERMAD['features'],
                'Mean_Rank': mean_ranks,
                'Mean_Gini': np.mean(all_imp, axis=0)
            }).sort_values('Mean_Rank', ascending=False)

            ranking_df.to_csv(os.path.join(SAVE_DIR, "PROTEIN_RANKING.csv"),
index=False)

            # Експеримент з Top-10
            top10_idx = ranking_df.index[:10].tolist()
            PERMAD_10 = PERMAD.copy()
            PERMAD_10['data'] = PERMAD['data'].iloc[top10_idx, :]
            PERMAD_10['features'] = np.array(PERMAD['features'])[top10_idx]

            run_simulation(PERMAD_10, "RF_top10", n_trials=60)


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

Лістинг А.6 – Файл process_tda.py

```
      i import os
      import json
      import pickle
      import numpy as np
      import networkx as nx
      import pandas as pd
      import torch
      import torch.nn.functional as F
      import torch.multiprocessing as mp
      import optuna
      import community.community_louvain as community_louvain
      import matplotlib.pyplot as plt
      import matplotlib.patches as mpatches
      from collections import Counter
      from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
      from torch_geometric.data import Data
      from torch_geometric.nn import GATConv
      from torch_geometric.utils import coalesce, homophily
      from sklearn.model_selection import StratifiedGroupKFold
      from sklearn.metrics import adjusted_rand_score, confusion_matrix, accuracy_score,
f1_score
      from joblib import Parallel, delayed
      import traceback


      CONFIG = {
          "cv_outer_splits": 5,
          "cv_outer_repeats": 5,
          "cv_inner_splits": 5,
          "optuna_trials": 100,
          "variance_penalty": 1.0,
          "results_dir": "results_tda_final",
          "n_jobs": 4,
          "random_state": 42,
          "window_size": 5,
          "log_transform": True,
          "feature_selection": {
              "use_fingerprint": True,
              "fingerprint_thresh": 0.95,
              "use_anova": True,
              "top_k_features": 15,
              "fix_global_features": True,
              "use_feature_weighting": True
          },
          "topology": {
              "lens_type": "velocity",
              "dist_metric": "dtw"
          },
          "max_epochs": 100,
          "early_stopping_patience": 15,
          "device": torch.device('cuda' if torch.cuda.is_available() else 'cpu'),
          "search_space": {
              # Діапазони параметрів для контрольного прогону графової моделі.
              "h": [4, 8],
              "lr": (0.0050, 0.0100),
              "wd": (0.0090, 0.0099),
              "dr": (0.1750, 0.2350),
```

```
            "l1_lambda": (0.00018, 0.00025),
            "k_neighbors": [6],
            "rbf_min_weight": (0.6400, 0.7600),
            "heads": [1],
            "w_corr": (0.7200, 0.7850),
            "class_penalty": (4.60, 5.10),
            "kde_bandwidth": (4.00, 5.30)
        },
        "vis": {
            "figsize": (22, 13),
            "node_size": 35,
            "edge_alpha": 0.15
        }
    }


    # ======================================================================
    # СЛУЖБОВІ ФУНКЦІЇ АНАЛІЗУ ТА ВІЗУАЛІЗАЦІЇ
    # ======================================================================

    def calculate_graph_metrics(d_graph):
        n_n, n_e = d_graph.x.size(0), d_graph.edge_index.size(1) // 2
        G = nx.Graph()
        G.add_nodes_from(range(n_n))
        if n_e > 0:
            G.add_edges_from(d_graph.edge_index.cpu().numpy().T)
        degrees = [d for n, d in G.degree()]
        y_np = d_graph.y.cpu().numpy()
        try:
            part = community_louvain.best_partition(G)
            mod = community_louvain.modularity(part, G)
            ari = adjusted_rand_score(y_np, [part.get(n, 0) for n in range(n_n)])
        except: mod, ari = 0.0, 0.0
        return {
            "nodes": n_n, "edges": n_e,
            "deg_mean": np.mean(degrees) if degrees else 0.0,
            "deg_max": max(degrees) if degrees else 0,
            "clustering": nx.average_clustering(G) if n_n > 0 else 0.0,
            "isolated": sum(1 for d in degrees if d == 0),
            "homophily":        float(homophily(d_graph.edge_index,        d_graph.y,
method='edge')) if n_e > 0 else 0.0,
            "modularity": mod, "ari": ari
        }

    def visualize_fold_graph(G, f_data, mets_te, mets_tr, val_mets, topo, params, te_m,
name):
        fig = plt.figure(figsize=CONFIG['vis']['figsize'])
        ax_g = fig.add_axes([0.05, 0.05, 0.60, 0.9])

        pos = nx.spring_layout(G, k=0.25, iterations=150, seed=CONFIG['random_state'])
        y_np = f_data.y.cpu().numpy()
        clrs, szs, lws = [], [], []
        for i in range(len(y_np)):
            is_test = te_m[i].item()
            cls = y_np[i]
            clrs.append(('#FF9800' if cls == 1 else '#03A9F4') if is_test else
('#FF5252' if cls == 1 else '#4CAF50'))
            szs.append(CONFIG['vis']['node_size'])
            lws.append(1.5 if is_test else 0.5)
```

```
            nx.draw(G, pos, ax=ax_g, node_color=clrs, node_size=szs, edgecolors='black',
                    width=CONFIG['vis']['edge_alpha'],              edge_color='#888888',
linewidths=lws, alpha=0.9)

            leg_items = [
                mpatches.Patch(color='#FF5252', label='Train (Sick)'),
                mpatches.Patch(color='#4CAF50', label='Train (Healthy)'),
                mpatches.Patch(color='#FF9800', label='Test (Sick)'),
                mpatches.Patch(color='#03A9F4', label='Test (Healthy)')
            ]
            ax_g.legend(handles=leg_items, loc='upper left')

            info = f"FOLD: {name}\n" + "="*35 + "\n"
            info += "[TEST PERFORMANCE]\n"
            for k, v in mets_te.items(): info += f"  {k:12}: {v:.4f}\n"
            info += "\n[TRAIN PERFORMANCE]\n"
            for k, v in mets_tr.items(): info += f"  {k:12}: {v:.4f}\n"
            info += "\n[VAL PERFORMANCE (AVG)]\n"
            for k, v in val_mets.items(): info += f"  {k:12}: {v:.4f}\n"
            info += "\n[GRAPH STRUCTURE]\n"
            for k, v in topo.items():
