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

Факультет комп'ютерно-інформаційних систем та програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

Магістр
(назва освітнього ступеня)

на тему: Розробка прикладної системи для автоматизованого аналізу ЕЕГ-сигналів із використанням топологічного та спектрального аналізу

Виконав(ла): студент(ка) 6 курсу, групи СПм-61
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

(підпис) Задворний О.М. (прізвище та ініціали)

Керівник (підпис) Бойко І.В. (прізвище та ініціали)

Нормоконтроль (підпис) Стоянов Ю. М. (прізвище та ініціали)

Завідувач кафедри (підпис) Петрик М. Р. (прізвище та ініціали)

Рецензент (підпис) Гром'як Р. С. (прізвище та ініціали)

Тернопіль
2025

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

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

Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Петрик М. Р.
(підпис) (прізвище та ініціали)
« » 20___ р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Магістр
(назва освітнього ступеня)

за спеціальністю 121 "Інженерія програмного забезпечення"
(шифр і назва спеціальності)

студенту Задворному Олександру Миколайовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка прикладної системи для автоматизованого аналізу ЕЕГ-сигналів із використанням топологічного та спектрального аналізу

Керівник роботи Бойко Ігор Володимирович, д.ф.-м.н, професор
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «*27*» *листопада* 2025 року № *4/7-1045*

2. Термін подання студентом завершеної роботи

3. Вихідні дані до роботи методи та засоби обробки ЕЕГ із використанням спектрального аналізу, топологічного аналізу та алгоритмів машинного навчання, відкрита база даних CHB-MIT Scalp EEG Database, Python, бібліотеки MNE, Gudhi, Scikit-learn, Imbalanced-learn, Streamlit

4. Зміст роботи (перелік питань, які потрібно розробити)
Аналіз предметної області, обґрунтування методів аналізу даних, проєктування архітектури, попередня обробка даних, екстракція ознак, машинне навчання, програмна реалізація, дослідження ефективності, охорона праці та безпека в надзвичайних ситуаціях

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Слайд 1: Титульний; Слайд 2: Актуальність та недоліки існуючих методів; Слайд 3: Мета, об'єкт, предмет та задачі дослідження; Слайд 4: Функціональні та нефункціональні вимоги до системи; Слайд 5: Діаграма варіантів використання; Слайд 6: Структурна схема конвеєра обробки ЕЕГ-сигналу; Слайд 7: Діаграма класів програмної системи; Слайд 8: Алгоритм роботи методу обробки даних та діаграма станів сесії; Слайд 9: Склад гібридного вектора ознак; Слайд 10: Порівняння спектральної моделі та TDA; Слайд 11: Технологічний стек; Слайд 12: Результати тестування: метрики Precision, Recall, F1-score; Слайд 13: Скріншоти інтерфейсу; Слайд 14: Наукова новизна та практичне значення результатів; Слайд 15: Апробація результатів та публікації; Слайд 16: Висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека в надзвичайних	Стручок В.С., ст. викл. каф. ОХ		
ситуаціях			
Охорона праці	Осухівська Г.М., к.т.н., доцент		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Збір та аналіз літературних джерел, постановка задач	29.11.2025	**Виконано**
	дослідження		
2	Вибір та підготовка набору даних CHB-MIT,	01.12.2025	**Виконано**
	стандартизація EDF-файлів		
3	Розробка алгоритмів попередньої обробки та фільтрації	02.12.2025	**Виконано**
	(ICA, Band-pass)		
4	Реалізація модулів спектральної та топологічної	04.12.2025	**Виконано**
	екстракції ознак (Gudhi)		
5	Навчання та оптимізація моделі Random Forest,	06.12.2025	**Виконано**
	балансування класів		
6	Розробка графічного інтерфейсу користувача (Streamlit)	07.12.2025	**Виконано**
7	Експериментальна перевірка ефективності, розрахунок	08.12.2025	**Виконано**
	метрик		
8	Написання розділу з охорони праці та безпеки в	10.12.2025	**Виконано**
	надзвичайних ситуаціях		
9	Оформлення пояснювальної записки та підготовка до	12.12.2025	**Виконано**
	захисту		
10	Захист кваліфікаційної роботи магістра	22.12.2025	

Студент _____ Задворний О.М.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Бойко І.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Задворний Олександр Миколайович. Розробка прикладної системи для автоматизованого аналізу ЕЕГ-даних із використанням топологічного та спектрального аналізу. Кваліфікаційна робота освітнього ступеня “Магістр”. Тернопільський національний технічний університет імені Івана Пулюя, кафедра програмної інженерії, група СПм-61, спеціальність 121 “Інженерія програмного забезпечення”. Тернопіль, 2025. Сторінок 78, таблиць 10, рисунків 14, додатків 2, бібліографічних посилань 42.

Метою роботи є підвищення ефективності автоматизованого виявлення епілептичних нападів шляхом розробки програмної системи, що використовує гібридний підхід на основі спектрального та топологічного аналізу даних із застосуванням машинного навчання.

Об’єктом дослідження є процес автоматизованого аналізу та класифікації ЕЕГ-сигналів. Предметом дослідження є методи та засоби обробки ЕЕГ із використанням спектрального аналізу, топологічного аналізу (TDA) та алгоритмів машинного навчання.

У роботі обґрунтовано доцільність використання топологічних ознак для виявлення структурних змін мозкової активності. Реалізовано архітектуру системи мовою Python (бібліотеки MNE, Gudhi, Scikit-learn), що виконує попередню обробку (стандартизація, ICA) та формує вектор гібридних ознак на основі спектральної густини потужності та персистентних ландшафтів.

Побудовано класифікатор Random Forest із використанням стратегій балансування даних (SMOTE, RandomUnderSampler). Експериментальне тестування на наборі CHB-MIT підтвердило ефективність підходу (f1-score 0.80). Розроблено веб-інтерфейс (Streamlit) для візуалізації результатів діагностики та інтерактивного аналізу сигналів.

Ключові слова: електроенцефалограма, епілепсія, машинне навчання, топологічний аналіз даних, TDA, персистентні гомології, Random Forest, ICA, Python, Streamlit.

ABSTRACT

Zadvornyi Oleksandr Mykolaiovych. Development of an applied system for the automated analysis of EEG data using topological and spectral analysis. Master's Qualification Paper. Ternopil Ivan Puluj National Technical University, Software Engineering Department, Group SPm-61, Specialty 121 "Software Engineering". Ternopil, 2025. Pages 78, tables 10, figures 14, appendices 2, bibliographic references 42.

The purpose of the work is to increase the efficiency of automated epileptic seizure detection by developing a software system that uses a hybrid approach based on spectral and topological data analysis using machine learning.

The object of research is the process of automated analysis and classification of EEG signals. The subject of research is the methods and tools for EEG processing using spectral analysis, Topological Data Analysis (TDA), and machine learning algorithms.

The work substantiates the feasibility of using topological features to detect structural changes in brain activity. The system architecture has been implemented in Python (MNE, Gudhi, Scikit-learn libraries), which performs preprocessing (standardization, ICA) and forms a vector of hybrid features based on power spectral density and persistence landscapes.

A Random Forest classifier has been built using data balancing strategies (SMOTE, RandomUnderSampler). Experimental testing on the CHB-MIT dataset confirmed the efficiency of the approach (F1-score 0.80). A web interface (Streamlit) has been developed for visualizing diagnostic results and interactive signal analysis.

Keywords: electroencephalogram, epilepsy, machine learning, topological data analysis, TDA, persistent homology, Random Forest, ICA, Python, Streamlit.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ЕЕГ	Електроенцефалограма (електроенцефалографія)
API	Application Programming Interface (інтерфейс прикладного програмування)
CHB-MIT	Children’s Hospital Boston – Massachusetts Institute of Technology (назва бази даних)
DFT	Discrete Fourier Transform (дискретне перетворення Фур’є)
EDF	European Data Format (європейський формат даних біосигналів)
EMG	Electromyogram (електроміограма, м’язова активність)
EOG	Electrooculogram (електроокулограма, рухи очей)
FFT	Fast Fourier Transform (швидке перетворення Фур’є)
GUI	Graphical User Interface (графічний інтерфейс користувача)
ICA	Independent Component Analysis (аналіз незалежних компонент)
MNE	Magnetoencephalography and Electroencephalography (бібліотека Python)
PSD	Power Spectral Density (спектральна густина потужності)
RUS	Random Under-Sampling (метод випадкового проріджування вибірки)
SMOTE	Synthetic Minority Over-sampling Technique (техніка синтетичного збільшення меншого класу)
TDA	Topological Data Analysis (топологічний аналіз даних)

ЗМІСТ

ВСТУП

Епілепсія є одним із найпоширеніших неврологічних захворювань у світі, що вражає мільйони людей. Основним методом діагностики епілепсії залишається електроенцефалографія (ЕЕГ), яка дозволяє реєструвати електричну активність мозку. Проте аналіз ЕЕГ-записів, що можуть тривати від кількох годин до кількох діб, є трудомістким процесом, який вимагає від лікарів-нейрофізіологів високої кваліфікації та значних часових витрат. Ручний аналіз часто супроводжується ризиком помилок через втому або наявність складних артефактів у сигналі.

Існуючі автоматизовані системи детекції нападів здебільшого базуються на спектральному аналізі та класичних методах обробки сигналів. Однак такі підходи не завжди здатні вловити складні нелінійні та структурні зміни в динаміці мозкової активності, що передують нападу або супроводжують його. У цьому контексті перспективним напрямом є використання топологічного аналізу даних (Topological Data Analysis, TDA), який дозволяє оцінювати “форму” даних у багатовимірному просторі та виявляти стійкі патерни, недоступні для традиційних методів. Поєднання спектральних методів із топологічними ознаками та сучасними алгоритмами машинного навчання відкриває нові можливості для підвищення точності та надійності автоматизованих систем діагностики.

Таким чином, розробка прикладної системи, що інтегрує методи TDA та машинного навчання для автоматизованого аналізу ЕЕГ, є актуальним науково-практичним завданням. Метою роботи є підвищення ефективності автоматизованого виявлення епілептичних нападів шляхом розробки прикладної програмної системи, що використовує гібридний підхід на основі спектрального аналізу, топологічного аналізу даних та машинного навчання.

Для досягнення поставленої мети необхідно вирішити такі задачі:

- Провести аналіз предметної області, дослідити природу ЕЕГ-сигналів та оглянути існуючі методи їх обробки.
- Обґрунтувати вибір методів топологічного аналізу як доповнення до класичних спектральних ознак.

- Розробити архітектуру програмної системи для обробки ЕЕГ-даних.
- Реалізувати алгоритми попередньої обробки сигналів, включаючи фільтрацію, стандартизацію каналів та видалення артефактів методом незалежних компонент (ICA).
- Розробити модуль екстракції ознак, що формує вектор ознак на основі спектральної густини потужності та топологічних ландшафтів.
- Побудувати та навчити модель класифікації з використанням методів балансування даних (SMOTE, RandomUnderSampler).
- Розробити інтерфейс системи для візуалізації результатів аналізу.
- Провести експериментальне дослідження ефективності розробленої системи та оцінити якість класифікації.

Об’єктом дослідження є процес автоматизованого аналізу та класифікації біомедичних сигналів (ЕЕГ). Предмет дослідження – методи та програмні засоби обробки ЕЕГ-сигналів із використанням спектрального аналізу, топологічного аналізу даних (TDA) та алгоритмів машинного навчання.

Наукова новизна одержаних результатів полягає в удосконаленні методу класифікації станів ЕЕГ шляхом комбінування класичних спектральних ознак із топологічними ознаками (персистентні ландшафти), отриманими з простору спектральної густини потужності, що дозволило підвищити чутливість системи до виявлення епілептиформної активності, а також в застосуванні підходу до аналізу ЕЕГ, де багатоканальний сигнал розглядається як хмара точок у частотному просторі, до якої застосовується апарат персистентних гомологій для виявлення стійких циклічних патернів (H_1-гомологій).

Практичне значення одержаних результатів полягає в розробці системи, яка дозволяє автоматично виявляти епілептичні напади, відображати результати у вигляді часових діаграм ймовірностей та спектрограм та бути використаною як прототип системи підтримки прийняття рішень для лікарів-епілептологів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИМОГ ДО СИСТЕМИ

Розробка ефективних програмних засобів для автоматизованої діагностики неврологічних патологій вимагає комплексного підходу, що поєднує глибоке розуміння фізіологічної природи біосигналів та застосування сучасних математичних методів їх обробки [1, 2]. У даному розділі проведено аналіз електроенцефалограми (ЕЕГ) як об’єкта дослідження, виявлено основні проблеми, пов’язані з ручною обробкою тривалих моніторингових записів та обґрунтовано необхідність автоматизації цього процесу.

Особлива увага приділяється огляду існуючих алгоритмічних підходів до виявлення епілептиформної активності. Зокрема, розглядаються класичні методи спектрального аналізу, що базуються на перетворенні Фур’є, а також новітні методи топологічного аналізу даних (TDA), які дозволяють оцінювати структурні характеристики сигналу у багатовимірному просторі [3].

1.1 Характеристика електроенцефалограми як об’єкта дослідження

Електроенцефалографія є неінвазивним методом реєстрації сумарної електричної активності головного мозку, що відводиться зі скальпа [4]. Цей сигнал є результатом синхронізованої активності великої кількості нейронів і несе важливу інформацію про функціональний стан центральної нервової системи.

ЕЕГ-сигнал класифікується на п’ять ритмічних хвиль із певним діапазоном частот, а саме: дельта (0,1-4 Гц), тета (4-8 Гц), альфа (8-12 Гц), бета (12-30 Гц) і гамма (30-60 Гц). Епілептичні напади виявляються як спайки (різкі підйоми) та гострі хвилі в синхронній електричній активності нейронів. Ці патерни суттєво відрізняються від фонової активності за амплітудою та морфологією. Окрім діагностики епілепсії, ЕЕГ використовується для прогнозування епілептичних нападів, характеристики явищ сну, діагностики енцефалопатій [5, 6]. Також методи аналізу сигналів кори головного мозку активно розвиваються у напрямку дослідження неврологічних рухів та врахування когнітивних зворотних зв’язків, а

також моніторингу глибини анестезії з використанням нелінійних методів для вивчення сигналу повільнохвильового сну [7].

1.1.1 Фізіологічна природа ЕЕГ та види епілептичних артефактів

Електроенцефалограма відображає сумарні постсинаптичні потенціали, що генеруються нейронами кори головного мозку. Однією з найважливіших задач комп'ютерної обробки таких біомедичних сигналів є скасування (усунення) або зменшення шуму та артефактів [8]. Це особливо критично у випадках, коли сигнал спотворений адитивним і мультиплікативним шумом, або коли корисна інформація про епілептичну активність становить лише незначну частину сигналу, а нерелевантні фрагменти вважаються артефактами.

Залежно від походження, артефакти ЕЕГ поділяються на два основні класи:

- Технічні артефакти (шум) – включають наведення від електромережі (50 Гц), шум електродів, рухи кабелів та перешкоди від обладнання. Хоча вплив технічного шуму можна значно зменшити шляхом удосконалення конструкції електродів, екранування приміщень і методів їх кріплення, повністю усунути його на етапі реєстрації неможливо [4].
- Фізіологічні артефакти – виникають внаслідок електричної активності інших біологічних джерел організму пацієнта. На відміну від технічних шумів, спотворювальний вплив артефактів фізіологічного походження є неминучим, оскільки вони генеруються самим суб'єктом дослідження.

Найпоширенішими типами артефактів, що ускладнюють аналіз ЕЕГ, є:

- Рухи очей та моргання (EOG) – генерують низькочастотні хвилі високої амплітуди, які особливо виражені у лобних відведеннях і можуть імітувати повільні хвилі дельта-діапазону.
- М'язова активність (EMG) – виникає від напруження м'язів шиї, обличчя або щелепи. Такі артефакти мають широкий частотний спектр і часто перекривають бета- та гамма-діапазони, що ускладнює виділення корисного сигналу.

- Серцева діяльність (ECG) – ритмічні потенціали серця, що можуть накладатись на сигнал, створюючи періодичні піки, які іноді помилково класифікуються як епілептичні спайки.
- Рухи головою – можуть вносити значні зміни в базову лінію сигналу.

Відповідно, переважна більшість алгоритмів попередньої обробки ЕЕГ, включаючи методи, використані в даній роботі (зокрема ICA – аналіз незалежних компонент), розроблена саме для боротьби з фізіологічними артефактами. Їх коректне усунення є критичним етапом, оскільки наявність артефактів може призвести до хибнопозитивних спрацювань системи детекції нападів.

1.1.2 Проблематика автоматизованої детекції нападів

Створення надійної системи автоматизованого аналізу ЕЕГ-сигналів супроводжується низкою викликів, зумовлених складною природою біомедичних даних та особливостями епілепсії. Можна виділити чотири ключові проблеми, що ускладнюють розробку універсальних алгоритмів детекції:

- Нестаціонарність та нелінійність сигналу – статистичні властивості ЕЕГ змінюються з часом, що робить сигнал нестаціонарним. Те, що є нормою для певного стану мозку (наприклад, сон), може нагадувати патологічну активність у іншому стані. Традиційні методи аналізу, такі як перетворення Фур’є, часто не здатні вловити динамічні зміни та природу епілептичного сигналу, що обмежує їх ефективність у ранній детекції [8, 9].
- Міжсуб’єктна варіативність – патерни судомної активності є індивідуальними, морфологія спайків, частота розрядів та локалізація зміни сигналу можуть суттєво відрізнятися у різних пацієнтів [10]. Це створює проблему “узагальнення” (generalization) для моделей машинного навчання: алгоритм, навчений на групі одних пацієнтів, може демонструвати низьку точність на нових даних.
- Дисбаланс класів – епілептичні напади є рідкісними подіями в часі, у добовому записі (24 години) сумарна тривалість нападів може становити

лише декілька хвилин, тоді як решта часу припадає на нормальну фонову активність. Такий значний перекіс даних призводить до того, що стандартні класифікатори схильні ігнорувати клас меншості (напад), максимізуючи загальну точність за рахунок класу більшості [4].

- Суб'єктивність розмітки даних – "золотим стандартом" для навчання моделей є розмітка, виконана лікарем-експертом. Однак візуальний аналіз є трудомістким і суб'єктивним процесом: навіть досвідчені нейрофізіологи можуть мати розбіжності у визначенні точного часу початку та кінця нападу, особливо за наявності артефактів [1].

Зазначені проблеми вимагають застосування адаптивних підходів та забезпечення високої стійкості алгоритмів штучного інтелекту, що є критичним для нейрокомп'ютерних інтерфейсів та діагностичних систем [11]. Зокрема, це досягається шляхом використання топологічних ознак (TDA) для захоплення інваріантних властивостей сигналу, та спеціальних стратегій навчання (SMOTE, крос-валідація по пацієнтах), які реалізовані в даній кваліфікаційній роботі.

1.2 Огляд існуючих методів аналізу часових рядів

Аналіз ЕЕГ сигналів, що представляють собою багатовимірні часові ряди, традиційно здійснюється в трьох основних доменах: часовому, частотному та часово-частотному. Кожен з цих підходів дозволяє виділити специфічні характеристики (ознаки), необхідні для класифікації станів мозку. Однак через нестаціонарну та нелінійну природу біосигналів, класичні методи часто виявляються недостатньо чутливими до тонких змін структури сигналу [4].

Це зумовлює зростаючий інтерес до новітніх математичних підходів, зокрема топологічного аналізу даних (TDA), який розглядає сигнал не як функцію часу, а як геометричний об'єкт (хмару точок) у фазовому просторі. У підрозділі здійснено порівняльний аналіз традиційних спектральних методів, які складають основу базової діагностики, та перспективних топологічних методів, що дозволяють виявляти стійкі інваріантні патерни, характерні для епілептичних нападів [5].

1.2.1 Методи спектрального аналізу

Спектральний аналіз є фундаментальним інструментом у цифровій обробці сигналів (DSP), який дозволяє перейти від представлення сигналу в часовій області (амплітуда як функція часу) до частотної області (енергія як функція частоти). Для ЕЕГ це є особливо важливим, оскільки різні фізіологічні стани мозку (сон, неспання, напад) характеризуються активністю в різних частотних діапазонах [7].

Основним методом переходу є дискретне перетворення Фур'є (DFT), яке на практиці реалізується за допомогою швидкого алгоритму (FFT). Проте, пряме застосування FFT до нестаціонарних сигналів має суттєвий недолік: воно дає усереднений спектр для всього запису, втрачаючи інформацію про зміни частот у часі. Крім того, класична періодограма має високу дисперсію шуму, що робить її менш надійною для аналізу зашумлених сигналів [4].

Для вирішення цих проблем у розробленій системі використано метод Велча (Welch's Method) для оцінки спектральної густини потужності (Power Spectral Density, PSD). Цей метод є вдосконаленням класичної періодограми і полягає у виконанні наступних кроків:

- Сегментація – вхідний сигнал розбивається на коротші сегменти.
- Віконне перетворення – до кожного сегмента застосовується спеціальна "віконна" функція (наприклад, вікно Ханнінга або Хеммінга), щоб зменшити ефект витоку спектру (spectral leakage) на краях сегментів.
- Усереднення – обчислюється періодограма для кожного віконного сегмента, після чого результати усереднюються.

Такий підхід дозволяє суттєво зменшити дисперсію спектральної оцінки, роблячи графік PSD більш "гладким" та інформативним для подальшого аналізу. На основі отриманої PSD здійснюється екстракція ознак для машинного навчання шляхом інтегрування потужності у клінічно значущих діапазонах: Delta, Theta, Alpha, Beta та Gamma. Для нормалізації даних та зменшення впливу викидів до розрахованих значень потужності застосовується логарифмічне перетворення (шкала децибел).

1.2.2 Методи топологічного аналізу даних (TDA)

Традиційно автоматизований аналіз ЕЕГ спирався виключно на спектральні показники (PSD) та лінійні методи оцінки зв'язності. Проте сучасні дослідження підкреслюють, що епілептичні напади супроводжуються складними структурними перебудовами динамічних мереж мозку, які неможливо повною мірою описати лише енергетичними характеристиками [6]. Це зумовлює необхідність доповнення спектрального аналізу методами, здатними оцінювати глобальну "форму" даних, такими як TDA.

Топологічний аналіз даних (Topological Data Analysis, TDA) – це сучасний підхід до аналізу даних, що базується на алгебраїчній топології та дозволяє вивчати форму даних. На відміну від спектрального аналізу, який фокусується на частотному складі, TDA досліджує зв'язність, наявність "дірок" (циклів) та порожнин у структурі даних, що є інваріантними до деформацій та стійкими до шуму [12].

У контексті даної роботи багатоканальний сигнал ЕЕГ (після перетворення у PSD) розглядається як набір точок у багатовимірному просторі, де кожна точка відповідає певному каналу ЕЕГ, а її координати – значенням потужності на різних частотах. Таке представлення називається хмарою точок (Point Cloud).

Для аналізу топології цієї хмари використовується метод персистентних гомологій (Persistent Homology). Ключовим поняттям тут є симпліціальний комплекс – структура, що будується шляхом з'єднання точок, які знаходяться на відстані не більше певного порогового значення ε. У роботі використано комплекс Вієторіса-Ріпса (Vietoris-Rips complex), оскільки він є найбільш обчислювально ефективним [12].

Процес аналізу полягає у зміні параметра фільтрації ε від 0 до максимуму:

1) При малому ε точки є ізольованими компонентами (гомології розмірності 0, H_0).
2) Зі збільшенням ε точки з'єднуються ребрами, утворюючи компоненти зв'язності.

3) При подальшому зростанні можуть утворюватися замкнуті цикли або "дірки" (гомології розмірності 1, H_1).

Саме 1-вимірні гомології (H_1), що відповідають циклам у просторі ознак, є найбільш інформативними для виявлення епілептичних патернів, оскільки вони відображають специфічні порушення синхронізації між каналами мозку [5].

Результатом роботи алгоритму є діаграма стійкості (Persistence Diagram), на якій кожна топологічна особливість (цикл) відображається точкою з координатами$(birth, death)$, де $birth$ — значення ε, при якому цикл з'явився, а $death$ – значення, при якому він "затягнувся".

Оскільки діаграми стійкості є складними для прямого використання у класичних алгоритмах машинного навчання (таких як Random Forest), у роботі застосовано метод векторизації – персистентні ландшафти (Persistence Landscapes). Це перетворення трансформує діаграму у функцію (або вектор чисел), яка описує "силу" та "тривалість" існування топологічних особливостей, роблячи їх придатними для навчання класифікатора [13].

1.3 Аналіз інструментальних засобів розробки

Розроблена програмна система – це інструмент з відкритим вихідним кодом на основі мови Python, створений для полегшення автоматизованого аналізу даних ЕЕГ. Основною метою системи є підвищення ефективності діагностики епілепсії шляхом інтеграції класичних методів обробки сигналів із сучасними методологіями топологічного аналізу даних (TDA).

Система використовує можливості потужних бібліотек, таких як MNE-Python [14] для попередньої обробки та очищення сигналів від артефактів, та бібліотеки Gudhi [15] для дослідження топологічних характеристик (персистентних гомологій) у просторі станів мозку. Інтегруючи TDA у традиційний конвеєр аналізу, розроблене рішення пропонує унікальний гібридний підхід до виявлення епілептиформної активності, що дозволяє фіксувати структурні зміни сигналу, недоступні для лінійних методів.

Основна цінність системи полягає в її здатності підвищувати точність класифікації станів (“напад” / “норма”) шляхом включення топологічних інваріантів у вектори ознак. Дані обробляються з використанням стандартної системи розміщення електродів 10–20. Для аналізу сигнали сегментуються на епохи тривалістю 5 секунд для виділення спектральних (PSD) та топологічних ознак.

Уможливлюючи характеристику активності мозку як на частотному, так і на структурному рівнях, цей інструмент відкриває нові можливості для клінічної практики, зокрема як система підтримки прийняття рішень для лікарів-епілептологів.

