Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: «Розробка програмного застосунку для класифікації множини 3D точок»

Виконав: студент 4 курсу
групи 4ПІ-21б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Гуменюк О. В.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Рейда О. М.
(прізвище та ініціали)
Рецензент: к.т.н., доц. каф. ОТ Томчук М. А.
(прізвище та ініціали)

Допущено до захисту

Зав. кафедри ПЗ Романюк О. Н.
«09» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший бакалаврський
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н._____
«21» березня 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Гуменюку Олександру Віталійовичу

1. Тема роботи – «Розробка програмного застосунку для класифікації множини 3D точок».

Керівник роботи: Рейда Олександр Миколайович, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. №97.

2. Строк подання студентом роботи 2 червня 2025.

3. Вихідні дані до роботи: алгоритми класифікації – Random Forest та K-NN; набори даних – MeshNet10, MeshNet40; методи візуалізації – Matplotlib; мова програмування – Python; середовище розробки – Visual Studio Code.

4. Зміст текстової частини: вступ; аналіз сучасного стану питання та обґрунтування завдання на роботу; розробка структури та алгоритмів роботи програмного застосунку; розробка програмних модулів реалізації застосунку; тестування застосунку; висновки; список використаних джерел; додатки.

5. Перелік ілюстративної частини: титульний слайд; актуальність; мета і завдання дослідження; предмет дослідження; об'єкт дослідження; задачі

дослідження; наукова новизна; практичне значення отриманих результатів, аналіз аналогів; методи та засоби розробки; тестування.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
	Рейда О. М. к.т.н., доцент кафедри ПЗ	24.03.25	01.06.25

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз сучасного тану питання та обґрунтування завдання на роботу	25.03.25 – 30.03.25	вип
2	Розробка структури та алгоритмів роботи програмного застосунку	31.03.25 – 16.04.25	вип
3	Розробка програмних модулів реалізації застосунку	17.04.25 – 07.05.25	вип
4	Аналіз і обґрунтування вибору засобів реалізації застосунку	08.05.25 – 24.05.25	вип
5	Тестування застосунку	25.05.25 – 27.05.25	вип
6	Оформлення матеріалів до захисту БКР	28.05.25 – 30.05.25	вип

Студент _____ Гуменюк О.В.
(підпис) (прізвище та ініціали)

Керівник бакалаврської кваліфікаційної роботи _____ Рейда О. М.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

УДК 004.4:519.688

Гуменюк О. В. Розробка програмного застосунку для класифікації множини 3D-точок: бакалаврська кваліфікаційна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2025 – 60 с.

Бакалаврська кваліфікаційна робота складається з 60 сторінок формату А4, 24 рисунків, 5 таблиць. Список використаних джерел містить 21 найменування.

У бакалаврській кваліфікаційній роботі проведено аналіз стану питання розробки програмного застосунку для класифікації множини 3D-точок. Було проведено аналіз аналогів, визначено їх переваги і недоліки та доведено доцільність розробки власного застосунку.

Обґрунтовано вибір мови програмування та середовища розробки, а також технологій, які були використані під час розробки застосунка.

Розроблено програмний продукт, який являє собою застосунок для класифікації множини 3D-точок.

Додаток реалізований за допомогою мови програмування Python. В якості середовища для розробки програмного забезпечення було обрано Visual Studio Code.

Отримані в бакалаврській дипломній роботі результати можуть бути використані для вдосконалення відстеження процесу читання книг, включаючи сфери освіти та розваг.

Отримані в бакалаврській кваліфікаційній роботі результати можуть бути використані для вдосконалення процесів аналізу та обробки 3D-хмар точок, зокрема в галузях комп’ютерного зору, геоінформаційних систем, тривимірного моделювання та автоматизованої інспекції.

Ключові слова: класифікація, 3D-точки, машинне навчання.

ABSTRACT

UDC 004.4:519.688

Humenyuk O. V. Development of a software application for the classification of a set of 3D points: bachelor's qualification thesis in specialty 121 – Software Engineering, educational program – Software Engineering. Vinnytsia: VNTU, 2025 – 60 p.

The bachelor's qualification thesis consists of 60 A4 pages, 24 figures, and 5 tables. The list of references includes 21 sources. This thesis presents an analysis of the current state of software development for the classification of 3D point clouds. A review of existing solutions was conducted, their advantages and disadvantages were identified, and the feasibility of developing a custom application was justified.

The choice of programming language, development environment, and technologies used in the implementation of the application was substantiated.

A software product was developed – an application for classifying sets of 3D points.

The application was implemented using the Python programming language. Visual Studio Code was chosen as the development environment.

The results obtained in this bachelor's qualification thesis can be used to improve the processes of analyzing and processing 3D point clouds, particularly in the fields of computer vision, geographic information systems, 3D modeling, and automated inspection.

Keywords: classification, 3D points, machine learning.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Сучасний стан розвитку інформаційних технологій характеризується стрімким зростанням обсягів тривимірних даних, зокрема множин 3D-точок, які генеруються за допомогою передових технологій сканування, таких як LIDAR [1], фотограмметрія, структуроване світло та системи глибинного зору. Ці дані є основою для вирішення складних задач у таких галузях, як робототехніка, автономний транспорт, комп'ютерний зір, медична візуалізація, архітектурне проектування, віртуальна та доповнена реальність, а також геоінформаційні системи. Класифікація множин 3D-точок відіграє ключову роль у перелічений галузях, оскільки дозволяє ідентифікувати об'єкти, сегментувати сцени, створювати цифрові моделі та забезпечувати надійну основу для прийняття рішень у реальному часі. Однак обробка таких даних пов'язана з рядом проблем, зокрема великими обсягами інформації, наявністю шумів, неоднорідністю точкових хмар і необхідністю адаптації алгоритмів до різноманітних сценаріїв використання.

Розвиток методів машинного навчання, зокрема керованих і некерованих алгоритмів класифікації, відкриває нові можливості для ефективної обробки множин 3D-точок. Такі методи дозволяють автоматизувати аналіз даних, підвищувати точність класифікації та оптимізувати обчислювальні ресурси. Проте більшість сучасних рішень для класифікації 3D-точок мають обмеження, пов'язані з недостатньою інтерактивністю, складністю інтеграції в прикладні системи або відсутністю зручних засобів візуалізації результатів. Це зумовлює потребу в розробці ефективних, автоматизованих і масштабованих програмних рішень для класифікації множин 3D-точок, які відповідають вимогам швидкості, точності та зручності використання.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є підвищення точності і якості класифікації множини 3D-точок у процесі обробки хмари тривимірних даних з використанням методів машинного навчання.

Основними задачами дослідження є:

– провести аналіз існуючих методів і засобів класифікації множин 3D-точок для визначення їх переваг та недоліків;

– модифікувати метод класифікації множин 3D-точок для підвищення точності класифікації;

– розробити алгоритми класифікації множин 3D-точок на основі методів машинного навчання;

– розробити програмний застосунок з інтуїтивно зрозумілим інтерфейсом для обробки та класифікації 3D-точок;

– провести експериментальне тестування розробленого програмного застосунку для оцінки його ефективності та точності.

Об'єкт дослідження – процес розробки програмного застосунку для класифікації множини 3D-точок.

Предмет дослідження – методи та засоби для класифікації множин 3D-точок з використанням алгоритмів машинного навчання.

Методи дослідження. У процесі виконання роботи застосовувалися методи машинного навчання, зокрема алгоритми некерованої класифікації; методи обробки даних, такі як фільтрація та нормалізація; програмування для реалізації програмного застосунку; а також методи експериментального тестування для оцінки ефективності розроблених алгоритмів.

Новизна отриманих результатів. Модифіковано метод класифікації множин 3D-точок, який на відміну від існуючого, поєднує метод статистичної фільтрації для усунення «викидів» точок та машинне навчання за допомогою моделі «K-NN» та «RandomForest» для підвищення точності класифікації.

Практична цінність отриманих результатів. Розроблений програмний застосунок може бути використаний у різних галузях, таких як робототехніка, автономні системи, медична діагностика та геоінформаційні системи, для

автоматизації обробки та класифікації множин 3D-точок. Застосунок забезпечує зручний інтерфейс, високу точність та швидкість обробки даних, що сприяє оптимізації робочих процесів та підвищенню ефективності аналізу тривимірних даних.

Особистий внесок здобувача. Усі наукові результати, викладені у БКР, отримані автором особисто.

Апробація матеріалів бакалаврської кваліфікаційної роботи. Описані у дослідженні положення доповідались на конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)» та опубліковані в тезах доповіді.

Публікації. За тематикою дослідження опубліковано 1 наукову працю, що доповідалась на конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)» та опубліковано в тезах доповіді [2].

1 АНАЛІЗ СУЧАСНОГО СТАНУ ПИТАННЯ ТА ОБҐРУНТУВАННЯ ЗАВДАННЯ НА РОБОТУ

1.1 Аналіз стану систем класифікації 3D-даних

У сучасних умовах швидкого розвитку технологій обробки даних та машинного навчання класифікація множин 3D-точок є важливим завданням у багатьох галузях, таких як комп’ютерний зір, робототехніка, геоінформаційні системи та медична візуалізація. Аналіз та класифікація тривимірних даних дозволяють виявляти складні просторові структури, ідентифікувати об’єкти та прогнозувати їхні характеристики на основі геометричних і топологічних властивостей. Розробка програмного забезпечення для класифікації 3D-точок із застосуванням методів машинного навчання, таких як алгоритм K-NN [3] та Random Forest [4], забезпечує ефективну обробку великих обсягів даних з підвищеною точністю і швидкістю аналізу.

Основні методи класифікації 3D-даних включають алгоритми, що базуються на різних підходах до обробки просторових даних. Алгоритм K-NN є одним із найпростіших і найефективніших методів, що використовує відстань між точками у тривимірному просторі для визначення їхньої належності до певного класу. Цей метод характеризується високою інтуїтивністю та простотою реалізації, однак його продуктивність може знижуватися при обробці великих наборів даних через високу обчислювальну складність. Для підвищення точності класифікації в умовах шумних або складних даних застосовуються ансамблеві методи, зокрема алгоритм Random Forest. Random Forest формує множину дерев, кожне з яких аналізує випадкову підмножину ознак, що дозволяє ефективно обробляти складні набори 3D-даних із високою розмірністю та нелінійними залежностями. Цей метод є стійким до перенавчання та забезпечує високу точність завдяки агрегації результатів кількох дерев.

Інші методи класифікації 3D-даних включають алгоритми на основі нейронних мереж, зокрема CNN та PointNet [5], які спеціально адаптовані для обробки хмар точок. Ці методи демонструють високу ефективність у задачах, де

необхідно враховувати глобальні та локальні особливості просторових даних. Проте їхнє використання вимагає значних обчислювальних ресурсів і великих обсягів анотованих даних для навчання. Для порівняння, K-NN та Random Forest є менш вимогливими до ресурсів, що робить їх оптимальними для реалізації в програмних застосунках із обмеженими обчислювальними можливостями.

Розробка програмного забезпечення для класифікації множин 3D-точок із застосуванням K-NN та Random Forest забезпечує гнучкість і адаптивність до різноманітних сценаріїв використання. Такі системи дозволяють дослідникам і розробникам швидко аналізувати тривимірні дані, оптимізувати параметри алгоритмів та візуалізувати результати класифікації. Крім того, ці методи легко інтегруються з інструментами обробки даних, такими як бібліотеки Python, зокрема scikit-learn та NumPy, що спрощує їхнє впровадження в проєкти.

Отже, дослідження сучасних підходів до класифікації 3D-даних демонструє високий потенціал використання алгоритмів машинного навчання, зокрема методів K-NN і Random Forest, у задачах обробки тривимірної інформації. Ці підходи поєднують точність, обчислювальну ефективність і відносну простоту впровадження, що робить їх придатними для реалізації продуктивних програмних рішень. Подальший розвиток таких систем передбачає впровадження новітніх технологій обробки даних, оптимізацію алгоритмів для роботи з великими масивами інформації та підвищення гнучкості щодо викликів, пов’язаних із сучасним аналізом просторових структур.

1.2 Порівняльний аналіз аналогів

Із розвитком тривимірного моделювання та застосування методів машинного навчання в обробці просторових даних зростає потреба у спеціалізованих програмних засобах для класифікації 3D-хмар точок. Такі програмні інструменти активно використовуються у галузях геоінформаційних систем, комп’ютерного бачення, робототехніки та інженерії. Вони дозволяють проводити попередню обробку, візуалізацію, а також навчання моделей класифікації, однак не всі вони

забезпечують повний спектр функціональності, необхідної для дослідницьких та практичних завдань.

Серед програмних рішень, що реалізують подібний функціонал, варто відзначити наступні аналоги: CloudCompare, Open3D, Point Data Abstraction Library та Potree. Кожен з цих інструментів має переваги та недоліки, але жоден не надає комплексного рішення з підтримкою повного циклу роботи з даними – від їх завантаження до збереження результатів. Це надає змогу для розробки нового програмного застосунку, який не буде мати конкурентів серед аналогів – 3D Point Cloud Classifier v1.0, що поєднує в собі функції, які раніше вимагали використання декількох інструментів одночасно.

На рисунку 1.1 зображено застосунок CloudCompare [6], який є потужним інструментом для обробки та аналізу 3D-хмар точок. Програма дозволяє виконувати широкий спектр операцій, включаючи візуалізацію, реєстрацію, порівняння поверхонь та класифікацію просторових даних.

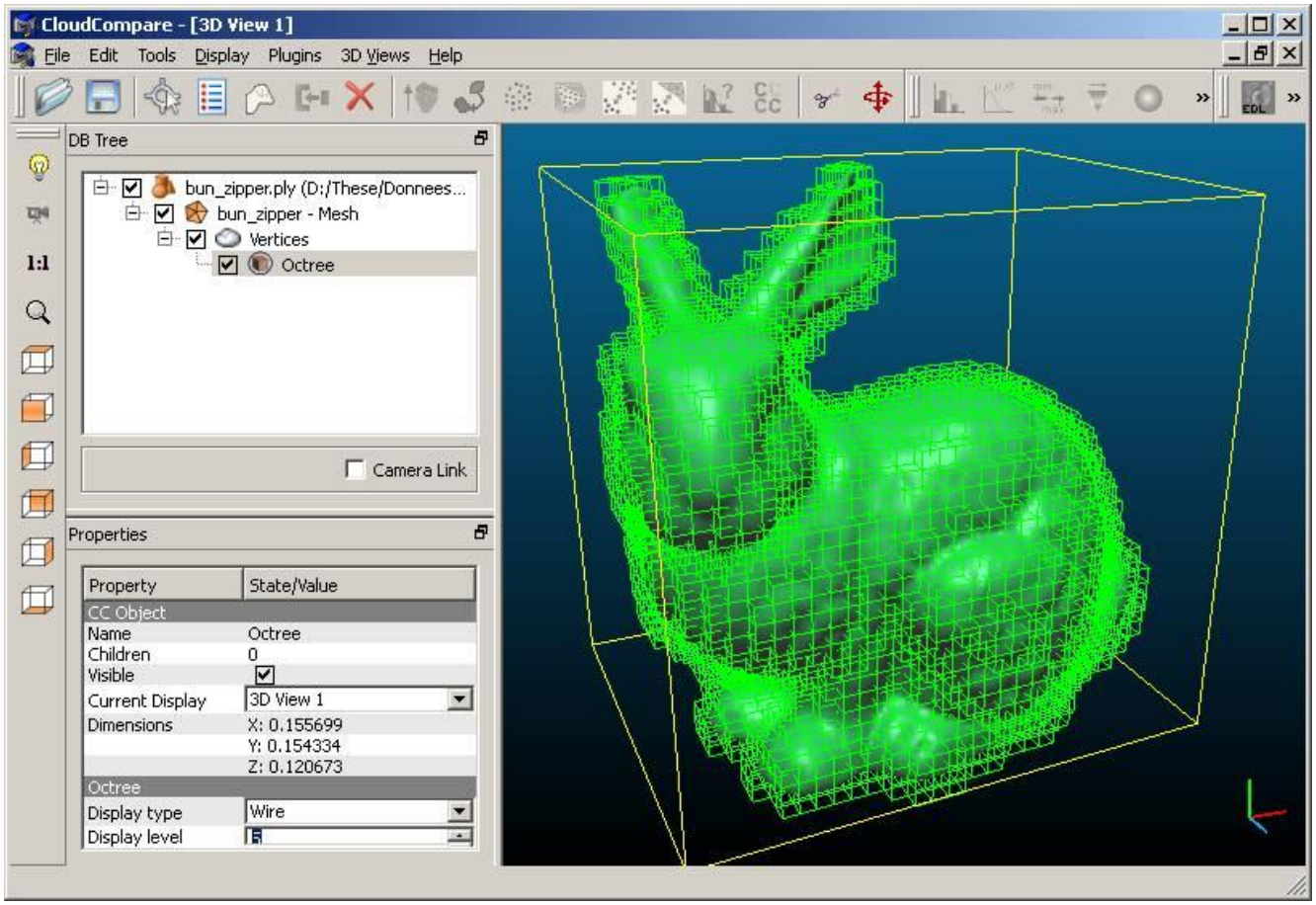


Рисунок 1.1 – Інтерфейс CloudCompare з візуалізацією хмари точок

CloudCompare підтримує обробку великих наборів даних з високою щільністю, а також має відкритий код, що дозволяє модифікувати функціональність відповідно до потреб користувача. Завдяки своїй гнучкості та розширюваності за допомогою плагінів, програма часто використовується у геодезії, архітектурі та наукових дослідженнях. Проте, незважаючи на велику кількість інструментів, інтерфейс CloudCompare є доволі складним для новачків, а сама система не забезпечує повноцінного циклу роботи з даними – від завантаження до класифікації й збереження результатів, для цього потрібно залучення сторонніх програмних рішень.

На рисунку 1.2 зображено програмне середовище Open3D [7], яке призначене для обробки тривимірних даних, зокрема 3D-хмар точок, сіток, воксельних представлень та моделей. Це програмне забезпечення з відкритим кодом, орієнтоване переважно на дослідників, розробників і інженерів, що працюють у галузі комп'ютерного зору та 3D-реконструкції.

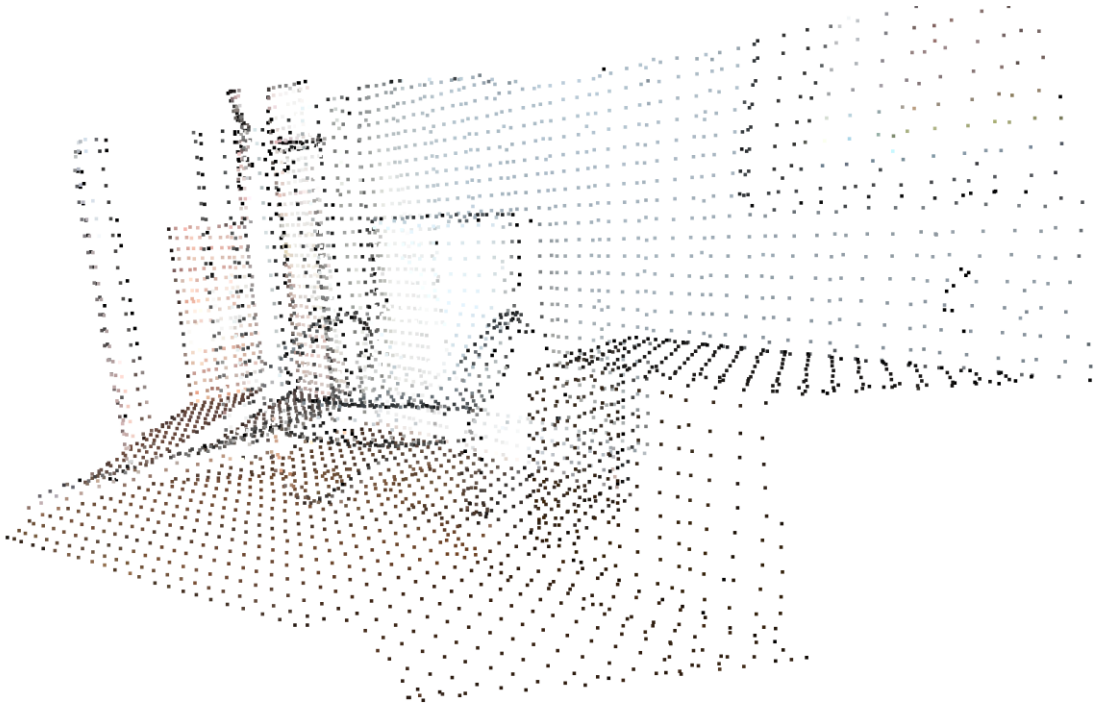


Рисунок 1.2 – Відображення хмари точок в Open3D

Open3D забезпечує користувача базовими й просунутими алгоритмами для обробки просторових даних, серед яких — нормалізація, зменшення щільності, реєстрація, фільтрація шумів, побудова поверхонь і класифікація. Завдяки підтримці мов Python та C++, бібліотека легко інтегрується в наукові проєкти та використовується для машинного навчання. Однак, як бібліотека, Open3D не надає готового інтерфейсу користувача і вимагає навичок програмування для створення власних рішень. Таким чином, її використання не є зручним для користувачів без навичок в програмуванні, що обмежує її використання в задачах із швидкою обробкою та візуалізацією даних без написання коду.

На рисунку 1.3 представлено програмне рішення PDAL [8], яке використовується для обробки та трансформації великих обсягів геопросторових 3D-хмар точок. PDAL є бібліотекою з відкритим кодом і виконує роль програмного фреймворку для побудови конвеєрів обробки даних, що робить її потужним інструментом у сфері геоінформатики та обробки LIDAR-даних.