                v_str = f"{v:.4f}" if isinstance(v, float) else str(v)
                info += f"  {k:12}: {v_str}\n"
            info += "\n[HYPERPARAMETERS]\n"
            for k, v in params.items():
                v_str = f"{v:.5f}" if isinstance(v, float) else str(v)
                info += f"  {k:15}: {v_str}\n"

            fig.text(0.67,      0.95,      info,      fontsize=9,      family='monospace',
verticalalignment='top',
                     bbox=dict(boxstyle='round',      facecolor='white',      alpha=0.95,
edgecolor='#CCCCCC'))

            plt.savefig(os.path.join(CONFIG['results_dir'],  f"{name}_viz.png"),  dpi=200,
bbox_inches='tight')
            plt.close()

        # ======================================================================
        # ЯДРО ОБРОБКИ
        # ======================================================================

        def compute_focal_loss(logits, targets, alpha=0.5, gamma=2.0):
            bce = F.binary_cross_entropy_with_logits(logits, targets, reduction='none')
            pt = torch.exp(-bce)
            alpha_t = torch.where(targets >= 0.5, alpha, 1.0 - alpha)
            return (alpha_t * (1 - pt)**gamma * bce).mean()

        def calculate_metrics_gpu(y_true, y_pred):
            tp = ((y_true == 1) & (y_pred == 1)).sum().float()
            tn = ((y_true == 0) & (y_pred == 0)).sum().float()
            fp = ((y_true == 0) & (y_pred == 1)).sum().float()
            fn = ((y_true == 1) & (y_pred == 0)).sum().float()

            sens = tp / (tp + fn + 1e-8)
            spec = tn / (tn + fp + 1e-8)
            acc = (tp + tn) / (y_true.size(0) + 1e-8)
            f1 = 2 * tp / (2 * tp + fp + fn + 1e-8)
            ss2 = (sens + spec) / 2
```

```
            return {
                "Acc": acc.item(),
                "f1": f1.item(),
                "Sens2": sens.item(),
                "Sens1": spec.item(),
                "SS2": ss2.item()
            }

        def get_permutation_importance(model, data, te_m, base_ss2, protein_names):
            model.eval()
            f_dim, w_size = len(protein_names), CONFIG['window_size']
            importances = {}
            with torch.no_grad():
                for i in range(f_dim):
                    target_indices = [i, f_dim + i] + [2 * f_dim + i + k * f_dim for k in
range(w_size - 1)]
                    x_perm = data.x.clone()
                    x_perm[:,  target_indices]  =  x_perm[torch.randperm(data.x.size(0)),
:][:, target_indices]
                    m      =      calculate_metrics_gpu(data.y[te_m],      (model(x_perm,
data.edge_index, data.edge_attr)[te_m] > 0.0).float())
                    importances[protein_names[i]] = float(max(0, base_ss2 - m['SS2']))
            return dict(sorted(importances.items(), key=lambda x: x[1], reverse=True))

        @torch.no_grad()
        def get_stable_global_features_and_weights(X_raw, y_raw, g_raw):
            device = CONFIG['device']
            X_t = torch.as_tensor(X_raw, dtype=torch.float32, device=device)
            if CONFIG['log_transform']:
                X_t = torch.log1p(X_t)

            y_t = torch.as_tensor(y_raw, device=device)
            g_t = torch.as_tensor(g_raw, device=device)

            feat_conf = CONFIG['feature_selection']
            n_features = X_t.size(1)

            # Якщо фільтрація вимкнена, використовуються всі білки.
            if (not feat_conf['use_fingerprint']) and (not feat_conf['use_anova']):
                final_mask = torch.ones(n_features, dtype=torch.bool, device=device)
                weights = torch.ones(n_features, dtype=torch.float32, device=device)
                return final_mask, weights

            kf = StratifiedGroupKFold(n_splits=5, shuffle=True, random_state=42)
            feature_wins = Counter()
            feature_f_scores = torch.zeros(n_features, device=device)
            fold_count = 0
            valid_anova_folds = 0

            for tr_idx, _ in kf.split(np.zeros(len(y_raw)), y_raw, g_raw):
                fold_count += 1

                X_tr = X_t[tr_idx]
                y_tr = y_t[tr_idx]
                g_tr = g_t[tr_idx]

                mask = torch.ones(n_features, dtype=torch.bool, device=device)
```

```
                if feat_conf['use_fingerprint']:
                    _, g_idx = torch.unique(g_tr, return_inverse=True)
                    num_g = g_idx.max() + 1

                    sums = torch.zeros((num_g, n_features), device=device).scatter_add_(
                        0,
                        g_idx.unsqueeze(1).expand_as(X_tr),
                        X_tr
                    )

                    cnts = torch.zeros(num_g, device=device).scatter_add_(
                        0,
                        g_idx,
                        torch.ones_like(g_idx, dtype=torch.float32)
                    ).unsqueeze(1)

                    group_means = sums / cnts.clamp(min=1)
                    fi = torch.var(group_means, dim=0) / (torch.var(X_tr, dim=0) + 1e-8)
                    mask &= (fi <= feat_conf['fingerprint_thresh'])

                if feat_conf['use_anova']:
                    c0 = X_tr[y_tr == 0]
                    c1 = X_tr[y_tr == 1]

                    if c0.size(0) < 2 or c1.size(0) < 2:
                        continue

                    m0 = c0.mean(0)
                    m1 = c1.mean(0)
                    mall = X_tr.mean(0)

                    ssb = c0.size(0) * (m0 - mall) ** 2 + c1.size(0) * (m1 - mall) ** 2
                    ssw = ((c0 - m0) ** 2).sum(0) + ((c1 - m1) ** 2).sum(0)

                    f_scores = ssb / (ssw / max(X_tr.size(0) - 2, 1) + 1e-8)
                    f_scores[~mask] = -1.0

                    actual_k = min(feat_conf['top_k_features'], int(mask.sum().item()))

                    if actual_k > 0:
                        _, top = torch.topk(f_scores, actual_k)
                        for idx in top.cpu().numpy():
                            feature_wins[int(idx)] += 1

                        feature_f_scores += torch.clamp(f_scores, min=0.0)
                        valid_anova_folds += 1

                else:
                    # Якщо ANOVA вимкнена, залишаються ознаки, що пройшли попередню
фільтрацію.