1.3.1 Огляд мови Python та бібліотек для Data Science

На сьогодні у галузі аналізу нейровізуалізації існує широкий спектр фреймворків, кожен з яких спеціалізується на окремих аспектах обробки даних мозку. Наприклад, Brainstorm широко визнаний завдяки своїй надійній підтримці аналізу МЕГ та ЕЕГ, що полегшує локалізацію джерел. FieldTrip демонструє високу ефективність у частотному аналізі та непараметричному статистичному тестуванні, що робить його цінним для дослідження нейронних осциляцій.

Платформа SPM надає передові інструменти статистичного моделювання для виявлення регіональних змін активності мозку (переважно для МРТ), а FSL спеціалізується на дифузійній візуалізації. Серед інструментів для ЕЕГ також виділяється EEGLAB завдяки зручному графічному інтерфейсу та великій бібліотеці плагінів.

Однак для розробки програмної системи в даній роботі було обрано бібліотеку MNE-Python [14]. Вона забезпечує найкращу інтеграцію з екосистемою Data Science (NumPy, Scikit-learn), надає комплексні інструменти для обробки сигналів та дозволяє ефективно працювати з багатовимірними масивами даних [17], що є критичним для реалізації гібридного підходу з використанням TDA.

Мова програмування Python була обрана як основний інструмент розробки через її високорівневі синтаксис, кросплатформеність та, найголовніше, наявність потужних відкритих бібліотек для наукових обчислень. Це дозволяє зосередитися на реалізації алгоритмів аналізу ЕЕГ, а не на низькорівневих деталях керування пам’яттю.

У таблиці 1.1 наведено перелік основних бібліотек, використаних у розробленій системі, із зазначенням їх конкретного функціонального призначення в рамках даного проєкту.

Таблиця 1.1 – Технологічний стек програмної системи

Бібліотека	**Модуль**	**Призначення в системі**
MNE-Python	mne.io, mne.preprocessing	Завантаження файлів формату EDF, стандартизація назв каналів, частотна фільтрація, видалення артефактів (ICA), розбиття сигналу на епохи.
Gudhi	gudhi.RipsComplex	Реалізація алгоритмів топологічного аналізу даних. Побудова комплексу Вієторіса-Ріпса та обчислення діаграм стійкості (Persistent Homology).
Scikit-learn	sklearn.ensemble, sklearn.pipeline	Побудова моделі машинного навчання, створення конвеєра обробки даних, оцінка метрик якості (F1-score, Precision, Recall).
Imbalanced-learn	imblearn.over_sampling	Вирішення проблеми дисбалансу класів за допомогою алгоритмів SMOTE (синтетичне збільшення меншого класу) та RandomUnderSampler [16].
NumPy	numpy	Ефективні операції з багатовимірними масивами даних (матриці ознак, вектори сигналів), математичні обчислення [17].
SciPy	scipy.signal	Додаткова цифрова обробка сигналів: обчислення спектрограм, медіанна фільтрація результатів передбачення для згладжування шумів.
Streamlit	streamlit	Розробка інтерактивного веб-інтерфейсу користувача для завантаження файлів та візуалізації результатів аналізу в реальному часі.

Особливістю даної реалізації є інтеграція бібліотеки Gudhi [15], яка є спеціалізованим інструментом для обчислювальної топології, з класичним стеком Scikit-learn. Це дозволяє створити єдиний автоматизований конвеєр (Pipeline), де топологічні ознаки генеруються на льоту та передаються класифікатору нарівні зі спектральними характеристиками.

1.3.2 Інструменти для топологічного аналізу

Для реалізації модуля топологічного аналізу в системі було обрано бібліотеку Gudhi (Geometric Understanding in Higher Dimensions). Це відкрита бібліотека, що розробляється дослідницькою групою INRIA, і є одним із найпотужніших інструментів для обчислювальної топології на сьогодні.

Вибір Gudhi зумовлений наступними факторами:

- Продуктивність – ядро бібліотеки написане на мові C++, що забезпечує високу швидкість обчислень, це є критично важливим фактором для даної роботи, оскільки обробка одного пацієнта може включати аналіз тисяч епох ЕЕГ.
- Структура даних – бібліотека реалізує ефективну структуру даних Simplex Tree, яка дозволяє компактно зберігати та швидко оперувати симпліціальними комплексами великої розмірності.
- Python-інтерфейс – наявність зручних прив’язок (bindings) до Python дозволяє легко інтегрувати топологічні обчислення в загальний конвеєр обробки даних разом з NumPy та Scikit-learn.

У лістингу 1.1 наведено фрагмент програмного коду розробленої системи, який демонструє використання бібліотеки Gudhi для обчислення персистентних діаграм.

Лістинг 1.1 – Реалізація методу обчислення персистентних діаграм

```
import gudhi as gd

def _compute_persistence_diagram(self, point_cloud):
    # 1. Побудова комплексу Вієторіса-Ріпса (max_dim=2 для H1)
    rips = gd.RipsComplex(points=point_cloud,
                          max_edge_length=1)
    st = rips.create_simplex_tree(max_dimension=2)

    # 2. Обчислення персистентності
    persistence = st.persistence()

    # 3. Фільтрація: відбір 1-вимірних циклів (H1)
    h1_cycles = []
    for dim, (birth, death) in persistence:
```

```
            # Ігноруємо H0, H2 та нескінченні цикли
            if dim == 1 and death != float('inf'):
                h1_cycles.append((dim, (birth, death)))

    return h1_cycles
```

Як видно з лістингу, клас “gd.RipsComplex” приймає на вхід хмару точок (в нашому випадку – нормалізовані вектори спектральної потужності) і будує фільтрацію. Параметр “max_dimension” вказує на те, що обчислення обмежуються топологічними особливостями низьких розмірностей (компоненти зв’язності та цикли), що є достатнім для задач аналізу ЕЕГ та дозволяє економити обчислювальні ресурси.

1.4 Постановка задачі та формування вимог до системи

На основі проведеного аналізу предметної області та обраного інструментарію, постає необхідність формалізації вимог до розроблюваної програмної системи. У даному підрозділі здійснюється постановка задачі проєктування, визначаються межі системи та її цільова аудиторія.

Чітке формулювання функціональних та нефункціональних вимог, а також моделювання сценаріїв взаємодії користувачів із системою, є ключовим етапом, що передує безпосередній програмній реалізації. Це дозволяє гарантувати, що створений продукт відповідатиме потребам кінцевих користувачів – лікарів-нейрофізіологів та дослідників, забезпечуючи високу точність діагностики та зручність використання.

1.4.1 Визначення акторів та сценаріїв використання

Моделювання системи починається з ідентифікації дійових осіб (акторів), які взаємодіють із програмним продуктом. Враховуючи специфіку прикладного застосунку та наявність інтуїтивно зрозумілого інтерфейсу налаштувань, у системі визначено єдиного актора – лікар-нейрофізіолог. Це основний та єдиний

користувач системи, він володіє знаннями предметної області, що дозволяє йому не лише завантажувати дані, але й самостійно коригувати параметри чутливості алгоритму (Threshold) залежно від клінічної задачі.

Взаємодія лікаря із системою декомпозована на набір сценаріїв використання (Use Cases), які представлені у таблиці 1.2. Ці сценарії охоплюють повний цикл роботи: від завантаження даних до детального аналізу результатів. Для зменшення складності, основні процеси розбиті на простіші сценарії з використанням відношень включення (include) та розширення (extend).

Таблиця 1.2 – Сценарії використання програмної системи

ID	**Назва сценарію**	**Опис**	**Тип зв'язку**	**Пов’язаний сценарій**
UC-1	Завантаження файлу ЕЕГ	Користувач обирає та завантажує файл .edf через веб-інтерфейс.	-	-
UC-1.1	Перевірка формату та цілісності	Система автоматично перевіряє структуру файлу та наявність необхідних каналів.	<<include>>	UC-1
UC-2	Налаштування конфігурації	Користувач змінює поріг чутливості (Sensitivity) та параметри згладжування (Smoothing Kernel) через бічну панель.	-	-
UC-3	Автоматизований аналіз	Система виконує повний цикл обробки даних після завантаження.	-	-
UC-3.1	Попередня обробка сигналу	Фільтрація шумів та застосування ICA для видалення артефактів.	<<include>>	UC-3
UC-3.2	Екстракція гібридних ознак	Обчислення спектральної потужності та топологічних інваріантів (Persistence Landscapes).	<<include>>	UC-3
UC-3.3	Класифікація станів	Застосування навченої моделі для визначення ймовірності нападу.	<<include>>	UC-3
UC-4	Візуалізація результатів	Відображення часової шкали (Timeline) з маркуванням виявлених епілептичних подій.	-	-
UC-5	Детальна інспекція події	Перегляд користувачем детальної інформації про конкретний часовий сегмент.	<<extend>>	UC-4
UC-5.1	Візуалізація сирого сигналу	Відображення багатоканального ЕЕГ у часовій області для обраного моменту.	<<include>>	UC-5
UC-5.2	Спектральний аналіз сегменту	Побудова та відображення спектрограми для обраного каналу.	<<include>>	UC-5

Така структура сценаріїв дозволяє чітко розділити етапи автоматичної

обробки (UC-3) та інтерактивної роботи лікаря з результатами (UC-4, UC-5). Наочна схема зв’язків між актором та визначеними прецедентами зображена на Рисунку 1.1.

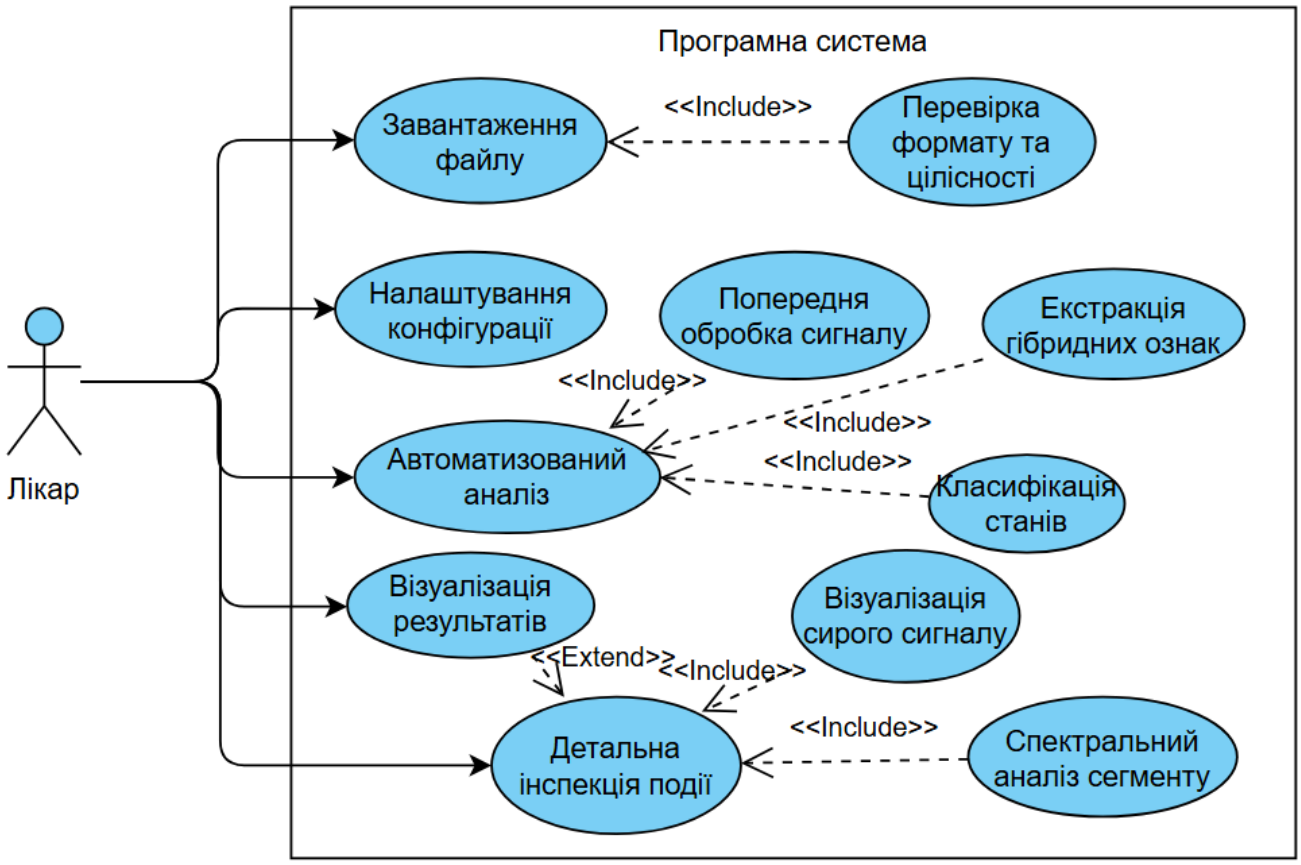

Рисунок 1.1 – Діаграма варіантів використання системи

Представлені сценарії слугують основою для формування детальних вимог до системи, які гарантують, що розроблене програмне забезпечення задовольнятиме клінічні та технічні потреби користувачів. Ці вимоги детально розглядаються у наступному пункті.

1.4.2 Функціональні та нефункціональні вимоги до системи

На етапі аналізу вимог важливо чітко розмежувати, що система повинна робити (функціональні вимоги), та як вона повинна працювати (нефункціональні вимоги). Це забезпечує основу для подальшого вибору архітектурних рішень.

Функціональні вимоги визначають набір операцій, які система має виконувати для підтримки сценаріїв використання. Повний перелік функціональних вимог наведено у таблиці 1.3.

Таблиця 1.3 – Функціональні вимоги до системи

ID	**Категорія**	**Опис вимоги**
FR-01	Введення даних	Система повинна підтримувати завантаження файлів у форматі EDF (European Data Format) через веб-інтерфейс.
FR-02	Валідація	Система повинна автоматично перевіряти наявність 18 стандартних каналів (система “10-20”) у завантаженому файлі.
FR-03	Обробка сигналу	Система повинна виконувати смугову фільтрацію (Band-pass filtering) у діапазоні 1-50 Гц.
FR-04	Видалення артефактів	Система повинна застосовувати алгоритм ICA для автоматичного очищення сигналу від фізіологічних артефактів.
FR-05	Розбиття на епохи	Сигнал повинен автоматично розбиватися на епохи фіксованої тривалості (за замовчуванням 5 секунд).
FR-06	Аналіз даних	Для кожної епохи повинні обчислюватися спектральні ознаки (PSD) та топологічні характеристики (діаграми стійкості).
FR-07	Класифікація	Система повинна класифікувати кожну епоху (“напад” / “норма”) за допомогою попередньо навченої моделі.
FR-08	Візуалізація	Система повинна відображати інтерактивний графік (Timeline).
FR-09	Інспекція	Система повинна надавати можливість перегляду сирого сигналу та спектрограми для обраного моменту часу.

Нефункціональні вимоги описують якісні характеристики системи, такі як продуктивність, зручність та безпека. Вони наведені у таблиці 1.4.

Таблиця 1.4 – Нефункціональні вимоги до системи

ID	**Атрибут якості**	**Опис вимоги**
NFR-01	Продуктивність	Час обробки однієї години запису ЕЕГ не повинен перевищувати 10 хвилин (завдяки використанню ядра C++ бібліотеки Gudhi).
NFR-02	Зручність	Інтерфейс має бути реалізований як Single Page Application (SPA) з інтуїтивним керуванням параметрами через графічні елементи.
NFR-03	Конфіденційність	Система повинна забезпечувати автоматичне видалення метаданих пацієнта з оперативної пам’яті перед початком аналізу.
NFR-04	Локальність	Обробка даних повинна відбуватися локально або в ізольованому контейнері без збереження файлів на сервері після завершення сесії.

Узагальнюючи вищезазначене, сформульовані вимоги створюють фундамент

для розробки надійної та ефективної діагностичної системи. Поєднання функціоналу глибокого аналізу даних з вимогами до швидкодії та приватності дозволяє створити інструмент, який реально допоможе лікарям у їхній повсякденній практиці, зменшуючи рутинне навантаження та підвищуючи точність діагнозів.

Підсумовуючи результати, можна стверджувати, що виконаний комплексний аналіз предметної області та існуючих методів діагностики підтвердив необхідність автоматизації процесу виявлення епілептичних нападів. Встановлено, що інтеграція класичного спектрального аналізу з методами топологічного аналізу даних (TDA) на базі обраного стеку технологій (Python, MNE, Gudhi) дозволить підвищити чутливість системи до структурних змін сигналу. Сформульовані функціональні та нефункціональні вимоги, разом із визначеними сценаріями використання, створюють чітке технічне завдання та теоретичне підґрунтя для проєктування архітектури та програмної реалізації системи, що буде детально розглянуто у наступному розділі.

2 ПРОЄКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

На основі проведеного аналізу предметної області та сформованих вимог, у даному розділі розглядається процес проєктування архітектури програмного засобу для автоматизованої детекції епілептичних нападів. Метою цього етапу є трансформація теоретичних засад та функціональних вимог у конкретні технічні рішення, що забезпечать надійність, швидкодію та точність діагностики.

У розділі обґрунтовано вибір технологічного стека та описано загальну структуру конвеєра обробки даних, який охоплює всі етапи: від завантаження “сирих” ЕЕГ-сигналів до отримання результатів класифікації. Детально розглядаються алгоритми попередньої обробки, зокрема стандартизація каналів та фільтрація артефактів методом незалежних компонент (ICA). Особливу увагу приділено проєктуванню підсистеми екстракції ознак, де реалізовано гібридний підхід поєднання класичних спектральних характеристик із новітніми топологічними інваріантами. Також описано стратегію побудови моделі машинного навчання, методи подолання дисбалансу класів та налаштування гіперпараметрів класифікатора для досягнення оптимальних метрик якості.

2.1 Архітектура програмної системи та вибір технологічного стека

Ефективність та надійність системи значною мірою визначаються правильністю обраних архітектурних рішень та інструментальних засобів. У цьому підрозділі обґрунтовано вибір архітектури програмного комплексу, яка базується на модульному принципі та об’єктно-орієнтованому підході. Така структура забезпечує чітке розмежування логіки обробки даних, алгоритмів машинного навчання та інтерфейсу користувача, що спрощує підтримку та майбутнє масштабування системи.

Технологічним базисом розробки обрано екосистему мови програмування Python, яка є стандартом у сфері наукових обчислень (Data Science). Ядро системи спирається на інтеграцію спеціалізованих бібліотек: MNE-Python для цифрової

обробки сигналів, Gudhi для обчислення топологічних інваріантів та Scikit-learn для побудови класифікаційних моделей. Реалізація графічного інтерфейсу виконана на базі фреймворку Streamlit [18], що дозволяє функціонування системи у форматі вебзастосунку. Далі детально розглянуто схему інформаційних потоків та взаємодію основних компонентів системи.

2.1.1 Загальна схема конвеєра обробки даних

Архітектура розробленої системи спроєктована з урахуванням гнучкості та обчислювальної ефективності, інтегруючи різнорідні компоненти, що працюють узгоджено для забезпечення наскрізного конвеєра (пайплайну) аналізу біомедичних сигналів. Вона побудована на фундаменті перевірених наукових бібліотек, таких як MNE-Python та Scikit-learn, а також імплементує новітні методи топологічного аналізу даних (TDA) через використання бібліотеки Gudhi.

Програмне забезпечення можна структурувати на кілька ключових функціональних модулів, взаємодія яких відображає логіку обробки даних:

- Попередня обробка даних (Data Preprocessing) – цей модуль відповідає за нормалізацію “сирих” даних ЕЕГ. Він включає процедури стандартизації монтажу електродів, частотну фільтрацію та очищення сигналу від фізіологічних артефактів за допомогою методу незалежних компонент (ICA) [8]. Це гарантує, що дані, які подаються на вхід алгоритмів, позбавлені шуму, що може спотворити результати аналізу.
- Аналіз спектральної густини потужності (PSD Analysis) – модуль обчислює розподіл енергії сигналу за частотними діапазонами для кожного каналу. Цей етап є критичним для ідентифікації ключових мозкових ритмів (дельта, тета, альфа, бета, гамма), зміни в яких корелюють з епілептичною активністю.
- Вилучення топологічних ознак (Topological Feature Extraction) – інтеграція методів TDA є центральним інноваційним елементом системи. Для аналізу структурних характеристик багатоканального сигналу використовуються

діаграми стійкості та персистентні ландшафти (persistence landscapes). Це дозволяє ідентифікувати інваріантні геометричні патерни мозкової активності, недоступні для класичних лінійних методів.

- Класифікація методами машинного навчання (ML Classification) – на фінальному етапі сформовані гібридні вектори ознак передаються до моделі класифікації. У роботі використано метод випадкового лісу (Random Forest), який забезпечує високу стійкість до перенавчання та дозволяє ефективно розрізняти стани “норма” та “напад” навіть в умовах дисбалансу класів.

Архітектура є модульною, що дозволяє незалежно використовувати різні етапи або модифікувати окремі алгоритми (наприклад, змінити параметри фільтрації чи метод векторизації діаграм) без порушення загальної цілісності системи. Така модульність гарантує, що програмне забезпечення можна легко адаптувати для широкого спектра дослідницьких завдань. Графічне представлення розробленої схеми конвеєра зображено на рисунку 2.1.

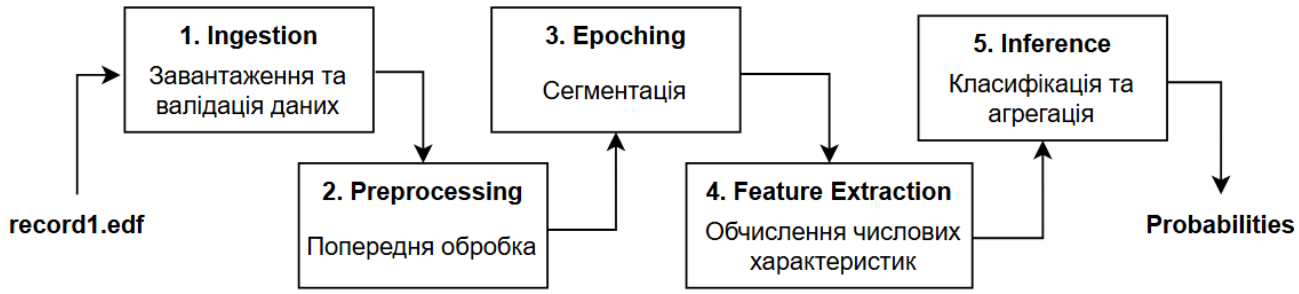

Рисунок 2.1 – Структурна схема конвеєра обробки ЕЕГ-сигналу

З точки зору реалізації, система спирається на екосистему Python для наукових обчислень: NumPy та SciPy забезпечують ефективні матричні операції, MNE-Python надає спеціалізований інструментарій для роботи з форматом EDF, а бібліотека Gudhi гарантує високу швидкодію топологічних обчислень. Програмне забезпечення оптимізоване для відтворюваності результатів: кожен крок чітко

визначений, а фіксація параметрів генерації випадкових чисел (random seeds) у моделях машинного навчання забезпечує валідність порівняльного аналізу.

Програмна реалізація даного конвеєра інкапсульована у класі EEGPipeline, що дозволяє абстрагуватися від деталей низькорівневої обробки у головному файлі застосунку. Ключовим методом, що виконує виклик усіх підетапів, є метод “process”. У лістингу 2.1 наведено фрагмент коду, що демонструє послідовність викликів у головному циклі обробки.

Лістинг 2.1 – Реалізація методу обробки даних у класі EEGPipeline

```
def process(self, edf_path: str, seizure_intervals=None):
    # 1. Завантаження, очищення та сегментація
    raw = self.preprocessor.load_and_clean(edf_path)
    if raw is None: return None, None
    epochs = mne.make_fixed_length_epochs(raw,
        duration=self.config.epoch_duration,
        preload=True, verbose=False)

    # 2. Екстракція спектральних та топологічних ознак
    spec_feats, psd_data = self._extract_spectral(epochs)
    tda_feats = self.tda_extractor.extract(psd_data)

    # 3. Обчислення часових характеристик
    ep_data = epochs.get_data()
    ll = np.sum(np.abs(np.diff(ep_data, axis=2)), axis=2)
    var = np.var(ep_data, axis=2)

    # 4. Об'єднання всіх ознак у фінальний вектор
    return np.hstack((spec_feats, tda_feats, ll, var))
```

Така організація коду дозволяє легко модифікувати окремі етапи (наприклад, змінити алгоритм фільтрації або додати нові ознаки) без порушення загальної логіки роботи системи. Отриманий на виході масив є повністю підготовленим для передачі у модуль класифікації.

2.1.2 Проєктування класів та модулів системи

Для забезпечення гнучкості, розширюваності та зручності супроводу програмного коду, архітектура системи спроєктована з використанням принципів

об’єктно-орієнтованого програмування. Функціональність системи розділена на два основні рівні: рівень бізнес-логіки (backend), що відповідає за обробку даних, та рівень представлення (frontend), що реалізує взаємодію з користувачем.

Основні сутності предметної області інкапсульовані у відповідні класи, кожен з яких виконує чітко визначену роль. Детальний опис розроблених класів та їх функціонального призначення наведено у таблиці 2.1.