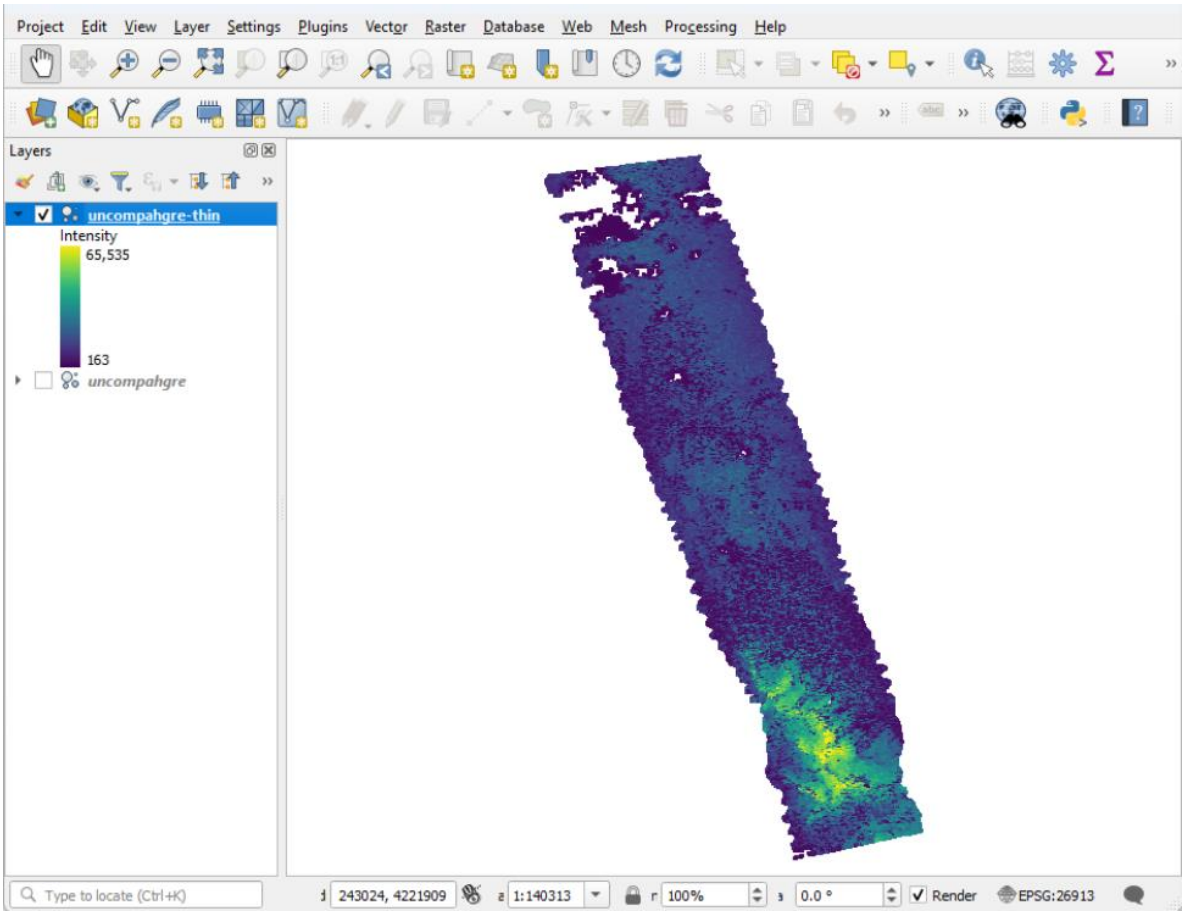


Рисунок 1.3 – Приклад обробки даних у PDAL

PDAL підтримує читання, фільтрацію, трансформацію, аналіз і запис хмар точок у різноманітних форматах, зокрема LAS/LAZ, E57, GeoTIFF та інших. Однією з ключових переваг PDAL є можливість автоматизованої пакетної обробки даних за допомогою конфігураційних файлів у форматі JSON, що дозволяє створювати гнучкі та відтворювані робочі процеси. Проте PDAL не має власного графічного інтерфейсу й орієнтована здебільшого на розробників і досвідчених користувачів, які володіють навичками командного рядка та знаннями форматів просторових даних. Це обмежує її застосування у проектах, де необхідна інтерактивна візуалізація чи швидкий перегляд результатів класифікації.

На рисунку 1.4 зображено застосунок Potree [9], який є інструментом для веб-візуалізації великих 3D-хмар точок. Potree заснований на WebGL і JavaScript, що дозволяє завантажувати та інтерактивно досліджувати просторові дані безпосередньо в браузері, не потребуючи встановлення додаткового програмного забезпечення.

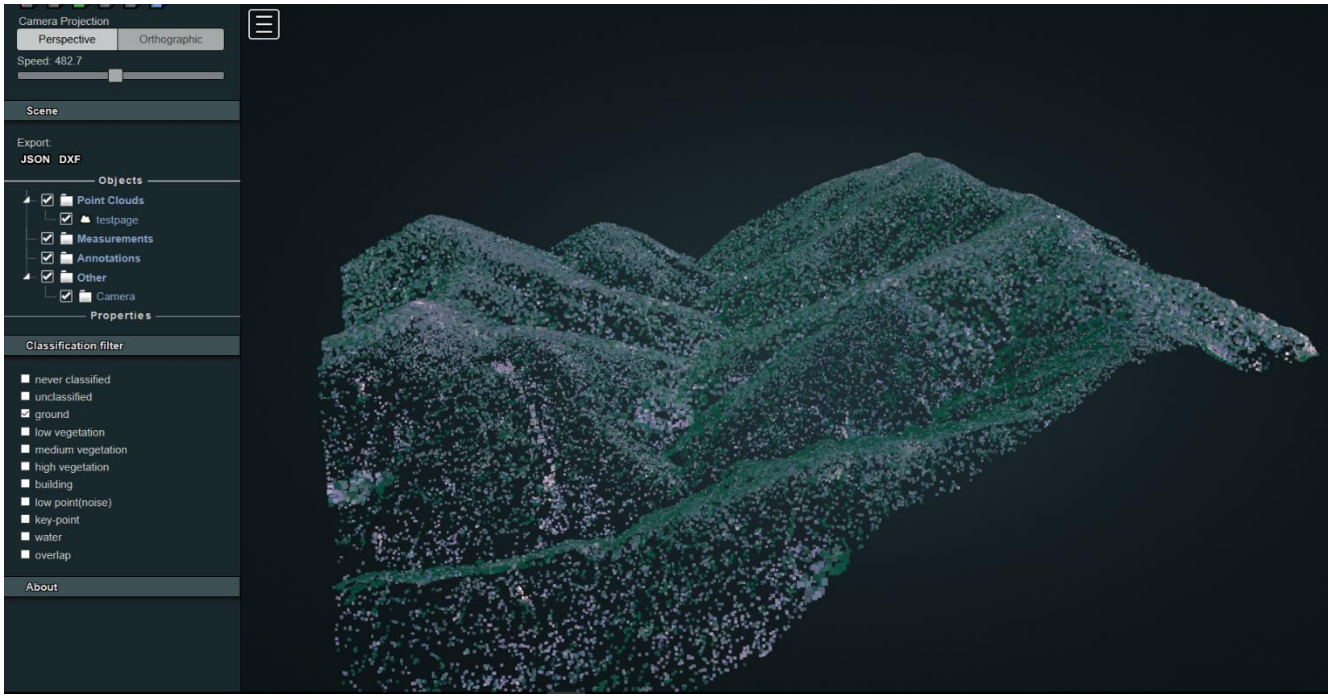


Рисунок 1.4 – Інтерфейс перегляду хмари точок у Potree

Однією з ключових переваг Potree є його здатність ефективно працювати з мільйонами точок, забезпечуючи плавну навігацію, масштабування та

вимірювання в тривимірному просторі. Інструмент підтримує налаштування відображення, кольорову сегментацію за висотою, інтенсивністю або іншими атрибутами, а також дозволяє робити анотації та вимірювання безпосередньо в сцені. Однак Potree орієнтований насамперед на візуалізацію, а не на глибоку обробку або класифікацію даних. Для підготовки вихідних даних часто необхідно використовувати сторонні утиліти, як-от PDAL або CloudCompare, що знижує рівень інтеграції та автономності системи.

Ці рішення демонструють, що є потреба в комплексному та зручному у використанні застосунку, який повинен охоплювати усі етапи роботи з хмарами 3D-точок – від їх завантаження та попередньої обробки до класифікації й збереження результатів. Розробка власного застосунку є актуальною темою, застосунок покликаний стати інтегрованим рішенням, що буде поєднувати переваги наявних інструментів, одночасно спрощуючи і роблячи їх доступними для користувачів з різним рівнем теоретичних та практичних знань. У результаті аналізу аналогів було створено таблицю 1.1 для порівняння їхнього функціоналу з розроблюваним програмним забезпеченням.

Таблиця 1.1 – Порівняльний аналіз характеристик програмного забезпечення

	CloudCompare	Open3D	PDAL	Potree	Власна розробка
Попередня обробка точок	1	1	1	1	1
Навчання моделей ML	0	1	0	0	1
Підтримка форматів off, ply, xyz, pcd, obj	1	1	1	1	1
3D-візуалізація	1	1	0	1	1
Аналіз навченої моделі	0	0	0	0	1

Продовження таблиці 1.1

Збереження ML-моделі	0	1	0	0	1
Завантаження ML-моделі	0	1	0	0	1
Сумарний коефіцієнт	42%	85%	28%	42%	100%

Згідно з результатами порівняльного аналізу, наведеного в таблиці 1.1, власна розробка програмного забезпечення для класифікації 3D-хмар точок має помітні переваги над існуючими аналогами – CloudCompare, Open3D, PDAL та Potree.

Рішення охоплює всі основні функціональні критерії на 100%, тоді як інші програмні продукти мають певні обмеження. Найкращий результат серед існуючих рішень показує Open3D – 85%, адже ця бібліотека пропонує широкі можливості для машинного навчання та 3D-візуалізації, проте не має зручного користувацького інтерфейсу. CloudCompare та Potree отримали по 42%, оскільки зосереджені переважно на 3D-візуалізації та наявності інтерфейсу, але не підтримують машинне навчання й класифікацію. PDAL, з показником 28%, в першу чергу призначений для автоматизованої обробки великих геопросторових даних, але не забезпечує інтерактивної роботи з візуалізацією.

Власна розробка поєднує в собі всі ключові функції: зручний і зрозумілий інтерфейс, можливості для навчання та класифікації моделей машинного навчання, 3D-візуалізацію, аналіз навченої моделі, а також збереження й завантаження ML-моделей. Завдяки цьому, вона підходить як для досвідчених користувачів, так і для тих, хто тільки починає працювати з 3D-хмарами.

Отже, результати порівняльного аналізу підтверджують, що створення власного програмного продукту є актуальним та необхідним для забезпечення повного циклу роботи з 3D-хмарами точок – від їхньої обробки до класифікації і збереження результатів.

1.3 Аналіз методів розв’язання задачі

Задача автоматичної класифікації об'єктів у 3D-хмарах точок сьогодні є актуальною у багатьох сферах: від картографії й архітектури до робототехніки й доповненої реальності. Визначити, що саме зображено у вигляді 3D-моделі – чи то дерево, автомобіль, пляшка, чи ліжко – є складним завданням, адже дані можуть бути неоднорідними, зашумленими та неповними. Саме тому для розв’язання цієї задачі важливо підібрати надійні методи машинного навчання, які зможуть працювати зі складними просторовими даними.

У межах даного програмного продукту було вирішено зосередитись на двох класичних, але перевірених підходах – алгоритмі K-NN та методі Random Forest. Обидва ці підходи мають свої переваги та дозволяють виконувати класифікацію без необхідності занурення в складні нейронні мережі на кшталт PointNet, що потребують більше ресурсів і складної підготовки даних.

Алгоритм K-NN є одним із найпростіших методів у машинному навчанні, проте, не зважаючи на це, він показує непогані результати для задач класифікації, де форма об'єктів і просторове положення точок мають значення. Суть його роботи полягає в тому, що для кожної нової точки або набору точок знаходяться найближчі по ознакам з тих, що вже були помічені під час навчання. Далі система просто визначає до якого класу належить більшість сусідів, і на цій основі приймає рішення. У випадку з 3D-хмарами точок K-NN може аналізувати координати, нормалі до поверхні, колір або інші характеристики, які вдається витягнути з просторової моделі.

K-NN має кілька сильних сторін. По-перше, він інтуїтивно зрозумілий і легко реалізується. По-друге, він не вимагає довгого навчання, адже запам’ятовує всі приклади, з якими працював. Але є й недоліки: алгоритм повільно працює з великими обсягами даних, а також чутливий до шуму – якщо в навчальному наборі багато помилок або випадкових точок, це може негативно вплинути на якість класифікації.

Для того щоб підвищити точність і стійкість системи, було також використано метод Random Forest. Це більш складний алгоритм, що базується на

ідеї ансамблю рішень. Простіше кажучи, замість одного дерева рішень, яке класифікує об'єкт, створюється багато таких дерев, кожне з яких обирає варіант, який вважає правильним. Остаточне рішення приймається на основі більшості, тобто обирається варіант, який обрало більше дерев. Завдяки цій структурі Random Forest не так сильно реагує на шум у даних і краще узагальнює нову інформацію.

У порівнянні з K-NN, Random Forest потребує більше часу на навчання, але натомість швидше працює на етапі класифікації. Цей метод також дозволяє зрозуміти, які саме характеристики мають найбільше значення при прийнятті рішень, що робить його хорошим інструментом для аналізу даних. У застосунку реалізовано можливість зберігати вже натреновану модель, а також завантажувати її повторно без необхідності проводити навчання щоразу.

Особливість запропонованої реалізації полягає в тому, що обидва алгоритми інтегровані у зручний графічний інтерфейс. Завдяки цьому застосунок стає доступним не лише для фахівців з машинного навчання, а й для користувачів з базовими знаннями, які хочуть працювати з 3D-даними у зручний спосіб.

Крім навчання і класифікації, програма дозволяє переглядати результат у 3D-форматі та аналізувати, як саме була класифікована кожна точка. Це значно полегшує налагодження та перевірку моделі, адже результат видно не у вигляді таблиці, а безпосередньо у візуальному середовищі.

Загалом, вибір K-NN та Random Forest був зроблений на користь перевірених рішень, які легко реалізувати, пояснити й адаптувати до нових умов. У контексті задачі класифікації 3D-хмар точок вони дозволяють досягати високої точності без потреби у великих обчислювальних ресурсах. І хоча в майбутньому не виключається можливість додавання більш складних підходів, на сьогодні ці методи забезпечують збалансовану комбінацію простоти, надійності та ефективності.

Отже, розроблений застосунок спирається на методи, що вже добре зарекомендували себе в задачах класифікації, й водночас пропонує зручний інтерфейс, що відкриває можливості для широкого кола користувачів. Це дозволяє

не лише вирішувати технічну задачу, а й робити роботу з просторовими даними більш доступною й зрозумілою.

1.4 Постановка задач дослідження

Метою даного дослідження є розробка застосунку для класифікації множини 3D-точок із використанням алгоритмів машинного навчання, зокрема Random Forest та K-NN. Програмне забезпечення має забезпечити повний цикл роботи з тривимірними хмарами точок: від завантаження даних і навчання моделі до класифікації нових наборів та візуалізації результатів.

Розробка відбувається з використанням мови програмування Python у середовищі Visual Studio Code, з використанням бібліотек scikit-learn для реалізації алгоритмів машинного навчання, Matplotlib для ефективної візуалізації, а також Tkinter для створення графічного інтерфейсу користувача. Додатково використовуються NumPy та Pandas для обробки та трансформації даних, що значно спрощує маніпуляції з великими масивами 3D-точок.

Створений застосунок повинен забезпечувати наступну функціональність:

1. Завантаження хмар точок у форматах .OFF, .PLY, .XYZ, .PCD, .OBJ;

2. Навчання моделей класифікації з використанням Random Forest та KNN на наданих наборах даних;

3. Класифікація нових наборів точок за допомогою збережених моделей;

4. Візуалізація результатів класифікації у 3D, що дозволяє користувачу інтуїтивно оцінити якість роботи алгоритмів;

5. Збереження навчених моделей у зручному для подальшого використання форматі;

6. Завантаження попередньо збережених моделей, що дає змогу швидко застосовувати їх для дослідження;

7. Аналіз навченої моделі, включаючи відображення точності та матриці неточностей.

Для досягнення поставленої мети необхідно реалізувати наступні завдання:

– провести аналіз існуючих методів і засобів класифікації множин 3D-точок для визначення їх переваг та недоліків;

– модифікувати метод класифікації множин 3D-точок для підвищення точності класифікації;

– розробити алгоритми класифікації множин 3D-точок на основі методів машинного навчання;

– розробити програмний застосунок з інтуїтивно зрозумілим інтерфейсом для обробки та класифікації 3D-точок;

– провести експериментальне тестування розробленого програмного застосунку для оцінки його ефективності та точності.

Таким чином, розробка програмного забезпечення для класифікації 3D-хмар точок із використанням алгоритмів машинного навчання передбачає використання мови програмування Python у середовищі розробки Visual Studio Code та бібліотек Tkinter, scikit-learn і Matplotlib, що являє собою комплексне завдання, що складається з кількох етапів. Застосунок буде корисним для досліджень у галузі комп’ютерного зору, робототехніки, геопросторового аналізу та інших напрямків, де активно використовуються 3D-дані.

1.5 Висновки

У першому розділі було проведено аналіз існуючих програмних рішень для класифікації множин 3D-точок, зосереджуючись на вивчені таких аналогів, як CloudCompare, Open3D, Point Data Abstraction Library та Potree. Дослідження дозволило виявити, що ці інструменти мають деякі переваги, але все ж демонструють суттєві недоліки, які обмежують їх ефективність для вирішення поставленої задачі.

Проведений порівняльний аналіз підтвердив потребу у власному програмному застосунку для класифікації 3D-точок з використанням Python у середовищі Visual Studio Code з бібліотеками Tkinter, scikit-learn і Matplotlib. Такий застосунок об’єднає найкращі функції існуючих рішень та вдосконалить, реалізувавши їх через через зручний графічний інтерфейс.

2 РОЗРОБКА СТРУКТУРИ ТА АЛГОРИТМІВ РОБОТИ ПРОГРАМНОГО ЗАСТОСУНКУ

2.1 Аналіз вхідних даних

У процесі розробки програмного застосунку для класифікації тривимірних даних було використано відкриті набори даних з колекції MeshNet10 та MeshNet40. Ці набори даних спеціалізуються на 3D-моделях, поданих у форматі хмар точок, що охоплюють різноманітні об’єкти з різною кількістю класів, загальна кількість класів налічує 40 об'єктів. Такий підхід дозволяє протестувати ефективність класифікації у випадках з різним рівнем складності та деталізації.

Основний формат збереження даних – OFF, який є широко вживаним для представлення 3D-геометрії. Він містить інформацію про вершини координати об'єкта, X, Y та Z. Окрім OFF, розроблюваний застосунок також підтримує імпорт файлів у форматах PLY, XYZ, PCD та OBJ, що забезпечує гнучкість та розширює потенційні сценарії використання програми. Таким чином, користувач може працювати з 3D-даними з різних джерел без необхідності конвертації.

У вхідному файлі переважно зберігається лише базова інформація про геометрію – тривимірні координати точок X, Y та Z. Ці дані виступають основою для побудови ознак, які використовуються моделями машинного навчання. Інші атрибути, такі як колір, нормалі або текстури, в межах цієї реалізації не використовуються, що дозволяє зосередитись саме на геометричних характеристиках.

Перед подачею даних до моделей класифікації, виконується базова попередня обробка. По-перше, координати кожної точки нормалізуються, щоб масштаб об’єктів не впливав на точність класифікації. Це дозволяє зробити аналіз незалежним від абсолютного розміру моделі. По-друге, здійснюється видалення шуму, що дозволяє покращити якість вхідних даних і знизити вплив сторонніх або непотрібних для класифікації точок. Для забезпечення коректного навчання алгоритмів, дані було розділено на навчальну та тестову вибірки, що дає змогу

провести оцінку якості класифікації за допомогою стандартних метрик точності та повноти.

Отже, аналіз вхідних даних показав, що обраний набір даних є зручними для дослідження у сфері 3D-класифікації завдяки простій структурі, достатній кількості об’єктів і добре підготовленій анотації. Незважаючи на обмеженість атрибутів, а саме: відсутність текстур чи колірної інформації, навіть базові координати дозволяють досягати хорошого рівня класифікації за умови якісної обробки та налаштування моделей.

2.2 Моделювання архітектури застосунку

Під час проєктування програмного застосунку для класифікації 3D-даних особлива увага була приділена структурі системи, модульності коду та гнучкості у розширенні функціональності. Архітектура застосунку була спроєктована таким чином, щоб забезпечити зручність для кінцевого користувача, незалежність окремих компонентів та ефективну взаємодію між модулями.

Застосунок реалізовано мовою програмування Python з використанням бібліотек scikit-learn для машинного навчання, NumPy та Pandas для обробки числових даних, а також Tkinter для побудови графічного інтерфейсу користувача. Всі основні компоненти поділені на логічні модулі: обробка вхідних даних, візуалізація, навчання моделей, класифікація та збереження результатів.

Кожен модуль реалізує окрему функціональність, що полегшує тестування, налагодження та можливість повторного використання коду. Такий підхід сприяє підвищенню стабільності роботи системи та забезпечує можливість масштабування проєкту в майбутньому. Завдяки використанню Tkinter користувачі можуть взаємодіяти із застосунком у зрозумілому та інтуїтивному середовищі, не потребуючи глибоких технічних знань.

На рисунку 2.1 зображено блок-схему роботи застосунку, що зображає основні етапи обробки та взаємодію між складовими. На схемі показано усі кнопки та функції які вони викликають.