                    valid_indices = torch.where(mask)[0].cpu().numpy()
                    for idx in valid_indices:
                        feature_wins[int(idx)] += 1

            final_mask = torch.zeros(n_features, dtype=torch.bool, device=device)

            if feat_conf['use_anova']:
                top_indices = [idx for idx, _ in
feature_wins.most_common(feat_conf['top_k_features'])]
```

```
            else:
                threshold = (fold_count // 2) + 1
                top_indices = [idx for idx, wins in feature_wins.items() if wins >=
threshold]

            # Якщо фільтрація не залишила ознак, повертаємо повний набір.
            if len(top_indices) == 0:
                final_mask.fill_(True)
            else:
                final_mask[top_indices] = True

            weights = torch.ones(n_features, dtype=torch.float32, device=device)

            if feat_conf['use_feature_weighting'] and feat_conf['use_anova'] and
valid_anova_folds > 0:
                avg_f_scores = feature_f_scores / float(valid_anova_folds)
                selected_f_scores = avg_f_scores[final_mask]

                if selected_f_scores.numel() > 0 and selected_f_scores.max() > 0:
                    norm_weights = selected_f_scores / selected_f_scores.max()
                    weights[final_mask] = torch.clamp(norm_weights, min=0.1)

            return final_mask, weights

        @torch.no_grad()
        def precompute_geometry(X, y, g, t, feat_mask, weights, device, tr_p_ids):
            X_s = X[:, feat_mask]
            w_s = weights[feat_mask]

            perm = torch.argsort(t, stable=True)[torch.argsort(g[torch.argsort(t,
stable=True)], stable=True)]
            X_s, y_s, g_s, t_s = X_s[perm], y[perm], g[perm], t[perm]

            tr_mask = torch.isin(g_s, tr_p_ids)
            X_norm = (X_s - X_s[tr_mask].mean(0)) / (X_s[tr_mask].std(0) + 1e-8)
            X_weighted = X_norm * w_s.unsqueeze(0)

            N, W, F_dim = X_s.size(0), CONFIG['window_size'], X_s.size(1)
            X_takens = torch.empty((N, F_dim * W), device=device)
            X_gnn = torch.empty((N, F_dim * 2 + F_dim * (W - 1)), device=device)
            t_diff = torch.empty((N, W - 1), device=device)
            vel_lens = torch.empty(N, device=device)

            for i in range(N):
                w = torch.where((g_s == g_s[i]) & (t_s <= t_s[i]))[0]
                if w.size(0) > W: w = w[-W:]
                if w.size(0) < W: w = torch.cat([w[:1].expand(W - w.size(0)), w])

                X_takens[i] = X_weighted[w].flatten()
                X_gnn[i] = torch.cat([X_norm[w[0]], X_norm[w[-1]], (X_norm[w[1:]] -
X_norm[w[:-1]]).flatten()])
                t_diff[i] = torch.clamp((t_s[w[1:]] - t_s[w[:-1]]) / 365.0, min=1e-4)
                vel_lens[i] = (X_norm[w[1:]] - X_norm[w[:-1]]).abs().mean()

            if CONFIG['topology']['lens_type'] == 'velocity':
                lens = (vel_lens - vel_lens.mean()) / (vel_lens.std() + 1e-8)
            else:
                lda = LinearDiscriminantAnalysis(n_components=1)
```

```
            lda.fit(X_takens[torch.isin(g_s,                tr_p_ids)].cpu().numpy(),
y_s[torch.isin(g_s, tr_p_ids)].cpu().numpy())
            lens                                                                    =
torch.from_numpy(lda.transform(X_takens.cpu().numpy())).to(device).squeeze().to(torch.f
loat32)

        d_lens     =     torch.cdist(lens.unsqueeze(1),     lens.unsqueeze(1))     /
(torch.cdist(lens.unsqueeze(1), lens.unsqueeze(1)).max() + 1e-8)
        d_cos  =  torch.clamp(1.0  -  torch.mm(F.normalize(X_takens,  p=2,  dim=1),
F.normalize(X_takens, p=2, dim=1).t()), min=0.0)

        # X_takens формується через flatten() часового вікна у порядку [час, ознаки],
        # тому для DTW повертаємо його у форму [N, W, F_dim].
        X_seq = X_takens.view(N, W, F_dim).contiguous()
        X_flat = F.normalize(X_seq.reshape(-1, F_dim), p=2, dim=1)
        cost = torch.clamp(1.0 - torch.mm(X_flat, X_flat.t()), min=0.0).view(N, W, N,
W).permute(0, 2, 1, 3)

        dtw = torch.full((N, N, W + 1, W + 1), float('inf'), device=device)
        dtw[:, :, 0, 0] = 0.0
        for r in range(1, W + 1):
            for c in range(1, W + 1):
                dtw[:, :, r, c] = cost[:, :, r-1, c-1] + torch.min(torch.min(dtw[:, :,
r-1, c], dtw[:, :, r, c-1]), dtw[:, :, r-1, c-1])
        d_dtw = (dtw[:, :, W, W] + dtw[:, :, W, W].t()) / 2.0
        d_dtw = d_dtw / (d_dtw.max() + 1e-8)

        return torch.cat([X_gnn, t_diff], 1), d_lens, d_cos, d_dtw, g_s.unsqueeze(0)
== g_s.unsqueeze(1), y_s, g_s, t_s, perm

    class TransductiveGNN(torch.nn.Module):
        def __init__(self, in_c, h_c, dr, heads=1):
            super().__init__()
            self.conv1  =  GATConv(in_c,  h_c,  heads=heads,  concat=False,  dropout=dr,
edge_dim=1)
            self.lin, self.skip = torch.nn.Linear(h_c, 1), torch.nn.Linear(in_c, h_c)
        def forward(self, x, edge_index, edge_attr):
            if edge_attr is not None and edge_attr.dim() == 1:
                edge_attr = edge_attr.view(-1, 1)
            x = F.elu(self.conv1(x, edge_index, edge_attr=edge_attr) + self.skip(x))
            return self.lin(F.dropout(x, p=0.3, training=self.training)).squeeze(-1)

    def train_eval(data, params, tr_m, val_m=None, is_test=False):
        device = data.x.device
        model    =    TransductiveGNN(data.x.size(1),    params['h'],    params['dr'],
heads=params['heads']).to(device)
        opt       =       torch.optim.Adam(model.parameters(),       lr=params['lr'],
weight_decay=params['wd'])

        n_neg,   n_pos   =   (data.y[tr_m]   ==   0).sum().float(),   (data.y[tr_m]   ==
1).sum().float()
        alpha = n_neg / (n_neg + n_pos + 1e-8)
        targets = data.y[tr_m].float() * 0.9 + 0.05

        best_val_loss, patience_counter = float('inf'), 0
        eval_mask = val_m if val_m is not None else tr_m