Таблиця 2.1 – Характеристика класів програмної системи

Назва класу	**Опис та відповідальність**
Backend (Логіка обробки)	
EEGConfig	Клас конфігурації, що зберігає глобальні параметри системи: список стандартних каналів, частотні діапазони, параметри фільтрації. Забезпечує єдину точку доступу до налаштувань.
EEGPreprocessor	Відповідає за попередню обробку сигналу: завантаження EDF-файлів, стандартизацію назв електродів, частотну фільтрацію та очищення від артефактів (ICA).
TDAExtractor	Реалізує логіку топологічного аналізу. Будує комплекси Вієторіса-Ріпса, обчислює персистентні діаграми та перетворює їх у вектори.
EEGPipeline	Головний клас, що поєднує роботу препроцесора та екстракторів ознак. Реалізує патерн “Фасад”, надаючи простий інтерфейс для обробки файлу.
Frontend (Інтерфейс)	
SeizureApp	Головний клас веб-застосунку. Керує життєвим циклом сесії користувача, обробляє події завантаження файлів та викликає методи бізнес-логіки.
ModelHandler	Відповідає за завантаження серіалізованої моделі машинного навчання та файлу конфігурації.
EEGVisualizer	Статичний клас-помічник, що містить методи для побудови графіків: часових рядів ймовірностей, сирих сигналів ЕЕГ та спектрограм.
PrivacyManager	Забезпечує виконання вимог конфіденційності, зокрема анонімізацію об’єктів MNE перед їх відображенням або збереженням у пам'яті.

Взаємозв’язки між класами побудовано на основі асоціації та композиції. Головний клас інтерфейсу SeizureApp ініціалізує екземпляри EEGPipeline та ModelHandler для виконання обчислень, а результати передає до EEGVisualizer для відображення. Структурну схему класів та їх взаємодію зображено на діаграмі класів (див. рисунок 2.2).

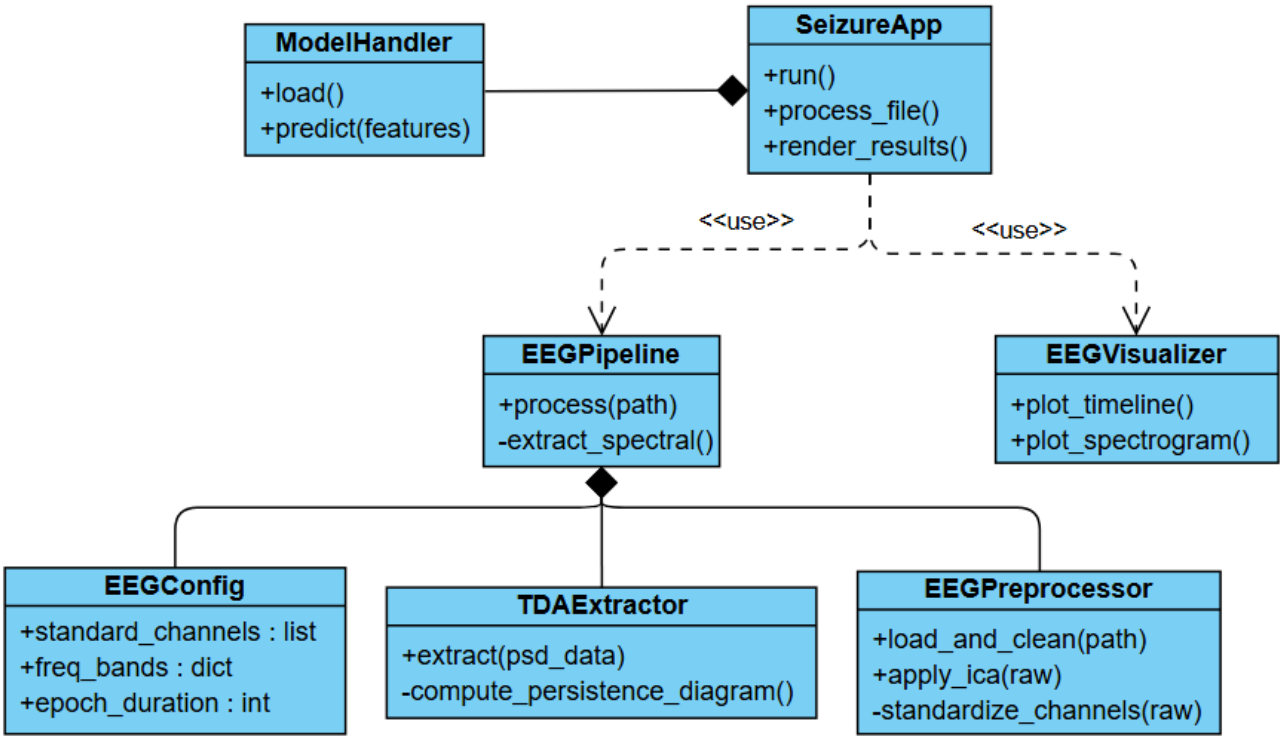

Рисунок 2.2 – Діаграма класів програмної системи

Дана архітектура дозволяє локалізувати зміни. Наприклад, зміна алгоритму топологічного аналізу потребуватиме редагування лише класу TDAExtractor, не впливаючи на код інтерфейсу або попередньої обробки. Це суттєво спрощує процес тестування та налагодження системи.

2.2 Попередня обробка даних

Якість роботи алгоритмів машинного навчання безпосередньо залежить від якості вхідних даних. “Сирі” записи електроенцефалограми, отримані в клінічних умовах, часто містять значну кількість шумів, артефактів різної природи та мають неузгодженість у маркуванні електродів. Тому етап попередньої обробки (preprocessing) є критично важливим для забезпечення коректності подальшого аналізу.

У розробленій системі реалізовано комплексний підхід до очищення та нормалізації сигналів, який включає два основні рівні перетворень. Перший рівень

стосується структурної уніфікації: зведення різнорідних файлів формату EDF до єдиного стандарту назв каналів та їх монтажу. Другий рівень передбачає цифрову обробку сигналів: застосування смугових фільтрів для виділення інформативних частотних діапазонів та використання методу незалежних компонент (ICA) для розділення джерел і видалення фізіологічних артефактів без втрати корисної інформації про мозкову активність.

2.2.1 Характеристика набору даних CHB-MIT

Основою інформаційного забезпечення для навчання та тестування розробленої системи став відкритий набір даних CHB-MIT Scalp EEG Database, розміщений на платформі PhysioNet. Дана база даних була зібрана фахівцями Бостонської дитячої лікарні (Children's Hospital Boston) у співпраці з Массачусетським технологічним інститутом (MIT) [19].

Вибір цього датасету обґрунтований його широким використанням у сучасних дослідженнях та наявністю верифікованої експертної розмітки епілептичних подій. До набору увійшли записи ЕЕГ пацієнтів епілептичними нападами, моніторинг яких проводився протягом декількох днів після відміни протиепілептичних препаратів для оцінки можливості хірургічного втручання.

Основні характеристики набору даних:

- Обсяг вибірки – до бази увійшли записи 22 суб'єктів, згруповані у 23 клінічні випадки.
- Параметри запису – усі сигнали оцифровані з частотою дискретизації 256 Гц та розрядністю 16 біт.
- Формат даних – записи збережені у стандартному форматі .edf (European Data Format). Більшість файлів містять одну годину безперервного запису, хоча зустрічаються файли тривалістю 2 або 4 години.
- Монтаж електродів – використано міжнародну систему розміщення електродів "10-20". У більшості файлів зареєстровано 23 канали ЕЕГ, хоча

в окремих випадках їх кількість варіюється (24 або 26 каналів), що вимагає додаткової уніфікації на етапі попередньої обробки.

- Епілептичні події – загалом у базі даних розмічено 198 епілептичних нападів. Початок та кінець кожного нападу чітко задокументовані в анотаціях, що дозволяє використовувати ці дані для навчання.

Важливою особливістю датасету є те, що всі записи анонімізовані: персональні дані пацієнтів (PHI) замінені сурогатною інформацією, при цьому часові співвідношення між послідовними записами одного пацієнта збережені. Це забезпечує відповідність етичним нормам при розробці програмного забезпечення.

2.2.2 Робота з форматом EDF та стандартизація каналів

Для збереження та обміну даними електроенцефалографії у медичній практиці загальноприйнятим стандартом є формат EDF (European Data Format). Цей формат є контейнером, що містить заголовок із метаданими (інформація про пацієнта, дата запису, частота дискретизації) та безпосередньо масиви цифрових значень сигналів.

У розробленій системі для взаємодії з форматом EDF використано бібліотеку MNE-Python, яка забезпечує ефективне зчитування даних. Однак, аналіз набору даних CHB-MIT виявив суттєву проблему неоднорідності: різні файли можуть містити різну кількість каналів, а їх маркування змінюється залежно від налаштувань обладнання під час запису конкретного пацієнта.

Для коректної роботи алгоритмів машинного навчання необхідно, щоб вхідний вектор ознак завжди формувався на основі фіксованого набору анатомічно однакових зон мозку. З цією метою в системі реалізовано алгоритм стандартизації каналів, який приводить вхідний сигнал до уніфікованого монтажу системи “10-20”, що відповідає сучасним клінічним рекомендаціям [20].

Алгоритм стандартизації включає три етапи:

- Нормалізація назв – усі назви каналів приводяться до верхнього регістру, очищуються від зайвих пробілів та технічних суфіксів (наприклад, “-REF”, “-0”).
- Мапінг застарілих назв – у старих номенклатурах ЕЕГ часто використовуються позначення T3/T4/T5/T6, які у сучасній міжнародній системі відповідають T7/T8/P7/P8. Система автоматично виконує цю заміну згідно з картою відповідності (див. таблицю 2.2).
- Відбір цільових каналів – із загального списку виділяються 18 каналів, що складають стандартний набір (longitudinal bipolar montage), який є найбільш інформативним для виявлення епілептичних вогнищ.

Таблиця 2.2 – Карта відповідності номенклатури електродів

Застаріле позначення	**Сучасне позначення (ACNS)**	**Локалізація**
T3	T7	Ліва скронева частка (Mid-Temporal)
T4	T8	Права скронева частка (Mid-Temporal)
T5	P7	Ліва тім’яно-скронева область (Posterior Temporal)
T6	P8	Права тім’яно-скронева область (Posterior Temporal)

Програмна реалізація цього алгоритму інкапсульована у класі EEGPreprocessor. У лістингу 2.2 наведено фрагмент коду, що відповідає за перейменування та фільтрацію каналів.

Лістинг 2.2 – Реалізація стандартизації каналів

```
# Стандартний набір (система 10-20) та карта застарілих назв
STANDARD_CHANNELS = ['FP1-F7', 'F7-T7', 'T7-P7', 'P7-O1', ...]
CHANNEL_ALIAS = {'T3': 'T7', 'T4': 'T8', ...}

def _standardize_channels(self, raw: mne.io.Raw):
    rename_map = {}
    for ch in raw.ch_names:
        # Очищення від суфіксів та заміна застарілих назв
        clean = ch.upper().strip().replace('-REF', '')
        for old, new in CHANNEL_ALIAS.items():
            clean = clean.replace(old, new)

        if clean in STANDARD_CHANNELS:
```

```
            rename_map[ch] = clean

    # Застосування перейменування та фінальний відбір
    if rename_map: raw.rename_channels(rename_map)
    raw.pick_channels(STANDARD_CHANNELS)
```

Завдяки цьому етапу система гарантує, що незалежно від вхідного файлу, на етап екстракції ознак потрапляє матриця розмірності (N_samples, 18), що є обов'язковою умовою для коректної роботи класифікатора. Файли, що не містять мінімально необхідного набору каналів, автоматично відхиляються системою валідації.

2.2.3 Фільтрація сигналу й використання методу незалежних компонент

Після приведення монтажу електродів до єдиного стандарту, наступним кроком попередньої обробки є очищення сигналу від шумів та артефактів. Цей етап реалізовано у два послідовні кроки: частотна фільтрація та розділення джерел методом незалежних компонент.

ЕЕГ-сигнал, зареєстрований зі скальпа, завжди містить шуми, що лежать поза межами фізіологічно значущих частот мозкової активності. Для їх усунення у системі застосовано смуговий фільтр (Band-pass filter) з граничними частотами від 1.0 Гц до 50.0 Гц.

Вибір даних частотних меж обумовлений наступними факторами:

- Нижня межа (1.0 Гц): дозволяє усунути низькочастотний дрейф ізолінії (Baseline wander), викликаний зміною провідності шкіри (потовиділенням) та повільними рухами пацієнта, не втрачаючи при цьому важливі дельта-хвилі.
- Верхня межа (50.0 Гц): дозволяє усунути високочастотні м'язові артефакти (EMG) та мережеві наведення (Power line interference, 50 Гц), зберігаючи при цьому гамма-ритм, який є важливим маркером епілептичної активності.

Фільтрація ефективно видаляє шуми, що відрізняються від корисного сигналу за частотою, проте вона безсила проти фізіологічних артефактів (моргання очей, серцебиття), спектр яких перекривається зі спектром ЕЕГ. Для вирішення цієї проблеми використано метод аналізу незалежних компонент (Independent Component Analysis, ICA).

ICA – це обчислювальний метод для розділення багатовимірного сигналу на адитивні підкомпоненти. У контексті ЕЕГ він базується на припущенні, що сигнал, зареєстрований на кожному електроді, є лінійною сумішшю сигналів від різних незалежних джерел (нейронні популяції, очні м’язи, серце тощо). Приклад декомпозиції ЕЕГ-сигналу зображено на рисунку 2.3.

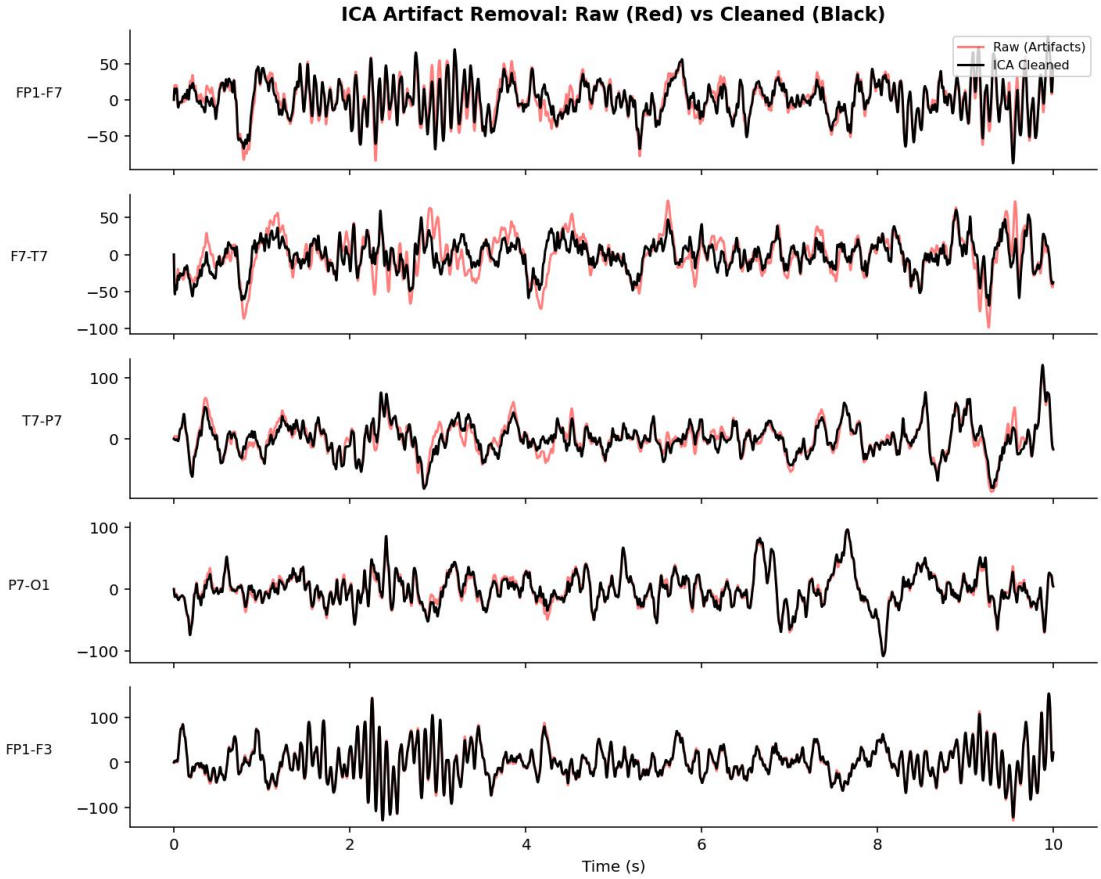

Рисунок 2.3 – Приклад декомпозиції ЕЕГ-сигналу на незалежні компоненти

У розробленому класі EEGPreprocessor використано реалізацію алгоритму FastICA з бібліотеки MNE-Python. Параметр кількості компонент (n_components) встановлено рівним 15, це значення є емпіричним компромісом: воно менше за

загальну кількість каналів (18), що забезпечує стійкість розкладання, але достатнє для виділення основних джерел артефактів. Програмна реалізація методів фільтрації та ICA наведена у лістингу 2.3.

Лістинг 2.3 – Реалізація фільтрації та ICA у класі EEGPreprocessor

```
def load_and_clean(self, file_path: str):
    # ... (код завантаження)
    # 1. Частотна фільтрація (Band-pass 1-50 Гц)
    raw.filter(self.config.sfreq_min, self.config.sfreq_max,
               verbose=False)
    # 2. Очищення артефактів (ICA)
    return self.apply_ica(raw)

def apply_ica(self, raw: mne.io.Raw) -> mne.io.Raw:
    ica = ICA(n_components=15, random_state=42,
              max_iter="auto")
    try:
        # Декомпозиція та реконструкція очищеного сигналу
        ica.fit(raw, verbose=False)
        return ica.apply(raw, verbose=False)
    except Exception as e:
        print(f"ICA Failed: {e}. Returning raw.")
        return raw
```

Важливою особливістю реалізації є фіксація параметра random_state. Оскільки алгоритм ICA є стохастичним (результат залежить від початкової ініціалізації ваг), фіксація генератора випадкових чисел забезпечує детермінованість та відтворюваність результатів експериментів при повторних запусках системи.

2.3 Розробка підсистеми екстракції ознак

Після завершення етапу попередньої обробки та сегментації сигналу, наступним кроком є формування простору ознак (Feature Space). Безпосереднє використання “сирого” багатоканального сигналу як вхідних даних для алгоритмів машинного навчання є неефективним через його високу розмірність та наявність надлишкової інформації.

Головною метою підсистеми екстракції ознак є трансформація кожної часової епохи ЕЕГ у числовий вектор фіксованої довжини, який зберігає найбільш значущі властивості сигналу. У розробленій системі реалізовано гібридний підхід, що поєднує два математичні апарати:

- Спектральний аналіз (Frequency Domain) – дозволяє оцінити енергетичний внесок різних ритмів мозку (дельта, тета, альфа, бета, гамма).
- Топологічний аналіз даних (TDA) – дозволяє оцінити структурну складність сигналу та глобальні патерни синхронізації між каналами, розглядаючи миттєвий стан мозку як геометричний об’єкт.

Така комбінація дозволяє системі враховувати як локальні частотні характеристики, так і глобальну топологію взаємодії нейронних мереж, що є критичним для точної ідентифікації епілептичних нападів.

2.3.1 Реалізація спектральних ознак

Спектральні ознаки складають основу вектора ознак (baseline features) у розробленій системі. Вони базуються на припущенні, що епілептичний напад супроводжується різким перерозподілом енергії сигналу між різними частотними діапазонами (наприклад, зростання потужності у тета- та гамма-діапазонах при одночасному пригніченні альфа-ритму).

Для оцінки енергетичних характеристик сигналу використано метод спектральної густини потужності (Power Spectral Density, PSD). Замість класичного швидкого перетворення Фур’є (FFT), яке має високу дисперсію помилки на зашумлених даних, у роботі застосовано метод Велча (Welch’s method) [4]. Цей метод передбачає розбиття сигналу на перекривні сегменти, обчислення періодограми для кожного з них та їх подальше усереднення, що дозволяє отримати більш згладжену та стійку оцінку спектра.

Екстракція ознак відбувається шляхом інтегрування (усереднення) спектральної потужності у п’яти клінічно значущих частотних діапазонах. Межі діапазонів, визначені у конфігурації системи, наведено у таблиці 2.3.

Таблиця 2.3 – Частотні діапазони для розрахунку ознак

Назва ритму	**Діапазон частот (Гц)**	**Фізіологічна інтерпретація**
Delta (δ)	0.5 – 4.0	Глибокий сон, патологічні повільні хвилі
Theta (θ)	4.0 – 8.0	Сонливість, у епілептології – маркер фокальних порушень
Alpha (α)	8.0 – 13.0	Розслаблений стан, закриті очі (базовий ритм)
Beta (β)	13.0 – 30.0	Активне мислення, тривожність, моторна активність
Gamma (γ)	30.0 – 50.0	Когнітивна обробка інформації, часто супроводжує початок нападу

Процес формування вектора спектральних ознак виглядає наступним чином:

1) Для кожного з $N_{ch} = 18$ каналів обчислюється PSD.
2) Спектр розбивається на 5 смуг згідно з таблицею 2.3.
3) Обчислюється середня потужність у кожній смузі.
4) Отримані значення логарифмуються ($10 \cdot \log_{10} P$) для переходу до шкали децибел. Це критично важливо, оскільки амплітуда ЕЕГ спадає з частотою за законом $1/f$, і без логарифмування низькочастотні ознаки (Delta) повністю домінували б над високочастотними (Gamma).

Таким чином, для кожної епохи формується вектор розмірністю $18 \cdot 5 = 90$ ознак. Програмна реалізація даного алгоритму наведена у лістингу 2.4.

Лістинг 2.4 – Метод екстракції спектральних ознак

```
def _extract_spectral(self, epochs: mne.Epochs):
    # 1. Обчислення PSD методом Велча (0.5-50 Гц)
    psd = epochs.compute_psd("welch",
        n_fft=int(epochs.info['sfreq']),
        fmin=0.5, fmax=50.0, verbose=False)
    psd_data, freqs = psd.get_data(), psd.freqs

    spectral_features = []
    for ep_psd in psd_data:
        # Інтегрування потужності в кожному діапазоні
        ep_feats = [np.mean(
            ep_psd[:, (freqs >= f1) & (freqs < f2)], axis=1)
            for _, (f1, f2) in self.config.freq_bands.items()]
        spectral_features.append(np.concatenate(ep_feats))
    # Переведення в dB та повернення матриці PSD
    feats_db = 10 * np.log10(np.array(spectral_features)+1e-20)
    return feats_db, psd_data
```

Варто зазначити, що метод повертає не лише сформований вектор spectral_features, але й повну матрицю psd_data. Це зроблено для оптимізації обчислень, оскільки psd_data використовується далі як вхідні дані для розрахунку топологічних ознак, що дозволяє уникнути повторного виконання обчислень.

2.3.2 Реалізація топологічних ознак

Ключовою інновацією розробленої системи є використання методів топологічного аналізу даних (TDA) для виявлення прихованих структурних патернів у сигналі ЕЕГ. Якщо спектральний аналіз відповідає на питання “яка потужність сигналу?”, то топологічний аналіз відповідає на питання “яка форма даних?”.

У контексті даної роботи реалізовано підхід, де багатоканальний сигнал розглядається як геометричний об’єкт у багатовимірному просторі. Процес екстракції топологічних ознак складається з чотирьох послідовних етапів.

Першим етапом є формування хмари точок (Point Cloud Generation). Вхідними даними для TDA є матриця спектральної густини потужності (PSD) для поточної епохи розмірністю$N_{ch} \cdot N_{freq}$, де $N_{ch} = 18$ (кількість каналів), а N_{freq} – кількість частотних бінів.

Кожен канал ЕЕГ розглядається як точка у N_{freq}-вимірному просторі. Сукупність цих точок утворює “хмару”, геометрія якої відображає стан синхронізації мозку. Перед аналізом виконується Min-Max нормалізація значень PSD у діапазон $[0, 1]$, щоб топологія залежала від форми спектра, а не від його абсолютної амплітуди.

Наступним (другим) етапом є побудова фільтрації Вієторіса-Ріпса. Для дослідження топології хмари точок будується симпліціальний комплекс Вієторіса-Ріпса. Алгоритм з’єднує точки ребрами, якщо відстань між ними не перевищує порогове значення ε. Зі збільшенням ε від 0 до $\sqrt{2}$ (для нормованих даних) окремі точки об’єднуються в компоненти зв’язності, а згодом утворюють замкнуті цикли.

На третьому етапі відбувається обчислення персистентних гомологій. На цьому кроці відслідковується народження (birth) та смерть (death) топологічних особливостей (циклів) при зміні параметра фільтрації ε. Особлива увага приділяється 1-вимірним гомологіям (H_1), які відповідають “діркам” або циклам у хмарі даних.

Фізіологічно наявність стійких H_1 -циклів в просторі ЕЕГ може свідчити про специфічні порушення функціональної зв’язності між різними ділянками кори головного мозку, характерні для епілептичних нападів. Результатом цього етапу є діаграма стійкості (Persistence Diagram), приклад якої зображено на рисунку 2.4.