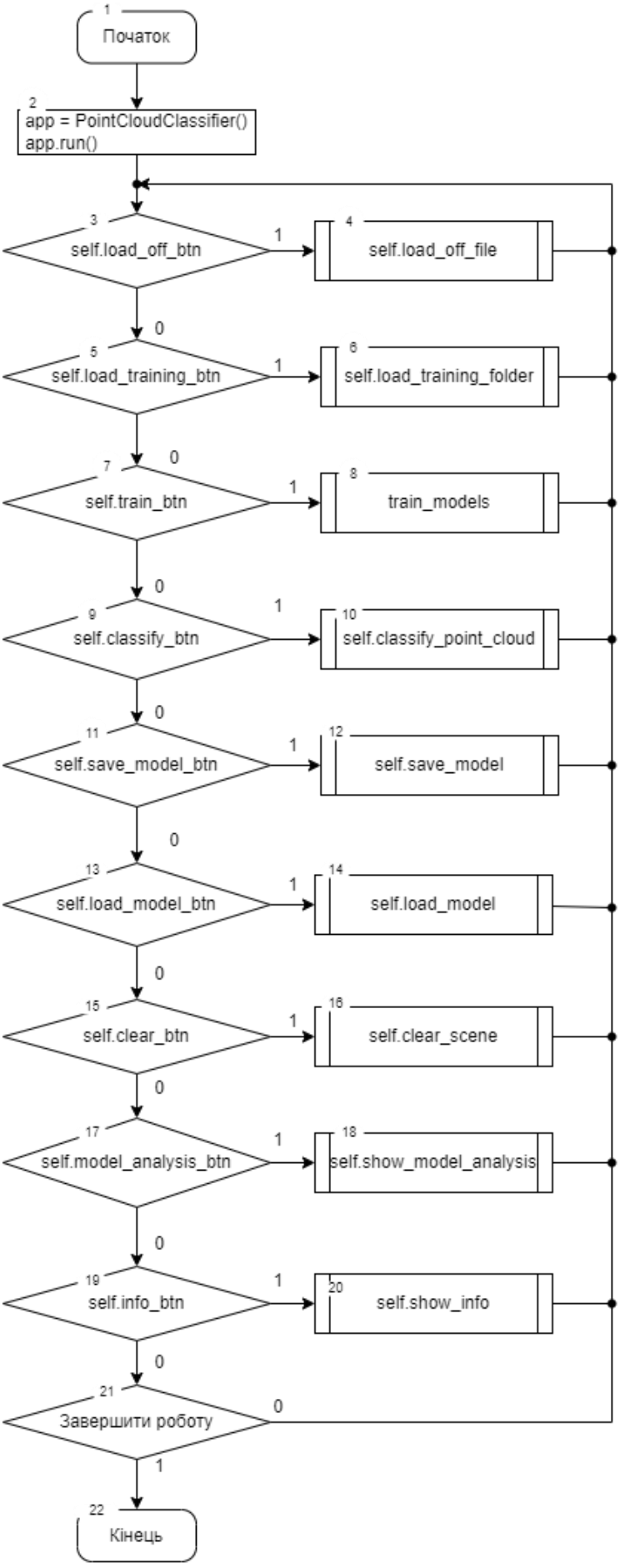


Рисунок 2.1 – Блок-схема роботи застосунку

Основні етапи роботи застосунку включають:

1. Імпорт вхідного файлу – користувач обирає файл одного з підтримуваних форматів (.OFF, .PLY, .XYZ, .OBJ, .PCD), після чого система зчитує координати точок і формує хмару.

2. Попередня обробка даних – реалізовано нормалізацію координат та очищення хмари від шуму. Це забезпечує покращення якості подальшої класифікації.

3. Навчання моделей – застосовуються алгоритми K-NN та Random Forest, які навчаються на одному і тому ж наборі даних. Після завершення навчання обидві моделі зберігаються у спільному файлі, що оптимізує доступ до них у подальших сесіях.

4. Класифікація нових даних – користувач може завантажити новий файл, після чого застосунок виконує класифікацію за обома моделями та дозволяє порівняти результати.

5. Візуалізація та аналіз – передбачено графічне відображення результатів класифікації у вигляді кольорової 3D-хмари точок. Користувач може переглядати статистику, зберігати результати та експортувати їх.

Розроблена архітектура дозволяє масштабувати функціональність наприклад, додати нові алгоритми, розширити підтримку форматів або інтегрувати обробку текстурованих моделей. Завдяки чіткому розподілу ролей між модулями забезпечується стабільність роботи та зручність у тестуванні й супроводі застосунку.

2.3 Розробка підсистеми для формування ознак множини 3D-точок

У рамках створення програмного забезпечення для класифікації 3D-хмар точок важливу роль відіграє етап попередньої обробки даних, зокрема витягнення інформативних ознак. Підсистема, що реалізує даний етап, відповідає за перетворення необробленої множини тривимірних точок у набір числових характеристик, які будуть використані у подальших етапах машинного навчання, таких як класифікація або кластеризація.

Вхідними даними для підсистеми є масив тривимірних координат точок, що утворюють певний об'єкт або фрагмент сцени. Основним завданням є побудова вектору ознак, який максимально точно описує просторову структуру вхідних даних. З цією метою у межах розробки було створено функцію, яка виконує послідовну обробку вхідної множини точок.

На рисунку 2.2 зображено блок-схему розроблюваної підсистеми, яка демонструє етапи аналізу хмари точок та побудови вектору ознак.

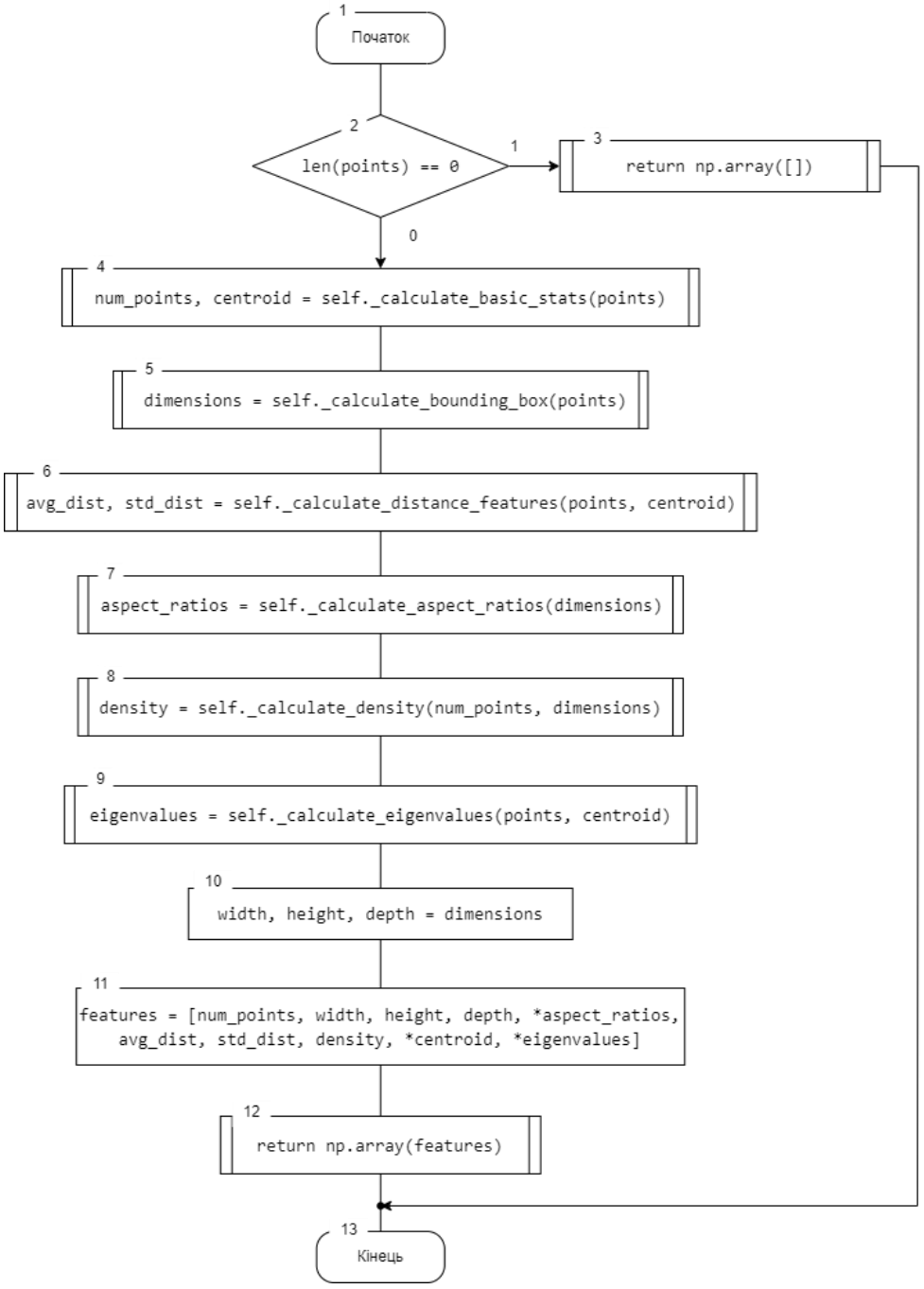


Рисунок 2.2 – Блок-схема підсистеми для формування ознак множини 3D-точок

Етапи роботи підсистеми включають:

1. Попередню перевірку вхідних даних на їх наявність, що дозволяє уникнути помилок при обробці.

2. Обчислення базових статистик, таких як кількість точок у хмарі та координати центроїда.

3. Оцінку просторових характеристик за допомогою розрахунку розмірів bounding box, який описує найменший контейнер, що охоплює всі точки.

4. Обчислення відстаней від кожної точки до центроїда та визначення їхнього середнього значення і стандартного відхилення. Ці характеристики дозволяють оцінити компактність або розкиданість хмари.

5. Аналіз пропорційності хмари шляхом обчислення відношень ширини до висоти, ширини до глибини та висоти до глибини.

6. Визначення щільності хмари точок, що розраховується як кількість точок на одиницю об’єму bounding box.

7. Оцінку форми хмари за допомогою спектрального аналізу – знаходження власних чисел коваріаційної матриці відцентрованих точок. Ці величини дозволяють оцінити напрямки основних варіацій у розташуванні точок.

Кожна з вказаних характеристик додається вектора ознак, який подається на вхід класифікаційної моделі. Такий підхід дозволяє перетворити просторову структуру хмари точок у компактну числову форму, придатну для подальшого машинного навчання.

Варто відзначити, що реалізований набір ознак має універсальний характер, не залежить від конкретної геометрії об’єкта, що дозволяє використовувати підсистему для широкого кола задач. Розширення цієї підсистеми може включати додаткові геометричні або топологічні ознаки, а також використання статистичних моментів вищих порядків.

Отже, розроблена підсистема ефективно вирішує задачу перетворення непідготовленої множини 3D-точок у структурований вектор ознак, який слугує основою для подальших етапів класифікації. Її реалізація охоплює широкий спектр геометричних і статистичних характеристик, що забезпечує достатню

інформативність для аналізу об’єктів різної форми, розміру і щільності. У поєднанні з модулями навчання та візуалізації, ця підсистема є важливою складовою архітектури програмного продукту.

2.4 Розробка підсистеми навчання моделей

Одним із ключових етапів реалізації програмного забезпечення для класифікації 3D-хмар точок є побудова та навчання моделей машинного навчання, які можуть ефективно розпізнавати просторові структури на основі векторів ознак, отриманих у попередньому етапі. В межах даної підсистеми реалізовано автоматизоване навчання кількох моделей з подальшою оцінкою їх ефективності та логуванням процесу.

На рисунку 2.3 зображено блок-схему розроблюваної підсистеми, що ілюструє основні етапи підготовки та навчання моделей класифікації.

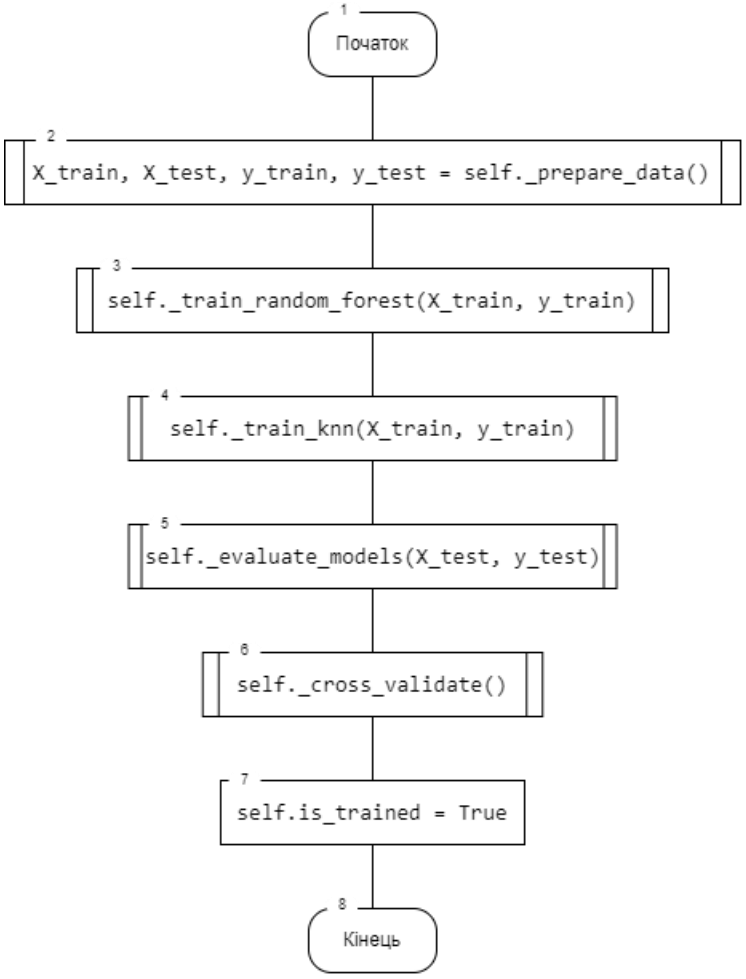


Рисунок 2.3 – Блок-схема підсистсеми навчання моделей

Розроблена підсистема реалізована у вигляді окремого методу, що викликається у фоновому потоці, не блокуючи основний інтерфейс користувача. Це дозволяє забезпечити зручність взаємодії із застосунком навіть під час інтенсивного обчислювального процесу навчання.

Підсистема включає такі етапи:

1. Підготовка навчальних даних. Масиви ознак X та міток класів у формуються із накопичених прикладів. Нормалізація ознак. Розділення на навчальну та тестову вибірки.

2. Навчання моделі Random Forest. Створюється класифікатор RandomForestClassifier, який навчається на тренувальній вибірці. Цей тип моделі дозволяє обробляти дані з великою кількістю ознак і має хорошу стійкість до перенавчання.

3. Навчання моделі K-NN. Виконується навчання моделі K-NN з метрикою відстані та адаптивним значенням параметра k, що враховує кількість класів.

4. Оцінювання точності моделей. На основі тестової вибірки розраховуються значення точності для обох моделей. Ці значення виводяться в лог користувача.

5. Крос-валідація. Для додаткової оцінки стабільності моделей виконується 5-кратна крос-валідація на повному наборі ознак. Користувачу надаються як середні значення точності, так і стандартне відхилення, що дозволяє оцінити надійність навчання.

6. Зміна прапорця стану, щоб повідомити користувача про успішне завершення навчання моделей.

Отже, розроблена підсистема навчання моделей є важливою частиною програмного комплексу, яка забезпечує побудову інтелектуальної логіки на основі попередньо обчислених ознак. Її гнучка архітектура, ефективна реалізація у фоновому потоці та підтримка кількох моделей дозволяють адаптуватися до різних типів задач та користувацьких вимог. Використання крос-валідації гарантує об'єктивну оцінку якості навчання, що, у свою чергу, підвищує надійність застосунку загалом.

2.5 Розробка графічної схеми інтерфейсу

На етапі розробки програмного продукту особливу увагу було приділено побудові структурній схемі застосунку та інтерфейсу користувача.

Основна мета створення структурної схеми – забезпечення чіткого уявлення про структуру програмного застосунку.

На рисунку 2.4 зображено структурну схему застосунку.

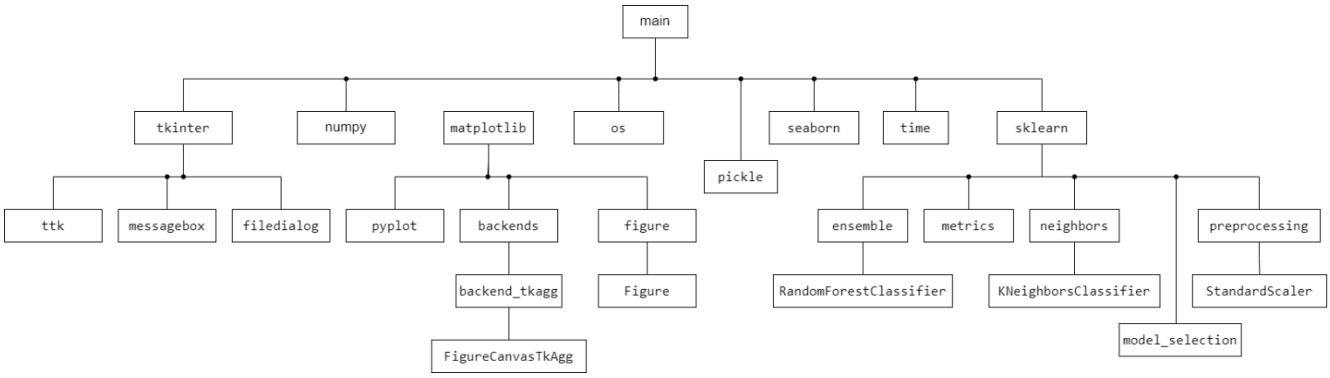


Рисунок 2.4 – Структурна схема застосунку

Графічна схема інтерфейсу – це візуальне зображення структури та елементів користувацького інтерфейсу програмного забезпечення. Вона слугує інструментом для розробників і дизайнерів, дозволяючи краще уявити зовнішній вигляд програми та зрозуміти логіку взаємодії користувача з її елементами.

Інтерфейс реалізовано відповідно до принципів зручності, мінімалізму та функціональної насиченості. Кожен елемент виконує окрему функцію і згрупований відповідно до логіки процесу класифікації хмари точок. Зліва розташована панель керування з кнопками для завантаження даних, навчання, збереження та завантаження моделі, а також очищення сцени та аналізу. Центральну частину інтерфейсу займає вікно візуалізації, де відображається множина 3D-точок. Нижче візуалізації передбачено додаткові кнопки для керування виглядом сцени. Також реалізовано окремі області для виведення результатів класифікації, статистики, подій логування та інформації про застосунок.

На рисунку 2.5 зображено розроблену графічну схему інтерфейсу застосунку.

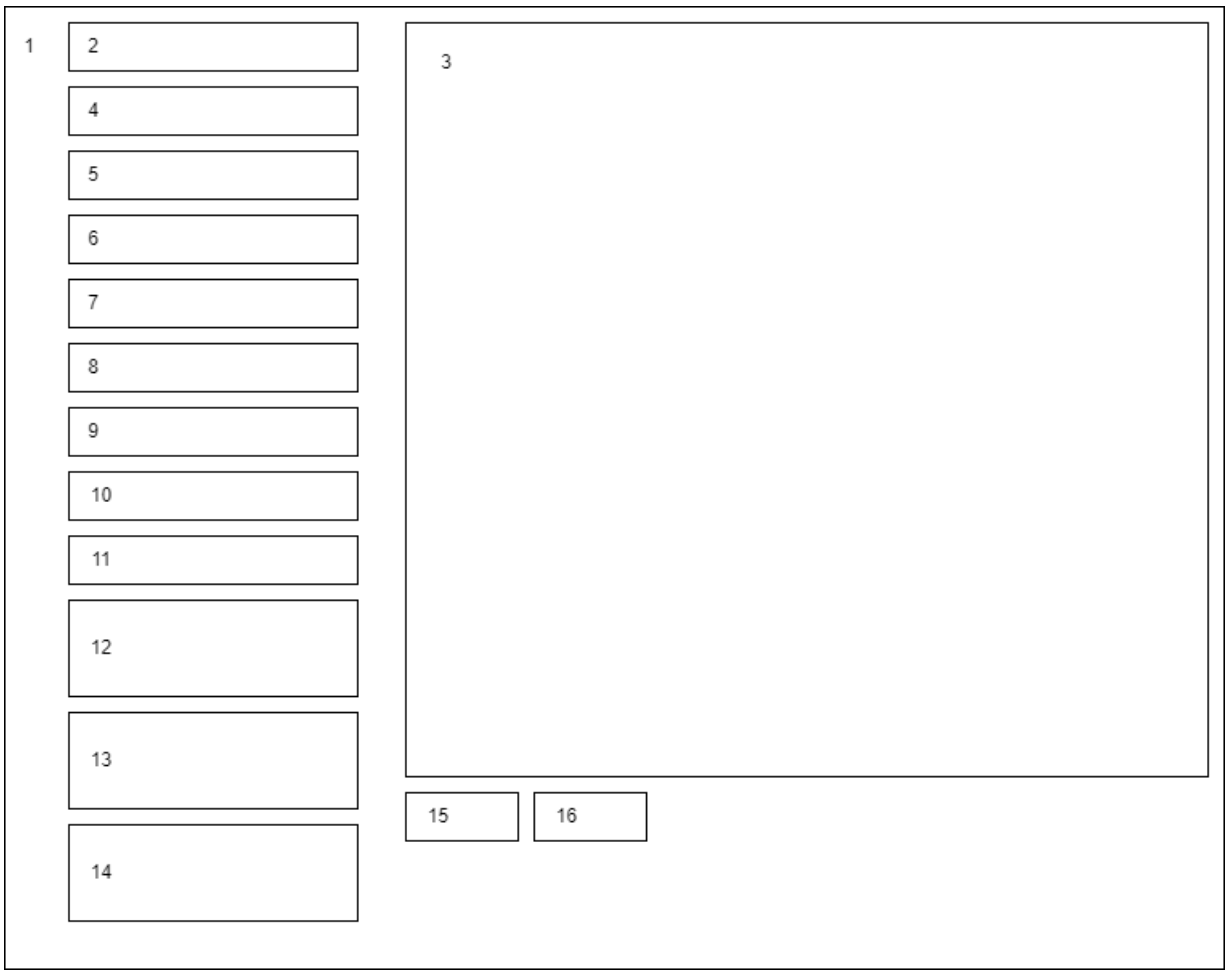


Рисунок 2.5 – Графічна схема інтерфейсу

Опис елементів інтерфейсу:

1. Основне вікно програми, на якому розміщено всі інші елементи.