        eval_targets = data.y[eval_mask].float() * 0.9 + 0.05

        for _ in range(CONFIG['max_epochs']):
```

```
                model.train()
                opt.zero_grad()
                logits = model(data.x, data.edge_index, data.edge_attr)
                loss = compute_focal_loss(logits[tr_m], targets, alpha=alpha)

                if params.get('l1_lambda', 0) > 0:
                    loss += params['l1_lambda'] * sum(p.abs().sum() for p in
model.parameters())

                loss.backward()
                opt.step()

                if not is_test:
                    model.eval()
                    with torch.no_grad():
                        val_logits = model(data.x, data.edge_index, data.edge_attr)
                        val_loss       =       compute_focal_loss(val_logits[eval_mask],
eval_targets, alpha=alpha).item()

                    if val_loss < best_val_loss - 1e-4:
                        best_val_loss = val_loss
                        patience_counter = 0
                    else:
                        patience_counter += 1
                    if patience_counter >= CONFIG['early_stopping_patience']:
                            break

            model.eval()
            with torch.no_grad():
                out = model(data.x, data.edge_index, data.edge_attr)
                preds = (out > 0.0).float()
                return calculate_metrics_gpu(data.y[eval_mask], preds[eval_mask]) if not
is_test else (preds, model)

        def run_fold(info, X_np, y_np, g_np, t_np, mask_t, weights_t, p_names_top,
all_feature_names):
            name, tr_p_np, te_p_np = info

            meta_path = os.path.join(CONFIG['results_dir'], f"{name}_meta.json")
            if os.path.exists(meta_path):
                try:
                    with open(meta_path, "r") as f:
                        saved_data = json.load(f)
                    return    {"fold":    name,    "status":    "success",    "params":
saved_data.get("params",    {}),    "features":    saved_data.get("features",    []),
"metrics_test":       saved_data.get("test_metrics",       {}),       "metrics_train":
saved_data.get("train_metrics", {}), "val_metrics": saved_data.get("val_metrics", {}),
"topology": saved_data.get("topology", {}), "importance": saved_data.get("importance",
{})}
                except Exception:
                    pass

            torch.set_num_threads(1)
            device = CONFIG['device']
            try:
                X_t,   y_t,   t_t,   g_t   =   torch.as_tensor(X_np,   device=device),
torch.as_tensor(y_np,     device=device),     torch.as_tensor(t_np,     device=device),
torch.as_tensor(g_np, device=device)
                if CONFIG['log_transform']: X_t = torch.log1p(X_t)
```

```
            if not CONFIG['feature_selection'].get('fix_global_features', True):
                tr_rows = np.isin(g_np, tr_p_np)
                mask_t,                          weights_t                          =
get_stable_global_features_and_weights(X_np[tr_rows], y_np[tr_rows], g_np[tr_rows])
                p_names_top     =     [all_feature_names[idx]     for     idx     in
torch.where(mask_t)[0].cpu().numpy()]

            tr_p_t = torch.as_tensor(tr_p_np, device=device)
            x_node,  d_lens,  d_cos,  d_dtw,  same_pat,  y_s,  g_s,  t_s,  perm  =
precompute_geometry(X_t, y_t, g_t, t_t, mask_t, weights_t, device, tr_p_t)
            tr_m,     te_m     =     torch.isin(g_s,     tr_p_t),     torch.isin(g_s,
torch.as_tensor(te_p_np, device=device))

            inner = []
            kf_inner = StratifiedGroupKFold(CONFIG['cv_inner_splits'], shuffle=True,
random_state=CONFIG['random_state'])
            for     it,     iv     in     kf_inner.split(np.zeros(tr_m.sum().item()),
y_s[tr_m].cpu().numpy(), g_s[tr_m].cpu().numpy()):
                it_t, iv_t = torch.zeros_like(tr_m), torch.zeros_like(tr_m)
                it_t[torch.where(tr_m)[0][it]],    iv_t[torch.where(tr_m)[0][iv]]    =
True, True
                inner.append((it_t, iv_t))

            def build_topo(p, current_train_mask):
                base = (d_cos + d_dtw) / 2.0 if CONFIG['topology']['dist_metric'] ==
'dtw' else d_cos
                dist = p['w_corr'] * base + (1.0 - p['w_corr']) * d_lens
                knn_d              =              dist.clone().masked_fill_(same_pat,
float('inf')).fill_diagonal_(float('inf'))

                val, idx = torch.topk(knn_d, min(int(p['k_neighbors']), x_node.size(0)
- 1), dim=1, largest=False)
                u = torch.arange(x_node.size(0), device=device).view(-1, 1).expand(-
1, idx.size(1)).flatten()
                v = idx.flatten()

                sig = 1.0 / (torch.exp(-(dist[u, v].view(x_node.size(0), -1)**2) / (2
* p['kde_bandwidth']**2)).mean(1) + 1e-4)
                w = torch.exp(-(dist[u, v]**2) / (sig[u] * sig[v]))
                m = (w.view(x_node.size(0), -1) >= p['rbf_min_weight']).flatten()
                e_raw = torch.stack([u[m], v[m]], 0)
                e_idx,  e_at  =  coalesce(torch.cat([e_raw,  e_raw.flip(0)],  dim=1),
torch.cat([w[m], w[m]]), num_nodes=x_node.size(0), reduce='max')

                if current_train_mask is not None and p.get('class_penalty', 0) > 1.0:
                    both_train        =        current_train_mask[e_idx[0]]        &
current_train_mask[e_idx[1]]
                    diff_class = y_s[e_idx[0]] != y_s[e_idx[1]]
                    penalty_mask = both_train & diff_class
                    e_at = torch.where(penalty_mask, e_at / p['class_penalty'], e_at)

                return Data(x=x_node, edge_index=e_idx, edge_attr=e_at, y=y_s)

            def objective(t):
                p  =  {k:  t.suggest_categorical(k,  v)  if  isinstance(v,  list)  else
t.suggest_float(k, *v) for k, v in CONFIG['search_space'].items()}
                scores = []
                base_data = build_topo(p, None)
```

```