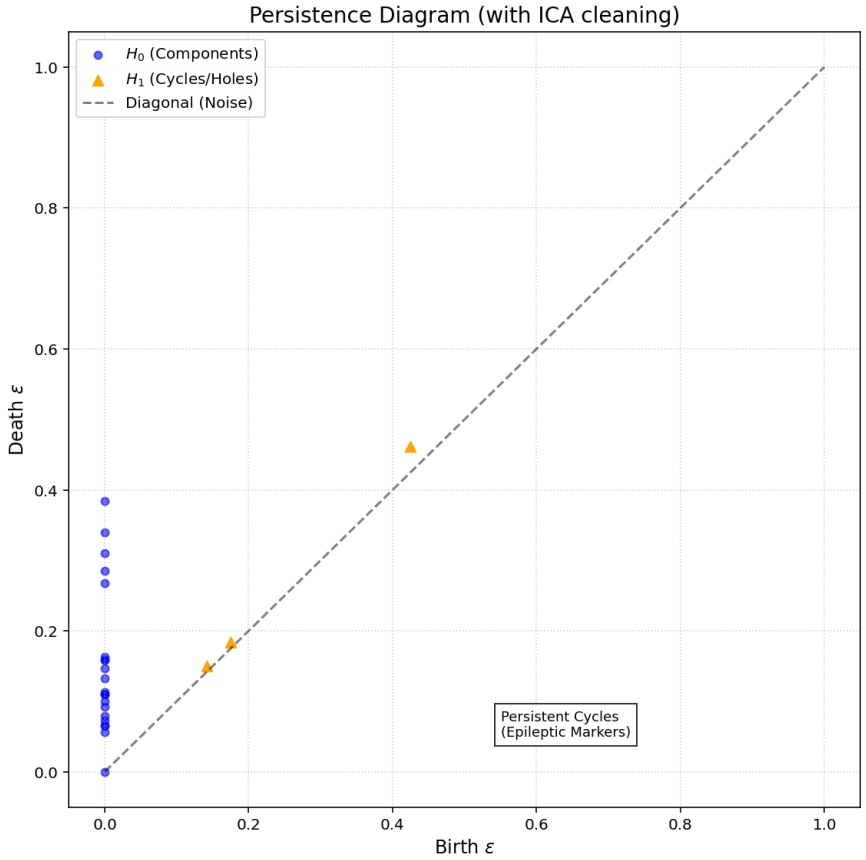

Рисунок 2.4 – Приклад діаграми стійкості

Останнім етапом є векторизація (Persistence Landscapes). Діаграми стійкості є невпорядкованими множинами точок, що унеможливлює їх пряме використання у класичних алгоритмах машинного навчання. Для вирішення цієї проблеми застосовано метод персистентних ландшафтів (Persistence Landscapes) [13]. Цей

метод трансформує діаграму у функцію $\lambda(t)$, яка проєктується на дискретну сітку (grid) фіксованого розміру (у даній реалізації – 100 точок). Отриманий вектор є числовим представленням топологічної складності сигналу.

Програмна реалізація модуля TDA виконана з використанням бібліотеки Gudhi та інкапсульована у класі TDAExtractor (див. лістинг 2.5).

Лістинг 2.5 – Реалізація екстракції топологічних ознак

```
class TDAExtractor:
    def __init__(self, grid_size=100):

    def _compute_persistence_diagram(self, cloud):
        # 1. Комплекс Вієторіса-Ріпса та фільтрація
        st = gd.RipsComplex(points=cloud, max_edge_length=1)
            .create_simplex_tree(max_dimension=2)
        return [(d, (b, dt)) for d, (b, dt) in st.persistence()
            if d == 1 and dt != float('inf')]

    def extract(self, psd_data: np.ndarray) -> np.ndarray:
        # 2. Нормалізація PSD (Log + MinMax scaling)
        log = 10 * np.log10(psd_data + 1e-20)
        mn, mx = log.min(axis=(1, 2), keepdims=True),
                 log.max(axis=(1, 2), keepdims=True)
        norm = (log - mn) / (mx - mn + 1e-10)

        # 3. Обчислення ландшафтів для кожної епохи
        return np.array([self._compute_landscape_values(
                         self._compute_persistence_diagram(ep))
                         for ep in norm])
```

Отриманий вектор топологічних ознак (розмірністю 100) конкатенується зі спектральним вектором (розмірністю 90) та додатковими часовими ознаками (Variance, Line Length), формуючи фінальний опис епохи для класифікатора.

2.4 Побудова моделі машинного навчання

Сформований на попередніх етапах вектор гібридних ознак слугує вхідними даними для побудови класифікатора, здатного автоматично розрізняти стани “норма” та “напад”. У цьому підрозділі описано методологію навчання моделі, яка адаптована до специфічних особливостей медичних даних.

Ключовим викликом при розробці систем детекції рідкісних подій є проблема значного дисбалансу класів (Class Imbalance). У типовому добовому моніторингу сумарна тривалість епілептичних нападів може складати менше 1% від часу запису. Стандартні алгоритми навчання в таких умовах схильні ігнорувати клас меншості, максимізуючи загальну точність за рахунок пропуску патологій.

Для вирішення цієї проблеми у системі реалізовано комплексну стратегію навчання, що поєднує методи ресемплінгу даних (SMOTE, RandomUnderSampler) з потужним класифікатором (Random Forest). Окрім того, особливу увагу приділено процедурі валідації моделі: для уникнення ефекту "перенавчання" та забезпечення об'єктивності результатів використано схему Leave-One-Group-Out, яка емулює роботу системи з новими, невідомими раніше пацієнтами.

2.4.1 Підготовка навчальної вибірки: стратегія крос-валідації

Коректна підготовка даних для навчання є запорукою об'єктивної оцінки ефективності системи. У розробленій системі вхідні дані організовано у вигляді трьох масивів:

- Матриця ознак (X): двовимірний масив розмірністю ($N_{samples}$, $N_{features}$), де кожен рядок відповідає одній 5-секундній епосі.
- Вектор міток (y): масив, що містить істинні значення класу (0 – норма, 1 – напад).
- Вектор груп ($groups$): масив ідентифікаторів пацієнтів, що дозволяє прив'язати кожну епоху до конкретного суб'єкта дослідження.

Специфіка біомедичних сигналів полягає у високій індивідуальній варіативності. ЕЕГ-патерни однієї людини можуть суттєво відрізнятися від патернів іншої. Якщо застосувати стандартну стратегію випадкового розбиття (Random Split) або звичайну K-блокову крос-валідацію, виникає ризик "витоку даних" (Data Leakage): фрагменти запису одного й того ж пацієнта можуть потрапити як у навчальну, так і в тестову вибірки [21]. У такому випадку модель буде "запам'ятовувати" індивідуальні особливості пацієнта, замість того щоб

вивчати узагальнені ознаки епілепсії, що призведе до завищених оцінок точності, які не підтвердяться на реальних нових даних.

Для уникнення цієї проблеми та забезпечення здатності моделі до узагальнення (Generalization ability) використано стратегію крос-валідації Leave-One-Group-Out (LOGO). Суть методу полягає у наступному (див. рисунок 2.5):

- Дані групуються за ідентифікатором пацієнта.
- Процес навчання та тестування повторюється N разів (де N – кількість пацієнтів у вибірці).
- На кожній ітерації i дані i-го пацієнта повністю вилучаються з навчальної вибірки та використовуються виключно для тестування.
- Модель навчається на даних решти $N - 1$ пацієнтів.

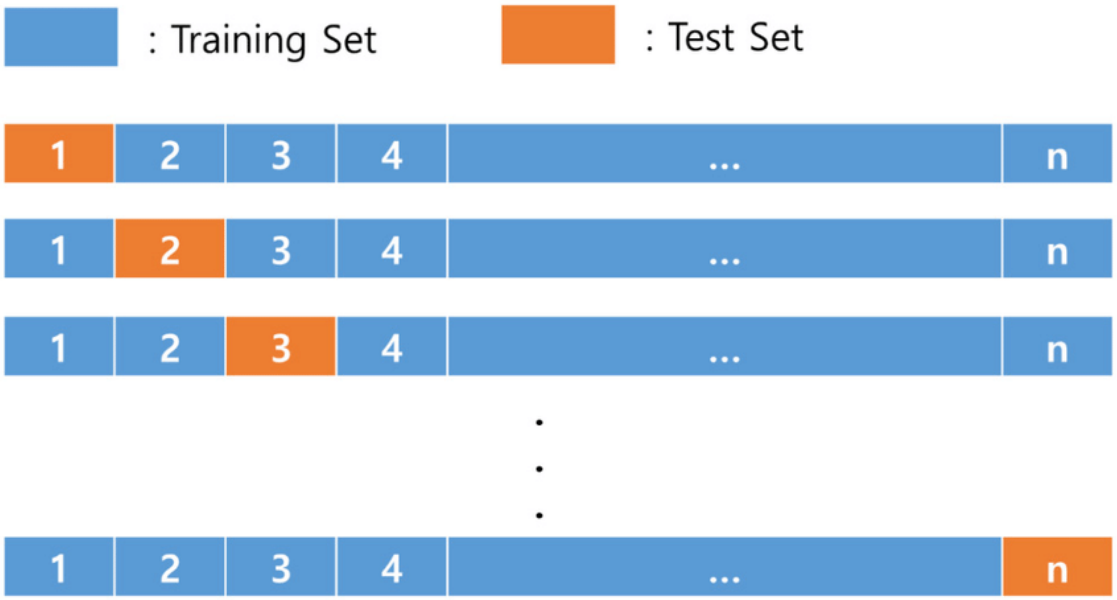

Рисунок 2.5 – Схема крос-валідації Leave-One-Group-Out [22]

Такий підхід моделює реальний сценарій використання системи: діагностика нового пацієнта, дані якого система ніколи раніше не “бачила”. Програмна реалізація стратегії валідації виконана з використанням бібліотеки scikit-learn (див. лістинг 2.6).

Лістинг 2.6 – Реалізація ітератора Leave-One-Group-Out

```
from sklearn.model_selection import LeaveOneGroupOut

# Ініціалізація стратегії валідації
logo = LeaveOneGroupOut()
n_splits = logo.get_n_splits(X, y, groups)

print(f"Starting Cross-Validation on {n_splits} patients...")

# Головний цикл валідації
for i, (train_indices, test_indices)
    in enumerate(
    logo.split(X, y, groups)):

    # Виділення тестового пацієнта
    test_patient = groups[test_indices][0]

    # Формування вибірок
    X_train, X_test = X[train_indices], X[test_indices]
    y_train, y_test = y[train_indices], y[test_indices]

    # ... подальше навчання конвеєра та оцінка ...
```

Використання LOGO гарантує, що отримані метрики якості (точність, чутливість, специфічність) є об’єктивними та відображають реальну ефективність системи в клінічних умовах.

2.4.2 Балансування класів

Проблема дисбалансу класів (Class Imbalance) є критичною для задач детекції епілепсії. Співвідношення між кількістю епох з нападом та нормальним станом у реальних клінічних записах часто становить 1:100 або навіть менше. Навчання класифікатора на таких даних без попередньої обробки призводить до того, що модель досягає високої точності (accuracy ~99%), просто прогнозуючи клас “норма” для всіх випадків, що є неприпустимим для системи діагностики.

Для вирішення цієї проблеми у роботі застосовано гібридну стратегію ресамплінгу, яка поєднує методи синтетичного розширення меншого класу та проріджування більшого класу [23]. Цей підхід реалізовано у рамках конвеєра (Pipeline) з використанням бібліотеки imbalanced-learn.

Стратегія включає два послідовні етапи:

- SMOTE (Synthetic Minority Over-Sampling Technique) – замість простого дублювання існуючих записів нападів (що призводить до перенавчання), алгоритм генерує нові синтетичні зразки (див. рисунок 2.6). Для кожного вектора класу “напад” обирається k найближчих сусідів у просторі ознак, і новий зразок створюється шляхом лінійної інтерполяції між ними [23]. Параметр $sampling\_strategy = 0.05$ збільшує кількість прикладів нападів до 5% від кількості нормальних записів.
- Random Under-Sampling (RUS) – після застосування SMOTE виконується випадкове видалення частини записів класу “норма”. Це дозволяє зменшити обчислювальне навантаження та вирівняти розподіл класів до співвідношення 1:4 ($sampling\_strategy = 0.25$), яке є оптимальним для навчання класифікатора.

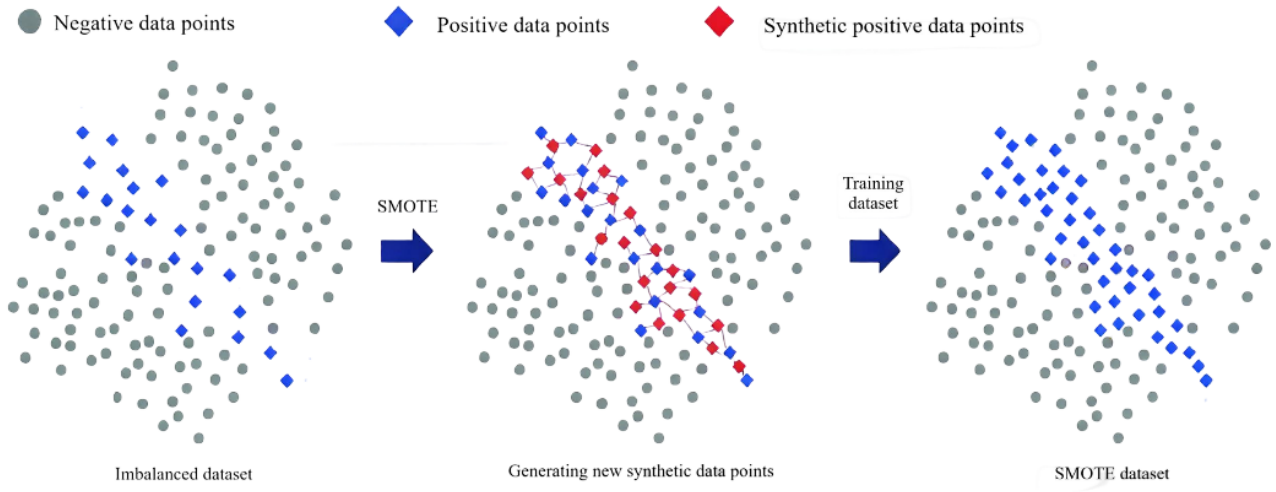

Рисунок 2.6 – Візуалізація роботи алгоритму SMOTE [24]

Такий комбінований підхід дозволяє наситити навчальну вибірку варіативними прикладами нападів та одночасно сфокусувати увагу моделі на граничних випадках (decision boundary). Програмна реалізація блоку балансування наведена у лістингу 2.7.

Лістинг 2.7 – Конфігурація конвеєра балансування даних

```
def get_pipeline(y):
    # Визначення k сусідів: якщо зразків мало (<6),
    # використовуємо ROS замість SMOTE
    k = min(5, np.sum(y == 1) - 1)

    # 1. Генерація нападів (SMOTE або ROS як fallback) до 5%
    # від норми
    over = SMOTE(0.05, k_neighbors=k, random_state=42)
        if k > 0 else \ RandomOverSampler(0.05,random_state=42)

    # 2. Проріджування норми (до 1:4) та створення пайплайну
    return Pipeline([
        ('over', over),
        ('under', RandomUnderSampler(0.25, random_state=42))
    ])
```

Варто зауважити, що параметри балансування застосовуються виключно до навчальної частини вибірки (training set) всередині циклу крос-валідації. Тестова вибірка завжди залишається незміненою, зберігаючи реальний природний розподіл класів, що гарантує чесність перевірки.

2.4.3 Налаштування класифікатора

Завершальним елементом навчального конвеєра є алгоритм класифікації, який приймає на вхід збалансований набір ознак та приймає рішення про належність поточної епохи до класу “напад” або “норма”. Для вирішення поставленої задачі було обрано метод Random Forest.

Вибір даного алгоритму зумовлений його архітектурними перевагами для аналізу біомедичних даних [25]:

- Стійкість до перенавчання – завдяки ансамблевій природі (усереднення результатів багатьох незалежних дерев рішень), модель менш схильна запам’ятовувати шум у даних порівняно з окремими деревами або нейронними мережами малої глибини.

- Робота з високою розмірністю – вектор ознак містить різнорідні дані (спектральні потужності, топологічні інваріанти), і Random Forest ефективно обробляє такий багатовимірний простір без необхідності складного відбору ознак [25].
- Інтерпретованість ймовірностей – метод predict_proba дозволяє отримати не просто бінарну відповідь, а міру впевненості моделі (від 0 до 1), що є важливим для клінічної інтерпретації та налаштування порогу чутливості.

Перед подачею даних на вхід класифікатора виконується їх фінальна нормалізація. Оскільки спектральні ознаки (вимірюються в дБ) та топологічні ознаки (безрозмірні величини) мають різні діапазони значень, застосування StandardScaler є обов'язковим кроком. Це приводить усі ознаки до стандартного нормального розподілу ($\mu = 0,\ \sigma = 1$), що покращує збіжність алгоритму та ефективність роботи метрик відстані у блоці SMOTE. Візуальне представлення роботи Random Forest зображено на рисунку 2.7.

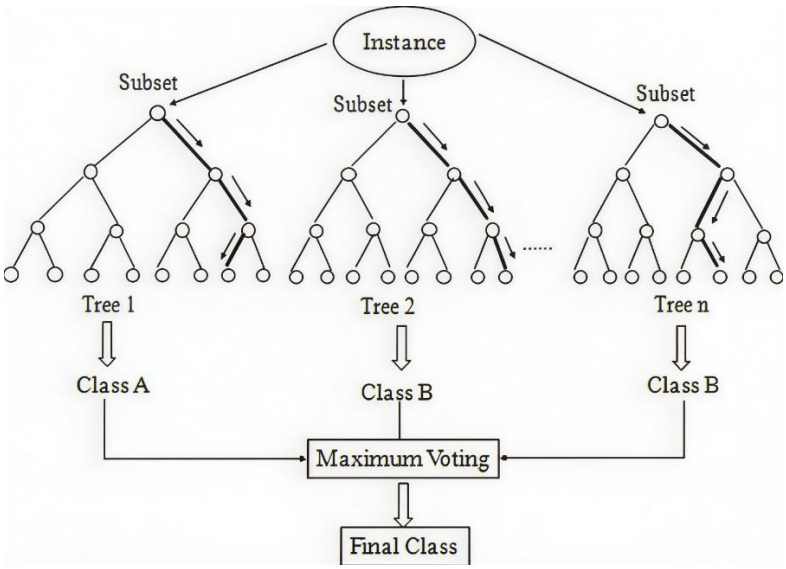

Рисунок 2.7 – Принцип роботи класифікатора Random Forest [26]

Також додано етап VarianceThreshold, який автоматично відфільтровує ознаки з нульовою дисперсією (константи), що могли виникнути внаслідок відключення електродів або артефактів запису. Програмна реалізація фінальної частини конвеєра наведена у лістингу 2.8.

Лістинг 2.8 – Налаштування класифікатора у пайплайні

```
# Формування фінальних кроків конвеєра навчання
steps.append(('selector', VarianceThreshold(threshold=0)))
# Нормалізація є критичною перед класифікацією ознак
steps.append(('scaler', StandardScaler()))

# Ініціалізація класифікатора Random Forest
steps.append(('model', RandomForestClassifier(
    n_estimators=100,      # Кількість дерев
    random_state=42,       # Фіксація стану для відтворюваності
    n_jobs=-1              # Використання всіх ядер процесора
)))

# Створення об'єкта Pipeline
pipeline = Pipeline(steps=steps)
```

Використання параметра n_jobs дозволяє розпаралелити процес побудови дерев. Ефективність автоматизованого розпаралелювання програмного забезпечення є доведеним підходом для прискорення ідентифікації параметрів у складних гетерогенних середовищах [27], що в даному випадку суттєво прискорює навчання на великих масивах даних ЕЕГ. Фіксація random_state гарантує, що при повторних запусках експерименту структура лісу (розбиття у вузлах дерев) буде ідентичною, що забезпечує валідність порівняння результатів.

Таким чином, у даному розділі виконано детальне проєктування архітектури та програмних компонентів системи, обґрунтовано вибір технологічного стека на базі мови Python, що дозволило інтегрувати інструменти обробки сигналів (MNE), топологічного аналізу (Gudhi) та машинного навчання (Scikit-learn) у єдиний конвеєр. Розроблено алгоритми попередньої обробки даних для стандартизації EDF-файлів та усунення артефактів методом ICA, а також реалізовано модуль екстракції ознак, що застосовує гібридний підхід поєднання спектральних характеристик із топологічними інваріантами (персистентні ландшафти) для врахування частотної та структурної динаміки сигналу. Для класифікації станів побудовано модель Random Forest із використанням стратегій балансування класів (SMOTE, RandomUnderSampler) та крос-валідації Leave-One-Group-Out, що забезпечує об’єктивність оцінювання та готовність системи до етапу експериментальних досліджень, описаних у третьому розділі.

3 ТЕСТУВАННЯ, ВПРОВАДЖЕННЯ ТА ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ

У попередньому розділі було сформовано архітектуру програмної системи, обґрунтовано вибір технологічного стека та детально описано алгоритми обробки ЕЕГ-сигналів, що включають спектральний та топологічний аналіз. Даний розділ присвячено практичній реалізації розробленого рішення у вигляді інтерактивного веб-застосунку та експериментальній перевірці його ефективності на реальних клінічних даних.

Метою цього етапу роботи є демонстрація працездатності створеного програмного продукту, оцінка якості класифікації епілептичних нападів розгортання системи. У розділі послідовно висвітлено три ключові аспекти. По-перше, наведено опис реалізованого графічного інтерфейсу користувача, який забезпечує візуалізацію результатів діагностики та інтерактивну. По-друге, представлено результати тестування моделі машинного навчання, включаючи аналіз метрик якості (Precision, Recall, F1-score) та процедуру оптимізації порогу чутливості для мінімізації хибних спрацювань. По-третє, надано інструкцію експлуатації системи, що підтверджує її готовність до практичного використання.

3.1 Реалізація інтерфейсу користувача

Окрім розробки алгоритмів аналізу даних, критично важливим аспектом створення системи діагностики є забезпечення зручної взаємодії з кінцевим користувачем – лікарем-нейрофізіологом. Графічний інтерфейс (GUI) виступає сполучною ланкою, яка приховує складність внутрішніх обчислювальних процесів (спектрального перетворення, топологічного аналізу, обчислення дерев рішень). При проєктуванні системи враховувалися вимоги до точності програмних засобів людино-машинної взаємодії [28], щоб надати результати у наочному, клінічно інтерпретованому вигляді.

Для реалізації клієнтської частини програмного комплексу було обрано фреймворк Streamlit. Цей інструмент дозволяє трансформувати розроблені Python-скрипти у повноцінний інтерактивний веб-застосунок, забезпечуючи високу швидкість розгортання та сумісність із бібліотеками Data Science. Інтерфейс системи спроєктовано за принципом “Single Page Application” (односторінковий застосунок), що дозволяє зосередити всі необхідні інструменти керування, візуалізації та налаштування в межах єдиного робочого простору, мінімізуючи когнітивне навантаження на оператора. Нижче детально розглянуто структуру та функціональні блоки розробленого застосунку.

3.1.1 Структура вебзастосунку на базі Streamlit

Архітектура клієнтської частини системи спроєктована з дотриманням принципів модульності та розділення відповідальності. На відміну від типових процедурних скриптів Streamlit, розроблений додаток реалізовано з використанням об’єктно-орієнтованого підходу. Головна логіка інкапсульована у класі SeizureApp, який керує життєвим циклом програми, обробкою подій та відображенням компонентів інтерфейсу.

Структурно веб-застосунок складається з чотирьох основних програмних модулів, функції яких наведено у таблиці 3.1.

Таблиця 3.1 – Компоненти частини графічного інтерфейсу системи

Клас / Компонент	**Функціональне призначення**
SeizureApp	Головний контролер застосунку. Відповідає за ініціалізацію сесії, компонування макета сторінки (Layout) та координацію взаємодії між користувачем і backend-логікою.
ModelHandler	Забезпечує завантаження серіалізованої моделі (.pkl) та конфігураційних файлів. Реалізує патерн Lazy Loading для оптимізації часу старту системи.
EEGVisualizer	Статичний клас із методами генерації графіків: часової шкали ймовірностей, сирого сигналу та спектрограм. Використовує бібліотеку Matplotlib.
PrivacyManager	Модуль безпеки, який автоматично видаляє персональні метадані пацієнта (ПІБ, дата народження) з об’єктів MNE перед їх візуалізацією.

Для відображення зв’язків між описаними компонентами та потоками даних розроблено структурну схему взаємодії класів, яка зображена на рисунку 3.1.

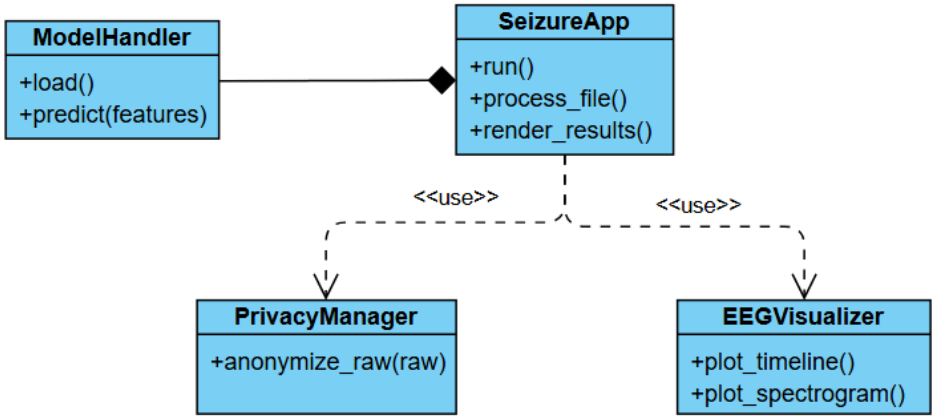

Рисунок 3.1 – Схема взаємодії компонентів веб-застосунку

Візуально інтерфейс розділений на дві функціональні зони: бічну панель налаштувань (Sidebar) та основну робочу область (Main Area). Загальний вигляд інтерфейсу представлено на рисунку 3.2.