2. Кнопка «Завантажити множину точок» – для завантаження множини точок і подальшої класифікації.

3. Область візуалізації 3D хмари точок – для візуалізації завантаженої множини точок.

4. Кнопка «Завантажити папку навчання» – призначення для завантаження датасету, на якому будуть навчатись моделі.

5. Кнопка «Навчити модель» – запускає навчання моделей, при умові якщо завантажено папку для навчання.

6. Кнопка «Класифікувати» – класифікує завантажену множину точок, при умові якщо є навчена модель.

7. Кнопка «Зберегти модель» – дозволяє зберегти навчену модель.

8. Кнопка «Завантажити модель» – дозволяє завантажити навчену модель.

9. Кнопка «Очистити сцену» – очищає вікно візуалізації разом із завантаженою множиною точок.

10. Кнопка «Аналіз моделі» – дозволяє проаналізувати модель для класифікації.

11. Кнопка «Про застосунок» – надає основну інформацію про застосунок.

12. Блок результатів класифікації – виводить результати класифікації.

13. Блок статистики – відображає кількість завантажених точок, кількість навчальних зразків та чи навчена модель.

14. Лог подій – інформує користуваача про те що відбулось в певний час та дозволяє оцінити час завтрачений на наавчання моделі.

15. Кнопка «Скинути вид» – повертає сцену до початкового стану.

16. Кнопка «Обернути» – вмикає автоматичне обертання сцени.

Отже, створена графічна схема інтерфейсу забезпечує наочне уявлення про компонування та функціональність програми, що дозволяє швидко орієнтуватися в її можливостях, сприяє зручності користування та відповідає сучасним вимогам до UX-дизайну [10].

2.6 Висновки

У другому розділі проведено аналіз і реалізацію ключових компонентів програмного забезпечення для класифікації множини 3D-точок. Розглянуто формат вхідних даних, їх особливості та вимоги до обробки. Змодельовано архітектуру застосунку, що забезпечує масштабованість. Реалізовано підсистему формування ознак для підготовки даних до навчання моделей, а також підсистему машинного навчання з можливістю тренування, збереження та завантаження моделей. Завершено розробкою графічного інтерфейсу, який забезпечує зручну взаємодію з функціоналом, включно з візуалізацією, класифікацією та переглядом результатів.

3 РОЗРОБКА ПРОГРАМНИХ МОДУЛІВ РЕАЛІЗАЦІЇ ЗАСТОСУНКУ

3.1 Аналіз і обґрунтування вибору засобів реалізації застосунку

У процесі створення програмного забезпечення важливо не лише продумати функціональність і логіку роботи, але й правильно обрати інструменти для реалізації. Від вибору мови програмування, середовища розробки та допоміжних бібліотек значною мірою залежить зручність створення застосунку, його продуктивність, можливість масштабування та підтримки в майбутньому.

Застосунок, що розробляється в межах цієї роботи, орієнтований на класифікацію 3D-хмар точок із використанням алгоритмів машинного навчання, візуалізацію результатів та подальший аналіз.

У процесі аналізу технологічного стеку для реалізації застосунку було розглянуто кілька мов програмування, серед яких основну увагу було приділено Python, C++, Java та JavaScript. Кожна з них має своє призначення в сучасній розробці програмного забезпечення, однак не всі здатні повною мірою забезпечити зручність і функціональність, необхідні для розв’язання поставлених задач.

Мова Python [11] останніми роками набула особливої популярності в науковому середовищі, зокрема в сфері машинного навчання та аналізу даних. Це обумовлено наявністю широкого спектра бібліотек, серед яких можна виокремити такі як scikit-learn, NumPy, Pandas та Matplotlib, які забезпечують повний цикл обробки даних – від підготовки до візуалізації результатів. Окрім того, простий і зрозумілий синтаксис мови значно знижує поріг входу для нових розробників і дозволяє швидко створювати прототипи алгоритмів, що особливо важливо в контексті експериментальної роботи над алгоритмами класифікації.

Мова C++ [12] забезпечує високу продуктивність і контроль над ресурсами, однак її використання у завданнях машинного навчання потребує значно більших зусиль для розробки та інтеграції необхідного функціоналу. До того ж, підтримка бібліотек машинного навчання у C++ є обмеженою, а візуалізація даних, як правило, реалізується через зовнішні компоненти або сторонні сервіси.

Java [13], як і C++, демонструє хорошу продуктивність і має розвинену екосистему, проте в задачах наукових обчислень і машинного навчання поступається Python за зручністю та функціональністю. Попри наявність таких бібліотек як Weka або Deeplearning4j, екосистема Java менш гнучка для швидкого створення наукових прототипів.

JavaScript [14] загалом не призначений для реалізації моделей машинного навчання чи обробки наукових даних. Його основна задача – клієнтська частина веб-інтерфейсів. Хоча існують експериментальні бібліотеки на зразок TensorFlow.js, їх функціональність значно обмежена в порівнянні з повноцінними фреймворками для Python.

Результатом аналізу є таблиця 3.1 для порівняння мов програмування.

Таблиця 3.1 – Порівняння мов програмування

Критерій	Python	C++	Java	JavaScript
ML бібліотеки	1	0	1	0
Візуалізація даних	1	0	0	1
Простий синтаксис	1	0	0	1
Наукові обчислення	1	1	-	0
Об'єктно-орієнтована	1	1	1	1
Мультипроцесність	1	1	1	0
Сумарний коефіцієнт	6	3	3	3

Провівши порівняльний аналіз мов програмування, Python має найвищий сумарний коефіцієнт відносно C++, Java та JavaScript на 50%.

Для розробки застосунку було обрано середовище розробки Visual Studio Code [15], яке є потужним та універсальним інструментом, що підтримує широкий спектр мов програмування та технологій. VS Code забезпечує зручний інтерфейс користувача, високий рівень налаштовуваності та інтеграцію з численними розширеннями, що дозволяють адаптувати середовище під конкретні потреби розробника. Особливо важливою перевагою VS Code є підтримка інтегрованого

терміналу, інструментів налагодження, систем контролю версій, таких як Git, а також можливість підключення до різноманітних сервісів і платформ. Завдяки відкритій архітектурі та активній спільноті, VS Code постійно оновлюється, отримуючи нові функції та поліпшення продуктивності. Для розробки застосунку на мові Python VS Code надає інструменти автодоповнення коду, перевірки синтаксису, а також підтримку віртуальних середовищ і налагодження, що суттєво прискорює процес розробки і підвищує якість програмного продукту. Таким чином, вибір VS Code як середовища розробки сприяє ефективній реалізації поставлених задач і забезпечує комфортні умови для подальшого розвитку застосунку.

Для реалізації функціональних компонентів програмного застосунку було обрано низку бібліотек, які забезпечують ефективну роботу з даними, побудову моделей машинного навчання, візуалізацію результатів і створення графічного інтерфейсу користувача. Основними з них стали scikit-learn, NumPy, Matplotlib та Tkinter.

Бібліотека scikit-learn [16] була обрана як основний інструмент для реалізації алгоритмів машинного навчання. Вона містить велику кількість готових моделей для класифікації, регресії, кластеризації та попередньої обробки даних, що дає змогу швидко інтегрувати складні математичні алгоритми в програму без потреби у ручній реалізації з нуля. Простий і зрозумілий API бібліотеки дозволяє легко експериментувати з параметрами моделей, проводити оцінку якості класифікації та здійснювати порівняльний аналіз різних підходів.

NumPy [17] використовується для ефективної роботи з багатовимірними масивами даних, що є основою при математичних обчисленнях у процесі підготовки вхідних даних для моделей. Ця бібліотека забезпечує високу продуктивність за рахунок оптимізованого виконання операцій на рівні C/Fortran і є фундаментом для більшості наукових бібліотек Python.

Для візуалізації результатів роботи алгоритмів була обрана бібліотека Matplotlib [18], яка дозволяє створювати графіки, гістограми, діаграми розсіювання та інші типи візуального представлення даних. Наявність широкого спектра

параметрів налаштування графічних елементів дає змогу створювати інформативні й адаптивні візуалізації, що значно полегшують аналіз результатів класифікації та надають користувачеві наочне уявлення про поведінку алгоритмів.

Для реалізації графічного інтерфейсу користувача застосовується Tkinter, що являє собою стандартну бібліотеку для побудови GUI у Python. Її перевагами є простота у використанні, кросплатформенність і тісна інтеграція з базовим інтерпретатором Python. Використання Tkinter дозволяє швидко створювати форми, кнопки, меню та інші елементи інтерфейсу, забезпечуючи інтуїтивну взаємодію користувача з програмою.

Отже, обраний технологічний стек на основі мови програмування Python, середовища розробки Visual Studio Code та бібліотек scikit-learn, NumPy, Matplotlib і Tkinter повністю задовольняє вимоги до створення застосунку для класифікації 3D-хмар точок. Такий вибір забезпечує зручність розробки, високу функціональність, підтримку наукових обчислень, якісну візуалізацію результатів та створення інтуїтивного графічного інтерфейсу користувача.

3.2 Розробка модулів застосунку

На рисунку 3.1 зображено діаграму функціонального класу «PointCloudClassifier», яка описує основний функціонал застосунку.

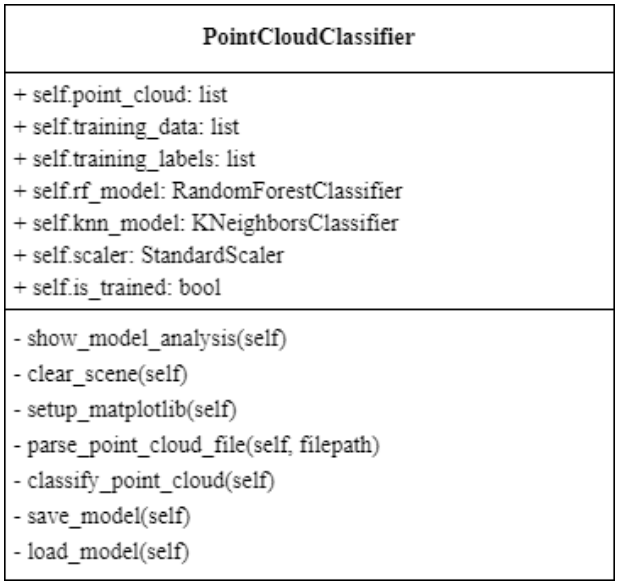


Рисунок 3.1 – Діаграму класу «PointCloudClassifier»

Клас PointCloudClassifier містить поля:

– point_cloud – хмара точок, завантажена користувачем для обробки або класифікації;

– training_data – список векторів ознак, обчислених із навчальних хмар точок;

– training_labels – список відповідних міток класів для кожного вектору ознак у training_data;

– rf_model – модель класифікації типу Random Forest, яка навчається на підготовлених даних;

– knn_model – модель класифікації типу K-Nearest Neighbors для порівняння результатів;

– scaler – об'єкт для нормалізації ознак перед передачею в моделі;

– is_trained – логічний прапорець, що вказує, чи були моделі успішно навчені.

На рисунку 3.2 представлено метод «show_model_analysis», який відповідає за візуалізацію та аналіз ефективності моделей класифікації, зокрема Random Forest та K-NN. Основною метою цього методу є порівняльна оцінка продуктивності алгоритмів машинного навчання, що використовуються у застосунку.

Робота методу починається з перевірки, чи була попередньо навчена модель. Якщо ні – користувачу виводиться попередження. У разі готовності моделі створюється окреме вікно на базі tk.Toplevel, стилізоване для візуального комфорту, в якому розміщується графічний аналіз.

Дані, що використовувалися для навчання, «training_data» та «training_labels», перетворюються на масиви NumPy та масштабуються за допомогою стандартного скейлера. Далі виконується поділ на навчальну та тестову вибірки із збереженням пропорцій класів.

Метод формує прогнози для двох моделей – Random Forest і KNN – на тестовій підвибірці. На основі отриманих результатів будується фігура з чотирма графіками:

1) матриця помилок для моделі Random Forest;

2) матриця помилок для моделі KNN;

3) морівняльна оцінка точності обох моделей;

4) мізуалізація класифікаційного звіту.

Кожен із підграфіків створюється за допомогою допоміжних методів. Для уникнення накладень графічних елементів використовується plt.tight_layout.

Готова фігура вбудовується у графічний інтерфейс Tkinter через FigureCanvasTkAgg, що дозволяє користувачу взаємодіяти з аналітикою безпосередньо в додатку. У випадку помилки відображається відповідне повідомлення, а інформація фіксується у журналі подій через «add_log».

```
def show_model_analysis(self):
    if not self.is_trained:
        messagebox.showwarning("Попередження", "Спочатку навчіть модель")
        return
    try:
        model_window = tk.Toplevel(self.root)
        model_window.title("Аналіз моделі")
        model_window.geometry("1400x900")
        model_window.configure(bg='#2b2b2b')
        X = np.array(self.training_data)
        y = np.array(self.training_labels)
        X_scaled = self.scaler.transform(X)
        X_train, X_test, y_train, y_test = train_test_split(
            X_scaled, y, test_size=0.2, random_state=42, stratify=y
        )
        rf_pred = self.rf_model.predict(X_test)
        knn_pred = self.knn_model.predict(X_test)
        fig, axes = plt.subplots(2, 2, figsize=(18, 10))
        fig.patch.set_facecolor('#2b2b2b')
        fig.suptitle('Аналіз продуктивності моделей', fontsize=16, color='white')
        cm_rf = confusion_matrix(y_test, rf_pred)
        self._plot_confusion_matrix(axes[0, 0], cm_rf, list(set(y)), 'Random Forest')
        cm_knn = confusion_matrix(y_test, knn_pred)
        self._plot_confusion_matrix(axes[0, 1], cm_knn, list(set(y)), 'KNN')
        self._plot_accuracy_comparison(axes[1, 0], y_test, rf_pred, knn_pred)
        self._plot_classification_report(axes[1, 1], y_test, rf_pred, knn_pred)
        plt.tight_layout(rect=[0, 0, 1, 0.95])
        canvas = FigureCanvasTkAgg(fig, model_window)
        canvas.draw()
        canvas.get_tk_widget().pack(fill=tk.BOTH, expand=True, padx=10, pady=10)
        self.add_log("Аналіз моделі відображено")
    except Exception as e:
        messagebox.showerror("Помилка", f"Помилка аналізу моделі: {str(e)}")
        self.add_log(f"Помилка аналізу моделі: {str(e)}")
```

Рисунок 3.2 – Метод «show_model_analysis»

На рисунку 3.3 зображено метод «clear_scene», який виконує повне очищення сцени у графічному інтерфейсі користувача. Основне призначення методу є скидання поточного стану застосунку, включно з хмарою точок, результатами класифікації та відповідними елементами інтерфейсу. Це дозволяє розпочати нову сесію без залишкових даних.

Після виклику функції виводиться діалог підтвердження. Якщо користувач погоджується, метод встановлює point_cloud у None, скидає текстові мітки для результатів класифікації та кількості точок, а також зупиняє анімацію обертання, якщо вона активна. За наявності колірної шкали вона видаляється, щоб уникнути візуальних конфліктів при наступному завантаженні даних.

Далі викликається метод «show_placeholder», який повертає інтерфейс у початковий стан. Успішне очищення фіксується у журналі подій за допомогою «add_log». У разі помилки виводиться відповідне повідомлення.

Завдяки такій логіці метод забезпечує швидке та безпечне скидання середовища для подальшої роботи з новими даними.

```
def clear_scene(self):
    result = messagebox.askyesno("Підтвердження",
                                 "Ви дійсно хочете очистити сцену?\n"
                                 "Це видалить завантажену хмару точок та результати класифікації.")
    if not result:
        return
    try:
        self.point_cloud = None
        self.rf_result_label.config(text="Random Forest: -")
        self.knn_result_label.config(text="KNN: -")
        self.points_label.config(text="Точок завантажено: 0")
        if hasattr(self, '_rotating'):
            self._rotating = False
        try:
            if hasattr(self, '_colorbar') and self._colorbar is not None:
                try:
                    self._colorbar.remove()
                except:
                    pass
                self._colorbar = None
        except:
            pass
        self.show_placeholder()
        self.add_log("Сцену очищено")
    except Exception as e:
        self.add_log(f"Помилка очищення: {str(e)}")
        messagebox.showerror("Помилка", f"Помилка очищення сцени: {str(e)}")
```

Рисунок 3.3 – Метод «clear_scene»

На рисунку 3.4 представлено метод «setup_matplotlib», який відповідає за ініціалізацію та налаштування тривимірної області побудови для візуалізації хмари точок у графічному інтерфейсі користувача. Метод створює об’єкт фігури Figure з підключеним підграфом у тривимірному просторі, налаштовуючи його зовнішній вигляд відповідно до темної кольорової схеми інтерфейсу.

Для покращення візуального сприйняття координатні осі та мітки встановлюються у білий колір, а сітка – напівпрозора. Панелі осей очищуються від фону, зберігаючи лише їх межі. Це дозволяє зробити графік чистішим та сфокусованим на даних.

Далі створюється об’єкт FigureCanvasTkAgg, який вбудовується у праву панель інтерфейсу за допомогою «pack()», що забезпечує інтеграцію графіка з GUI-структурою на основі Tkinter. Після цього формується панель інструментів із кнопками для скидання виду та активації або деактивації обертання сцени, що забезпечують базову інтерактивність.

```
def setup_matplotlib(self):
    self.fig = Figure(figsize=(10, 8), dpi=100, facecolor='#1e1e1e')
    self.ax = self.fig.add_subplot(111, projection='3d', facecolor='#1e1e1e')
    self.ax.xaxis.label.set_color('white')
    self.ax.yaxis.label.set_color('white')
    self.ax.zaxis.label.set_color('white')
    self.ax.tick_params(axis='x', colors='white')
    self.ax.tick_params(axis='y', colors='white')
    self.ax.tick_params(axis='z', colors='white')
    self.ax.xaxis.pane.fill = False
    self.ax.yaxis.pane.fill = False
    self.ax.zaxis.pane.fill = False
    self.ax.xaxis.pane.set_edgecolor('gray')
    self.ax.yaxis.pane.set_edgecolor('gray')
    self.ax.zaxis.pane.set_edgecolor('gray')
    self.ax.grid(True, alpha=0.3)
    self.canvas = FigureCanvasTkAgg(self.fig, self.right_panel)
    self.canvas.draw()
    self.canvas.get_tk_widget().pack(fill=tk.BOTH, expand=True, padx=10, pady=10)
    toolbar_frame = tk.Frame(self.right_panel, bg='#1e1e1e')
    toolbar_frame.pack(fill=tk.X, padx=10, pady=(0, 10))
    btn_style = {'font': ('Arial', 9), 'width': 12, 'height': 1,
                 'relief': tk.FLAT, 'cursor': 'hand2', 'bg': '#4a4a4a', 'fg': 'white'}
    reset_btn = tk.Button(toolbar_frame, text="🔄 Скинути вид",
                          command=self.reset_view, **btn_style)
    reset_btn.pack(side=tk.LEFT, padx=5)
    rotate_btn = tk.Button(toolbar_frame, text="🔄 Обертати",
                           command=self.toggle_rotation, **btn_style)
    rotate_btn.pack(side=tk.LEFT, padx=5)
    self.show_placeholder()
```

Рисунок 3.4 – Метод «setup_matplotlib»

На рисунку 3.5 зображено метод «parse_point_cloud_file», який реалізує логіку обробки вхідного файлу з хмарою точок. Його основне призначення – визначити тип файлу за розширенням та передати його для обробки відповідному допоміжному методу.

Метод підтримує найпоширеніші формати: .off, .ply, .xyz, .pcd, .obj. Для кожного з них передбачений окремий парсер, що дозволяє гнучко масштабувати програму під різні джерела 3D-даних. У разі, якщо формат не підтримується, користувач отримує повідомлення про помилку.

Обробка винятків реалізована через try-except блок. У випадку виникнення будь-якої помилки читання, вона фіксується з уточненням типу файлу, що спрощує діагностику проблем.

Таким чином, «parse_point_cloud_file» виконує роль універсального завантажувача хмари точок, забезпечуючи гнучкість, модульність та контроль помилок при роботі з файлами різних форматів.

```
def parse_point_cloud_file(self, filepath):
    file_ext = os.path.splitext(filepath)[1].lower()
    try:
        if file_ext == '.off':
            return self._parse_off(filepath)
        elif file_ext == '.ply':
            return self._parse_ply(filepath)
        elif file_ext == '.xyz':
            return self._parse_xyz(filepath)
        elif file_ext == '.pcd':
            return self._parse_pcd(filepath)
        elif file_ext == '.obj':
            return self._parse_obj(filepath)
        else:
            raise ValueError(f"Непідтримуваний формат файлу: {file_ext}")

    except Exception as e:
        raise Exception(f"Помилка читання файлу {file_ext}: {str(e)}")
```

Рисунок 3.5 – Метод «parse_point_cloud_file»

На рисунку 3.6 представлено метод «classify_point_cloud», який відповідає за класифікацію поточної хмари точок за допомогою двох моделей машинного навчання – Random Forest і K-Nearest Neighbors. Цей метод є однією з основних складових системи, оскільки саме він дозволяє користувачу швидко отримати

оцінку про те, до якого з наявних класів належить об'єкт, представлений у вигляді тривимірних координат.