                    for it_m, iv_m in inner:
                        if p.get('class_penalty', 0) > 1.0:
                            e_at = base_data.edge_attr.clone()
                            both_train       =       it_m[base_data.edge_index[0]]       &
it_m[base_data.edge_index[1]]
                            diff_class       =       y_s[base_data.edge_index[0]]       !=
y_s[base_data.edge_index[1]]
                            e_at   =   torch.where(both_train   &   diff_class,   e_at   /
p['class_penalty'], e_at)
                            data_cv                  =                  Data(x=base_data.x,
edge_index=base_data.edge_index, edge_attr=e_at, y=base_data.y)
                        else:
                            data_cv = base_data
                        scores.append(train_eval(data_cv, p, it_m, iv_m)['SS2'])
                    avg_ss2 = float(np.mean(scores))
                    t.set_user_attr("val_SS2", avg_ss2)
                    return avg_ss2 - CONFIG['variance_penalty'] * np.std(scores)

                optuna.logging.set_verbosity(optuna.logging.INFO)
                study = optuna.create_study(direction="maximize")
                study.optimize(objective, n_trials=CONFIG['optuna_trials'])

                bp = study.best_params
                f_data = build_topo(bp, tr_m)
                preds, m_f = train_eval(f_data, bp, tr_m, val_m=te_m, is_test=True)
                mets_te,  mets_tr  =  calculate_metrics_gpu(f_data.y[te_m],  preds[te_m]),
calculate_metrics_gpu(f_data.y[tr_m], preds[tr_m])
                val_mets    =    {k.replace('val_',    ''):    v    for    k,    v    in
study.best_trial.user_attrs.items() if k.startswith('val_')}

                topo = calculate_graph_metrics(f_data)
                imp  =  get_permutation_importance(m_f,  f_data,  te_m,  mets_te['SS2'],
p_names_top)

                with  open(os.path.join(CONFIG['results_dir'],  f"{name}_meta.json"),  "w")
as f:
                    json.dump({"fold":  name,  "params":  bp,  "features":  p_names_top,
"test_metrics": mets_te, "train_metrics": mets_tr, "val_metrics": val_mets, "topology":
topo, "importance": imp}, f, indent=2)

                G = nx.Graph()
                G.add_nodes_from(range(f_data.x.size(0)))
                if f_data.edge_index.size(1) > 0:
                    G.add_edges_from(f_data.edge_index.cpu().numpy().T)

                nx.write_gml(G, os.path.join(CONFIG['results_dir'], f"{name}_graph.gml"))

                y_out,   g_out,   t_out   =   f_data.y.cpu().numpy(),   g_s.cpu().numpy(),
t_s.cpu().numpy()
                v_df = pd.concat([pd.DataFrame([{"Node": i, "Patient": g_out[i], "Class":
y_out[i], "Time": t_out[i], "IsTest": te_m[i].item()} for i in range(len(y_out))]),
                                  pd.DataFrame(X_np[perm.cpu().numpy()][:,
mask_t.cpu().numpy()])], axis=1)
                v_df.to_pickle(os.path.join(CONFIG['results_dir'],
f"{name}_vertices.pkl"))

                with    open(os.path.join(CONFIG['results_dir'],    f"{name}_result.pkl"),
"wb") as f:
```

```
                    pickle.dump({"fold": name, "params": bp, "features": p_names_top,
"test_metrics": mets_te, "train_metrics": mets_tr, "val_metrics": val_mets, "topology":
topo, "importance": imp}, f)

                visualize_fold_graph(G, f_data, mets_te, mets_tr, val_mets, topo, bp,
te_m, name)

                return {"fold": name, "status": "success", "params": bp, "features":
p_names_top, "metrics_test": mets_te, "metrics_train": mets_tr, "val_metrics": val_mets,
"topology": topo, "importance": imp}
            except Exception:
                with open(os.path.join(CONFIG['results_dir'], "error_log.txt"), "a") as f:
                    f.write(f"FAIL {name}: {traceback.format_exc()}\n")
                return None

        def run_pipeline():
            mp.set_start_method('spawn', force=True)
            os.makedirs(CONFIG['results_dir'], exist_ok=True)

            d = pickle.load(open("data/PERMAD_graph_raw.pkl", "rb"))
            X, y, g, t = d['data'].values.T, d['labs'].astype(int), d['sample_id'],
d['time_ct']

            all_feature_names = d['data'].index.tolist()
            if CONFIG['feature_selection'].get('fix_global_features', True):
                mask_t, weights_t = get_stable_global_features_and_weights(X, y, g)
                p_names = [all_feature_names[idx] for idx in
torch.where(mask_t)[0].cpu().numpy()]
            else:
                mask_t, weights_t, p_names = None, None, []

            runs = []

            for repeat in range(CONFIG['cv_outer_repeats']):
                # Кожний повтор використовує окремий random_state.
                current_random_state = CONFIG['random_state'] + repeat
                kf_outer = StratifiedGroupKFold(n_splits=CONFIG['cv_outer_splits'],
shuffle=True, random_state=current_random_state)

                for f, (tr, te) in enumerate(kf_outer.split(X, y, g)):
                    # Назва фолду містить номер повтору та номер фолду.
                    run_name = f"R{repeat+1}_F{f+1}"
                    runs.append((run_name, np.unique(g[tr]), np.unique(g[te])))

            results = Parallel(n_jobs=CONFIG['n_jobs'],
backend="loky")(delayed(run_fold)(run, X, y, g, t, mask_t, weights_t, p_names,
all_feature_names) for run in runs)
            valid_results = [res for res in results if res is not None]
            pickle.dump(valid_results, open(os.path.join(CONFIG['results_dir'],
"RESULT_Velocity_DTW.pkl"), "wb"))
            with open(os.path.join(CONFIG['results_dir'], "RESULT_Velocity_DTW.json"),
"w") as f:
                json.dump(valid_results, f, indent=2)

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

ДОДАТОК Б − ПОВНІ РЕЗУЛЬТАТИ КЛАСИЧНИХ МОДЕЛЕЙ

У додатку Б наведено додаткові таблиці результатів класичних моделей машинного навчання. Ці таблиці доповнюють основні результати, подані у Розділі 3.