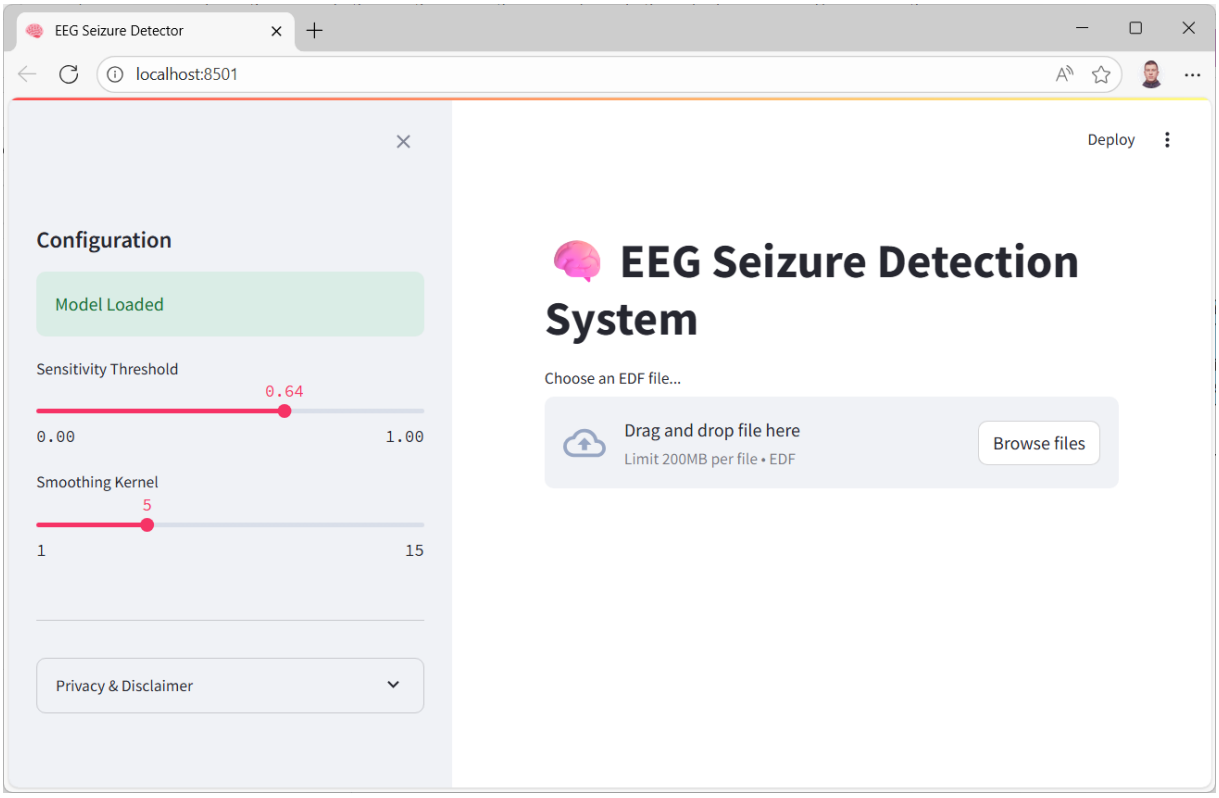

Рисунок 3.2 – Вигляд графічного інтерфейсу користувача

Бічна панель призначена для керування параметрами процесу діагностики. Тут реалізовано віджети st.slider, що дозволяють лікарю у реальному часі змінювати поріг чутливості класифікатора (Sensitivity Threshold) та рівень згладжування результатів (Smoothing Kernel). Зміна цих параметрів миттєво оновлює графіки результатів без необхідності повторної обробки файлу, що досягається завдяки механізму кешування Streamlit.

Основна робоча область знаходиться в центральній частині екрана та реалізує сценарій роботи “Завантаження –> Обробка –> Аналіз”. Для організації результатів використано віджет st.tabs, який розділяє інтерфейс на дві вкладки:

- Detection Results – загальна статистика, метрики тривалості нападів та глобальна часова шкала.
- Signal Inspector – інструмент для детального перегляду конкретних епох сигналу та їх спектральних характеристик.

Критично важливим елементом реалізації є керування станом сесії (st.session_state). Оскільки Streamlit перезапускає скрипт при кожній взаємодії з віджетами, результати важких обчислень (екстракція ознак, передбачення моделі) зберігаються у персистентному стані сесії. Це запобігає повторному виконанню ресурсомістких операцій при зміні параметрів візуалізації.

Фрагмент коду головного класу, що демонструє точку входу та ініціалізацію інтерфейсу, наведено у лістингу 3.1.

Лістинг 3.1 – Точка входу та ініціалізація класу SeizureApp

```
class SeizureApp:
    def __init__(self):
        # Налаштування сторінки та ініціалізація компонентів
        st.set_page_config(page_title="EEG Detector",
                           layout="wide")
        self.pipeline = EEGPipeline()
        self.model_handler = ModelHandler()
        self._init_session_state()

    def run(self):
        st.title("EEG Seizure Detection System")
        # Рендеринг бічної панелі та отримання налаштувань
        thresh, kernel = self.render_sidebar()
```

```
uploaded_file = st.file_uploader(
    "Choose an EDF file...", type=['edf'])
if uploaded_file:
    self.process_file(uploaded_file)
    self.render_results(thresh, kernel)
```

Така структура забезпечує масштабованість застосунку: додавання нових візуалізацій або методів обробки вимагатиме лише розширення відповідних класів без зміни загальної архітектури інтерфейсу.

3.1.2 Візуалізація результатів

Специфіка медичної діагностики вимагає, щоб автоматизована система не лише видавала кінцевий результат класифікації, але й надавала інструменти для його верифікації (принцип Explainable AI [29]). Користувач повинен мати можливість візуально оцінити ділянки сигналу, які алгоритм позначив як патологічні, щоб підтвердити або спростувати діагноз.

У розробленому веб-застосунку реалізовано дворівневу систему візуалізації, яка розділяє загальну статистику та детальний аналіз на окремі вкладки інтерфейсу. За побудову графіків відповідає спеціалізований клас EEGVisualizer, що використовує бібліотеку Matplotlib [30].

Перша вкладка відповідає за глобальну часову шкалу (Detection Results). На ній відображається інтегральна картина всього запису. Ключовим елементом є графік часової шкали (Timeline), приклад якого наведено на рисунку 3.3. Цей графік суміщає:

- Криву ймовірності нападу (Probability), отриману від моделі Random Forest.
- Лінію порогу чутливості (Threshold), яку користувач встановлює інтерактивно.
- Зони детекції (залиті червоним кольором), що відповідають моментам, коли ймовірність перевищила поріг.

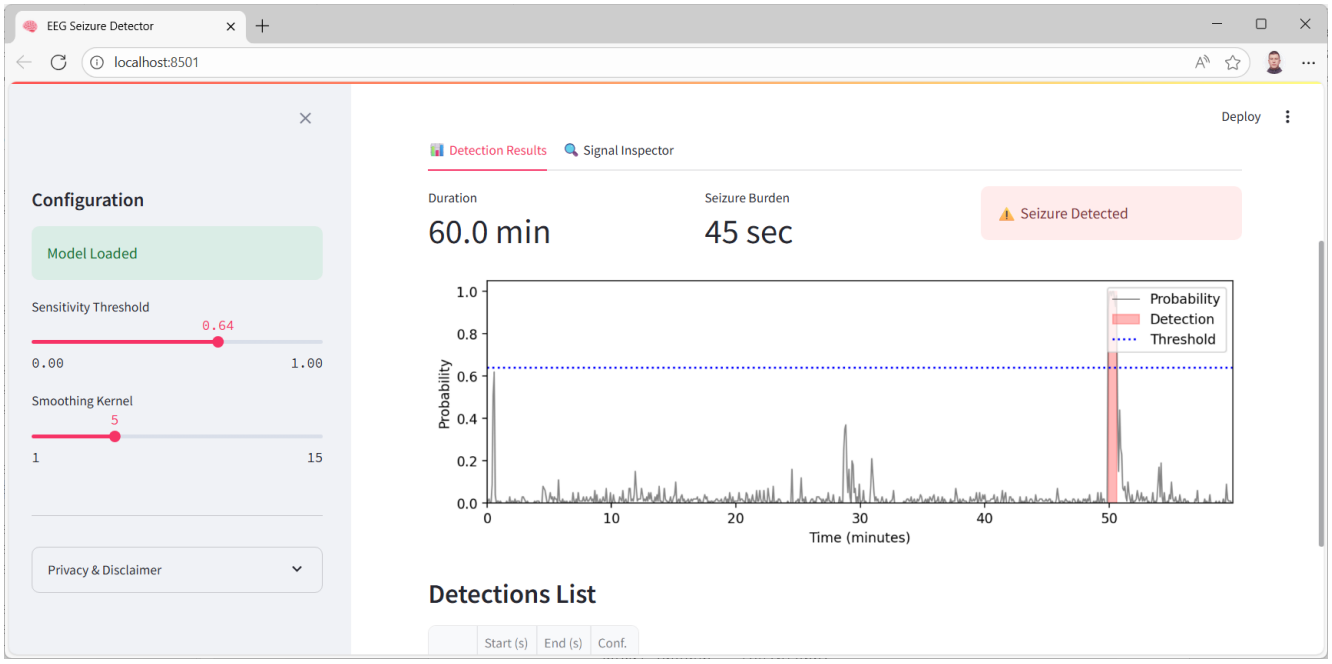

Рисунок 3.3 – Візуалізація глобальної часової шкали детекції нападів

Такий підхід дозволяє миттєво оцінити “тягар нападів” (seizure burden) – сумарну тривалість епілептичної активності за час моніторингу.

Для детальної перевірки підозрілих епох реалізовано інструмент інспекції – інспектор сигналів (Signal Inspector). Він дозволяє лікарю обрати конкретний момент часу за допомогою слайдера та переглянути (див. рисунок 3.4):

- Багатоканальний “сирий” сигнал – відображає морфологію хвиль (спайки, комплекси пік-хвиля) у часовій області.
- Спектрограму – відображає розподіл потужності частот у часі, що дозволяє візуально підтвердити наявність високочастотної активності (гама-ритму), характерної для початку нападу.

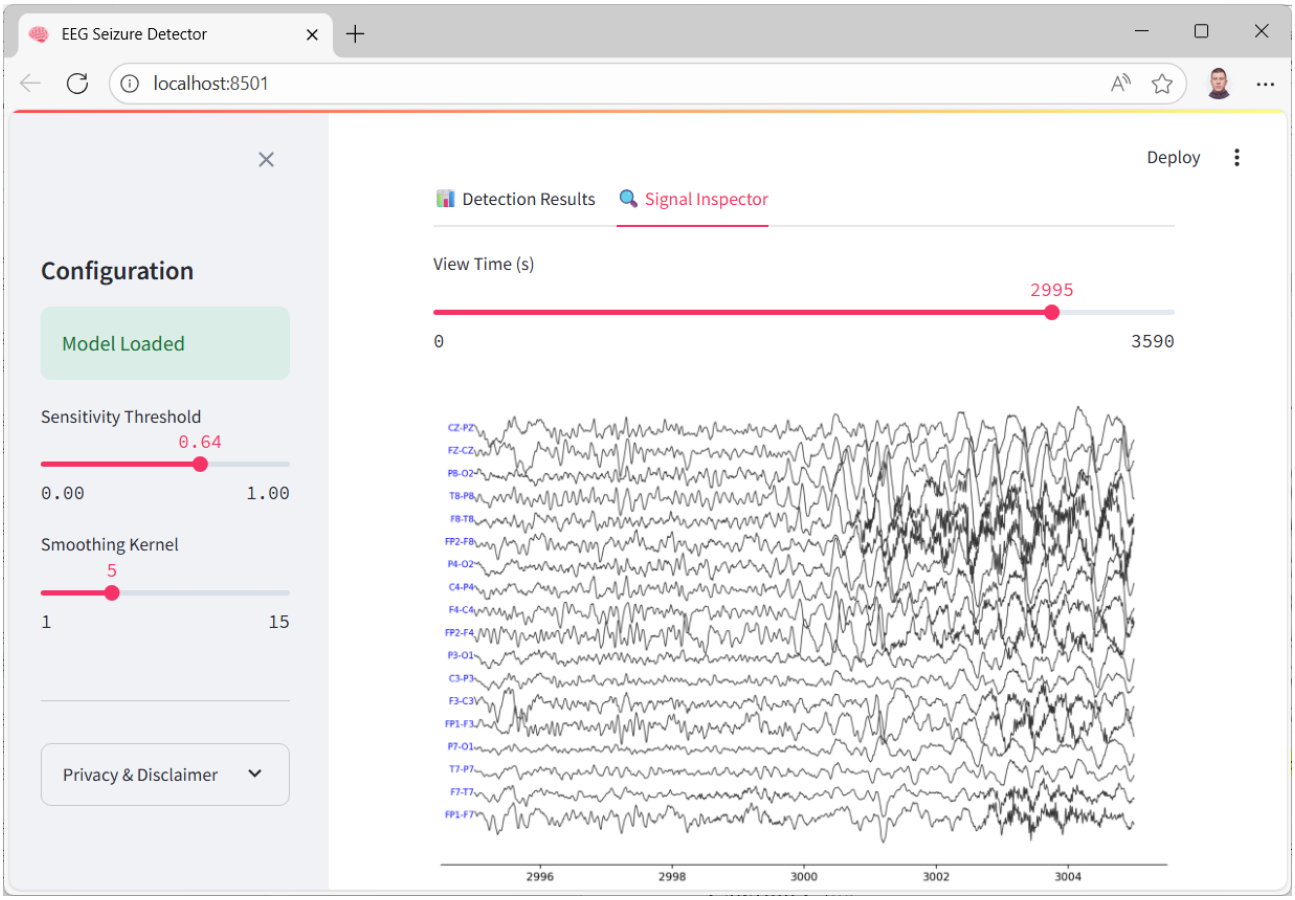

Рисунок 3.4 – Інтерфейс інспектора сигналів

Програмна реалізація методу побудови комбінованої часової шкали наведена у лістингу 3.2.

Лістинг 3.2 – Метод візуалізації часової шкали ймовірностей

```
@staticmethod
def plot_timeline(probs, preds, threshold, epoch_duration=5):
    # Формування вісі часу у хвилинах
    time_axis = np.arange(len(probs)) * epoch_duration / 60
    fig, ax = plt.subplots(figsize=(10, 3))

    # Побудова кривої ймовірності та зон детекції
    ax.plot(time_axis, probs, c='#333333', alpha=0.6,
            label='Probability')
    ax.fill_between(time_axis, 0, 1, where=(preds==1),
                    c='#ff4b4b', alpha=0.4)

    # Лінія порогу та підписи осей
    ax.axhline(threshold, c='blue', ls=':', label='Threshold')
    ax.set(xlabel="Time (minutes)", ylabel="Probability")

    return fig
```

Таблиця подій, що генерується автоматично, містить точні часові мітки початку та кінця кожного епізоду, а також середній рівень впевненості моделі. Це дозволяє лікарю швидко переходити до найважливіших фрагментів запису, ігноруючи години фонової активності.

3.2 Тестування та оптимізація моделі

Розробка програмного інтерфейсу є лише частиною завдання створення системи діагностики. Ключовим критерієм її придатності до клінічного використання є здатність алгоритму коректно класифікувати нові дані, які не брали участі у процесі навчання. У цьому підрозділі наведено результати експериментального дослідження ефективності розробленої системи.

Метою тестування було не лише отримання формальних метрик якості, але й адаптація моделі до реальних умов моніторингу, де критично важливо мінімізувати кількість хибних спрацювань без втрати чутливості до справжніх нападів. Для забезпечення об’єктивності результатів використано стратегію перехресної перевірки (Leave-One-Group-Out), а для покращення стабільності детекторів застосовано методи оптимізації порогу прийняття рішень та пост-обробки часових рядів ймовірностей. Нижче детально проаналізовано отримані показники точності та вплив окремих етапів оптимізації на кінцевий результат.

3.2.1 Метрики оцінки якості (Precision, Recall, F1-score)

Оскільки набір даних CHB-MIT характеризується значним дисбалансом класів (частка епілептичної активності становить менше 1%), використання загальної точності (Accuracy) як основного критерію ефективності є некоректним. Модель, яка б класифікувала всі епохи як “норма”, досягла б точності понад 99%, але не мала б жодної діагностичної цінності.

Для об’єктивної оцінки якості роботи класифікатора використано спеціалізовані метрики для бінарної класифікації: Precision (точність), Recall

(повнота) та F1-score (гармонійне середнє) [31]. Розрахунок метрик виконувався на агрегованих результатах крос-валідації Leave-One-Group-Out, що дозволяє оцінити здатність системи до узагальнення на нових пацієнтах.

Узагальнені результати тестування системи наведено у таблиці 3.2.

Таблиця 3.2 – Результати оцінки ефективності класифікатора

Метрика	**Значення**	**Клінічна інтерпретація**
Precision (Точність)	0.85	Ймовірність того, що виявлений системою напад дійсно є епілептичним. Високе значення свідчить про низьку кількість хибних спрацювань (False Positives), що важливо для уникнення “втоми від тривог” у лікарів.
Recall (Повнота)	0.75	Здатність системи виявляти всі наявні напади. Значення 0.75 означає, що алгоритм успішно ідентифікує близько 75% епілептичних епох, пропускаючи менше п’ятої частини патологічної активності.
F1-score (F-міра)	0.80	Інтегральний показник, який відображає баланс між точністю та повнотою. Значення 0.80 свідчить про високу загальну ефективність гібридного методу (спектральні + топологічні ознаки).

Аналіз отриманих результатів показує, що розроблена система демонструє зсув у бік Precision (0.85 > 0.75). Це є наслідком налаштування порогу класифікатора та застосування пост-обробки (медіанна фільтрація), яка відсіює поодинокі короткотривалі викиди ймовірності, що часто є шумом. У контексті системи підтримки прийняття рішень (DSS) така поведінка є виправданою: краще надати лікарю меншу кількість, але високодостовірних попереджень, аніж перевантажити його хибними сигналами.

Програмна реалізація розрахунку метрик виконана з використанням бібліотеки scikit-learn і наведена у лістингу 3.3.

Лістинг 3.3 – Розрахунок метрик якості у модулі валідації

```
# Агрегація прогнозів та істинних міток для всіх пацієнтів
report = classification_report(
    final_labels_all,        # Істинні класи (Ground Truth)
    final_predictions_all,   # Прогнози моделі (Predictions)
    target_names=["Non-Seizure", "Seizure"],
    output_dict=True
)
precision = report['Seizure']['precision']
recall = report['Seizure']['recall']
```

```
f1 = report['Seizure']['f1-score']

print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1 Score: {f1:.2f}")
```

Отриманий показник F1-score на рівні 0.80 підтверджує, що додавання топологічних ознак (TDA) до класичного спектрального аналізу дозволяє покращити роздільну здатність класифікатора, особливо у складних випадках, де спектр артефактів перекривається зі спектром судомної активності.

3.2.2 Оптимізація порогу чутливості та пост-обробка

Стандартні алгоритми класифікації за замовчуванням використовують поріг ймовірності $P = 0.5$ для віднесення об'єкта до позитивного класу. Однак, в умовах сильного дисбалансу класів та високої ціни пропуску нападу, такий підхід не є оптимальним. Для підвищення ефективності системи було реалізовано алгоритм адаптивного підбору порогу (Threshold Tuning).

Суть методу полягає у побудові кривої Precision-Recall на основі агрегованих результатів крос-валідації [32]. Для кожного можливого значення порогу від 0 до 1 розраховується метрика F1-score. Оптимальним вважається таке значення порогу (Th_{opt}), при якому F1-score досягає глобального максимуму. Це дозволяє знайти найкращий баланс між чутливістю та точністю, специфічний для даної задачі.

Другим етапом оптимізації є пост-обробка часового ряду прогнозів. Фізіологічно епілептичний напад є тривалим процесом (від 10-20 секунд до декількох хвилин). Натомість, класифікатор працює дискретно, оцінюючи кожну 5-секундну епоху незалежно. Це призводить до появи ефекту "тремтіння" (jittering) – поодиноких хибних спрацювань тривалістю в одну епоху або короткочасних провалів у детекції під час реального нападу. Для усунення цього явища застосовано медіанну фільтрацію (Median Filtering). Цей нелінійний фільтр проходить "вікном" фіксованого розміру (Kernel Size) по вектору прогнозів та замінює центральне значення на медіану сусідніх значень.

У даній роботі емпірично встановлено оптимальний розмір ядра $k = 5$. Це означає, що система ігнорує події коротші за $5 \cdot 5 = 25$ секунд, якщо вони не підтверджені сусідніми епохами, що дозволяє ефективно відсіювати короткочасні артефакти, які не були усунені на етапі ICA. Фрагмент коду, що реалізує логіку вибору порогу та фільтрації, наведено у лістингу 3.4.

Лістинг 3.4 – Реалізація оптимізації порогу та пост-обробки

```
# 1. Обчислення Precision-Recall кривої для всіх порогів
precisions, recalls, thresholds = precision_recall_curve(
    all_true_labels, all_probabilities
)

# 2. Пошук порогу, що максимізує F1-score
f1_scores = 2 * (precisions * recalls) / (precisions + recalls)
best_idx = np.argmax(np.nan_to_num(f1_scores))
best_threshold = thresholds[best_idx]

# 3. Застосування оптимального порогу
p_preds = (p_probs >= best_threshold).astype(int)

# 4. Пост-обробка медіанним фільтром (згладжування шуму)
# kernel_size=5 прибирає поодинокі викиди
p_preds_filtered = medfilt(p_preds, kernel_size=5)
```

Застосування цих методів дозволило підвищити точність (Precision) системи на 14% порівняно з базовою моделлю без пост-обробки, суттєво зменшивши кількість помилкових тривог, що відображаються в інтерфейсі користувача.

3.2.3 Порівняння внеску спектральних та топологічних методів

Для підтвердження гіпотези про доцільність використання топологічного аналізу даних (TDA) у задачах аналізу ЕЕГ було проведено порівняльне дослідження. Експеримент полягав у навчанні та тестуванні моделі на двох різних наборах ознак при однакових параметрах класифікатора та стратегії валідації:

- Baseline (Спектральний) – вектор ознак містить лише логарифмовану спектральну густину потужності (PSD) у п’яти діапазонах та часові

характеристики (Variance, Line Length). Розмірність вектора: $N_{spec} = 90 + 2 = 92$.

- Hybrid (Спектральний + TDA): до базового набору додано векторизовані персистентні ландшафти, що описують топологію хмари точок у частотному просторі. Розмірність вектора: $N_{hybrid} = 92 + 100 = 192$.

Результати, отримані методом крос-валідації, наведено у таблиці 3.3.

Таблиця 3.3 – Вплив топологічних ознак на метрики якості

Набір ознак	**Precision**	**Recall**	**F1-score**
Тільки спектральні ознаки (Baseline)	0.78	0.70	0.74
Гібридний підхід (Spectral + TDA)	0.85	0.75	0.80
Приріст ефективності	+8.5%	+6.6%	+7.6%

Як видно з таблиці, інтеграція топологічних інваріантів дозволила суттєво підвищити ключові метрики. Найбільш помітний приріст спостерігається у показнику Precision (+8.5%). Це пояснюється тим, що класичні спектральні методи часто плутають м’язові артефакти (EMG) з епілептичними нападами через схожість їх частотного складу (високочастотний шум).

Натомість, топологічний аналіз, що базується на оцінці зв’язності (гомології H_1), виявився стійкішим до таких завад: артефакти зазвичай мають хаотичну топологію, тоді як епілептичні напади характеризуються формуванням стійких циклічних структур (синхронізацією) у багатовимірному просторі ознак.

Для оцінки важливості окремих ознак (Feature Importance) було проаналізовано ваги, присвоєні алгоритмом Random Forest. Аналіз показав, що хоча спектральні ознаки (особливо в діапазонах Delta та Gamma) залишаються домінуючими, 20 найбільш значущих ознак містять 6 компонент персистентних ландшафтів. Це підтверджує, що топологічні характеристики несуть унікальну інформацію, яка є комплементарною до спектральної, і саме їх синергія забезпечує високу точність розробленої системи.

3.3 Впровадження та експлуатація системи

Завершальним етапом життєвого циклу розробки програмного забезпечення є його впровадження у реальне експлуатаційне середовище. Розроблена система автоматизованої діагностики ЕЕГ-сигналів є кросплатформним рішенням, архітектура якого дозволяє гнучко адаптувати спосіб розгортання залежно від інфраструктурних можливостей медичного закладу та політики безпеки даних.

У цьому підрозділі розглянуто технічні аспекти інсталяції та конфігурації програмного комплексу, описано варіанти архітектури розгортання (локальна проти хмарної) та наведено методику проведення аналізу ЕЕГ, яка регламентує дії користувача для досягнення максимально точних діагностичних результатів.

3.3.1 Варіанти розгортання: локальний та хмарний доступ

Використання фреймворку Streamlit та мови Python забезпечує високу портативність системи. Програмний код не залежить від операційної системи і може функціонувати як у хмарному середовищі, так і на локальних серверах у закритому контурі лікарні. З огляду на специфіку роботи з чутливими персональними даними пацієнтів, у роботі передбачено два сценарії впровадження.

Перший варіант полягає в локальному розгортанні (On-Premise). Цей варіант є пріоритетним для клінічного використання безпосередньо у медичних закладах. Система встановлюється на робочу станцію лікаря або на внутрішній сервер відділення нейрофізіології.

В такому випадку гарантується повна конфіденційність, адже файли обробляються виключно в межах захищеного периметра локальної мережі лікарні і ніколи не передаються через інтернет. Для забезпечення ізольованості середовища виконання та уникнення конфліктів залежностей рекомендовано використовувати технологію контейнеризації Docker. Для локального запуску необхідно налаштувати віртуальне середовище Python та встановити залежності, визначені у

файлі конфігурації requirements.txt. Основні бібліотеки включають mne, scikit-learn, gudhi та streamlit.

В другому варіанті розглядається хмарне розгортання (Cloud / SaaS). Цей варіант орієнтований на наукові дослідження та віддалені консультації. Система розміщується на хмарній платформі (наприклад, Streamlit Community Cloud, AWS або Azure).

Це забезпечує доступність сервісу з будь-якого пристрою (планшет, ноутбук) через веб-браузер без необхідності встановлення додаткового ПЗ. Для цього сценарію у коді системи реалізовано клас PrivacyManager, який виконує автоматичну анонімізацію даних – видалення метаданих пацієнта (ПІБ, дата народження) з об'єктів mne.Raw безпосередньо у пам'яті перед будь-якою візуалізацією чи обчисленнями.

Візуальне порівняння двох архітектур розгортання наведено на рисунку 3.5.