Перед тим як перейти до класифікації, метод перевіряє, чи все готово для виконання цієї операції. Спершу він переконується, що користувач завантажив необхідну хмару точок. Якщо цього не було зроблено, з'являється повідомлення, яке нагадує про потребу завантаження даних. Далі відбувається перевірка, чи були моделі навчальні раніше. Якщо жоден із цих кроків не виконаний, метод зупиняє виконання, аби уникнути помилок або хибних результатів.

Якщо ж всі умови виконані, розпочинається основна частина роботи. З хмари точок витягуються числові характеристики, які описують її основні властивості. Потім ці значення масштабуються за допомогою збереженого скейлера. Це робиться для того, щоб привести всі ознаки до одного масштабу, що позитивно впливає на точність обох моделей.

Після цього модель Random Forest і модель KNN незалежно одна від одної здійснюють передбачення на основі підготовлених даних. Обидві моделі визначають, до якого класу, на їхню думку, належать оброблені дані. Крім самого класу, вони також оцінюють ступінь своєї впевненості у передбаченні. Найбільше значення з цієї оцінки відображає, наскільки модель впевнена у своєму виборі.

Отримані результати відображаються у графічному інтерфейсі користувача. Для кожної з моделей виводиться передбачений клас та рівень впевненості у простій та зрозумілій формі. Це дає змогу користувачу миттєво оцінити ситуацію без потреби в додатковому аналізі або спеціальних знаннях.

На завершення метод додає записи до журналу подій, щоб за потреби можна було простежити історію класифікацій. У разі виникнення помилки користувач отримає відповідне повідомлення, а сама помилка також буде збережена в журналі. Такий підхід допомагає не лише уникнути збоїв, а й швидко знаходити причини проблем, якщо такі виникнуть.

У підсумку, метод «classify_point_cloud» автоматизує складний процес класифікації, роблячи його простим і доступним навіть для користувача без глибоких знань у сфері обробки даних.

```
def classify_point_cloud(self):
    if self.point_cloud is None:
        messagebox.showwarning("Попередження", "Спочатку завантажте хмару точок")
        return

    if not self.is_trained:
        messagebox.showwarning("Попередження", "Спочатку навчіть модель")
        return

    try:
        features = self.extract_features(self.point_cloud)
        features_scaled = self.scaler.transform([features])

        rf_prediction = self.rf_model.predict(features_scaled)[0]
        knn_prediction = self.knn_model.predict(features_scaled)[0]
        rf_proba = self.rf_model.predict_proba(features_scaled)[0]
        knn_proba = self.knn_model.predict_proba(features_scaled)[0]
        rf_confidence = max(rf_proba)
        knn_confidence = max(knn_proba)
        self.rf_result_label.config(
            text=f"Random Forest: {rf_prediction} ({rf_confidence:.2f})"
        )
        self.knn_result_label.config(
            text=f"KNN: {knn_prediction} ({knn_confidence:.2f})"
        )
        self.add_log(f"Класифікація завершена:")
        self.add_log(f"  RF: {rf_prediction} (впевненість: {rf_confidence:.2f})")
        self.add_log(f"  KNN: {knn_prediction} (впевненість: {knn_confidence:.2f})")
    except Exception as e:
        messagebox.showerror("Помилка", str(e))
        self.add_log(f"Помилка класифікації: {str(e)}")
```

Рисунок 3.6 – Метод «classify_point_cloud»

На рисунку 3.7 метод «save_model» призначений для збереження навченої моделі машинного навчання, яка була побудована на основі оброблених даних. Його роль у програмному забезпеченні є дуже важливою, адже він дозволяє зберігати результати тривалого навчання моделі, щоби згодом їх можна було повторно використати без потреби запускати процес тренування з нуля. Це суттєво економить час користувача і підвищує зручність роботи із застосунком.

Перед тим як виконати основну дію, метод перевіряє, чи взагалі є модель, яку можливо зберігати. Це реалізується через змінну is_trained, яка вказує на те, що класифікатори вже пройшли навчання. Якщо модель ще не навчена, користувачу одразу показується попереджувальне повідомлення, і процес збереження переривається. Таким чином, програма уникає помилок і гарантує, що зберігаються лише коректно підготовлені моделі.

Якщо ж модель справді готова до збереження, метод відкриває діалогове вікно, в якому користувач може вибрати місце та назву для збереження файлу.

Пропонується розширення файлу .pkl, що відповідає стандартному інструменту серіалізації об'єктів у Python. Користувач має змогу змінити це розширення або відмовитися від збереження, натиснувши скасування. Якщо шлях не обрано, метод завершує виконання без подальших дій.

У разі, коли користувач вказав шлях до файлу, створюється структура даних, яка включає всі важливі елементи навченої моделі. Зберігаються як самі моделі Random Forest і KNN, так і об’єкт масштабування ознак. Крім того, зберігаються вихідні дані, на яких виконувалося навчання, включно з ознаками і мітками класів. Це дозволяє не тільки використовувати модель для передбачення, а й за потреби відтворити умови, в яких вона була побудована. Також додається список унікальних класів, на основі яких проводилася класифікація.

Збереження відбувається за допомогою модуля Pickle. Всі об’єкти пакуються у файл, і цей файл записується на диск у двійковому форматі. Якщо процес збереження проходить успішно, користувач бачить відповідний запис у журналі дій, де вказується назва збереженого файлу. Це допомагає контролювати, які саме дії було виконано в межах сесії.

У разі виникнення помилки, наприклад, при спробі записати файл у недоступне місце або у випадку пошкодження одного з об’єктів, метод обробляє виняток. Користувачу відображається повідомлення про помилку, а також додається запис до логів. Такий підхід робить роботу системи більш стійкою до непередбачуваних ситуацій та гарантує збереження важливої інформації.

Загалом метод «save_model» виконує одну з ключових допоміжних функцій, яка робить програмне забезпечення практичним і гнучким. Завдяки можливості зберігати модель, користувач може будувати набір рішень, переносити результати роботи на інші пристрої та забезпечити повторне використання моделей у майбутньому без повторного навчання.

```
def save_model(self):
    if not self.is_trained:
        messagebox.showwarning("Попередження", "Немає навченої моделі для збереження")
        return
    filepath = filedialog.asksaveasfilename(
        title="Зберегти модель",
        defaultextension=".pkl",
        filetypes=[("Pickle files", "*.pkl"), ("All files", "*.*")]
    )
    if not filepath:
        return
    try:
        model_data = {
            'rf_model': self.rf_model,
            'knn_model': self.knn_model,
            'scaler': self.scaler,
            'training_data': self.training_data,
            'training_labels': self.training_labels,
            'classes': list(set(self.training_labels))
        }
        with open(filepath, 'wb') as f:
            pickle.dump(model_data, f)
        self.add_log(f"Модель збережена: {os.path.basename(filepath)}")
    except Exception as e:
        messagebox.showerror("Помилка", str(e))
        self.add_log(f"Помилка збереження: {str(e)}")
```

Рисунок 3.7 – Метод «save_model»

На рисунку 3.8 – метод «load_model» забезпечує завантаження раніше збереженої моделі машинного навчання, що дозволяє відновити її стан без необхідності повторного навчання. Це особливо корисно у тих випадках, коли обробка даних та тренування моделі зайняли значну кількість часу, і користувач хоче продовжити роботу з уже готовими результатами.

Після активації функції користувачу пропонується вибрати файл моделі на локальному пристрої. Вікно вибору файлу реалізоване через стандартний механізм взаємодії з файловою системою, де вказано фільтр для файлів з розширенням pkl. Це дозволяє користувачу швидко знаходити потрібні об’єкти без ризику відкрити неправильний тип файлу.

Якщо користувач не обирає жодного файлу або натискає скасування, метод завершує роботу, не виконуючи жодних змін у системі. Це є запобіжним заходом проти випадкового втручання в уже існуючий стан програми.

У випадку, якщо файл обрано, програма переходить до етапу завантаження даних. Відкриття файлу здійснюється у двійковому режимі, після чого дані зчитуються за допомогою модуля Pickle. Завантажений об’єкт містить у собі всю необхідну інформацію: моделі класифікаторів, масштабувальник ознак, навчальні дані, мітки класів та список самих класів. Усі ці компоненти переносяться до внутрішнього середовища програми, тобто відновлюється стан, який існував на момент збереження.

Після завантаження усіх необхідних об’єктів встановлюється прапорець, що вказує на те, що модель є навченою. Це дозволяє іншим частинам застосунку розуміти, що моделі готові до використання без необхідності повторного тренування. Крім того, в інтерфейсі оновлюється інформація для користувача. Зокрема, у відповідному полі зазначається, що модель є навченою, а також виводиться кількість навчальних зразків, які були завантажені разом із нею.
Всі основні дії, пов’язані із завантаженням, фіксуються у журналі подій. Туди заноситься як назва завантаженого файлу, так і перелік класів, що були використані під час навчання моделі. Це дозволяє користувачу відслідковувати історію дій та за потреби повертатися до попередніх кроків.

У випадку виникнення помилки, наприклад через спробу завантажити пошкоджений файл або об’єкт, що не містить потрібної структури, метод перехоплює виняток. Користувачу одразу показується повідомлення про помилку, а відповідний запис потрапляє до логів. Це дозволяє зберегти стабільність роботи застосунку навіть у складних ситуаціях і швидко діагностувати причину проблеми.

Метод «load_model» є важливим елементом загального функціоналу системи. Він дозволяє зручно відновлювати раніше побудовані моделі та швидко повертатися до роботи, уникаючи повторних обчислень. Це робить систему більш практичною, адаптивною та зручною для користувача.

```
def load_model(self):
    filepath = filedialog.askopenfilename(
        title="Завантажити модель",
        filetypes=[("Pickle files", "*.pkl"), ("All files", "*.*")]
    )

    if not filepath:
        return

    try:
        with open(filepath, 'rb') as f:
            model_data = pickle.load(f)

        self.rf_model = model_data['rf_model']
        self.knn_model = model_data['knn_model']
        self.scaler = model_data['scaler']
        self.training_data = model_data['training_data']
        self.training_labels = model_data['training_labels']

        self.is_trained = True
        self.model_label.config(text="Модель навчена: Так")
        self.training_label.config(text=f"Навчальних зразків: {len(self.training_data)}")

        self.add_log(f"Модель завантажена: {os.path.basename(filepath)}")
        self.add_log(f"Класи: {model_data.get('classes', [])}")

    except Exception as e:
        messagebox.showerror("Помилка", str(e))
        self.add_log(f"Помилка завантаження: {str(e)}")
```

Рисунок 3.8 – Метод «load_model»

Отже, у межах роботи було реалізовано окремі програмні модулі, що забезпечують обробку, класифікацію та збереження 3D-хмар точок. Кожен модуль відповідає за свою функціональність: від завантаження даних і витягу ознак до навчання моделей, їх збереження та подальшого використання для класифікації. Взаємодія між модулями забезпечує цілісний процес аналізу просторових даних. Таким чином, створена архітектура дозволяє легко масштабувати та підтримувати систему.

3.3 Висновки

У третьому розділі було реалізовано функціональні програмні модулі, що відповідають за основні етапи обробки та класифікації 3D-хмар точок. Для розробки застосунку було обрано мову програмування Python, яка завдяки своїй

гнучкості та великій кількості готових бібліотек є зручною для створення інструментів обробки просторових даних. Розробка здійснювалася в середовищі Visual Studio Code, що надало комфортні умови для написання, налагодження та структурування коду. До роботи були залучені такі бібліотеки, як scikit-learn для реалізації алгоритмів машинного навчання, NumPy і Pandas для роботи з даними, Matplotlib для побудови графіків і візуалізації результатів, а також Tkinter для створення графічного інтерфейсу. Завдяки такому поєднанню засобів програмна система вийшла функціонально повною, зручною для користувача та такою, що легко піддається подальшому розширенню або адаптації під нові задачі.

4 ТЕСТУВАННЯ ЗАСТОСУНКУ

4.1 Функціональне тестування застосунку

Функціональне тестування є важливою частиною процесу забезпечення якості програмного забезпечення. Воно зосереджується на перевірці того, як система виконує свої функції згідно з вимогами, які були визначені на етапі розробки. Основна мета цього виду тестування – переконатися, що всі функції працюють саме так, як очікується, і забезпечують коректну взаємодію між компонентами системи.

Цей підхід до тестування орієнтований на поведінку програми з точки зору користувача. Іншими словами, замість того, щоб досліджувати внутрішню структуру або код, функціональне тестування зосереджується на тому, що система повинна робити. Наприклад, якщо програма має виконувати обчислення, зберігати дані або надавати результати у зручній формі, то тестування має підтвердити, що ці дії виконуються правильно у різних ситуаціях.

Функціональне тестування [19] включає в себе перевірку всіх основних сценаріїв використання, таких як вхід користувача, обробка інформації, генерація звітів, взаємодія з базами даних або зовнішніми сервісами. Воно також охоплює перевірку граничних значень, виняткових випадків і поведінки системи при некоректних введеннях. Завдяки цьому вдається виявити потенційні помилки, які можуть виникнути під час роботи реальних користувачів.

Таким чином, функціональне тестування дозволяє переконатися в тому, що програмний продукт відповідає очікуванням замовника та кінцевого користувача. Його результати слугують основою для подальших етапів тестування або вдосконалення програмного забезпечення, а також дають змогу розробникам і тестувальникам впевнено говорити про якість створеної системи.

Для проведення тестування було розроблено таблицю 4.1, яка відображає основні функції програмного застосунку, способи проведення тестування і очікувані результати.

Таблиця 4.1 – Функціональне тестування

Іденти-фікатор	Назва	Методика проведення тестування	Очікуваний результат	Результат
1	2	3	4	5
ТВ-1	Завантаження OFF файлу	1. Натисніть «Завантажити множину точок». 2. Виберіть файл.	1. Файл завантажується. 2. Оновлюється статистика. 3. З'являється 3D візуалізація. 4. В логу з'являється повідомлення.	Виконано
ТВ-2	Завантаження різних форматів	1. Почергово завантажте PLY, XYZ, PCD, OBJ файли.	1. Всі формати обробляються коректно.	Виконано
ТВ-3	Завантаження папки навчання	1. Натисніть «Завантажити папку навчання». 2. Виберіть підготовлену папку.	1. Оновлюється «Навчальних зразків». 2. В логу відображаються знайдені класи.	Виконано
ТВ-4	Завантаження порожньої папки	1. Натисніть «Завантажити папку навчання». 2. Виберіть порожню папку або папку.	1. Повідомлення «Не знайдено жодного OFF файлу».	Виконано

Продовження таблиці 4.1

1	2	3	4	5
ТВ-5	Навчання з достатньою кількістю даних	1. Після завантаження навчальних даних натисніть «Навчити модель».	1. В логу відображаються показники точності 2. Статус змінюється на «Модель навчена: Так».	Виконано
ТВ-6	Навчання без даних	1. Натисніть «Навчити модель» без завантаження навчальних даних.	1. Попередження «Потрібно мінімум 2 навчальні зразки».	Виконано
ТВ-7	Класифікація після навчання	1. Завантажте тестову хмару точок. 2. Натисніть «Класифікувати»	1. Відображаються результати Random Forest та KNN. 2. Показуються коефіцієнти впевненості. 3. Результати логуються.	Виконано
ТВ-8	Збереження моделі	1. Після навчання натисніть «Зберегти модель».	1. Модель зберігається в .pkl файл	Виконано

Продовження таблиці 4.1

ТВ-9	Завантаження моделі	1. Натисніть «Завантажити модель». 2. Виберіть збережений файл.	1. Модель завантажується. 2. Оновлюється статистика. 3. Статус змінюється на «Модель навчена: Так».	Виконано
ТВ-10	3D візуалізація	1. Завантажте хмару точок та перевірте візуалізацію.	1. Точки відображаються в 3D просторі. 2. Кольорова карта працює коректно. 3. Осі підписані та видимі.	Виконано
ТВ-11	Елементи керування візуалізацією	1. Натисніть кнопки «Скинути вид» та «Обертати».	1. Вид скидається до початкового. 2. Автоматичне обертання працює.	Виконано
ТВ-12	Очищення сцени	1. Натисніть «Очистити сцену».	1. З'являється діалог підтвердження. 2. Сцена очищається. 3. Статистика скидається.	Виконано
ТВ-13	Відображення аналізу моделі	1. Після навчання натисніть «Аналіз моделі».	1. Відкривається нове вікно з графіками. 2. Відображаються матриці помилок. 3. Показується порівняння точності.	Виконано

Отже, на основі результатів функціонального тестування системи класифікації хмар точок можна зробити висновок, що всі 13 тестових випадків були успішно виконані, що свідчить про високу якість розробленої системи та її повну функціональну відповідність технічним вимогам.

4.2 Інструкція користувача

3D Point Cloud Classifier – це застосунок для класифікації тривимірних хмар точок за допомогою алгоритмів машинного навчання. Програма підтримує різні формати файлів, дозволяє навчати моделі та класифікувати нові дані з візуалізацією результатів.

Керівництво користувача [20] включає опис технічних умов, необхідних для стабільної роботи застосунку. У ньому зазначено конфігурацію комп'ютера, на який може бути встановлений та ефективно функціонувати розроблений застосунок. Детальну інформацію щодо мінімальних та рекомендованих характеристик комп'ютера для запуску застосунку наведено у таблицях 4.1 та 4.2.

Таблиця 4.1 – Мінімальна конфігурація

Тип процесора	Intel Core i3 або AMD Athlon з підтримкою SSE4
Відеокарта	NVIDIA GeForce GT 1030
Об'єм оперативної пам'яті	4 ГБ
Місце на жорсткому диску	500 МБ
Операційна система	Windows 10 / Ubuntu 20.04

Таблиця 4.2 – Рекомендована конфігурація

Тип процесора	Intel Core i5 або AMD Ryzen 5
Відеокарта	NVIDIA GeForce GTX 1650
Об'єм оперативної пам'яті	8-16 ГБ
Місце на жорсткому диску	1 ГБ
Операційна система	Windows 10 / Ubuntu 22.04

Інсталяція застосунку, покрокова інструкція користувача:

1. Розпакуйте 3DPointCloudClassifier.rar архів за допомогою WinRAR архіватора.

2. Перейдіть до папки, у яку ви розпакували архів.

3. Знайдіть файл 3DPointCloudClassifier.exe та запустіть його, після чого відкриється головне вікно застосунку, зображене на рисунку 4.1.

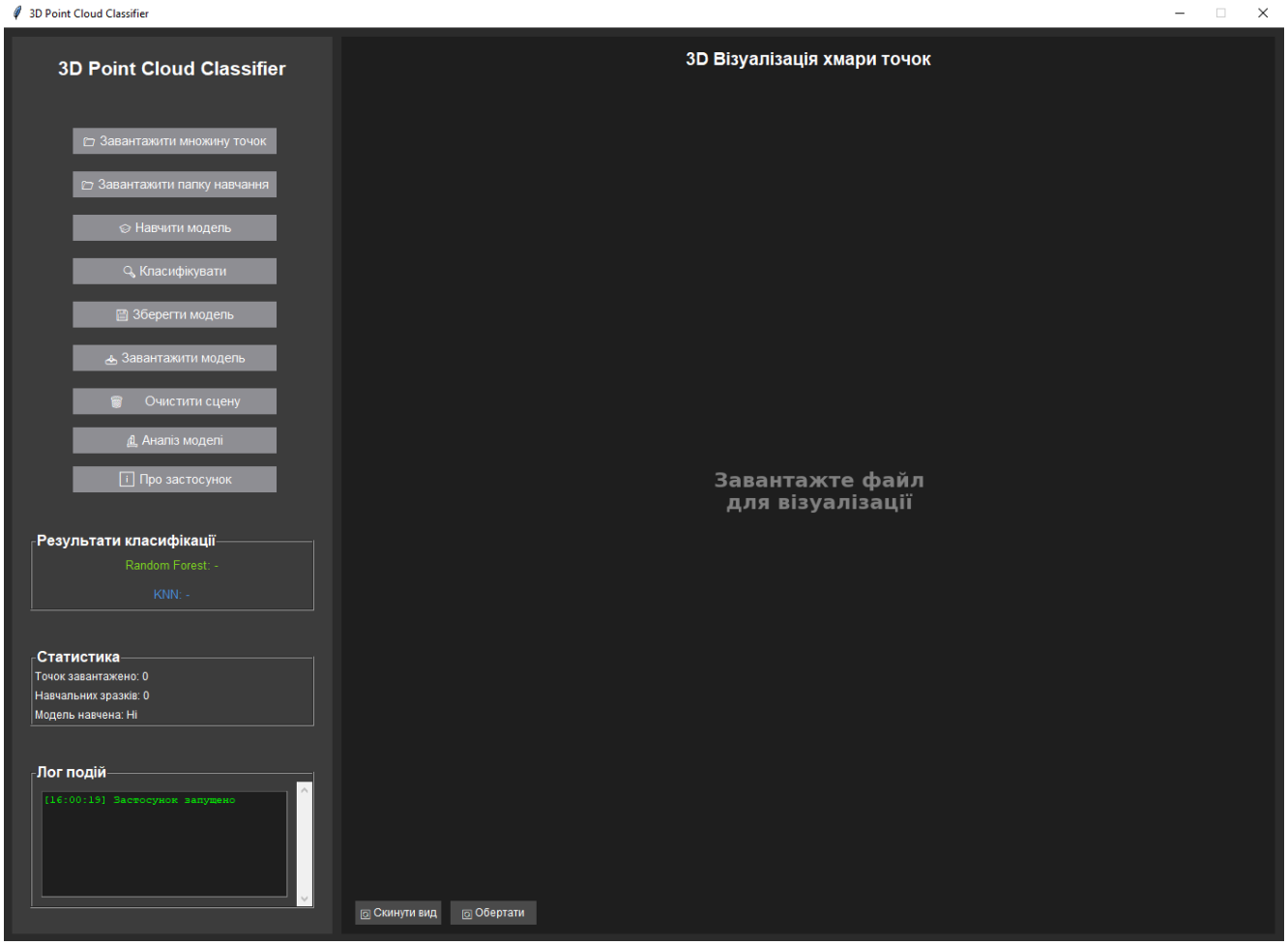


Рисунок 4.1 – Головне вікно

Покрокова інструкція використання:

1. Завантаження хмари точок для подальшої візуалізації. Натисніть «Завантажити множину точок», у діалоговому вікні виберіть файл у форматі OFF, PLY, XYZ, PCD чи OBJ, а після підтвердження дочекайтеся завершення імпорту. Після успішного завантаження хмара точок одразу з’являється у 3D-вікні, у

статистиці показується кількість точок, а в журналі подій фіксується повідомлення про успіх.