Таблиця Б.1 – Навчальні результати негрупового відтворення класичних моделей

Метрика	**RF Full**	**RF Top-10**	**SVM**	**k-NN**
Accuracy	79.3 [78.9; 79.8]	75.3 [74.6; 76.4]	67.9 [67.1; 68.9]	60.9 [59.9; 61.9]
Sensitivity	85.3 [84.0; 86.0]	81.8 [78.6; 85.8]	51.9 [50.4; 53.5]	45.3 [39.9; 48.9]
Specificity	75.1 [74.5; 76.1]	72.4 [67.1; 74.0]	79.1 [78.4; 79.5]	71.6 [67.4; 76.5]
SS2	80.3 [79.6; 80.5]	76.3 [76.0; 77.1]	65.2 [64.4; 66.2]	58.2 [57.7; 58.8]

Таблиця Б.2 – Тестові результати негрупового відтворення класичних моделей

Метрика	**RF Full**	**RF Top-10**	**SVM**	**k-NN**
Accuracy	79.7 [76.6; 82.8]	75.9 [72.4; 80.7]	67.2 [63.8; 69.5]	58.6 [55.2; 62.7]
Sensitivity	83.3 [79.2; 90.4]	82.6 [73.9; 91.3]	48.9 [41.7; 54.2]	41.3 [34.8; 50.0]
Specificity	77.1 [71.4; 82.9]	71.4 [66.4; 77.1]	80.0 [74.3; 82.9]	70.0 [65.7; 77.1]
SS2	80.5 [77.9; 83.5]	77.1 [73.5; 81.2]	64.6 [59.6; 67.0]	56.0 [52.6; 60.1]

Таблиця Б.3 – Top-10 біомаркерів у негруповому відтворенні

Позиція	**Біомаркер**	**Mean Rank**	**Mean Gini**
1	Antileukoproteinase (ALP)	100.46	0.023418
2	Insulin-like Growth Factor-Binding Protein 1 (IGFBP-1)	98.38	0.020311
3	Tissue type Plasminogen activator (tPA)	98.24	0.020254
4	Vascular endothelial growth factor receptor 3 (VEGFR-3)	97.20	0.019746
5	Hepsin	93.96	0.017333
6	Pepsinogen I (PGI)	93.58	0.017471
7	Immunoglobulin M (IgM)	93.10	0.017306
8	Sex Hormone-Binding Globulin (SHBG)	92.06	0.016619
9	Myeloperoxidase (MPO)	90.60	0.015947
10	Cancer Antigen 15-3 (CA-15-3)	90.04	0.015819

Таблиця Б.4 – Навчальні результати класичних моделей із груповою ізоляцією пацієнтів

Метрика	**RF Full**	**RF Top-10**	**SVM**	**k-NN**
Accuracy	55.7 [54.3; 57.2]	61.3 [59.6; 62.6]	55.5 [53.7; 57.4]	47.4 [46.3; 48.4]
Sensitivity	36.0 [31.0; 44.8]	52.5 [48.1; 57.1]	61.0 [59.0; 62.6]	50.4 [47.5; 54.7]
Specificity	75.8 [67.1; 79.2]	68.5 [62.6; 73.3]	51.1 [48.7; 54.6]	46.1 [44.0; 49.9]
SS2	55.7 [54.2; 56.6]	60.7 [59.3; 61.7]	56.4 [54.7; 57.7]	49.0 [47.5; 49.8]

Таблиця Б.5 – Тестові результати класичних моделей із груповою ізоляцією пацієнтів

Метрика	**RF Full**	**RF Top-10**	**SVM**	**k-NN**
Accuracy	57.8 [49.5; 61.6]	61.2 [53.0; 67.4]	55.1 [48.3; 64.1]	46.1 [40.8; 54.5]
Sensitivity	32.6 [19.1; 50.9]	59.9 [33.0; 68.7]	64.8 [48.2; 82.1]	51.2 [40.0; 62.8]
Specificity	74.3 [61.5; 83.3]	66.7 [53.5; 84.7]	49.9 [32.4; 71.4]	45.7 [33.8; 56.5]
SS2	53.9 [48.6; 57.8]	60.6 [53.2; 66.4]	57.3 [52.4; 63.6]	46.3 [41.4; 54.1]

Таблиця Б.6 – Top-10 біомаркерів у груповій схемі класичних моделей

Позиція	**Біомаркер**	**Mean Rank**	**Mean Gini**
1	Antileukoproteinase (ALP)	99.22	0.023740
2	Tissue type Plasminogen activator (tPA)	96.82	0.022874
3	Vascular endothelial growth factor receptor 3 (VEGFR-3)	95.34	0.019796
4	Immunoglobulin M (IgM)	94.42	0.019704
5	Sex Hormone-Binding Globulin (SHBG)	93.46	0.019830
6	Pepsinogen I (PGI)	93.06	0.018730
7	Cancer Antigen 15-3 (CA-15-3)	92.76	0.019596
8	Hepsin	90.18	0.016921
9	Insulin-like Growth Factor-Binding Protein 1 (IGFBP-1)	89.74	0.017838
10	Myeloperoxidase (MPO)	89.00	0.016419

ДОДАТОК В − ПОВНІ РЕЗУЛЬТАТИ ГРАФОВОЇ МОДЕЛІ

У додатку В наведено додаткові результати основної графової моделі Velocity-DTW та контрольного графового експерименту без відбору ознак.