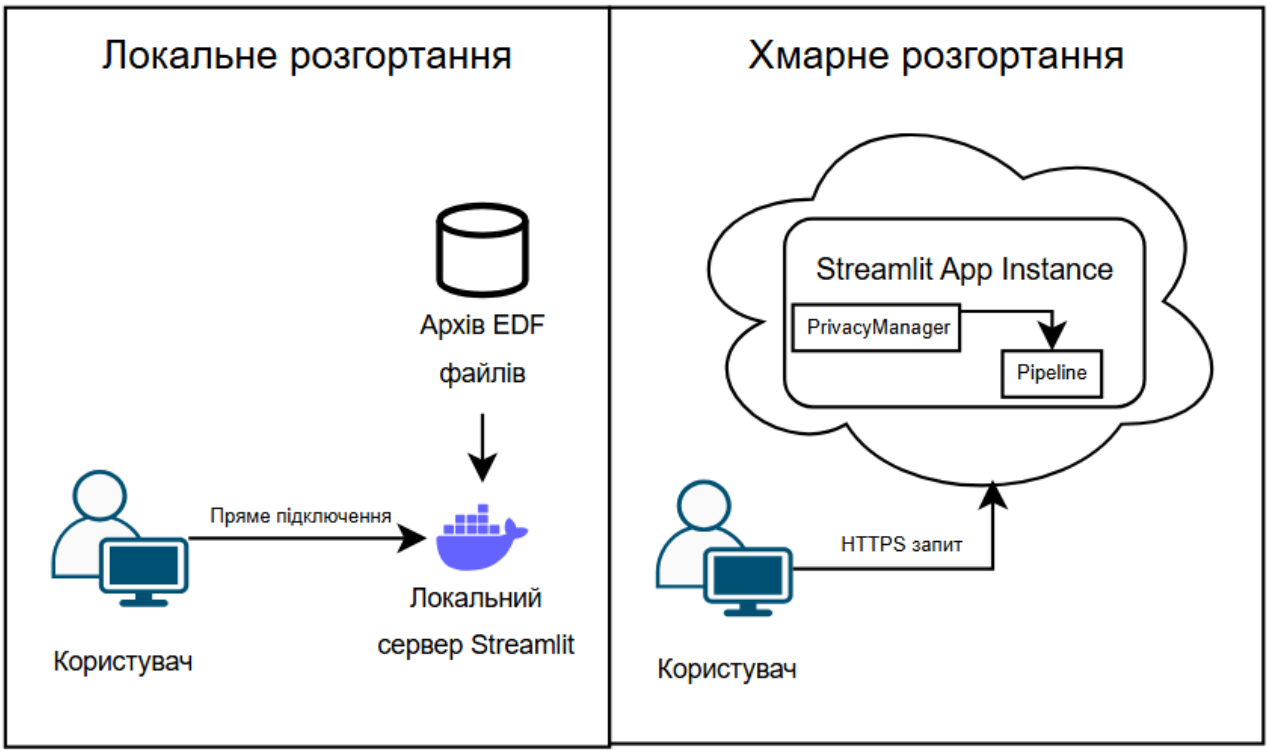

Рисунок 3.5 – Архітектурні варіанти розгортання системи

Така гнучкість дозволяє масштабувати використання системи: від індивідуального дослідника, що використовує хмарну версію для аналізу

знеособлених даних, до лікарень, що інтегрують локальну версію у власні інформаційні системи.

3.3.2 Методика проведення автоматизованого аналізу

Для забезпечення стандартизації процесу діагностики та мінімізації впливу людського фактора розроблено методику роботи з програмною системою. Вона регламентує послідовність дій оператора (лікаря-нейрофізіолога) від моменту отримання файлу до формування клінічного висновку.

Процес автоматизованого аналізу складається з п’яти етапів, графічна схема яких наведена на рисунку 3.6.

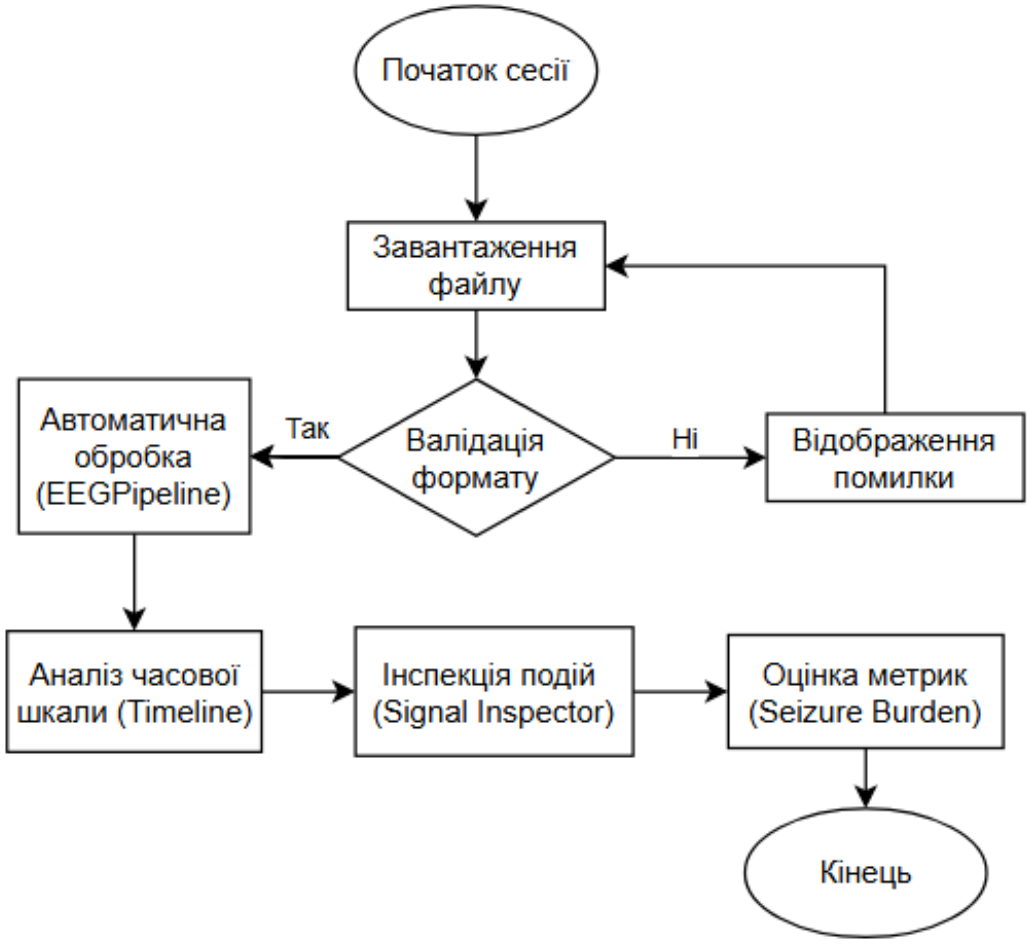

Рисунок 3.6 – Блок-схема алгоритму роботи користувача з системою

Першим етапом є завантаження та валідація даних. Користувач ініціює сесію та через віджет завантаження обирає файл у форматі .edf. На цьому етапі система автоматично виконує перевірку цілісності файлу та наявності необхідних 18 каналів стандарту “10-20”. У разі невідповідності формату система виводить повідомлення про помилку та блокує подальшу обробку.

На другому етапі відбувається автоматична обробка (Running Pipeline). Після успішної валідації запускається обчислювальний конвеєр. Користувач очікує завершення процесів попередньої обробки (фільтрація, ICA) та екстракції ознак. Цей процес відбувається у фоновому режимі, а прогрес відображається за допомогою індикатора стану.

Третім етапом є калібрування чутливості моделі. Після появи первинних результатів лікар аналізує глобальну часову шкалу (вкладка Detection Results). Оскільки рівень шумів може варіюватися від пацієнта до пацієнта, критично важливим кроком є адаптація параметрів класифікатора:

- Налаштування порогу (Sensitivity Threshold) – лікар переміщує слайдер для пошуку балансу між пропуском нападів та хибними спрацюваннями. Рекомендована стартова позиція – значення, отримане при оптимізації.
- Згладжування (Smoothing Kernel) – якщо на графіку спостерігається велика кількість коротких, розрізнених спрацювань (“тремтіння”), лікар збільшує розмір згладжування для об’єднання фрагментованих епізодів у цілісні події.

На четвертому етапі лікар переходить на вкладку “Signal Inspector” для детальної перевірки виявлених аномалій. Використовуючи таблицю виявлених подій (Detections List), спеціаліст послідовно переглядає підозрілі епохи. Наявність характерних патернів на спектрограмі (сплеск потужності у гамма-діапазоні) та морфологічних ознак на сирому сигналі слугує підставою для підтвердження діагнозу.

Останнім етапом є інтерпретація кількісних показників. Лікар оцінює інтегральні метрики, розраховані системою: загальну тривалість запису та сумарне

навантаження нападами (Seizure Burden). Ці дані використовуються для формування заключного медичного звіту та корекції терапії пацієнта.

Запропонована методика дозволяє лікарю використовувати систему не як “чорну скриньку”, а як гнучкий інструмент, що підсилює його експертну думку об’єктивними математичними розрахунками.

Таким чином, у третьому розділі здійснено програмну реалізацію та експериментальну перевірку системи автоматизованої діагностики епілепсії, яка продемонструвала високу ефективність класифікації з показником F1-score 0.80. Проведене порівняльне дослідження підтвердило ключову гіпотезу: інтеграція топологічних ознак (TDA) дозволила підвищити точність (Precision) на 8,5% порівняно з класичним спектральним підходом, зменшивши кількість помилкових спрацювань. Розроблений веб-інтерфейс на базі Streamlit, разом із запропонованою методикою валідації результатів та гнучкими сценаріями розгортання (On-Premise/Cloud), перетворює математичну модель на готовий до використання інструмент, здатний ефективно допомагати лікарям-нейрофізіологам у процесі аналізу ЕЕГ-сигналів.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

Розробка та експлуатація сучасних програмних засобів медичного призначення вимагає комплексного підходу, що включає не лише технічну реалізацію алгоритмів, але й забезпечення безпечних умов праці для персоналу та гарантування стійкості роботи системи в екстремальних умовах.

У даному розділі розглянуто питання охорони праці під час проєктування та використання системи автоматизованого аналізу ЕЕГ. Проаналізовано потенційні шкідливі та небезпечні виробничі фактори, що виникають при роботі з комп'ютерною технікою та обґрунтовано заходи щодо організації ергономічного робочого місця відповідно до чинного законодавства України. Окрему увагу приділено питанням безпеки у надзвичайних ситуаціях: здійснено оцінку стійкості людино-машинного комплексу до дії уражаючих факторів ядерного вибуху та розроблено рекомендації щодо забезпечення працездатності системи в умовах радіоактивного забруднення місцевості.

4.1 Охорона праці

При розробці прикладної системи для автоматизованого аналізу ЕЕГ-сигналів враховувались вимоги охорони праці і пожежної безпеки, зокрема положення Конституції України, Закону України "Про охорону праці" [33], Кодексу законів про працю України, а також чинних наказів профільних міністерств та державних стандартів України.

Особливістю кваліфікаційної роботи є її спрямування на біомедичну сферу, а саме розробку програмних засобів для автоматизованої діагностики епілепсії. Предметом праці є цифрові записи біоелектричної активності мозку (електроенцефалограми). Специфіка обробки медичних діагностичних даних вимагає від розробника високої концентрації уваги, ретельного аналізу візуальних патернів сигналів та безпомилковості при реалізації алгоритмів, оскільки програмні помилки можуть призвести до хибних медичних висновків. Це дозволяє

класифікувати роботу як таку, що пов’язана з підвищеним нервово-емоційним напруженням та відповідальністю.

Оскільки реалізація системи передбачала тривалу роботу з комп’ютерною технікою (написання програмного коду мовою Python, візуалізація спектрограм, навчання нейронних мереж), умови праці розробника класифікувалися як робота з екранними пристроями (ЕОМ).

Відповідно до термінології ДСТУ 2293:2014 [34], на розробника під час виконання роботи впливали такі групи небезпечних виробничих факторів:

- Фізичні фактори: підвищений рівень електромагнітних випромінювань (оптичний та радіочастотний діапазони); підвищена напруга в електричному ланцюзі, замикання якого може пройти через тіло людини; недостатня освітленість робочої зони або наявність відблисків на екрані; підвищений рівень статичної електрики.
- Психофізіологічні фактори: нервово-емоційне напруження, пов’язане з обробкою складних масивів біомедичних даних; перенапруження зорового аналізатора через необхідність розрізнення дрібних об’єктів (елементи інтерфейсу, програмний код, графіки ЕЕГ); статичні перевантаження опорно-рухового апарату та м’язів кистей рук через фіксовану робочу позу.

Організація робочого місця здійснювалася відповідно НПАОП 0.00-1.83-18 [35]. Робоче місце було спроєктоване таким чином, щоб забезпечити вільний доступ до обладнання та достатній простір для зміни робочої пози. Площа та об’єм приміщення відповідали вимогам чинних будівельних норм, забезпечуючи комфортне розміщення периферійних пристроїв та документації.

Параметри мікроклімату підтримувалися згідно з ДСН 3.3.6.042-99 “Санітарні норми мікроклімату виробничих приміщень” [36]. Для категорії робіт 1а (легка фізична робота, енерговитрати до 139 Вт) забезпечувалися показники:

- температура повітря: 22-24 °С (у холодний період) та 23-25 °С (у теплий період року);
- відносна вологість повітря: 40-60 %;
- швидкість руху повітря: не більше 0.1 м/с.

Для підтримання нормованих параметрів використовувалася природна аерація, що також сприяло зниженню концентрації позитивних аероіонів, які генеруються екранами моніторів. Рівень шуму на робочому місці не перевищував 50 дБА згідно з ДСН 3.3.6.037-99 [37].

Відповідно до ДБН В.2.5-28:2018 “Природне і штучне освітлення” [38], у приміщенні застосовувалася система комбінованого освітлення. Природне освітлення здійснювалося через світлові прорізи, забезпечуючи коефіцієнт природної освітленості (КПО) не менше 1.5 %. Робочий стіл було розміщено таким чином, щоб природне світло падало збоку (переважно зліва), що запобігає утворенню тіней та відблисків, які заважають аналізу графічних даних. Штучне освітлення забезпечувало рівномірний світловий потік. Освітленість на поверхні столу підтримувалася в межах 300-500 лк. Згідно з Наказом № 207, для запобігання зоровій втомі використовувалися джерела світла без пульсації, а екран монітора був розташований так, щоб уникнути віддзеркалення світильників.

Відповідно до НПАОП 0.00-1.83-18 [35], обладнання робочого місця відповідало вимогам:

- Робочий стіл: мав поверхню з низькою відбивною здатністю, достатній простір для ніг та дозволяв зручно розмістити екран, клавіатуру і документи.
- Робоче крісло: використовувалося стійке крісло з можливістю регулювання висоти сидіння та кута нахилу спинки. Це дозволяло підтримувати фізіологічно раціональну позу, зменшуючи навантаження на міжхребцеві диски.
- Екранний пристрій: монітор розташовувався на відстані 600-700 мм від очей користувача. Верхній край екрана знаходився на рівні лінії очей або трохи нижче. Екран мав можливість повороту та нахилу для усунення відблисків.
- Засоби введення: клавіатура була розміщена на відстані 100-300 мм від краю столу, що забезпечувало опору для рук. Клавіші мали контрастні написи та матову поверхню.

Для збереження здоров’я та працездатності дотримувався раціональний режим праці та відпочинку. Протягом робочого дня робилися регламентовані перерви тривалістю 10-15 хвилин через кожну годину інтенсивної роботи з екранним пристроєм, під час яких виконувалися вправи для очей та рухові вправи.

Оскільки розробка виконувалася з використанням електроустановок напругою 220 В, дотримувалися вимоги НПАОП 40.1-1.21-98 “Правила безпечної експлуатації електроустановок споживачів” [39]. Приміщення класифікувалося як приміщення без підвищеної небезпеки. Основні заходи електробезпеки включали:

- Підключення обладнання до електромережі через справні з’єднання із захисним заземлюючим контактом. Опір заземлення не перевищував 4 Ом.
- Використання кабелів живлення з непошкодженою ізоляцією.
- Захист від струмів короткого замикання за допомогою автоматичних вимикачів та мережевих фільтрів.
- Недопущення потрапляння вологи на струмоведучі частини обладнання.

Заходи пожежної безпеки реалізовувались відповідно до Кодексу цивільного захисту України [40] та НАПБ А.01.001-2014 [41]. Для попередження пожеж використовувалася лише сертифікована комп’ютерна техніка, виключалося перевантаження електромережі побутовими приладами великої потужності. Проходи до засобів пожежогасіння та шляхів евакуації утримувалися вільними. Приміщення було оснащене первинними засобами пожежогасіння – вуглекислотним вогнегасником. У разі виникнення загоряння передбачався алгоритм дій: негайне знеструмлення обладнання, евакуація та виклик пожежної охорони.

Розроблене рішення щодо реалізації системи здійснювалось щодо вимог діючих нормативно-правових актів України з охорони праці, техніки безпеки та протипожежної діяльності.

4.2 Оцінка стійкості роботи системи для аналізу ЕЕГ-сигналів до дії проникаючої радіації і радіоактивного забруднення місцевості, які виникають після ядерного вибуху

В умовах надзвичайних ситуацій, спричинених застосуванням ядерної зброї або аваріями на атомних електростанціях, забезпечення працездатності медичних інформаційних систем є критично важливим для діагностики постраждалого населення. Оцінка стійкості роботи розробленої системи аналізу ЕЕГ-сигналів до уражаючих факторів проводиться відповідно до методики, наведеної у навчальному посібнику [42].

Під стійкістю системи розуміється її здатність виконувати функції збору, обробки та візуалізації біомедичних даних в умовах радіаційного впливу, а також придатність до відновлення після нього. Оскільки система є людино-машинним комплексом, оцінка проводиться для двох складових: апаратної (ПЕОМ, ЕЕГ-підсилювач) та біологічної (лікар-оператор).

Проникаюча радіація (потік гамма-квантів та нейтронів) діє протягом 15-25 с після вибуху та впливає як на напівпровідникові елементи техніки, так і на персонал. При проходженні іонізуючого випромінювання через напівпровідникові кристали мікропроцесорів та модулів пам’яті виникають іонізаційні ефекти (генерація хибних електричних сигналів) та структурні пошкодження (дефекти кристалічної ґратки). Це може призводити до тимчасових збоїв у роботі програмного забезпечення або повної відмови обладнання.

Критерієм стійкості є умова:

$$D_{max} \leq D_{lim}, \tag{4.1}$$

де D_{max} – максимальна доза опромінення в точці розміщення системи;

D_{lim} – гранична доза збереження працездатності.

Згідно з [42], для сучасної обчислювальної техніки граничні дози становлять: $D_{lim} \approx 10^3$ рад (для збоїв) та $D_{lim} \approx 10^4$-10^5 рад (для незворотної відмови).

Персонал є найбільш вразливим елементом системи. Доза опромінення в 100-200 рад вже викликає променеву хворобу I ступеня, що унеможливлює

ефективну роботу лікаря. Таким чином, стійкість системи лімітується стійкістю персоналу ($D_{\lim(pers)} \ll D_{\lim(tech)}$).

Для забезпечення роботи системи необхідне використання захисних споруд. Коефіцієнт ослаблення дози радіації (K_{os}) розраховується як:

$$K_{os} \leq 2^{h/d_{pol}}, \quad (4.2)$$

де h – товщина захисного шару,

d_{pol} – шар половинного ослаблення (для бетону $d_{pol} \approx 5.6$ см).

Розміщення робочого місця у підвальному приміщенні з бетонним перекриттям 30 см дозволяє знизити дозу в $K_{os} \approx 40$ разів, що забезпечує стійкість системи у зонах значного радіаційного впливу.

Радіоактивне забруднення виникає внаслідок випадання радіоактивних опадів і характеризується тривалою дією. Ступінь небезпеки визначається рівнем радіації P (Р/год). Залежно від рівня радіації на 1 годину після вибуху (P_1), виділяють зони забруднення [42], які визначають режим роботи системи ЕЕГ-аналізу:

1) Зона М (помірне забруднення, $14 \leq P_1 < 140$ Р/год): робота системи продовжується у звичайному режимі за умови герметизації приміщення.
2) Зона А (сильне забруднення, $140 \leq P_1 < 420$ Р/год): робота припиняється або переноситься у захисну споруду.
3) Зони Б та В ($P_1 \geq 420$ Р/год): робота неможлива, проводиться евакуація.

Основним фактором ураження для ЕЕГ-системи є потрапляння радіоактивного пилу всередину корпусу комп’ютера через систему охолодження. Це призводить до:

- Радіоактивного розігріву мікросхем та погіршення тепловідводу.
- Зміни провідності ізоляційних матеріалів внаслідок іонізації, що викликає короткі замикання.
- Перетворення обладнання на вторинне джерело випромінювання (“брудна” техніка), що становить небезпеку для рук оператора.

Для планування роботи необхідно оцінити спад рівня радіації у часі за формулою:

$$P_t = P_0 \cdot \left(\frac{t}{t_0}\right)^{-0.4}, \tag{4.3}$$

де P_t – рівень радіації на час t.

Згідно з законом спаду, рівень радіації зменшується в 10 разів при збільшенні часу в 7 разів. Це дозволяє розрахувати час відновлення безпечної роботи з системою після проходження радіоактивної хмари.

Для забезпечення функціонування системи в умовах радіаційного впливу необхідно реалізувати комплекс інженерно-технічних та організаційних заходів.

- Герметизація та фільтрація: обладнання робочого місця лікаря у приміщеннях із коефіцієнтом захисту $K_{os} \geq 50$. Встановлення фільтрів на вентиляційні отвори приміщення та повітрозабірники системних блоків для запобігання потраплянню радіоактивного пилу.
- Захист інтерфейсів: використання захисних полімерних плівок (чохлів) на клавіатурі, маніпуляторі “миша” та сенсорних екранах для спрощення подальшої дезактивації.
- Резервування даних: створення резервних копій бази даних ЕЕГ-записів на змінних носіях (SSD, DVD), які зберігаються в екранованих металевих сейфах для захисту від іонізуючого випромінювання.
- Дезактивація: у разі забруднення техніка підлягає знепилюванню методом вакуумування (пилососом з HEPA-фільтром) та протиранню спеціальними дезактивуючими розчинами на спиртовій основі. Використання води для дезактивації електронної апаратури заборонено.

Проведена оцінка показала, що стійкість системи лімітується радіаційною безпекою персоналу. За умови розміщення у захисних спорудах та застосуванні засобів пилозахисту, розроблена система здатна функціонувати в зонах помірного та сильного забруднення.

ВИСНОВКИ

У даній кваліфікаційній роботі магістра розв’язано актуальну науково-технічну задачу підвищення ефективності автоматизованого діагностування епілепсії шляхом розробки та реалізації прикладної програмної системи, що базується на гібридному поєднанні спектрального аналізу, топологічного аналізу даних (TDA) та методів машинного навчання.

Основні наукові та практичні результати роботи полягають у наступному:

- Проведено комплексний аналіз предметної області, в ході якого виявлено обмеження класичних лінійних методів обробки сигналів (перетворення Фур’є) при роботі з ЕЕГ-даними. Обґрунтовано доцільність застосування топологічного аналізу даних як інструменту для виявлення структурних інваріантів та нелінійних зв’язків, характерних для епілептичної активності.
- Розроблено та програмно реалізовано алгоритмічний конвеєр обробки ЕЕГ-сигналів мовою Python. Реалізовано методи попередньої обробки, включаючи стандартизацію каналів, смугову фільтрацію (1-50 Гц) та видалення фізіологічних артефактів методом незалежних компонент (ICA), що забезпечило високу якість вхідних даних для аналізу.
- Створено модуль екстракції ознак на основі гібридного підходу: застосовано комбінування спектральної густини потужності (метод Велча) з топологічними ознаками (персистентні ландшафти), отриманими з діаграм стійкості 1-вимірних гомологій. Це дозволило сформувати вектор ознак, що враховує як енергетичні, так і структурні зміни мозкової активності.
- Побудовано модель класифікації на базі алгоритму Random Forest із застосуванням стратегій балансування даних (SMOTE, RandomUnderSampler) для вирішення проблеми дисбалансу класів. Для уникнення перенавчання використано стратегію крос-валідації Leave-One-Group-Out, що гарантує об’єктивність оцінки на нових пацієнтах.

- Експериментально підтверджено ефективність запропонованого методу на верифікованому наборі даних CHB-MIT. Система досягла показника F1-score на рівні 0.80. Порівняльний аналіз довів, що інтеграція топологічних ознак дозволила підвищити точність (Precision) системи на 8,5% та повноту (Recall) на 6,6% порівняно з базовим спектральним методом. Це свідчить про високу чутливість топологічних методів до специфічних патернів епілепсії та їх стійкість до шумів.
- Розроблено інтерактивний веб-застосунок на базі фреймворку Streamlit, який реалізує повний цикл діагностики: від завантаження EDF-файлів до візуалізації результатів. Інтерфейс включає інструменти для налаштування порогу чутливості, фільтрації результатів та детальної інспекції сигналів (спектрограми, часові діаграми ймовірностей).

Наукова новизна одержаних результатів полягає у вдосконаленні методу класифікації станів ЕЕГ шляхом застосування апарату персистентних гомологій до простору спектральної густини потужності, що, на відміну від існуючих аналогів, дозволило виявляти стійкі циклічні патерни синхронізації (H_1-цикли) та суттєво знизити рівень хибних спрацювань.

Практичне значення роботи полягає у створенні готового до використання кросплатформного програмного засобу, який може функціонувати як локально, так і у хмарному середовищі. Впровадження системи дозволяє скоротити час аналізу тривалих ЕЕГ-моніторингів та слугувати ефективною системою підтримки прийняття рішень для лікарів-епілептологів.