Щоб повернути камеру до початкової позиції, натисніть «Скинути вид»; для автоматичного обертання сцени – «Обертати». Мишу використовуйте так: ліва кнопка обертає сцену, коліщатко змінює масштаб, а права кнопка переміщує її.

На рисунку 4.2 зображено результат завантаження множини 3D-точок.

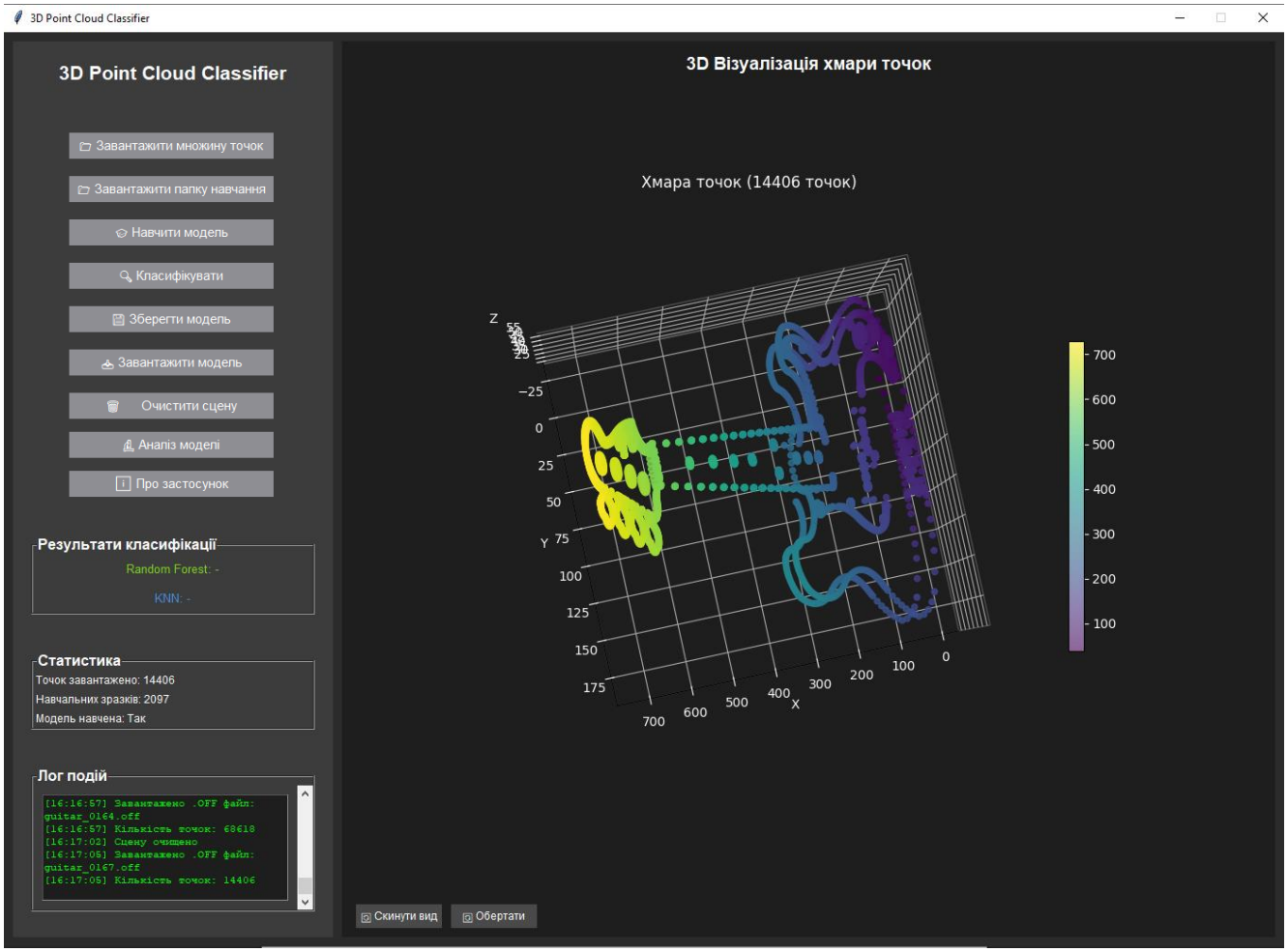


Рисунок 4.2 – Результат завантаження множини 3D-точок

2. Завантаження даних для навчання, для початку створіть структуру каталогів, де кожна підпапка представляє окремий клас, а її назва буде використовуватись як мітка цього класу під час навчання. У кожній папці повинні знаходитись файли з хмарами точок.

На рисунку 4.3 зображено приклад структури каталогів.

```
├── Навчальні_дані/
│   ├── Клас_1/
│   │   ├── зразок1.off
│   │   └── зразок2.off
├── Клас_2/
│   ├── зразок1.off
│   ├── зразок2.off
│   └── ....
└── Клас_N/
    └── ...
```

Рисунок 4.3 – Приклад структури каталогів

Після чого натисніть кнопку «Завантажити папку навчання» та виберіть кореневу папку з підготовленими даними. Після підтвердження відобразиться індикатор прогресу, журнал покаже оброблені класи, а на панелі статистики буде загальна кількість зразків.

На рисунку 4.4 зображено стан головного вікна після завантаження навчальних зразків.

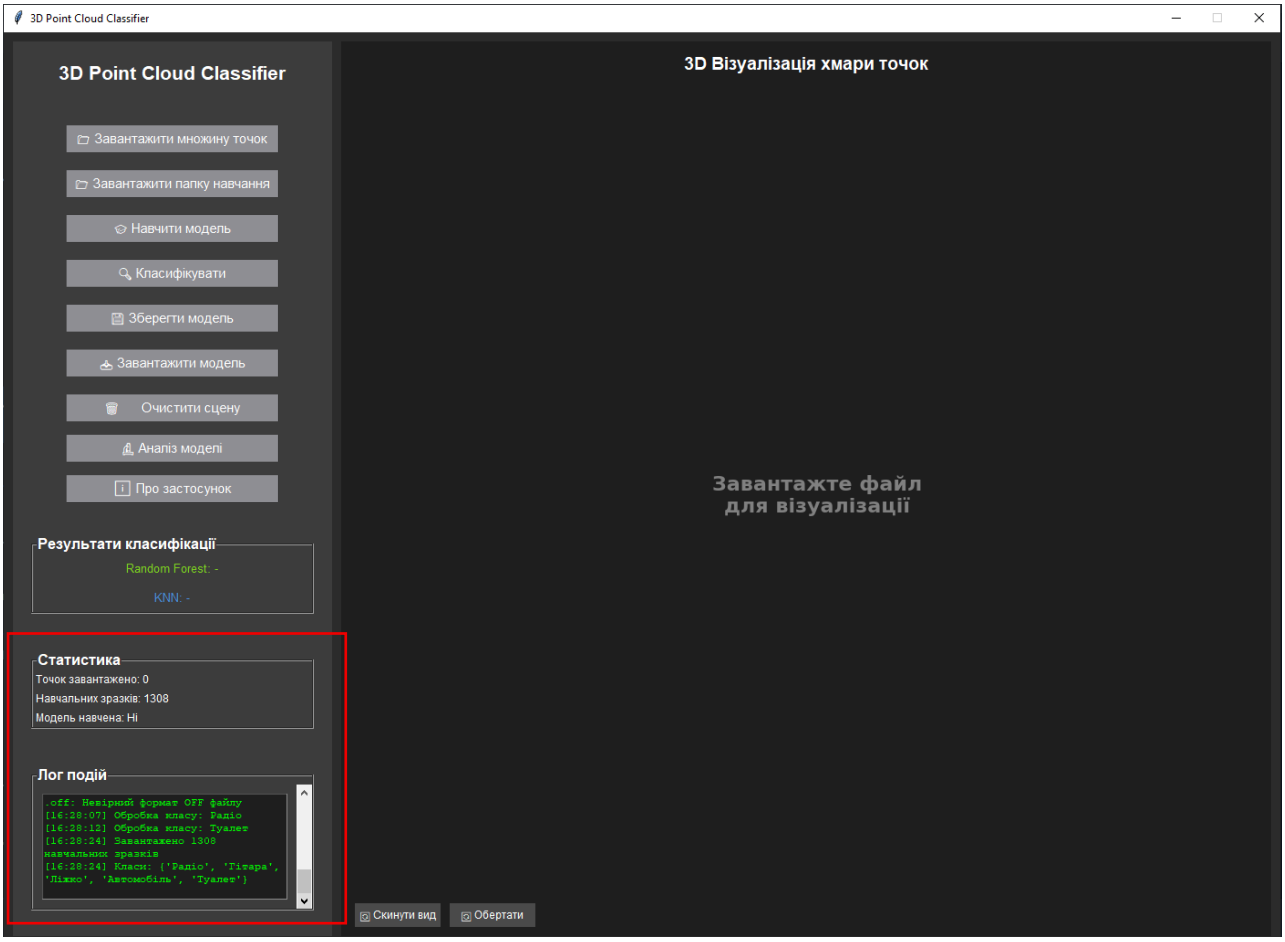


Рисунок 4.4 – Результат завантаження навчальних зразків

3. Навчання моделі, перед початком переконайтеся, що дані завантажено. Натисніть «Навчити модель» і дочекайтеся завершення процесу. У журналі буде виведено точність двох моделей – Random Forest і K-NN. Стан зміниться на «Модель навчена: Так».

Під час навчання відбувається:

– витягування 16 ознак з кожної хмари точок;

– нормалізація векторів ознак;

– розділення на навчальну та тестову вибірки;

– навчання моделі Random Forest;

– навчання моделі K-NN;

– крос-валідація для оцінки точності.

На рисунку 4.5 зображено результати навчання моделі.

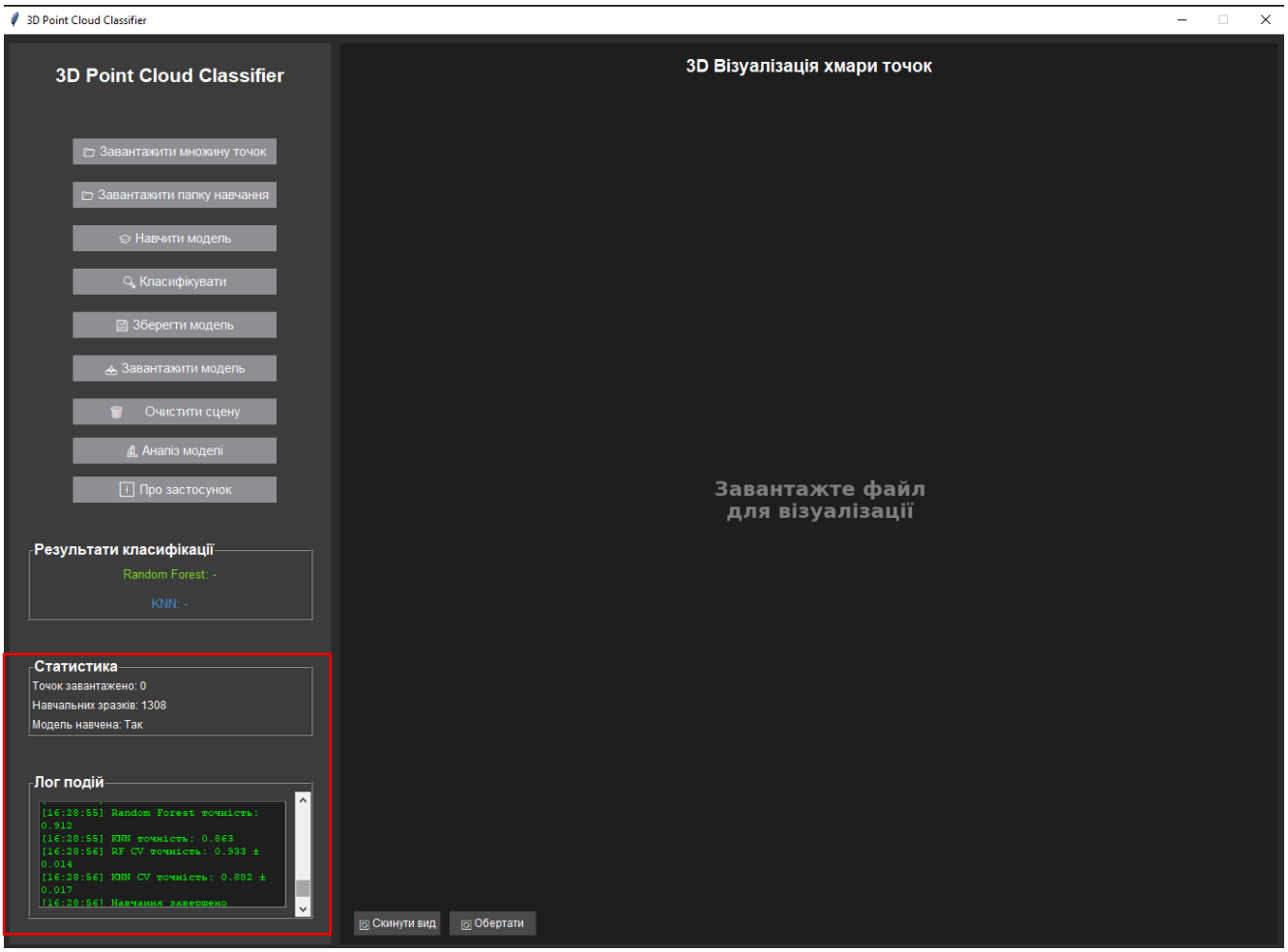


Рисунок 4.5 – Навчання моделі

4. Класифікація нових хмар точок, спотчатку завантажте хмару точок (див. пункт 1), переконайтеся, що модель вже навчена, і натисніть кнопку «Класифікувати». У панелі результатів буде відображено:

– результат класифікації від Random Forest з рівнем впевненості;

– результат від K-NN з відповідним значенням впевненості.

На рисунку 4.6 зображено результати тестової класифікації.

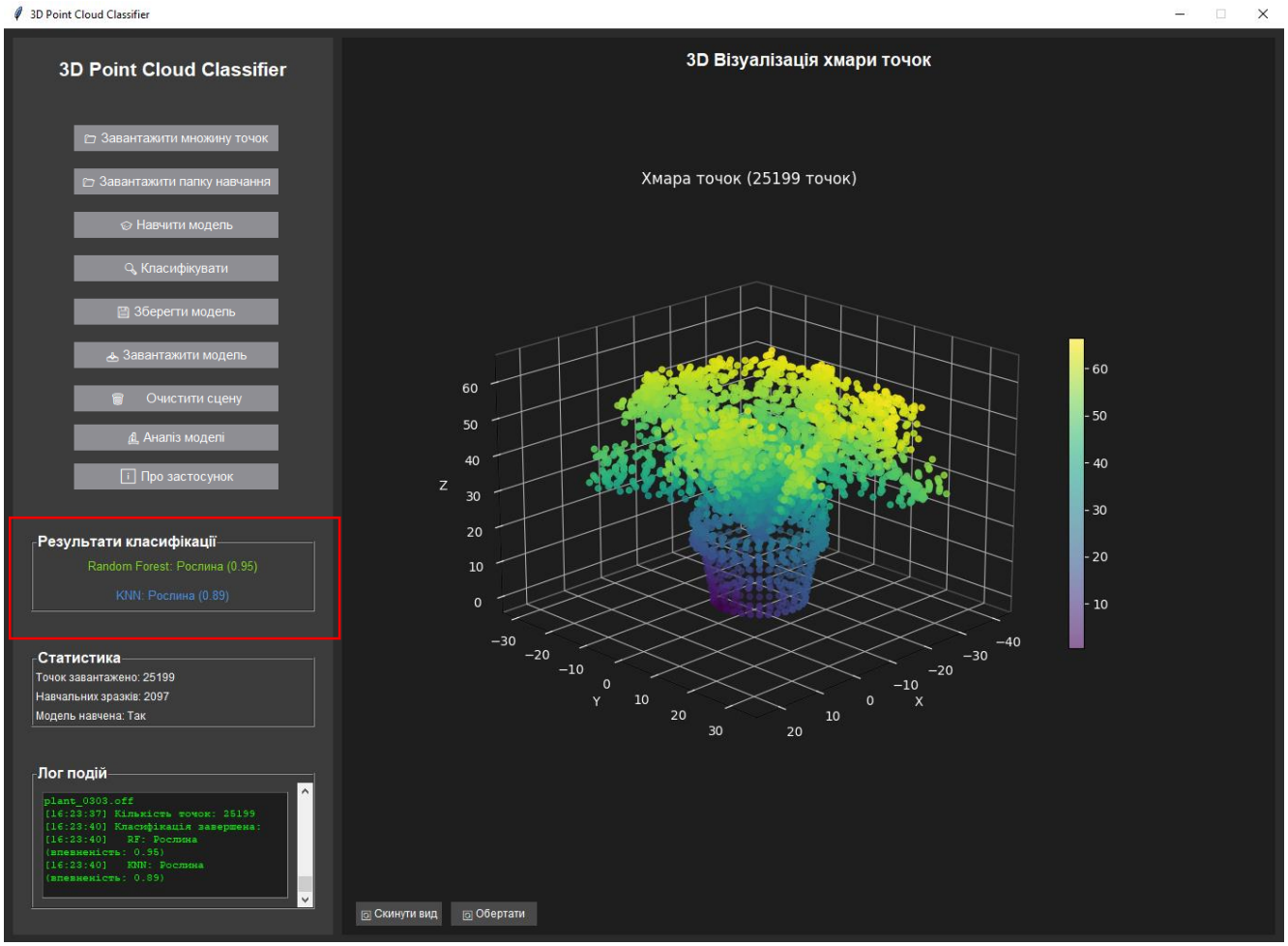


Рисунок 4.6 – Тестова класифікація

5. Збереження та завантаження моделі, щоб зберегти модель, натисніть «Зберегти модель», виберіть місце збереження та вкажіть ім’я файлу з розширенням .pkl. Для відновлення моделі натисніть «Завантажити модель», оберіть файл моделі та підтвердьте імпорт – система автоматично підготує її до подальшого використання.

6. Аналіз моделі, ця функція доступна після завершення навчання моделі. Натисніть «Аналіз моделі», після чого програма згенерує:

– матриці помилок для обох моделей;

– порівняння точності;

– статистику по кожному класу.

На рисунку 4.7 зображено результат аналізу моделі.

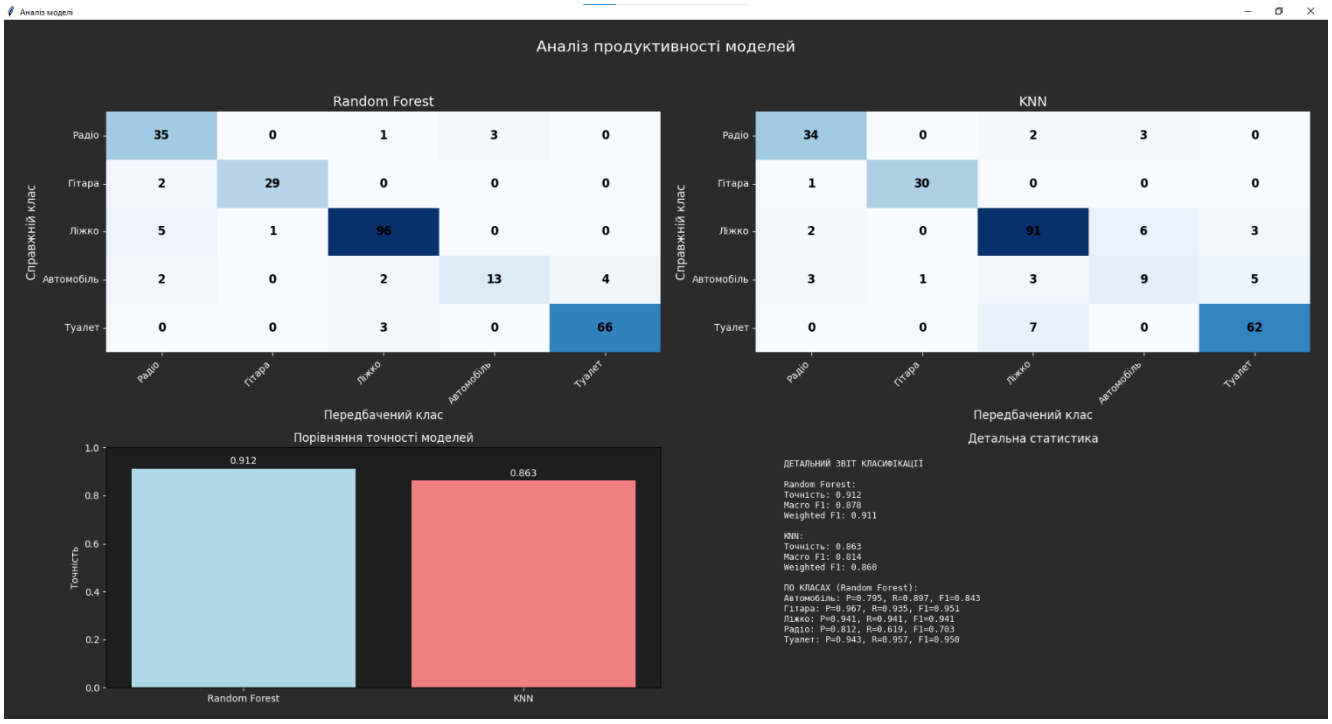


Рисунок 4.7 – Аналіз моделі

7. Очищення сцени, кнопка «Очистити сцену» дозволяє видалити поточну хмару точок і результати класифікації, щоб підготувати інтерфейс до нового завдання.