Таблиця В.1 – Підсумкові метрики основної графової моделі Velocity-DTW

Фаза	**Accuracy**	**Sensitivity**	**Specificity**	**SS2**
Training	79.5 [78.7; 80.0]	80.8 [79.1; 81.7]	78.7 [77.5; 80.1]	79.7 [78.9; 80.3]
Test	63.6 [58.1; 67.5]	63.2 [52.7; 75.9]	64.6 [56.1; 75.3]	63.9 [61.4; 67.8]

Таблиця В.2 – Підсумковий рейтинг білків основної графової моделі

Позиція	**Protein**	**Mean Importance**	**Std Importance**	**Folds Present**
1	Insulin-like Growth Factor-Binding Protein 1 (IGFBP-1)	0.0365	0.0377	25
2	Epiregulin (EPR)	0.0359	0.0319	25
3	Vascular Endothelial Growth Factor (VEGF)	0.0290	0.0267	25
4	Galectin-3	0.0272	0.0252	25
5	EN-RAGE	0.0263	0.0285	25
6	Serum Amyloid P-Component (SAP)	0.0243	0.0279	25
7	6Ckine	0.0212	0.0279	25
8	Tetranectin	0.0160	0.0185	25
9	Immunoglobulin A (IgA)	0.0149	0.0201	25
10	Urokinase-type plasminogen activator receptor (uPAR)	0.0127	0.0163	25
11	Leucine-rich alpha-2-glycoprotein (LRG1)	0.0081	0.0137	25

12	Neuropilin-1	0.0073	0.0114	25
13	Antileukoproteinase (ALP)	0.0072	0.0091	25
14	Mast/stem cell growth factor receptor (SCFR)	0.0070	0.0100	25
15	Haptoglobin	0.0065	0.0101	25

Продовження таблиці В.2

Таблиця В.3 – Метрики контрольної графової моделі без відбору ознак

Фаза	**Accuracy**	**Sensitivity**	**Specificity**	**SS2**
Training	98.4 [97.9; 98.9]	99.4 [99.0; 100.0]	97.8 [97.4; 98.7]	98.6 [98.2; 98.9]
Test	56.8 [51.1; 61.3]	62.3 [54.2; 72.9]	55.1 [41.5; 67.0]	58.7 [56.0; 61.9]

Таблиця В.4 – Підсумковий рейтинг білків контрольної графової моделі без відбору ознак

Позиція	**Protein**	**Mean Importance**	**Std Importance**	**Folds Present**