Достовірність результатів забезпечується використанням відкритої верифікованої бази даних CHB-MIT, коректною методологією валідації та порівнянням отриманих метрик з базовими підходами. Розроблена система рекомендується до використання у клінічній практиці як допоміжний інструмент для попереднього скринінгу добових записів ЕЕГ, а також у наукових дослідженнях для вивчення топологічних властивостей нейронних мереж.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. R. D. Thijs, R. Surges, T. J. O'Brien, and J. W. Sander, “Epilepsy in adults”, The Lancet, vol. 393, no. 10172, pp. 689–701, Feb. 2019. doi: 10.1016/S0140-6736(18)32596-0.

2. A. Shoeibi et al., “Epileptic seizures detection using deep learning techniques: A review”, Int. J. Environ. Res. Public Health, vol. 18, no. 11, p. 5780, 2021. doi: 10.3390/ijerph18115780.

3. Методичні вказівки до виконання кваліфікаційної роботи магістра для здобувачів спеціальності 121 – Інженерія програмного забезпечення, всіх форм навчання / укладачі Михалик Д.М., Цуприк Г.Б., Бревус В.М., Мудрик І.Я. – Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2024. – 44 с.

4. K. Rasheed, A. Qayyum, J. Qadir, S. Sivathamboo, and P. Kwan, “Machine learning for predicting epileptic seizures using EEG signals: A review”, IEEE Rev. Biomed. Eng., vol. 14, pp. 139–155, 2020. doi: 10.1109/RBME.2020.3008792.

5. Y. Wang et al., “Topological data analysis of Single-Trial Electroencephalographic Signals”, The Annals of Applied Statistics, vol. 12, no. 3, pp. 1506–1534, Sep. 2018. doi: 10.1214/17-AOAS1119.

6. A. Phinyomark, E. Ibanez-Marcelo, and G. Petri, “Resting-state fMRI functional connectivity: Big data preprocessing pipelines and topological data analysis”, IEEE Trans. Big Data, vol. 3, no. 4, pp. 415–428, Dec. 2017. doi: 10.1109/TBDATA.2017.2734883.

7. M. Sazgar and M. G. Young, Absolute Epilepsy and EEG Rotation Review: Essentials for Trainees. Springer International Publishing, 2019. doi: 10.1007/978-3-030-03511-2.

8. X. Jiang, G.-B. Bian, and Z. Tian, “Removal of artifacts from EEG signals: A review”, Sensors, vol. 19, no. 5, p. 987, 2019. doi: 10.3390/s19050987.

9. A. Craik, Y. He, and J. L. Contreras-Vidal, “Deep learning for electroencephalogram (EEG) classification tasks: a review”, J. Neural Eng., vol. 16, no. 3, p. 031001, 2019. doi: 10.1088/1741-2552/ab0ab5.

10. H. G. Daoud and M. A. Bayoumi, “Efficient epileptic seizure prediction based on deep learning”, IEEE Trans. Biomed. Circuits Syst., vol. 13, no. 5, pp. 804–813, Oct. 2019. doi: 10.1109/TBCAS.2019.2929053.

11. I. Stefanyshyn, O. Pastukh, V. Stefanyshyn, I. Baran, and I. Boyko, “Robustness of AI algorithms for neurocomputer interfaces based on software and hardware technologies”, in Proc. CEUR Workshop, vol. 3742, 2024, pp. 137–149.

12. F. Chazal and B. Michel, “An Introduction to Topological Data Analysis: Fundamental and Practical Aspects for Data Scientists”, Frontiers in Artificial Intelligence, vol. 4, 2021. doi: 10.3389/frai.2021.667963.

13. P. Bubenik, “Statistical topological data analysis using persistence landscapes”, J. Mach. Learn. Res., vol. 16, pp. 77–102, 2015. [Online]. Available: https://www.researchgate.net/publication/281766106

14. M. Jas et al., “MEG/EEG group study with MNE: recommendations, quality assessments and best practices”, Frontiers in Neuroscience, vol. 12, p. 530, 2018. doi: 10.3389/fnins.2018.00530.

15. The GUDHI Project, “GUDHI User and Developer Guide”, GUDHI Editorial Board, 2024. [Online]. Available: https://gudhi.inria.fr/doc/latest/

16. G. Lemaître, F. Nogueira, and C. Aridas, “Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning”, J. Mach. Learn. Res., vol. 18, no. 17, pp. 1–5, 2017. [Online]. Available: http://jmlr.org/papers/v18/16-365.html

17. C. Harris et al., “Array programming with NumPy”, Nature, vol. 585, pp. 357–362, Sep. 2020. doi: 10.1038/s41586-020-2649-2.

18. Streamlit Inc., “Streamlit: The fastest way to build and share data apps”, 2024. [Online]. Available: https://streamlit.io/

19. J. Guttag, “CHB-MIT Scalp EEG Database (version 1.0.0)”, PhysioNet, 2010. doi: 10.13026/C2K01R.

20. J. N. Acharya, A. Hani, J. Cheek, P. Thirumala, and T. N. Tsuchida, “American Clinical Neurophysiology Society Guideline 2: Guidelines for Standard Electrode Position Nomenclature”, J. Clin. Neurophysiol., vol. 33, no. 4, pp. 308–311, Aug. 2016. doi: 10.1097/WNP.0000000000000316.

21. R. Poldrack et al., “Scanning the Horizon: Towards transparent and reproducible neuroimaging research”, Nat. Rev. Neurosci., vol. 18, no. 2, pp. 115–126, 2017. doi: 10.1038/nrn.2016.167.

22. G.-W. Cha et al., “Development of a prediction model for demolition waste generation using a Random Forest algorithm based on small data sets”, Int. J. Environ. Res. Public Health, vol. 17, no. 19, p. 6997, 2020. doi: 10.3390/ijerph17196997.

23. A. Fernández, S. García, M. Galar, R. Prati, B. Krawczyk, and F. Herrera, Learning from Imbalanced Data Sets. Springer International Publishing, 2018. doi: 10.1007/978-3-319-98074-4.

24. H. Hairani, A. Anggrawan, and D. Priyanto, “Improvement Performance of the Random Forest Method on Unbalanced Diabetes Data Classification Using SMOTE-Tomek Link”, Int. J. Informatics Visualization, vol. 7, no. 1, pp. 258–264, 2023. doi: 10.30630/joiv.7.1.1069.

25. J. Speiser, “A random forest method with feature selection for developing medical prediction models with clustered and longitudinal data”, J. Biomed. Inform., vol. 117, p. 103763, 2021. doi: 10.1016/j.jbi.2021.103763.

26. S. S. Nain, D. Garg, and S. Kumar, “Performance evaluation of the WEDM process of aeronautics super alloy”, Materials and Manufacturing Processes, vol. 33, no. 3, pp. 1793–1808, 2018. doi: 10.1080/10426914.2018.1476761.

27. M. Petryk, A. Doroshenko, D. Mykhalyk, P. Ivanenko, and O. Yatsenko, “Automated Parallelization of Software for Identifying Parameters of Intraparticle Diffusion and Adsorption in Heterogeneous Nanoporous Media”, in Lecture Notes in Networks and Systems, vol. 667, 2023, pp. 33–47.

28. V. Stefanyshyn, I. Stefanyshyn, O. Pastukh, V. Yatsyshyn, and I. Yakymenko, “Accuracy of software and hardware of computer systems for human-machine interaction”, in Proc. CEUR Workshop, vol. 3842, 2024, pp. 178–183.

29. E. Tjoa and C. Guan, “A Survey on Explainable Artificial Intelligence (XAI): Toward Medical XAI”, IEEE Trans. Neural Netw. Learn. Syst., vol. 32, no. 11, pp. 4793-4813, Nov. 2021. doi: 10.1109/TNNLS.2020.3027314.

30. N. P. Rougier, Scientific Visualization: Python + Matplotlib. Bordeaux: Nicolas P. Rougier, 2021. [Online]. Available: https://hal.inria.fr/hal-03427242/

31. A. Tharwat, “Classification Assessment Methods: a detailed tutorial”, Appl. Comput. Inform., vol. 17, no. 1, pp. 168-192, 2020. doi: 10.1016/j.aci.2018.08.003.

32. H. Sofaer, J. Hoeting, and C. Jarnevich, “The area under the precision-recall curve as a performance metric for rare binary events”, Methods in Ecology and Evolution, vol. 10, no. 4, pp. 565-577, 2019. doi: 10.1111/2041-210X.13140.

33. “Про охорону праці”, Закон України № 2694-XII, 14 жовт. 1992.

34. Охорона праці. Терміни та визначення основних понять, ДСТУ 2293:2014, Київ, Україна: Мінекономрозвитку України, 2015.

35. Правила охорони праці під час експлуатації навантажувачів : НПАОП 0.00-1.83-18 : затв. наказом М-ва соц. політики України від 27.08.2018 № 1220. Київ : [Б. в.], 2018.

36. “Санітарні норми мікроклімату виробничих приміщень”, ДСН 3.3.6.042-99, 1 груд. 1999.

37. “Санітарні норми виробничого шуму, ультразвуку та інфразвуку”, ДСН 3.3.6.037-99, 1 груд. 1999.

38. Природне і штучне освітлення, ДБН В.2.5-28:2018, Київ, Україна: Мінрегіонбуд України, 2018.

39. “Правила безпечної експлуатації електроустановок споживачів”, ДНАОП 0.00-1.21-98, 9 січ. 1998.

40. “Кодекс цивільного захисту України”, Кодекс України № 5403-VI, 2 жовт. 2012.

41. Правила пожежної безпеки в Україні : НАПБ А.01.001-2014 : затв. наказом М-ва внутрішніх справ України від 30.12.2014 № 1417. Київ, 2015..

42. В. С. Стручок, Техноекологія та цивільна безпека. Частина “Цивільна безпека”: навч. посіб., Тернопіль, Україна: ТНТУ ім. І.Пулюя, 2022.

ДОДАТКИ

Додаток А

Публікація в науковому виданні

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний технічний університет імені Івана Пулюя
Маріборский університет (Словенія)
Технічний університет у Кошице (Словаччина)
Вільнюський технічний університет ім. Гедимінаса (Литва)
Краківський економічний університет (Польща)
Вроцлавський економічний університет (Польща)
Університет «Опольська Політехніка» (Польща)
Національний університет «Полтавська політехніка імені Юрія Кондратюка»
Вінницький національний аграрний університет
Львівський національний університет ім. І. Франка
Головне управління Пенсійного фонду в Тернопільській області
Наукове товариство ім. Шевченка
Тернопільський обласний комунальний інститут післядипломної педагогічної освіти
Сумський державний педагогічний університет
Запорізький національний університет

АКТУАЛЬНІ ЗАДАЧІ СУЧАСНИХ ТЕХНОЛОГІЙ

Збірник

тез доповідей

XIV Міжнародної науково-технічної конференції молодих учених та студентів

11-12 грудня 2025 року

УКРАЇНА
ТЕРНОПІЛЬ – 2025

Система охолодження та захисту. Оскільки під час роботи утворюється багато тепла, системи оснащуються вентиляторами, теплообмінниками або рідинним охолодженням.

Корпуси мають екранування від зовнішніх електромагнітних впливів та захист від вологи, пилу й механічних пошкоджень.

У військових зразках корпуси часто броньовані, щоб витримати удар або уламки.

Програмне забезпечення. Без програмного керування РЕБ не зможе працювати точно.

Програми відповідають за:

-аналіз спектра сигналів;

-класифікацію типів загроз;

--вибір оптимального режиму придушення;

синхронізацію з іншими системами — наприклад, радіорозвідкою чи ППО.

Програмне забезпечення постійно оновлюється для адаптації до нових типів сигналів противника.

Радіоелектронна боротьба є ключовим елементом сучасної війни, що дозволяє захищати війська й техніку від дронів та засобів зв'язку супротивника. Українські технології в цій сфері доводять свою ефективність і постійно удосконалюються, залишаючись одним із важливих факторів у забезпеченні обороноздатності.

Література

1. Радіоелектронна боротьба у війні: види, роль та найкращі системи 2025 року. URL: https://bezpeka-veritas.com.ua/radioelektronna-borotba-u-viini-vydy-rol-ta-naikrashchi-systemy-2025-roku/

2. Які існують види РЕБів: їхні класи й відмінності? URL: https://wartex.com/yaki-isnuyut-vidi-rebiv/

3. Радіоелектронна боротьба — ключ до сучасної війни URL: https://www.armyfm.com.ua/radioelektronna-borotba--kliuch-do-suchasnoi-viiny/

УДК 004.41

О.М. Задворний; І.В. Бойко, д.ф.-м.н, професор

(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ДЕТЕКЦІЇ ЕПІЛЕПТИЧНИХ НАПАДІВ НА ОСНОВІ ГІБРИДНИХ ТОПОЛОГІЧНО-СПЕКТРАЛЬНИХ ОЗНАК

O.M. Zadvornyi, I.V. Boyko, D.Sc., Prof.

INCREASING THE EFFICIENCY OF EPILEPTIC SEIZURE DETECTION BASED ON HYBRID TOPOLOGICAL-SPECTRAL FEATURES

Автоматизований аналіз ЕЕГ-сигналів є критично важливим для ефективної діагностики епілепсії. Хоча методи глибокого навчання демонструють високу результативність [1], класичні спектральні підходи часто втрачають інформацію про нелінійні структурні зміни мозкової активності [2]. Останні дослідження виділяють топологічний аналіз даних (TDA) як перспективний інструмент для ідентифікації стійких геометричних патернів у біомедичних сигналах, що залишаються недоступними для лінійних методів [3, 4].

У дослідженні використано базу даних CHB-MIT Scalp EEG [5]. Запропонований гібридний підхід поєднує оцінку спектральної густини потужності (метод Велча) з обчисленням персистентних гомологій. Багатоканальні епохи сигналу розглядаються як хмари точок у частотному просторі, для яких будуються комплекси Вієторіса-Ріпса. Екстракція топологічних ознак виконується шляхом трансформації діаграм стійкості у персистентні ландшафти [6]. Класифікація реалізована алгоритмом Random Forest із застосуванням стратегії балансування даних SMOTE та валідацією Leave-One-Group-Out для забезпечення узагальнення на нових пацієнтах.

Інтеграція топологічних інваріантів дозволила моделі фіксувати специфічні патерни синхронізації між каналами, характерні для судомної активності. Порівняльний аналіз із базовим спектральним методом демонструє приріст точності (Precision) на 8,5% (до 0,85) та повноти (Recall) на 6,6% (до 0,75), що забезпечило загальний F1-score на рівні 0,80. Застосування адаптивного порогу чутливості у поєднанні з медіанною фільтрацією дозволяє мінімізувати вплив короткочасних артефактів, знизивши кількість помилкових спрацювань при аналізі. Реалізований вебінтерфейс візуалізує часову динаміку ймовірностей нападів, надаючи інструмент для верифікації результатів.

Взаємодія спектрального аналізу та топологічних методів забезпечує вищу стійкість алгоритму до шумів та артефактів порівняно з традиційними підходами. Розроблена архітектура може слугувати основою для функціонування систем підтримки прийняття рішень у клінічній нейрофізіології.

Література

1. A. Shoeibi et al., "Epileptic seizures detection using deep learning techniques: A review," Int. J. Environ. Res. Public Health, vol. 18, no. 11, p. 5780, 2021. doi: 10.3390/ijerph18115780.
2. X. Jiang, G.-B. Bian, and Z. Tian, "Removal of artifacts from EEG signals: A review," Sensors, vol. 19, no. 5, p. 987, 2019. doi: 10.3390/s19050987.
3. Y. Wang et al., "Topological data analysis of Single-Trial Electroencephalographic Signals," The Annals of Applied Statistics, vol. 12, no. 3, pp. 1506–1534, Sep. 2018. doi: 10.1214/17-AOAS1119.
4. F. Chazal and B. Michel, "An Introduction to Topological Data Analysis: Fundamental and Practical Aspects for Data Scientists," Frontiers in Artificial Intelligence, vol. 4, 2021. doi: 10.3389/frai.2021.667963.
5. J. Guttag, "CHB-MIT Scalp EEG Database (version 1.0.0)," PhysioNet, 2010. doi: 10.13026/C2K01R.
6. P. Bubenik, "Statistical topological data analysis using persistence landscapes," J. Mach. Learn. Res., vol. 16, pp. 77–102, 2015. https://www.jmlr.org/papers/volume16/bubenik15a/bubenik15a.pdf

Додаток Б

Лістинг коду розробленої програми

Лістинг Б.1 – Код файлу “app.py” для тренування моделі

```
import numpy as np
from sklearn.model_selection import LeaveOneGroupOut
from sklearn.feature_selection import VarianceThreshold
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report,
                            recall_score,
                            precision_score
from sklearn.preprocessing import StandardScaler
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE, RandomOverSampler
from imblearn.pipeline import Pipeline
import os
import glob
import warnings
import sys
import joblib
from scipy.signal import medfilt
from sklearn.neural_network import MLPClassifier
import matplotlib.pyplot as plt
from sklearn.metrics import precision_recall_curve, f1_score
import json

from utils.parse_summary_file import parse_summary_file
from utils.process_edf_file import process_edf_file

SAVED_DATA_FILE = "chb_features_ica.npz"
DATA_DIR = "./CHB-MIT_Scalp_EEG_Database/"

# Check for pre-processed data to avoid re-running extraction
if os.path.exists(SAVED_DATA_FILE):
    print(
        f"Loading pre-processed data from {SAVED_DATA_FILE}..."
    )
    data = np.load(SAVED_DATA_FILE)
    X = data['X']
    y = data['y']
    groups = data['groups']
    print("Data loaded successfully.")
else:
    print("Running full processing pipeline...")

    master_features_list = []
    master_labels_list = []
    master_patient_groups = []

    patient_dirs = sorted(
```

```
    glob.glob(os.path.join(DATA_DIR, "chb*")))

# Iterate over each patient directory
for patient_dir in patient_dirs:
    if not os.path.isdir(patient_dir):
        continue

    patient_id = os.path.basename(patient_dir)
    print(f"\n--- Processing Patient: {patient_id} ---")

    # Find and parse the summary file
    summary_file = glob.glob(
        os.path.join(patient_dir, "*-summary.txt"))
    if not summary_file:
        print(f"No summary file found for {patient_id}.")
        continue

    seizure_dict = parse_summary_file(summary_file[0])

    # Find all .edf files for this patient
    edf_files = sorted(glob.glob(
        os.path.join(patient_dir, "*.edf")))

    if not edf_files:
        print(f"No .edf files found for {patient_id}")
        continue

    # Iterate over EDF files
    for edf_path in edf_files:
        filename = os.path.basename(edf_path)

        # Retrieve seizure intervals
        intervals = seizure_dict.get(filename, [])

        # Process file
        features, labels = process_edf_file(edf_path,
                                            intervals)

        # Collect results
        if features is not None and labels is not None:
            master_features_list.append(features)
            master_labels_list.append(labels)

            # Assign group ID based on patient number
            patient_group_id =
                int(patient_id.replace("chb", ""))
            master_patient_groups.append(
                np.full(len(labels), patient_group_id))

print("\n--- All Files Processed. Combining Data... ---")

if not master_features_list:
    print("No features were extracted!")
```

```
            sys.exit("Stopping script: No data to process.")

        X = np.vstack(master_features_list)
        y = np.concatenate(master_labels_list)
        groups = np.concatenate(master_patient_groups)

        print(f"Saving processed data to {SAVED_DATA_FILE}...")
        np.savez_compressed(SAVED_DATA_FILE, X=X, y=y,
                            groups=groups)
        print("Data saved successfully.")

    print(f"\nTotal features shape (X): {X.shape}")
    print(f"Total labels shape (y): {y.shape}")
    print(f"Total seizures found (label 1): {np.sum(y)}")

    # Data integrity checks
    print(f"Is there NaN in data? {np.isnan(X).any()}")
    std_devs = np.std(X, axis=0)
    print(f"Columns with zero variance: {np.sum(std_devs == 0)}")

    # Hyperparameters
    smote_ratio = 0.05
    rus_ratio = 0.25
    filter_kernel = 5

    def get_pipeline(y_train_data=None):
        """
        Constructs the processing pipeline.
        Adjusts SMOTE neighbors based on available seizure samples.
        """
        n_seizures = np.sum(y_train_data == 1)

        # Ensure k_neighbors is valid (< n_seizures)
        k_neighbors = min(5, n_seizures - 1)

        steps = []

        # VarianceThreshold(0) ensures features have non-zero
        # variance before scaling
        steps.append(('selector', VarianceThreshold(threshold=0)))
        steps.append(('scaler', StandardScaler()))

        if k_neighbors > 0:
            steps.append(('smote',
                SMOTE(sampling_strategy=smote_ratio,
                      k_neighbors=k_neighbors, random_state=42)))
        else:
            steps.append(('ros',
                RandomOverSampler(sampling_strategy=smote_ratio,
                                  random_state=42)))

        steps.append(('rus',
            RandomUnderSampler(sampling_strategy=rus_ratio,
```

```
                                random_state=42)))
    steps.append(('model',
        RandomForestClassifier(n_estimators=100,
                               random_state=42, n_jobs=-1)))

    return Pipeline(steps=steps)


# ======================================================
# PART 1: Validation with Threshold Tuning
# ======================================================
logo = LeaveOneGroupOut()
n_splits = logo.get_n_splits(X, y, groups)

print(f"\nStarting Cross-Validation on {n_splits} patients...")
print(f"Configuration: SMOTE={smote_ratio}, RUS={rus_ratio}")

# Storage for global optimization
all_true_labels = []
all_probabilities = []
patient_group_map = []

# Pass 1: Collect Probabilities via Cross-Validation
for i, (train_indices, test_indices) in enumerate(
    logo.split(X, y, groups)
):
    test_patient = groups[test_indices][0]

    X_train, X_test = X[train_indices], X[test_indices]
    y_train, y_test = y[train_indices], y[test_indices]

    pipeline = get_pipeline(y_train)
    pipeline.fit(X_train, y_train)

    # Predict probabilities for the positive class (Seizure)
    probs = pipeline.predict_proba(X_test)[:, 1]

    all_true_labels.extend(y_test)
    all_probabilities.extend(probs)
    patient_group_map.extend(groups[test_indices])

    print(f"Patient {test_patient} processed.")

all_true_labels = np.array(all_true_labels)
all_probabilities = np.array(all_probabilities)
patient_group_map = np.array(patient_group_map)

# Pass 2: Optimize Threshold Globally
print("\n--- Optimizing Threshold on Aggregated Data ---")
precisions, recalls, thresholds =
    precision_recall_curve(all_true_labels, all_probabilities)

with np.errstate(divide='ignore', invalid='ignore'):
```

```
    f1_scores = 2 * (precisions * recalls) /
        (precisions + recalls)
f1_scores = np.nan_to_num(f1_scores)

best_idx = np.argmax(f1_scores)
best_threshold = thresholds[best_idx]
best_f1 = f1_scores[best_idx]

print(f"Best Threshold Found: {best_threshold:.4f}")
print(f"Max Theoretical F1:   {best_f1:.4f}")

# Plot optimization curve
plt.figure(figsize=(10, 6))
plt.plot(thresholds, precisions[:-1], "b--", label="Precision")
plt.plot(thresholds, recalls[:-1], "g-", label="Recall")
plt.plot(thresholds, f1_scores[:-1], "r-", label="F1 Score")
plt.axvline(best_threshold, color='k', linestyle=':',
            label=f"Optimal ({best_threshold:.2f})")
plt.title("Threshold Optimization Curve")
plt.legend()
plt.savefig('./threshold_optimization.png')
print("Plot saved to ./Figures/threshold_optimization.png")

# Pass 3: Apply Best Threshold & Median Filter
print(f"\nRe-evaluating with Threshold {best_threshold:.4f}")

final_predictions_all = []
final_labels_all = []

# Apply median filter per patient to maintain integrity
unique_patients = np.unique(patient_group_map)

for pat in unique_patients:
    idx = (patient_group_map == pat)

    p_probs = all_probabilities[idx]

    # Apply optimized threshold
    p_preds = (p_probs >= best_threshold).astype(int)

    # Apply smoothing
    p_preds_filtered = medfilt(p_preds,
                               kernel_size=filter_kernel)

    final_predictions_all.extend(p_preds_filtered)
    final_labels_all.extend(all_true_labels[idx])

print("\n--- Final Validation Report (Optimized) ---")
print(classification_report(final_labels_all,
      final_predictions_all, target_names=["Non-Seizure",
      "Seizure"]))

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

```
# PART 2: Final Training & Saving
# ======================================================
print("\n--- Retraining Final Model on ALL Data ---")

# Instantiate pipeline passing full dataset labels for SMOTE
# calculation
final_pipeline = get_pipeline(y)

final_pipeline.fit(X, y)

model_filename = 'seizure_model_final.pkl'
joblib.dump(final_pipeline, model_filename)

# Save configuration and threshold separately as they are not
# part of the sklearn pipeline object
config = {
    "best_threshold": float(best_threshold),
    "filter_kernel": int(filter_kernel),
    "smote_ratio": float(smote_ratio),
    "rus_ratio": float(rus_ratio)
}

config_filename = 'model_config.json'
with open(config_filename, 'w') as f:
    json.dump(config, f)

print(f"Model saved to {model_filename}")
print(f"Configuration (Threshold) saved to {config_filename}")
```

Лістинг Б.2 – Код файлу “parse_summary_file.py” для зчитування міток

```
import re

def parse_summary_file(summary_path):
    """
    Parses a chbXX-summary.txt file to extract intervals.

    Args:
        summary_path (str): Path to the summary text file.