8. Кнопка «Про застосунок» відкриває довідкову інформацію про програмне забезпечення.

Отже, ця інструкція допоможе вам ефективно використовувати всі можливості 3D Point Cloud Classifier для вирішення задач класифікації хмар точок.

4.3 Висновки

У четвертому розділі було проведене функціональне тестування, яке показало, що розроблений застосунок повністю готовий до роботи та стійкий до відмов. Також було розроблено інструкцію користувача, яка допоможе ефективно використовувати всі можливості застосунку.

ВИСНОВКИ

У першому розділі було проведено аналіз існуючих програмних рішень для класифікації множин 3D-точок, зосереджуючись на вивчені таких аналогів, як CloudCompare, Open3D, Point Data Abstraction Library та Potree. Дослідження дозволило виявити, що ці інструменти мають деякі переваги, але все ж демонструють суттєві недоліки, які обмежують їх ефективність для вирішення поставленої задачі.

Проведений порівняльний аналіз підтвердив потребу у власному програмному застосунку для класифікації 3D-точок з використанням Python у середовищі Visual Studio Code з бібліотеками Tkinter, scikit-learn і Matplotlib. Такий застосунок об’єднає найкращі функції існуючих рішень та вдосконалить, реалізувавши їх через через зручний графічний інтерфейс, який буде оптимізовано для наукових і навчальних потреб, що забезпечить гнучкість і ефективність у класифікації множин 3D-точок.

У другому розділі проведено аналіз і реалізацію ключових компонентів програмного забезпечення для класифікації множини 3D-точок. Розглянуто формат вхідних даних, їх особливості та вимоги до обробки. Змодельовано архітектуру застосунку, що забезпечує масштабованість. Реалізовано підсистему формування ознак для підготовки даних до навчання моделей, а також підсистему машинного навчання з можливістю тренування, збереження та завантаження моделей. Завершено розробкою графічного інтерфейсу, який забезпечує зручну взаємодію з функціоналом, включно з візуалізацією, класифікацією та переглядом результатів.

У третьому розділі було реалізовано функціональні програмні модулі, що відповідають за основні етапи обробки та класифікації 3D-хмар точок. Для розробки застосунку було обрано мову програмування Python, яка завдяки своїй гнучкості та великій кількості готових бібліотек є зручною для створення інструментів обробки просторових даних. Розробка здійснювалася в середовищі Visual Studio Code, що надало комфортні умови для написання, налагодження та структурування коду. До роботи були залучені такі бібліотеки, як scikit-learn для

реалізації алгоритмів машинного навчання, NumPy і Pandas для роботи з даними, Matplotlib для побудови графіків і візуалізації результатів, а також Tkinter для створення графічного інтерфейсу. Завдяки такому поєднанню засобів програмна система вийшла функціонально повною, зручною для користувача та такою, що легко піддається подальшому розширенню або адаптації під нові задачі.

У четвертому розділі було проведене функціональне тестування, яке показало, що розроблений застосунок повністю готовий до роботи та стійкий до відмов. Також було розроблено інструкцію користувача, яка допоможе ефективно використовувати всі можливості застосунку.

Результати бакалаврської кваліфікаційної роботи оформлено згідно методичних вказівок [21].

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What is LIDAR? URL: https://oceanservice.noaa.gov/facts/lidar.html (дата звернення: 03.04.2025).

2. Гуменюк О. В. КЛАСИФІКАЦІЯ МНОЖИНИ 3D-ТОЧОК НА ОСНОВІ МЕТОДІВ МАШИННОГО НАВЧАННЯ / О. В. Гуменюк, О. М. Рейда // Молодь в науці: дослідження, проблеми, перспективи (МН-2025). URL: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/25027

3. K-NN Models URL: https://itwiki.dev/data-science/ml-reference/ml-glossary/knn-models (дата звернення: 04.04.2025).

4. Random Forest Algoritm in Machine Learning URL: https://www.geeksforgeeks.org/random-forest-algorithm-in-machine-learning/ (дата звернення: 04.04.2025).

5. PointNet and PointNet++. URL: https://www.pair.toronto.edu/csc2547-w21/assets/slides/lec2-csc2547-w21-dylan-pointnet.pdf (дата звернення: 04.04.2025).

6. CloudCompare URL: https://www.danielgm.net/cc/ (дата звернення: 06.04.2025).

7. Open3D URL: https://www.open3d.org/ (дата звернення: 06.04.2025).

8. PDAL URL: https://pdal.io/en/stable/ (дата звернення: 06.04.2025).

9. Potree URL: https://github.com/potree/potree/blob/develop/README.md (дата звернення: 06.04.2025).

10. UX-дизайн URL: https://prjctr.com/mag/uxui-questions (дата звернення: 13.04.2025).

11. Bernd Klein. Funktionale Programmierung mit Python. München: Carl Hanser Verlag; 1st ed., 2025. 368 p.

12. Farrier J. Data Structures and Algorithms with the C++ STL. Birmingham: Packt Publishing; 1st ed., 2024. 450 p.

13. Сьєра Кеті, Бейтс Берт. Head First Java, друге видання: науково популярне видання. Харків: ВД "Фабула", 2022. 720 с.

14. Federico Kereki. Mastering JavaScript Functional Programming. Birmingham : Packt Publishing; 3rd ed. Edition, 2023. 614 p.

15. Visual Studio Code URL: https://code.visualstudio.com/ (дата звернення: 15.04.2025).

16. Scikit-learn URL: https://scikit-learn.org/stable/ (дата звернення: 16.04.2025).

17. NumPy. URL: https://arxiv.org/pdf/2006.10256 (дата звернення: 16.04.2025).

18. Matplotlib URL: https://matplotlib.org/ (дата звернення: 16.04.2025).

19. Що таке функціональне тестування? URL: https://www.zaptest.com/uk (дата звернення: 25.04.2025).

20. Керівництво користувача. URL: https://wiki.cusu.edu.ua/index.php (дата звернення: 26.04.2025).

21. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів студентів спеціальності 121 "Інженерія програмного забезпечення" / Уклад. О. Н. Романюк, Г. О. Черноволик – Вінниця: ВНТУ, 2022. – 51с.

ДОДАТКИ

Додаток А – Технічне завдання

Додаток А – Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
«25» березня 2025 р.

Технічне завдання
на бакалаврську кваліфікаційну роботу «Розробка програмного застосунку для класифікації множини 3D точок»
за спеціальністю
121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
к.т.н., доц. Рейда О. М.
березня 2025 року.
Виконав:
студент гр. 4ПІ-21б Гуменюк О. В.
березня 2025 року.

м. Вінниця – 2025 рік

1. Найменування та галузь застосування.

Бакалаврська кваліфікаційна робота: «Розробка програмного застосунку для класифікації множини 3D точок».

Галузь застосування – машинне навчання.

2. Підстава для розробки.

Завдання на роботу, яке затверджене на засіданні кафедри програмного забезпечення – протокол №18 від 18 лютого 2025 р.

3. Мета та призначення розробки.

Метою роботи є підвищення точності і якості класифікації множини 3D-точок у процесі обробки хмари тривимірних даних з використанням методів машинного навчання.

4. Вихідні дані для проведення НДР.

1. Guo, Y.; Wang, H.; Hu, Q. et al. Deep Learning for 3D Point Clouds: A Survey // IEEE Transactions on Pattern Analysis and Machine Intelligence, 2020, 43(12):4338–4364.

2. Qi, C. R.; Su, H.; Mo, K.; Guibas, L. J. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 652–660.

3. Hackel, T.; Wegner, J. D.; Schindler, K. Fast Semantic Segmentation of 3D Point Clouds with Strongly Varying Density // ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, 2016, III-3:177–184.

4. Pedregosa, F. et al. Scikit-learn: Machine Learning in Python // Journal of Machine Learning Research, 2011, 12:2825–2830.

5. Rusu, R. B.; Cousins, S. 3D is here: Point Cloud Library (PCL) // IEEE International Conference on Robotics and Automation, 2011, pp. 1–4.

5. Технічні вимоги

Вхідні дані – множина 3D-точок.

Вихідні дані – клас приналежності.

6. Конструктивні вимоги.

Інтерфейс програми повинен відповідати естетичним та ергономічним вимогам, повинен бути зручним в використанні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

– пояснювальна записка до бакалаврської кваліфікаційної роботи;

– технічне завдання;

– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз сучасного тану питання та обґрунтування завдання на роботу	25.03.25 – 30.03.25
2	Розробка структури та алгоритмів роботи програмного застосунку	31.03.25 – 16.04.25
3	Розробка програмних модулів реалізації застосунку	17.04.25 – 07.05.25
4	Аналіз і обґрунтування вибору засобів реалізації застосунку	08.05.25 – 24.05.25
5	Тестування застосунку	25.05.25 – 27.05.25
6	Оформлення матеріалів до захисту БКР	28.05.25 – 30.05.25

10. Порядок контролю та прийняття.

Усі етапи бакалаврської кваліфікаційної роботи контролюються науковим керівником згідно плану по виконанню роботи. Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком захисту.

Додаток Б – Протокол перевірки на плагіат

Додаток Б – Протокол перевірки на плагіат

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: «Розробка програмного застосунку для класифікації множини 3D точок»

Тип роботи: бакалаврська кваліфікаційна робота

Підрозділ: кафедра програмного забезпечення, ФІТКІ, 4ПІ-21б

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 6,8 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

(прізвище, ініціали, посада) (підпис)

(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник _____ к.т.н., доцент каф. ПЗ Рейда О.М.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Гуменюк О. В.
(підпис) (прізвище, ініціали)

Додаток В – Лістинг програми

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.figure import Figure
import tkinter as tk
from tkinter import filedialog, messagebox, ttk
import os
import pickle
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.preprocessing import StandardScaler
import threading
import time
import seaborn as sns

class PointCloudClassifier:
    def __init__(self):
        self.root = tk.Tk()
        self.root.title("3D Point Cloud Classifier")
        self.root.geometry("1400x920")
        self.root.configure(bg='#2b2b2b')
        self.root.resizable(False, False)

        # Дані та моделі
        self.point_cloud = None
        self.training_data = []
        self.training_labels = []
        self.rf_model = None
        self.knn_model = None
        self.scaler = StandardScaler()
        self.is_trained = False

        # Matplotlib об'єкти
        self.fig = None
        self.canvas = None
```

```
        self.ax = None

        # Створення інтерфейсу
        self.create_interface()

    def create_interface(self):
        # Головна рамка
        main_frame = tk.Frame(self.root, bg='#2b2b2b')
        main_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

        # Ліва панель з кнопками
        left_panel = tk.Frame(main_frame, bg='#3c3c3c', width=350)
        left_panel.pack(side=tk.LEFT, fill=tk.Y, padx=(0, 10))
        left_panel.pack_propagate(False)

        # Заголовок
        title_label = tk.Label(left_panel, text="3D Point Cloud Classifier",
                               font=('Arial', 16, 'bold'), fg='white', bg='#3c3c3c')
        title_label.pack(pady=(20, 30))

        # Кнопки
        button_style = {
            'font': ('Arial', 11),
            'width': 25,
            'height': 1,
            'relief': tk.FLAT,
            'cursor': 'hand2'
        }

        # Завантаження файлу
        self.load_off_btn = tk.Button(left_panel, text="📂 Завантажити множину точок",
                                      command=self.load_off_file,
                                      bg='#4a90e2', fg='white', **button_style)
        self.load_off_btn.pack(pady=10, padx=20)

        # Завантаження папки навчання
        self.load_training_btn = tk.Button(left_panel, text="📂 Завантажити папку навчання",
                                           command=self.load_training_folder,
```

```
                        bg='#7ed321', fg='white', **button_style)
self.load_training_btn.pack(pady=10, padx=20)

# Навчання моделі
self.train_btn = tk.Button(left_panel, text="🎓 Навчити модель",
                           command=self.train_models,
                           bg='#9013fe', fg='white', **button_style)
self.train_btn.pack(pady=10, padx=20)

# Класифікація
self.classify_btn = tk.Button(left_panel, text="🔍 Класифікувати",
                              command=self.classify_point_cloud,
                              bg='#ff6900', fg='white', **button_style)
self.classify_btn.pack(pady=10, padx=20)

# Зберегти/Завантажити модель
self.save_model_btn = tk.Button(left_panel, text="💾 Зберегти модель",
                                command=self.save_model,
                                bg='#50e3c2', fg='white', **button_style)
self.save_model_btn.pack(pady=10, padx=20)

self.load_model_btn = tk.Button(left_panel, text="📥 Завантажити модель",
                                command=self.load_model,
                                bg='#b8e986', fg='white', **button_style)
self.load_model_btn.pack(pady=10, padx=20)

# Кнопка очищення сцени
self.clear_btn = tk.Button(left_panel, text="🗑️ Очистити сцену",
                           command=self.clear_scene,
                           bg='#f5a623', fg='white', **button_style)
self.clear_btn.pack(pady=10, padx=20)

# Кнопка аналізу моделі
self.model_analysis_btn = tk.Button(left_panel, text="🔬 Аналіз моделі",
                                    command=self.show_model_analysis,
                                    bg='#ff5722', fg='white', **button_style)
self.model_analysis_btn.pack(pady=5, padx=20)
```

```
# Кнопка інформації
self.info_btn = tk.Button(left_panel, text="ℹ Про застосунок",
                          command=self.show_info,
                          bg='#8e8e93', fg='white', **button_style)
self.info_btn.pack(pady=10, padx=20)

# Результати класифікації
self.result_frame = tk.LabelFrame(left_panel, text="Результати класифікації",
                                  font=('Arial', 12, 'bold'), fg='white', bg='#3c3c3c')
self.result_frame.pack(pady=20, padx=20, fill=tk.X)

self.rf_result_label = tk.Label(self.result_frame, text="Random Forest: -",
                                font=('Arial', 10), fg='#7ed321', bg='#3c3c3c')
self.rf_result_label.pack(pady=5)

self.knn_result_label = tk.Label(self.result_frame, text="KNN: -",
                                 font=('Arial', 10), fg='#4a90e2', bg='#3c3c3c')
self.knn_result_label.pack(pady=5)

# Статистика
self.stats_frame = tk.LabelFrame(left_panel, text="Статистика",
                                 font=('Arial', 12, 'bold'), fg='white', bg='#3c3c3c')
self.stats_frame.pack(pady=20, padx=20, fill=tk.X)

self.points_label = tk.Label(self.stats_frame, text="Точок завантажено: 0",
                             font=('Arial', 9), fg='white', bg='#3c3c3c')
self.points_label.pack(anchor=tk.W)

self.training_label = tk.Label(self.stats_frame, text="Навчальних зразків: 0",
                               font=('Arial', 9), fg='white', bg='#3c3c3c')
self.training_label.pack(anchor=tk.W)

self.model_label = tk.Label(self.stats_frame, text="Модель навчена: Ні",
                            font=('Arial', 9), fg='white', bg='#3c3c3c')
self.model_label.pack(anchor=tk.W)

# Лог подій
```

```
    log_frame = tk.LabelFrame(left_panel, text="Лог подій",
                              font=('Arial', 12, 'bold'), fg='white', bg='#3c3c3c')
    log_frame.pack(pady=20, padx=20, fill=tk.BOTH, expand=True)

    self.log_text = tk.Text(log_frame, height=8, width=40, font=('Courier', 8),
                            bg='#1e1e1e', fg='#00ff00', wrap=tk.WORD)
    scrollbar = tk.Scrollbar(log_frame)
    scrollbar.pack(side=tk.RIGHT, fill=tk.Y)
    self.log_text.pack(pady=10, padx=10, fill=tk.BOTH, expand=True)
    self.log_text.config(yscrollcommand=scrollbar.set)
    scrollbar.config(command=self.log_text.yview)

    # Прогрес бар
    self.progress = ttk.Progressbar(left_panel, mode='indeterminate')

    # Права панель для візуалізації
    self.right_panel = tk.Frame(main_frame, bg='#1e1e1e')
    self.right_panel.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True)

    # Заголовок візуалізації
    viz_title = tk.Label(self.right_panel, text="3D Візуалізація хмари точок",
                         font=('Arial', 14, 'bold'), fg='white', bg='#1e1e1e')
    viz_title.pack(pady=(10, 5))

    # Ініціалізація matplotlib
    self.setup_matplotlib()



    self.add_log("Застосунок запущено")

def show_model_analysis(self):
    if not self.is_trained:
        messagebox.showwarning("Попередження", "Спочатку навчіть модель")
        return
    try:
        model_window = tk.Toplevel(self.root)
        model_window.title("Аналіз моделі")
        model_window.geometry("1400x900")
```

```
        model_window.configure(bg='#2b2b2b')
        X = np.array(self.training_data)
        y = np.array(self.training_labels)
        X_scaled = self.scaler.transform(X)
        X_train, X_test, y_train, y_test = train_test_split(
            X_scaled, y, test_size=0.2, random_state=42, stratify=y
        )
        rf_pred = self.rf_model.predict(X_test)
        knn_pred = self.knn_model.predict(X_test)
        fig, axes = plt.subplots(2, 2, figsize=(18, 10))
        fig.patch.set_facecolor('#2b2b2b')
        fig.suptitle('Аналіз продуктивності моделей', fontsize=16, color='white')
        cm_rf = confusion_matrix(y_test, rf_pred)
        self._plot_confusion_matrix(axes[0, 0], cm_rf, list(set(y)), 'Random Forest')
        cm_knn = confusion_matrix(y_test, knn_pred)
        self._plot_confusion_matrix(axes[0, 1], cm_knn, list(set(y)), 'KNN')
        self._plot_accuracy_comparison(axes[1, 0], y_test, rf_pred, knn_pred)
        self._plot_classification_report(axes[1, 1], y_test, rf_pred, knn_pred)
        plt.tight_layout(rect=[0, 0, 1, 0.95])
        canvas = FigureCanvasTkAgg(fig, model_window)
        canvas.draw()
        canvas.get_tk_widget().pack(fill=tk.BOTH, expand=True, padx=10, pady=10)
        self.add_log("Аналіз моделі відображено")
    except Exception as e:
        messagebox.showerror("Помилка", f"Помилка аналізу моделі: {str(e)}")
        self.add_log(f"Помилка аналізу моделі: {str(e)}")

def _plot_confusion_matrix(self, ax, cm, classes, title):
    """Малює матрицю помилок"""
    """Малює матрицю помилок з покращеною читабельністю"""
    # Використовуємо seaborn heatmap для кращої візуалізації
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', ax=ax, cbar=False,
                annot_kws={"size": 12, "weight": "bold", "color": "black"})

    # Налаштування підписів осей
    ax.set_xlabel('Передбачений клас', fontsize=12, color='white')
    ax.set_ylabel('Справжній клас', fontsize=12, color='white')
    ax.set_title(title, fontsize=14, color='white')
```

```
    # Налаштування позначок на осях
    ax.set_xticklabels(classes, rotation=45, ha='right', fontsize=10, color='white')
    ax.set_yticklabels(classes, rotation=0, fontsize=10, color='white')

    # Колір позначок для контрасту
    ax.tick_params(axis='x', colors='white')
    ax.tick_params(axis='y', colors='white')

def _plot_feature_importance(self, ax):
    """Малює важливість ознак"""
    ax.set_facecolor('#1e1e1e')

    feature_names = [
        'Кількість точок', 'Ширина', 'Висота', 'Глибина',
        'Співвідношення Ш/В', 'Співвідношення Ш/Г', 'Співвідношення В/Г',
        'Середня відстань', 'Стд відстані', 'Густина',
        'Центроїд X', 'Центроїд Y', 'Центроїд Z',
        'Власне значення 1', 'Власне значення 2', 'Власне значення 3'
    ]

    importances = self.rf_model.feature_importances_
    indices = np.argsort(importances)[::-1]

    # Беремо топ-10 ознак
    top_indices = indices[:10]
    top_importances = importances[top_indices]
    top_names = [feature_names[i] for i in top_indices]

    bars = ax.bar(range(len(top_importances)), top_importances, color='lightblue')
    ax.set_xlabel('Ознаки', color='white')
    ax.set_ylabel('Важливість', color='white')
    ax.set_title('Важливість ознак (Random Forest)', color='white')
    ax.set_xticks(range(len(top_names)))
    ax.set_xticklabels([name[:10] + '...' if len(name) > 10 else name for name in top_names],
                       rotation=45, ha='right', color='white')
    ax.tick_params(colors='white')
```

```
def _plot_accuracy_comparison(self, ax, y_test, rf_pred, knn_pred):
    """Порівняння точності моделей"""
    ax.set_facecolor('#1e1e1e')

    rf_accuracy = accuracy_score(y_test, rf_pred)
    knn_accuracy = accuracy_score(y_test, knn_pred)

    models = ['Random Forest', 'KNN']
    accuracies = [rf_accuracy, knn_accuracy]
    colors = ['lightblue', 'lightcoral']

    bars = ax.bar(models, accuracies, color=colors)
    ax.set_ylabel('Точність', color='white')
    ax.set_title('Порівняння точності моделей', color='white')
    ax.set_ylim([0, 1])
    ax.tick_params(colors='white')