1	Insulin-like Growth Factor-Binding Protein 1 (IGFBP-1)	0.0307	0.0288	25
2	Vascular Endothelial Growth Factor (VEGF)	0.0268	0.0330	25
3	Macrophage inflammatory protein 3 beta (MIP-3 beta)	0.0146	0.0156	25
4	Complement C3 (C3)	0.0140	0.0154	25
5	Interleukin-1 receptor antagonist (IL-1ra)	0.0130	0.0174	25
6	Macrophage-Stimulating Protein (MSP)	0.0130	0.0171	25

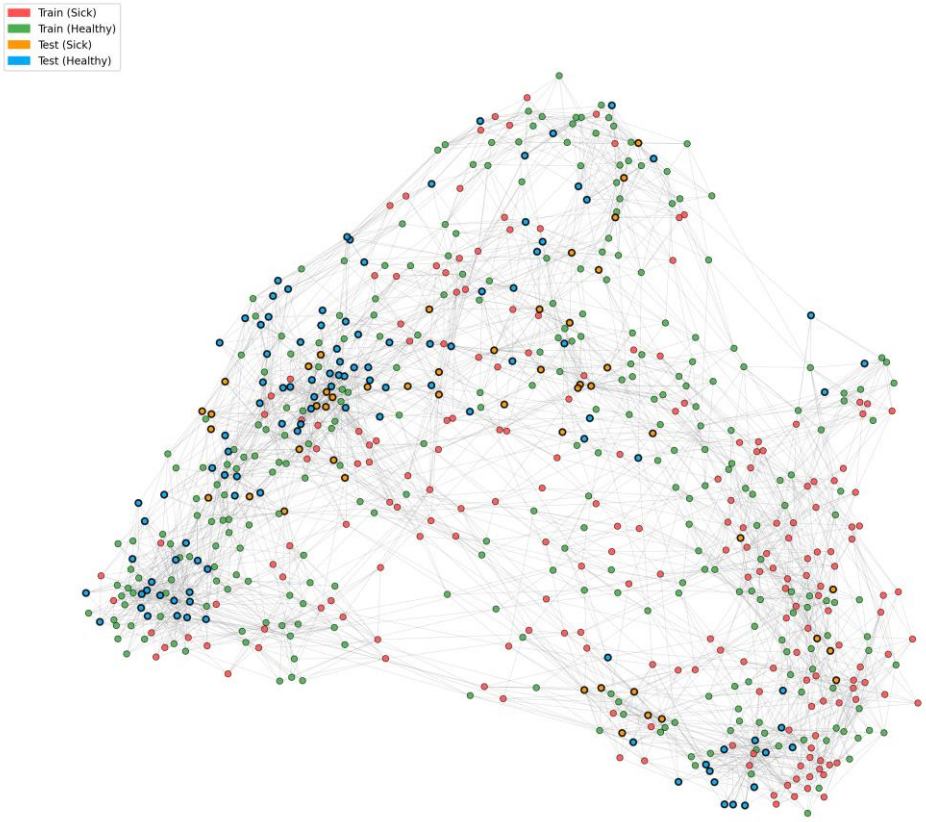
7	Epiregulin (EPR)	0.0126	0.0156	25
8	Adiponectin	0.0124	0.0181	25
9	Neuropilin-1	0.0119	0.0120	25
10	Galectin-3	0.0118	0.0200	25
11	B cell-activating factor (BAFF)	0.0115	0.0177	25
12	Cystatin-B	0.0110	0.0101	25
13	Pulmonary and Activation-Regulated Chemokine (PARC)	0.0107	0.0135	25
14	Fibulin-1C (Fib-1C)	0.0104	0.0114	25
15	Pepsinogen I (PGI)	0.0102	0.0132	25

Продовження таблиці В.4

ДОДАТОК Г – ВІЗУАЛІЗАЦІЇ ГРАФОВИХ МОДЕЛЕЙ

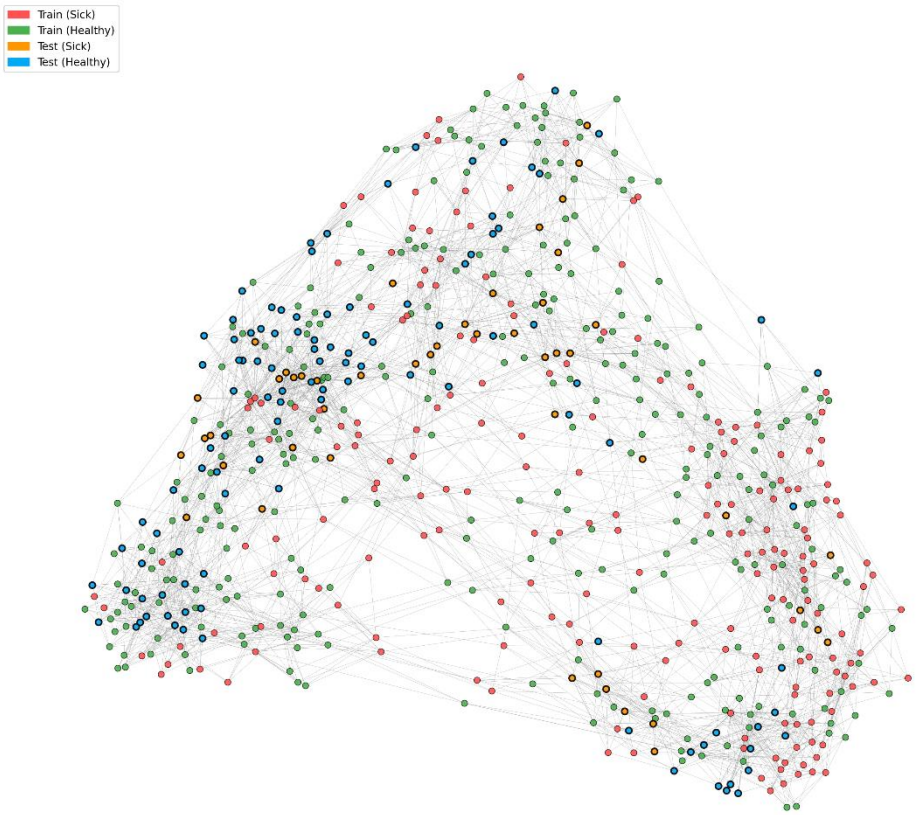
У додатку Г наведено приклади графів подібності, побудованих у межах основної графової моделі Velocity-DTW та контрольного експерименту без відбору ознак.

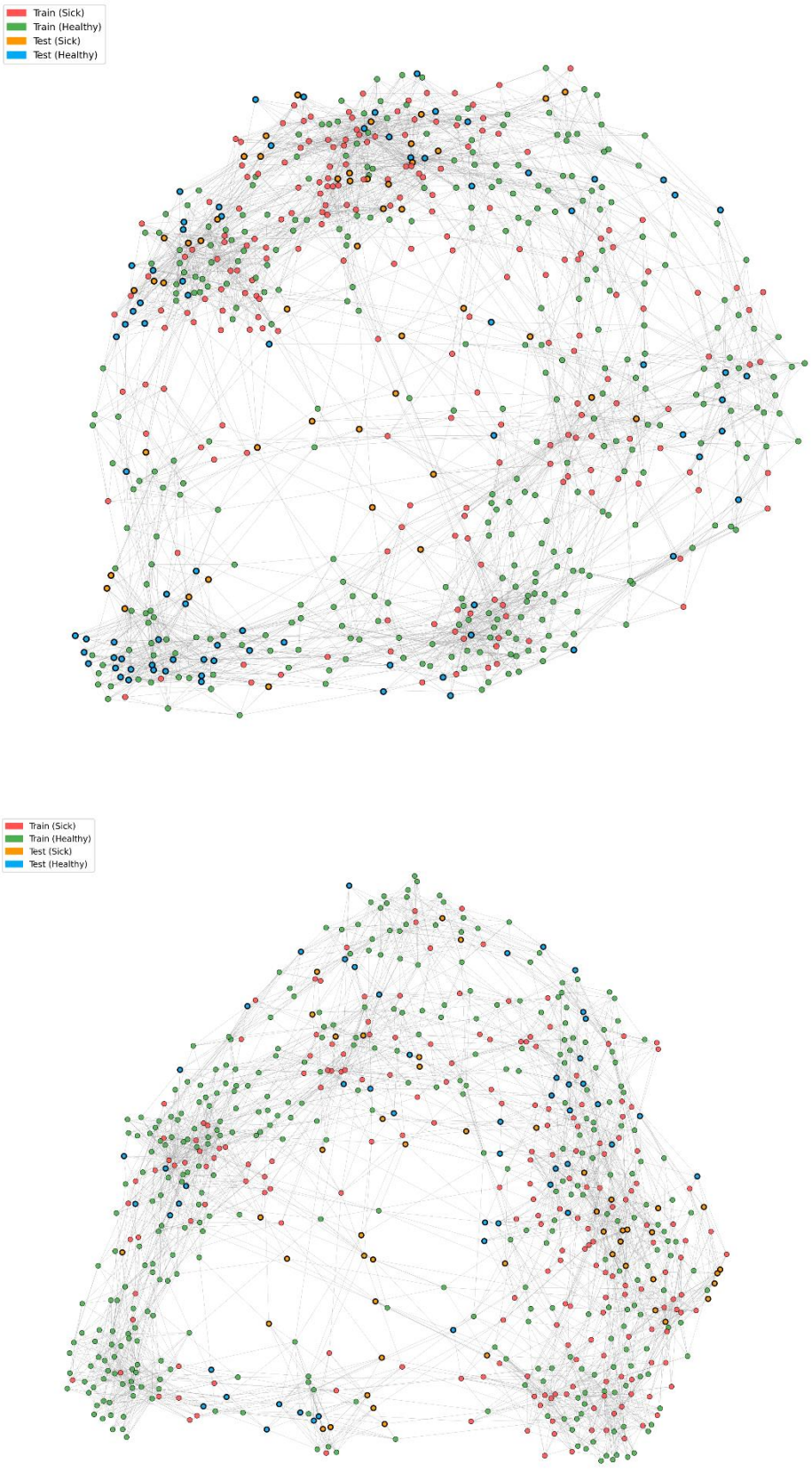
Рисунок Г.1 – Приклад графа подібності часових протеомних профілів у фінальній моделі Velocity-DTW



Примітка. Колір вузла відповідає класу та належності до навчальної або тестової частини відповідного фолду. Ребра відображають подібність між локальними часовими протеомними профілями.

Рисунок Г.2 – Приклади графів подібності для окремих фолдів основної моделі Velocity-DTW





Примітка. Рисунок ілюструє варіативність графової структури між різними фолдами групової крос-валідації.

Рисунок Г.3 – Приклад графа подібності у контрольному експерименті без відбору ознак