    Returns:
        dict: A dictionary where keys are filenames
              and values are lists of tuples (start, end)
              representing seizure times in seconds.
    """
    seizure_dict = {}
    current_filename = ""
    start_sec = None

    try:
        with open(summary_path, 'r') as f:
            for line in f:
```

```
                line = line.strip()

                if line.startswith("File Name:"):
                    current_filename = line
                        .split(":")[-1].strip()
                    seizure_dict[current_filename] = []
                    start_sec = None

                # Match seizure start time patterns
                elif re.match(
                    r"^Seizure \d* ?Start Time:", line):
                    start_sec = int(line
                        .split(":")[-1].strip().split(" ")[0])

                # Match seizure end time patterns
                elif re.match(
                    r"^Seizure \d* ?End Time:", line):
                    # Ensure corresponding start time exists
                    # before recording interval
                    if start_sec is not None:
                        end_sec = int(line.split(":")[-1]
                            .strip().split(" ")[0])
                        seizure_dict[current_filename].append(
                            (start_sec, end_sec))
                        start_sec = None

    except Exception as e:
        print(f"Error parsing file {summary_path}: {e}")

    return seizure_dict
```

Лістинг Б.3 – Код файлу “process_edf_file” для обробки ЕЕГ-даних

```
import mne
import numpy as np
import os
import gudhi as gd
from mne.preprocessing import ICA

def apply_ica_cleaning(raw, n_components=15, random_state=42):
    """
    Fits ICA on the raw object to prepare the data structure
    for artifact removal.
    """
    ica = ICA(n_components=n_components, r
              random_state=random_state, max_iter="auto")

    # Fit on the raw data
    ica.fit(raw, verbose=False)

    # Apply cleaning (uses manual exclusions if set)
    raw_cleaned = ica.apply(raw, verbose=False)
```

```
    return raw_cleaned

def compute_persistence_diagram(point_cloud):
    """
    Computes the persistence diagram for a given point cloud
    using a Rips Complex.
    Returns H1 (cycles/loops) features.
    """
    rips_complex = gd.RipsComplex(points=point_cloud,
                                  max_edge_length=1)
    simplex_tree = rips_complex.create_simplex_tree(
        max_dimension=2)
    persistence_diagram = simplex_tree.persistence()

    result = []
    for dim, (birth, death) in persistence_diagram:
        # Drop infinite cycles
        if death == float('inf'):
            continue

        # Filter for H1 dimension (cycles)
        if dim != 1:
            continue

        result.append((dim, (birth, death)))

    return result

def compute_landscape_values(diag, grid):
    """
    Computes persistence landscape values for a given diagram
    over a specified grid.
    """
    landscape_values = np.zeros_like(grid)

    for interval in diag:
        dim, (birth, death) = interval
        for i, t in enumerate(grid):
            if birth < t < death:
                landscape_values[i] = max(
                    landscape_values[i],
                    min(t - birth, death - t))

    return landscape_values

def process_edf_file(edf_path, seizure_intervals):
    """
    Processes a single .edf file:
    1. Loads and normalizes channels to International 10-20
       standard.
    2. Filters and applies ICA.
    3. Epochs the data.
    4. Extracts Spectral, TDA, and Time-domain features.
```

```
5. Labels epochs based on seizure intervals.
"""

print(f"\nProcessing {os.path.basename(edf_path)}...")

# Target standard channels (International 10-20)
STANDARD_CHANNELS = [
    'FP1-F7', 'F7-T7', 'T7-P7', 'P7-O1',
    'FP1-F3', 'F3-C3', 'C3-P3', 'P3-O1',
    'FP2-F4', 'F4-C4', 'C4-P4', 'P4-O2',
    'FP2-F8', 'F8-T8', 'T8-P8', 'P8-O2',
    'FZ-CZ', 'CZ-PZ'
]

# Alias mapping for inconsistencies in CHB-MIT (Old -> New)
CHANNEL_ALIAS = {
    'T3': 'T7', 'T4': 'T8', 'T5': 'P7', 'T6': 'P8'
}

try:
    # Load Data
    raw = mne.io.read_raw_edf(edf_path, preload=True,
                              verbose=False)

    # Normalize channel names
    print("Normalizing channel names...")
    rename_map = {}
    current_chans = raw.ch_names

    for ch in current_chans:
        clean_name = ch.upper().strip()
        clean_name = clean_name.replace('-0', '')
            .replace('-REF', '').replace('-Ref', '')

        # Apply aliases
        for old, new in CHANNEL_ALIAS.items():
            clean_name = clean_name.replace(old, new)

        if clean_name in STANDARD_CHANNELS:
            rename_map[ch] = clean_name

    if rename_map:
        raw.rename_channels(rename_map)

    # Validate required channels
    present_channels = raw.ch_names
    missing_chans = [ch for ch in STANDARD_CHANNELS
        if ch not in present_channels]

    if len(missing_chans) > 0:
        print(f"Missing required channels:{missing_chans}")
        return None, None
```

```
raw.pick_channels(STANDARD_CHANNELS)

# Filter Data
print("Filtering data (1.0-50 Hz)...")
raw.filter(l_freq=1.00, h_freq=50, verbose=False)

# Apply ICA
print("Applying ICA...")
try:
    raw = apply_ica_cleaning(raw, n_components=15)
except Exception as e:
    print(f"ICA Failed. Proceeding with raw data.")

# Epoching
epoch_duration = 5
epochs = mne.make_fixed_length_epochs(
    raw,
    duration=epoch_duration,
    preload=True,
    verbose=False
)

if len(epochs) == 0:
    print("No epochs extracted. Skipping file.")
    return None, None

# Feature Extraction: Spectral
print("Extracting spectral features...")

sfreq = epochs.info['sfreq']
bands = {
    "delta": (0.5, 4), "theta": (4, 8),
    "alpha": (8, 13), "beta": (13, 30),"gamma":(30, 50)
}

psd = epochs.compute_psd(method="welch",
                         n_fft=int(sfreq),
                         fmin=0.5,
                         fmax=50.0,
                         verbose=False)
psd_data = psd.get_data()
freqs = psd.freqs

spectral_features = []
for epoch_psd in psd_data:
    epoch_features = []
    for band in bands:
        fmin, fmax = bands[band]
        idx = np.logical_and(
            freqs >= fmin, freqs < fmax)
        band_power_per_channel = np.mean(
            epoch_psd[:, idx], axis=1)
        epoch_features.append(band_power_per_channel)
```

```
        spectral_features.append(
            np.concatenate(epoch_features))

    spectral_features = np.array(spectral_features)
    spectral_features = 10 * np.log10(
        spectral_features + 1e-20)

    # Feature Extraction: Topological (TDA)
    print("Extracting TDA features...")

    # Log transform and normalize PSD for shape analysis
    psd_log = 10 * np.log10(psd_data + 1e-20)

    p_min = psd_log.min(axis=(1,2), keepdims=True)
    p_max = psd_log.max(axis=(1,2), keepdims=True)
    psd_norm = (psd_log - p_min) / (p_max - p_min + 1e-10)

    grid = np.linspace(0, 1, 100)
    tda_features_list = []

    for epoch_idx in range(len(psd_data)):
        # Form point cloud from channel data in frequency
        point_cloud = psd_norm[epoch_idx]

        diag = compute_persistence_diagram(point_cloud)
        landscape = compute_landscape_values(diag, grid)

        tda_features_list.append(landscape)

    tda_features = np.array(tda_features_list)

    # Feature Extraction: Time Domain
    all_epoch_data = epochs.get_data()
    n_epochs, n_channels, _ = all_epoch_data.shape
    line_length = np.sum(np.abs(np.diff(
        all_epoch_data, axis=2)), axis=2)
    variance = np.var(all_epoch_data, axis=2)

    all_features = np.hstack((spectral_features,
        tda_features, line_length, variance))

    # Labeling
    labels = np.zeros(n_epochs, dtype=int)
    epoch_start_times = epochs.events[:, 0] /
        epochs.info['sfreq']

    for i in range(n_epochs):
        epoch_start = epoch_start_times[i]
        epoch_end = epoch_start + epoch_duration

        for seizure_start, seizure_end in
            seizure_intervals:
            if (epoch_start < seizure_end)
```

```
                    and (epoch_end > seizure_start):
                    labels[i] = 1
                    break

        print("Labeling complete.")
        print(f"Seizures: {np.sum(labels)} / {len(labels)}")

        return all_features, labels

    except Exception as e:
        print(f"Error processing {os.path.basename(edf_path)}")
        print(e)
        return None, None
```

Лістинг Б.4 – Код файлу “web_app.py” для інтерфейсу вебдодатку

```
import streamlit as st
import pandas as pd
import numpy as np
import joblib
import json
import os
import tempfile
import matplotlib.pyplot as plt
from scipy.signal import medfilt, spectrogram
import mne

# Attempt to import the custom pipeline
try:
    from utils.eeg_processor import EEGPipeline, EEGConfig
except ImportError:
    try:
        from utils.eeg_processor import EEGPipeline, EEGConfig
    except ImportError:
        st.error(
            "Critical Error: Could not import EEGPipeline.")

# --- Privacy / Security Class ---
class PrivacyManager:
    @staticmethod
    def anonymize_raw(raw):
        """Removes sensitive metadata from MNE Raw objects."""
        if raw.info['subject_info'] is not None:
            raw.info['subject_info'] = None
        return raw

# --- Visualization Class ---
class EEGVisualizer:
    @staticmethod
    def plot_timeline(probs, preds, threshold,
                      epoch_duration=5):
        total_epochs = len(probs)
```

```
        time_axis = np.arange(total_epochs) *
            epoch_duration / 60

        fig, ax = plt.subplots(figsize=(10, 3))

        ax.plot(
            time_axis,
            probs,
            color='#333333',
            alpha=0.6,
            linewidth=1,
            label='Probability'
        )
        ax.fill_between(
            time_axis,
            0,
            1,
            where=(preds == 1),
            color='#ff4b4b',
            alpha=0.4,
            label='Detection'
        )
        ax.axhline(
            threshold,
            color='blue',
            linestyle=':',
            label='Threshold'
        )

        ax.set_ylabel("Probability")
        ax.set_xlabel("Time (minutes)")
        ax.set_xlim(0, time_axis[-1])
        ax.set_ylim(0, 1.05)
        ax.legend(loc='upper right')
        return fig

    @staticmethod
    def plot_raw_signal(raw, start_time, duration=10):
        sfreq = raw.info['sfreq']
        start_sample = int(start_time * sfreq)
        stop_sample = int((start_time + duration) * sfreq)

        data = raw.get_data(start=start_sample,
            stop=stop_sample)
        times = np.arange(data.shape[1]) / sfreq + start_time

        fig, ax = plt.subplots(figsize=(12, 8))
        offset_step = 0.0002

        for i, ch_name in enumerate(raw.ch_names):
            offset = i * offset_step
            ax.plot(
                times,
```

```
                data[i] + offset,
                linewidth=0.8,
                color='#333'
            )
            ax.text(
                times[0],
                offset,
                ch_name,
                fontsize=8,
                color='blue',
                ha='right'
            )

        ax.set_yticks([])
        ax.spines['top'].set_visible(False)
        ax.spines['right'].set_visible(False)
        ax.spines['left'].set_visible(False)
        return fig

    @staticmethod
    def plot_spectrogram(data, sfreq, start_time):
        fig, ax = plt.subplots(figsize=(10, 4))
        f, t, Sxx = spectrogram(
            data,
            fs=sfreq,
            nperseg=128,
            noverlap=64
        )
        img = ax.pcolormesh(
            t + start_time,
            f,
            10 * np.log10(Sxx + 1e-10),
            shading='gouraud',
            cmap='inferno'
        )
        ax.set_ylabel('Frequency [Hz]')
        ax.set_xlabel('Time [sec]')
        plt.colorbar(img, ax=ax, label='Power (dB)')
        return fig

# --- Model Management Class ---
class ModelHandler:
    def __init__(
        self,
        model_path='seizure_model_final.pkl',
        config_path='model_config.json'
    ):
        self.model = None
        self.config = {}
        self._load(model_path, config_path)

    def _load(self, model_path, config_path):
        try:
```

```
            self.model = joblib.load(model_path)
            with open(config_path, 'r') as f:
                self.config = json.load(f)
        except FileNotFoundError:
            pass # Error handling delegated to UI

    def is_loaded(self):
        return self.model is not None

    def predict(self, features):
        if not self.is_loaded(): return None
        return self.model.predict_proba(features)[:, 1]

# --- Main Application Class ---
class SeizureApp:
    def __init__(self):
        st.set_page_config(
            page_title="EEG Seizure Detector",
            page_icon="□",
            layout="wide"
        )
        self.pipeline = EEGPipeline()
        self.model_handler = ModelHandler()
        self._init_session_state()

    def _init_session_state(self):
        if 'file_name' not in st.session_state:
            st.session_state.file_name = None
        if 'probs' not in st.session_state:
            st.session_state.probs = None
        if 'raw_viz' not in st.session_state:
            st.session_state.raw_viz = None
        if 'features_extracted' not in st.session_state:
            st.session_state.features_extracted = False

    def render_sidebar(self):
        st.sidebar.header("Configuration")

        if self.model_handler.is_loaded():
            st.sidebar.success("Model Loaded")
        else:
            st.sidebar.error("Model files not found.")
            st.stop()

        default_thresh = self.model_handler.config.get(
            'best_threshold', 0.5)
        thresh = st.sidebar.slider(
            "Sensitivity Threshold",
            0.0,
            1.0,
            default_thresh
        )
```

```
        default_kernel = self.model_handler.config.get(
            'filter_kernel', 5)
        kernel = st.sidebar.slider(
            "Smoothing Kernel",
            1,
            15,
            default_kernel,
            step=2
        )

        st.sidebar.markdown("---")
        with st.sidebar.expander("Privacy & Disclaimer"):
            st.warning("Not a Medical Device.")
            st.info(
                "Data processed locally. No cloud uploads.")

        return thresh, kernel

    def process_file(self, uploaded_file):
        # Avoid reprocessing the same file
        if st.session_state.file_name == uploaded_file.name:
            return

        with st.spinner(f"Processing {uploaded_file.name}..."):
            with tempfile.NamedTemporaryFile(
                delete=False,
                suffix=".edf"
            ) as tmp_file:
                tmp_file.write(uploaded_file.getvalue())
                tmp_path = tmp_file.name

            try:
                # 1. Feature Extraction
                features, _ = self.pipeline.process(
                    tmp_path,
                    seizure_intervals=[]
                )
                if features is None:
                    st.error("Feature extraction failed.")
                    st.stop()

                # 2. Prediction
                probs = self.model_handler.predict(features)

                # 3. Load Raw Data for Visualization
                # Uses preprocessor logic to ensure consistency
                # with model input
                raw_viz = self.pipeline.preprocessor
                    .load_and_clean(tmp_path)
                raw_viz = PrivacyManager.anonymize_raw(raw_viz)

                # 4. Update State
                st.session_state.probs = probs
```

```
                st.session_state.raw_viz = raw_viz
                st.session_state.file_name = uploaded_file.name
                st.session_state.features_extracted = True

            except Exception as e:
                st.error(f"Error: {e}")
            finally:
                if os.path.exists(tmp_path):
                    os.remove(tmp_path)

    def render_results(self, threshold, kernel):
        if not st.session_state.features_extracted:
            return

        probs = st.session_state.probs
        raw = st.session_state.raw_viz

        # Post-processing
        preds = medfilt(
            (probs >= threshold).astype(int),
            kernel_size=kernel
        )
        n_seizures = np.sum(preds)

        tab1, tab2 = st.tabs(
            ["Detection Results", "Signal Inspector"])

        with tab1:
            c1, c2, c3 = st.columns(3)
            c1.metric("Duration",
                      f"{(len(preds)*5)/60:.1f} min")
            c2.metric("Seizure Burden", f"{n_seizures*5} sec")

            if n_seizures > 0:
                c3.error("Seizure Detected")
            else:
                c3.success("Normal Record")

            st.pyplot(EEGVisualizer.plot_timeline(
                probs,
                preds,
                threshold
            ))

            if n_seizures > 0:
                self._render_detection_table(preds, probs)

        with tab2:
            self._render_inspector(raw, preds, n_seizures)

    def _render_detection_table(self, preds, probs):
        st.markdown("### Detections List")
        padded = np.pad(preds, (1, 1), 'constant')
```

```
        diffs = np.diff(padded)
        starts = np.where(diffs == 1)[0]
        ends = np.where(diffs == -1)[0]

        detections = []
        for s, e in zip(starts, ends):
            detections.append({
                "Start (s)": s*5,
                "End (s)": e*5,
                "Conf.": f"{np.mean(probs[s:e]):.2f}"
            })
        st.dataframe(pd.DataFrame(detections))

    def _render_inspector(self, raw, preds, n_seizures):
        max_time = int(len(preds) * 5)

        # Default to first seizure or start of file
        if n_seizures > 0:
            default = int(np.where(preds == 1)[0][0] * 5)
        else:
            default = 0

        start_view = st.slider(
            "View Time (s)",
            0,
            max_time - 10,
            default
        )

        st.pyplot(EEGVisualizer.plot_raw_signal(
            raw, start_view))

        spec_ch = st.selectbox("Spectrogram Channel",
                               raw.ch_names)
        ch_idx = raw.ch_names.index(spec_ch)

        sfreq = raw.info['sfreq']
        start_samp = int(start_view * sfreq)
        end_samp = int((start_view + 10) * sfreq)

        data_segment = raw.get_data(
            start=start_samp,
            stop=end_samp
        )[ch_idx]

        st.pyplot(EEGVisualizer.plot_spectrogram(
            data_segment,
            sfreq,
            start_view
        ))

    def run(self):
        st.title("□ EEG Seizure Detection System")
```

```
        thresh, kernel = self.render_sidebar()

        uploaded_file = st.file_uploader(
            "Choose an EDF file...",
            type=['edf']
        )
        if uploaded_file:
            self.process_file(uploaded_file)
            self.render_results(thresh, kernel)

# --- Entry Point ---
if __name__ == "__main__":
    app = SeizureApp()
    app.run()
```

Лістинг Б.5 – Код файлу “eeg_processor.py” обробки сигналів в додатку

```
import mne
import numpy as np
import os
import gudhi as gd
from mne.preprocessing import ICA
from dataclasses import dataclass, field
from typing import List, Tuple, Optional, Dict

# --- Configuration Class ---
@dataclass
class EEGConfig:
    """Holds constant configuration for EEG processing."""
    standard_channels: List[str] =
        field(default_factory=lambda: [
        'FP1-F7', 'F7-T7', 'T7-P7', 'P7-O1',
        'FP1-F3', 'F3-C3', 'C3-P3', 'P3-O1',
        'FP2-F4', 'F4-C4', 'C4-P4', 'P4-O2',
        'FP2-F8', 'F8-T8', 'T8-P8', 'P8-O2',
        'FZ-CZ', 'CZ-PZ'
    ])

    channel_alias: Dict[str, str] =
        field(default_factory=lambda: {
        'T3': 'T7', 'T4': 'T8', 'T5': 'P7', 'T6': 'P8'
    })

    freq_bands: Dict[str, Tuple[float, float]] = field(
        default_factory=lambda: {
            "delta": (0.5, 4), "theta": (4, 8),
            "alpha": (8, 13), "beta": (13, 30),
            "gamma": (30, 50)
        }
    )

    epoch_duration: int = 5
```

```
    sfreq_min: float = 1.0
    sfreq_max: float = 50.0

# --- Topological Analysis Class ---
class TDAExtractor:
    """Handles Topological Data Analysis"""

    def __init__(self, grid_size: int = 100):
        self.grid = np.linspace(0, 1, grid_size)

    def _compute_persistence_diagram(self, point_cloud):
        rips_complex = gd.RipsComplex(
            points=point_cloud,
            max_edge_length=1
        )
        simplex_tree = rips_complex
            .create_simplex_tree(max_dimension=2)
        persistence_diagram = simplex_tree.persistence()

        result = []
        for dim, (birth, death) in persistence_diagram:
            if death == float('inf'): continue
            if dim != 1: continue # Only H1 (cycles)
            result.append((dim, (birth, death)))
        return result

    def _compute_landscape_values(self, diag):
        landscape_values = np.zeros_like(self.grid)
        for _, (birth, death) in diag:
            for i, t in enumerate(self.grid):
                if birth < t < death:
                    val = min(t - birth, death - t)
                    landscape_values[i] = max(
                        landscape_values[i], val)
        return landscape_values

    def extract(self, psd_data: np.ndarray) -> np.ndarray:
        """
        Extracts TDA features from PSD data.
        psd_data shape: (n_epochs, n_channels, n_freqs)
        """
        # Log transform and Normalize
        psd_log = 10 * np.log10(psd_data + 1e-20)
        p_min = psd_log.min(axis=(1, 2), keepdims=True)
        p_max = psd_log.max(axis=(1, 2), keepdims=True)
        psd_norm = (psd_log - p_min) / (p_max - p_min + 1e-10)

        features_list = []
        for epoch_idx in range(len(psd_norm)):
            point_cloud = psd_norm[epoch_idx]
            diag = self._compute_persistence_diagram(
                point_cloud)
            landscape = self._compute_landscape_values(diag)
```

```
            features_list.append(landscape)

        return np.array(features_list)

# --- Preprocessing Class ---
class EEGPreprocessor:
    """Handles loading, cleaning, channel standardization and
       ICA."""

    def __init__(self, config: EEGConfig):
        self.config = config
        self.ica = ICA(
            n_components=15,
            random_state=42,
            max_iter="auto"
        )

    def _standardize_channels(self, raw: mne.io.Raw):
        rename_map = {}
        current_chans = raw.ch_names

        for ch in current_chans:
            clean_name = ch.upper().strip()
            clean_name = clean_name.replace('-0', '')
            clean_name = clean_name.replace('-REF', '')
            clean_name = clean_name.replace('-Ref', '')

            for old, new in self.config.channel_alias.items():
                clean_name = clean_name.replace(old, new)

            if clean_name in self.config.standard_channels:
                rename_map[ch] = clean_name

        if rename_map:
            raw.rename_channels(rename_map)

        # Check completeness
        missing = [
            ch for ch in self.config.standard_channels
            if ch not in raw.ch_names
        ]
        if missing:
            print(f"Missing channels: {missing}")
            return None

        raw.pick_channels(self.config.standard_channels)
        return raw

    def apply_ica(self, raw: mne.io.Raw) -> mne.io.Raw:
        try:
            self.ica.fit(raw, verbose=False)
            return self.ica.apply(raw, verbose=False)
        except Exception as e:
```

```
            print(f"ICA Failed: {e}. Returning raw.")
            return raw

    def load_and_clean(self, file_path: str):
        try:
            raw = mne.io.read_raw_edf(
                file_path,
                preload=True,
                verbose=False
            )
            raw = self._standardize_channels(raw)
            if raw is None: return None

            raw.filter(
                l_freq=self.config.sfreq_min,
                h_freq=self.config.sfreq_max,
                verbose=False
            )
            raw = self.apply_ica(raw)
            return raw
        except Exception as e:
            print(f"Error loading {file_path}: {e}")
            return None

# --- Main Pipeline Class ---
class EEGPipeline:
    """Orchestrates the conversion of EDF to Features."""

    def __init__(self):
        self.config = EEGConfig()
        self.preprocessor = EEGPreprocessor(self.config)
        self.tda_extractor = TDAExtractor()

    def _extract_spectral(self, epochs: mne.Epochs):
        sfreq = epochs.info['sfreq']
        psd = epochs.compute_psd(
            method="welch",
            n_fft=int(sfreq),
            fmin=0.5,
            fmax=50.0,
            verbose=False
        )
        psd_data = psd.get_data()
        freqs = psd.freqs

        spectral_features = []
        for epoch_psd in psd_data:
            epoch_feats = []
            for _, (fmin, fmax) in self.config
                .freq_bands.items():
                idx = np.logical_and(
                    freqs >= fmin, freqs < fmax)
                band_power = np.mean(epoch_psd[:, idx], axis=1)
```

```
                epoch_feats.append(band_power)
            spectral_features.append(
                np.concatenate(epoch_feats))

        spectral_features = 10 * np.log10(
            np.array(spectral_features) + 1e-20)
        return spectral_features, psd_data

    def _generate_labels(
        self,
        epochs: mne.Epochs,
        seizure_intervals: List[Tuple[float, float]]
    ):
        n_epochs = len(epochs)
        labels = np.zeros(n_epochs, dtype=int)

        if not seizure_intervals:
            return labels

        starts = epochs.events[:, 0] / epochs.info['sfreq']
        duration = self.config.epoch_duration

        for i in range(n_epochs):
            t_start = starts[i]
            t_end = t_start + duration
            for s_start, s_end in seizure_intervals:
                if (t_start < s_end) and (t_end > s_start):
                    labels[i] = 1
                    break
        return labels

    def process(
        self,
        edf_path: str,
        seizure_intervals: List[Tuple[float, float]] = None
    ):
        print(f"Processing {os.path.basename(edf_path)}...")

        # 1. Load & Clean
        raw = self.preprocessor.load_and_clean(edf_path)
        if raw is None: return None, None

        # 2. Epoch
        epochs = mne.make_fixed_length_epochs(
            raw,
            duration=self.config.epoch_duration,
            preload=True,
            verbose=False
        )
        if len(epochs) == 0: return None, None

        # 3. Spectral Features
        spec_feats, psd_data = self._extract_spectral(epochs)
```

```
        # 4. Topological Features
        tda_feats = self.tda_extractor.extract(psd_data)

        # 5. Time-domain stats
        epoch_data = epochs.get_data()
        line_length = np.sum(np.abs(np.diff(epoch_data,
            axis=2)), axis=2)
        variance = np.var(epoch_data, axis=2)

        # 6. Combine
        all_features = np.hstack((
            spec_feats,
            tda_feats,
            line_length,
            variance
        ))

        # 7. Labels
        labels = self._generate_labels(epochs,
            seizure_intervals or [])

        return all_features, labels

# Wrapper function for backward compatibility
def process_edf_file(edf_path, seizure_intervals):
    pipeline = EEGPipeline()
    return pipeline.process(edf_path, seizure_intervals)
```