    # Додаємо значення на стовпці
    for bar, acc in zip(bars, accuracies):
        ax.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.01,
                f'{acc:.3f}', ha='center', va='bottom', color='white')

def _plot_classification_report(self, ax, y_test, rf_pred, knn_pred):
    """Детальний звіт класифікації"""
    ax.set_facecolor('#1e1e1e')
    ax.axis('off')

    # Генеруємо звіти
    rf_report = classification_report(y_test, rf_pred, output_dict=True)
    knn_report = classification_report(y_test, knn_pred, output_dict=True)

    # Формуємо текст
    report_text = "ДЕТАЛЬНИЙ ЗВІТ КЛАСИФІКАЦІЇ\n\n"
    report_text += "Random Forest:\n"
    report_text += f"Точність: {rf_report['accuracy']:.3f}\n"
    report_text += f"Macro F1: {rf_report['macro avg']['f1-score']:.3f}\n"
    report_text += f"Weighted F1: {rf_report['weighted avg']['f1-score']:.3f}\n\n"
```

```
    report_text += "KNN:\n"
    report_text += f"Точність: {knn_report['accuracy']:.3f}\n"
    report_text += f"Macro F1: {knn_report['macro avg']['f1-score']:.3f}\n"
    report_text += f"Weighted F1: {knn_report['weighted avg']['f1-score']:.3f}\n\n"

    # Додаємо інформацію по класах
    report_text += "ПО КЛАСАХ (Random Forest):\n"
    for class_name in rf_report:
        if class_name not in ['accuracy', 'macro avg', 'weighted avg']:
            precision = rf_report[class_name]['precision']
            recall = rf_report[class_name]['recall']
            f1 = rf_report[class_name]['f1-score']
            report_text += f"{class_name}: P={precision:.3f}, R={recall:.3f}, F1={f1:.3f}\n"

    ax.text(0.05, 0.95, report_text, transform=ax.transAxes,
            verticalalignment='top', fontfamily='monospace',
            color='white', fontsize=9)
    ax.set_title('Детальна статистика', color='white')

def clear_scene(self):
    result = messagebox.askyesno("Підтвердження",
                                 "Ви дійсно хочете очистити сцену?\n"
                                 "Це видалить завантажену хмару точок та результати класифікації.")
    if not result:
        return
    try:
        self.point_cloud = None
        self.rf_result_label.config(text="Random Forest: -")
        self.knn_result_label.config(text="KNN: -")
        self.points_label.config(text="Точок завантажено: 0")
        if hasattr(self, '_rotating'):
            self._rotating = False
        try:
            if hasattr(self, '_colorbar') and self._colorbar is not None:
                try:
                    self._colorbar.remove()
                except:
                    pass
```

```
                self._colorbar = None
            except:
                pass
            self.show_placeholder()
            self.add_log("Сцену очищено")
        except Exception as e:
            self.add_log(f"Помилка очищення: {str(e)}")
            messagebox.showerror("Помилка", f"Помилка очищення сцени: {str(e)}")

    def setup_matplotlib(self):
        self.fig = Figure(figsize=(10, 8), dpi=100, facecolor='#1e1e1e')
        self.ax = self.fig.add_subplot(111, projection='3d', facecolor='#1e1e1e')
        self.ax.xaxis.label.set_color('white')
        self.ax.yaxis.label.set_color('white')
        self.ax.zaxis.label.set_color('white')
        self.ax.tick_params(axis='x', colors='white')
        self.ax.tick_params(axis='y', colors='white')
        self.ax.tick_params(axis='z', colors='white')
        self.ax.xaxis.pane.fill = False
        self.ax.yaxis.pane.fill = False
        self.ax.zaxis.pane.fill = False
        self.ax.xaxis.pane.set_edgecolor('gray')
        self.ax.yaxis.pane.set_edgecolor('gray')
        self.ax.zaxis.pane.set_edgecolor('gray')
        self.ax.grid(True, alpha=0.3)
        self.canvas = FigureCanvasTkAgg(self.fig, self.right_panel)
        self.canvas.draw()
        self.canvas.get_tk_widget().pack(fill=tk.BOTH, expand=True, padx=10, pady=10)
        toolbar_frame = tk.Frame(self.right_panel, bg='#1e1e1e')
        toolbar_frame.pack(fill=tk.X, padx=10, pady=(0, 10))
        btn_style = {'font': ('Arial', 9), 'width': 12, 'height': 1,
                     'relief': tk.FLAT, 'cursor': 'hand2', 'bg': '#4a4a4a', 'fg': 'white'}
        reset_btn = tk.Button(toolbar_frame, text="🔄 Скинути вид",
                              command=self.reset_view, **btn_style)
        reset_btn.pack(side=tk.LEFT, padx=5)
        rotate_btn = tk.Button(toolbar_frame, text="🔄 Обертати",
                               command=self.toggle_rotation, **btn_style)
        rotate_btn.pack(side=tk.LEFT, padx=5)
```

```
    self.show_placeholder()

def show_placeholder(self):
    """Показує placeholder текст коли немає даних"""
    self.ax.clear()
    self.ax.text2D(0.5, 0.5, "Завантажте файл\nдля візуалізації",
                   transform=self.ax.transAxes, ha='center', va='center',
                   fontsize=16, color='gray', weight='bold')
    self.ax.set_xlim([0, 1])
    self.ax.set_ylim([0, 1])
    self.ax.set_zlim([0, 1])
    self.ax.axis('off')
    self.canvas.draw()

def reset_view(self):
    """Скидає вид 3D графіка"""
    if self.point_cloud is not None:
        self.ax.view_init(elev=20, azim=45)
        self.canvas.draw()

def toggle_rotation(self):
    """Увімкнути/вимкнути автоматичне обертання"""
    # Простий приклад анімації
    if hasattr(self, '_rotating') and self._rotating:
        self._rotating = False
        return

    self._rotating = True

    def animate():
        angle = 0
        while self._rotating and self.point_cloud is not None:
            self.ax.view_init(elev=20, azim=angle)
            self.canvas.draw()
            angle = (angle + 2) % 360
            time.sleep(0.05)

    threading.Thread(target=animate, daemon=True).start()
```

```
def add_log(self, message):
    """Додає повідомлення до логу"""
    timestamp = time.strftime("%H:%M:%S")
    log_message = f"[{timestamp}] {message}\n"
    self.log_text.insert(tk.END, log_message)
    self.log_text.see(tk.END)
    self.root.update()

def parse_point_cloud_file(self, filepath):
    file_ext = os.path.splitext(filepath)[1].lower()
    try:
        if file_ext == '.off':
            return self._parse_off(filepath)
        elif file_ext == '.ply':
            return self._parse_ply(filepath)
        elif file_ext == '.xyz':
            return self._parse_xyz(filepath)
        elif file_ext == '.pcd':
            return self._parse_pcd(filepath)
        elif file_ext == '.obj':
            return self._parse_obj(filepath)
        else:
            raise ValueError(f"Непідтримуваний формат файлу: {file_ext}")

    except Exception as e:
        raise Exception(f"Помилка читання файлу {file_ext}: {str(e)}")

def _parse_off(self, filepath):
    """Парсить OFF файл"""
    with open(filepath, 'r') as file:
        lines = [line.strip() for line in file.readlines() if line.strip()]

    if lines[0] != 'OFF':
        raise ValueError("Невірний формат OFF файлу")

    header = lines[1].split()
    num_vertices = int(header[0])
```

```
    vertices = []
    for i in range(2, 2 + num_vertices):
        coords = list(map(float, lines[i].split()[:3]))
        vertices.append(coords)

    return np.array(vertices)

def _parse_ply(self, filepath):
    """Парсить PLY файл"""
    with open(filepath, 'r') as file:
        lines = file.readlines()

    # Знаходимо кінець заголовка
    header_end = 0
    num_vertices = 0
    for i, line in enumerate(lines):
        if line.startswith('element vertex'):
            num_vertices = int(line.split()[2])
        elif line.strip() == 'end_header':
            header_end = i + 1
            break

    # Читаємо вершини
    vertices = []
    for i in range(header_end, header_end + num_vertices):
        coords = list(map(float, lines[i].split()[:3]))
        vertices.append(coords)

    return np.array(vertices)

def _parse_xyz(self, filepath):
    """Парсить XYZ файл"""
    vertices = []
    with open(filepath, 'r') as file:
        for line in file:
            line = line.strip()
            if line and not line.startswith('#'):
```

```
                coords = list(map(float, line.split()[:3]))
                vertices.append(coords)

        return np.array(vertices)

    def _parse_pcd(self, filepath):
        """Парсить PCD файл"""
        with open(filepath, 'r') as file:
            lines = file.readlines()

        # Знаходимо початок даних
        data_start = 0
        num_points = 0
        for i, line in enumerate(lines):
            if line.startswith('POINTS'):
                num_points = int(line.split()[1])
            elif line.startswith('DATA'):
                data_start = i + 1
                break

        # Читаємо точки
        vertices = []
        for i in range(data_start, data_start + num_points):
            if i < len(lines):
                coords = list(map(float, lines[i].split()[:3]))
                vertices.append(coords)

        return np.array(vertices)

    def _parse_obj(self, filepath):
        """Парсить OBJ файл (тільки вершини)"""
        vertices = []
        with open(filepath, 'r') as file:
            for line in file:
                line = line.strip()
                if line.startswith('v '):  # Вершина
                    coords = list(map(float, line.split()[1:4]))
                    vertices.append(coords)
```

```
    return np.array(vertices)

def extract_features(self, points):
    """Витягує ознаки з хмари точок"""
    if len(points) == 0:
        return np.array([])

    num_points, centroid = self._calculate_basic_stats(points)
    dimensions = self._calculate_bounding_box(points)
    avg_dist, std_dist = self._calculate_distance_features(points, centroid)
    aspect_ratios = self._calculate_aspect_ratios(dimensions)
    density = self._calculate_density(num_points, dimensions)
    eigenvalues = self._calculate_eigenvalues(points, centroid)
    width, height, depth = dimensions

    features = [
        num_points,
        width, height, depth,
        *aspect_ratios,
        avg_dist, std_dist,
        density,
        *centroid,
        *eigenvalues
    ]

    return np.array(features)

def _calculate_basic_stats(self, points):
    num_points = len(points)
    centroid = np.mean(points, axis=0)
    return num_points, centroid

def _calculate_bounding_box(self, points):
    min_coords = np.min(points, axis=0)
    max_coords = np.max(points, axis=0)
    return max_coords - min_coords
```

```
def _calculate_distance_features(self, points, centroid):
    distances = np.linalg.norm(points - centroid, axis=1)
    return np.mean(distances), np.std(distances)

def _calculate_aspect_ratios(self, dimensions):
    width, height, depth = dimensions
    return [
        width / (height + 1e-8),
        width / (depth + 1e-8),
        height / (depth + 1e-8)
    ]

def _calculate_density(self, num_points, dimensions):
    width, height, depth = dimensions
    volume = width * height * depth
    return num_points / (volume + 1e-8)

def _calculate_eigenvalues(self, points, centroid):
    centered_points = points - centroid
    cov = np.cov(centered_points.T)
    eigvals = np.linalg.eigvals(cov)
    return np.sort(eigvals)[::-1]

def load_off_file(self):
    filepath = filedialog.askopenfilename(
        title="Виберіть файл хмари точок",
        filetypes=[
            ("Усі підтримувані", "*.off;*.ply;*.xyz;*.pcd;*.obj"),
            ("OFF files", "*.off"),
            ("PLY files", "*.ply"),
            ("XYZ files", "*.xyz"),
            ("PCD files", "*.pcd"),
            ("OBJ files", "*.obj"),
            ("All files", "*.*")
        ]
    )

    if not filepath:
```

```
            return

        try:
            self.point_cloud = self.parse_point_cloud_file(filepath)
            num_points = len(self.point_cloud)
            file_format = os.path.splitext(filepath)[1].upper()
            self.add_log(f"Завантажено {file_format} файл: {os.path.basename(filepath)}")
            self.add_log(f"Кількість точок: {num_points}")
            self.points_label.config(text=f"Точок завантажено: {num_points}")
            self.visualize_point_cloud()

        except Exception as e:
            messagebox.showerror("Помилка", str(e))
            self.add_log(f"Помилка завантаження: {str(e)}")

    def load_training_folder(self):
        """Завантажує папку з навчальними даними"""
        folder_path = filedialog.askdirectory(title="Виберіть папку з навчальними даними")

        if not folder_path:
            return

        try:
            self.progress.pack(pady=10, padx=20, fill=tk.X)
            self.progress.start()

            # Запускаємо завантаження в окремому потоці
            threading.Thread(target=self._load_training_data, args=(folder_path,), daemon=True).start()

        except Exception as e:
            messagebox.showerror("Помилка", str(e))
            self.add_log(f"Помилка завантаження папки: {str(e)}")
            self.progress.stop()
            self.progress.pack_forget()

    def _load_training_data(self, folder_path):
        """Завантажує навчальні дані (запускається в окремому потоці)"""
        try:
```

```
new_data = []
new_labels = []

# Проходимо по всіх підпапках
for class_name in os.listdir(folder_path):
    class_path = os.path.join(folder_path, class_name)
    if not os.path.isdir(class_path):
        continue

    self.add_log(f"Обробка класу: {class_name}")

    # Завантажуємо всі файли з цієї папки
    for filename in os.listdir(class_path):
        if any(filename.lower().endswith(ext) for ext in ['.off', '.ply', '.xyz', '.pcd', '.obj']):
            filepath = os.path.join(class_path, filename)
            try:
                points = self.parse_point_cloud_file(filepath)
                features = self.extract_features(points)

                new_data.append(features)
                new_labels.append(class_name)

            except Exception as e:
                self.add_log(f"Помилка файлу {filename}: {str(e)}")

# Оновлюємо дані
if new_data:
    self.training_data.extend(new_data)
    self.training_labels.extend(new_labels)

    self.add_log(f"Завантажено {len(new_data)} навчальних зразків")
    self.add_log(f"Класи: {set(new_labels)}")

    # Оновлюємо статистику
    self.training_label.config(text=f"Навчальних зразків: {len(self.training_data)}")
else:
    self.add_log("Не знайдено жодного OFF файлу")
```

```
        except Exception as e:
            self.add_log(f"Помилка завантаження: {str(e)}")
        finally:
            self.progress.stop()
            self.progress.pack_forget()

    def train_models(self):
        """Навчає моделі Random Forest та KNN"""
        if len(self.training_data) < 2:
            messagebox.showwarning("Попередження",
                "Потрібно мінімум 2 навчальні зразки")
            return



        self.progress.pack(pady=10, padx=20, fill=tk.X)
        self.progress.start()
        self.add_log("Початок навчання моделей...")
        try:
            # Запускаємо навчання в окремому потоці
            threading.Thread(target=self._train_models, daemon=True).start()
        except Exception as e:
            messagebox.showerror("Помилка", str(e))
        finally:
            self.progress.stop()
            self.progress.pack_forget()
            self.model_label.config(text="Модель навчена: Так")
            self.add_log("Навчання завершено успішно!")

    def _train_models(self):
        """Навчає моделі (запускається в окремому потоці)"""
        X_train, X_test, y_train, y_test = self._prepare_data()

        self._train_random_forest(X_train, y_train)
        self._train_knn(X_train, y_train)

        self._evaluate_models(X_test, y_test)

        self._cross_validate()
```

```
    self.is_trained = True

def _prepare_data(self):
    self.add_log("Підготовка даних...")
    X = np.array(self.training_data)
    y = np.array(self.training_labels)
    X_scaled = self.scaler.fit_transform(X)
    X_train, X_test, y_train, y_test = train_test_split(
        X_scaled, y, test_size=0.2, random_state=42, stratify=y
    )
    return X_train, X_test, y_train, y_test

def _train_random_forest(self, X_train, y_train):
    self.add_log("Навчання Random Forest...")
    self.rf_model = RandomForestClassifier(
        n_estimators=100, random_state=42, n_jobs=-1
    )
    self.rf_model.fit(X_train, y_train)

def _train_knn(self, X_train, y_train):
    self.add_log("Навчання KNN...")
    self.knn_model = KNeighborsClassifier(
        n_neighbors=min(5, len(set(y_train))), weights='distance'
    )
    self.knn_model.fit(X_train, y_train)

def _evaluate_models(self, X_test, y_test):
    rf_score = self.rf_model.score(X_test, y_test)
    knn_score = self.knn_model.score(X_test, y_test)
    self.add_log(f"Random Forest точність: {rf_score:.3f}")
    self.add_log(f"KNN точність: {knn_score:.3f}")

def _cross_validate(self):
    X = np.array(self.training_data)
    y = np.array(self.training_labels)
    X_scaled = self.scaler.transform(X)
    rf_scores = cross_val_score(self.rf_model, X_scaled, y, cv=5)
```

```
    knn_scores = cross_val_score(self.knn_model, X_scaled, y, cv=5)
    self.add_log(f"RF CV точність: {rf_scores.mean():.3f} ± {rf_scores.std():.3f}")
    self.add_log(f"KNN CV точність: {knn_scores.mean():.3f} ± {knn_scores.std():.3f}")


def classify_point_cloud(self):
    if self.point_cloud is None:
        messagebox.showwarning("Попередження", "Спочатку завантажте хмару точок")
        return

    if not self.is_trained:
        messagebox.showwarning("Попередження", "Спочатку навчіть модель")
        return

    try:
        features = self.extract_features(self.point_cloud)
        features_scaled = self.scaler.transform([features])

        rf_prediction = self.rf_model.predict(features_scaled)[0]
        knn_prediction = self.knn_model.predict(features_scaled)[0]
        rf_proba = self.rf_model.predict_proba(features_scaled)[0]
        knn_proba = self.knn_model.predict_proba(features_scaled)[0]
        rf_confidence = max(rf_proba)
        knn_confidence = max(knn_proba)
        self.rf_result_label.config(
            text=f"Random Forest: {rf_prediction} ({rf_confidence:.2f})"
        )
        self.knn_result_label.config(
            text=f"KNN: {knn_prediction} ({knn_confidence:.2f})"
        )
        self.add_log(f"Класифікація завершена:")
        self.add_log(f"  RF: {rf_prediction} (впевненість: {rf_confidence:.2f})")
        self.add_log(f"  KNN: {knn_prediction} (впевненість: {knn_confidence:.2f})")
    except Exception as e:
        messagebox.showerror("Помилка", str(e))
        self.add_log(f"Помилка класифікації: {str(e)}")

def save_model(self):
```

```
    if not self.is_trained:
        messagebox.showwarning("Попередження", "Немає навченої моделі для збереження")
        return
    filepath = filedialog.asksaveasfilename(
        title="Зберегти модель",
        defaultextension=".pkl",
        filetypes=[("Pickle files", "*.pkl"), ("All files", "*.*")]
    )
    if not filepath:
        return
    try:
        model_data = {
            'rf_model': self.rf_model,
            'knn_model': self.knn_model,
            'scaler': self.scaler,
            'training_data': self.training_data,
            'training_labels': self.training_labels,
            'classes': list(set(self.training_labels))
        }
        with open(filepath, 'wb') as f:
            pickle.dump(model_data, f)
        self.add_log(f"Модель збережена: {os.path.basename(filepath)}")
    except Exception as e:
        messagebox.showerror("Помилка", str(e))
        self.add_log(f"Помилка збереження: {str(e)}")

def load_model(self):
    filepath = filedialog.askopenfilename(
        title="Завантажити модель",
        filetypes=[("Pickle files", "*.pkl"), ("All files", "*.*")]
    )

    if not filepath:
        return

    try:
        with open(filepath, 'rb') as f:
            model_data = pickle.load(f)
```

```
            self.rf_model = model_data['rf_model']
            self.knn_model = model_data['knn_model']
            self.scaler = model_data['scaler']
            self.training_data = model_data['training_data']
            self.training_labels = model_data['training_labels']

            self.is_trained = True
            self.model_label.config(text="Модель навчена: Так")
            self.training_label.config(text=f"Навчальних зразків: {len(self.training_data)}")

            self.add_log(f"Модель завантажена: {os.path.basename(filepath)}")
            self.add_log(f"Класи: {model_data.get('classes', [])}")

        except Exception as e:
            messagebox.showerror("Помилка", str(e))
            self.add_log(f"Помилка завантаження: {str(e)}")

    def visualize_point_cloud(self):
        """Візуалізує хмару точок в інтегрованому віджеті"""
        if self.point_cloud is None:
            return

        try:
            # Очищуємо попередній графік
            self.ax.clear()

            # Налаштовуємо темну тему знову
            self.ax.xaxis.label.set_color('white')
            self.ax.yaxis.label.set_color('white')
            self.ax.zaxis.label.set_color('white')
            self.ax.tick_params(axis='x', colors='white')
            self.ax.tick_params(axis='y', colors='white')
            self.ax.tick_params(axis='z', colors='white')

            # Продовження методу visualize_point_cloud
            self.ax.xaxis.pane.fill = False
            self.ax.yaxis.pane.fill = False
```

```
self.ax.zaxis.pane.fill = False
self.ax.xaxis.pane.set_edgecolor('gray')
self.ax.yaxis.pane.set_edgecolor('gray')
self.ax.zaxis.pane.set_edgecolor('gray')
self.ax.grid(True, alpha=0.3)

# Візуалізуємо хмару точок
x, y, z = self.point_cloud[:, 0], self.point_cloud[:, 1], self.point_cloud[:, 2]

# Кольорова карта для красивого відображення
colors = np.sqrt(x**2 + y**2 + z**2)  # Відстань від початку координат

scatter = self.ax.scatter(x, y, z, c=colors, cmap='viridis', s=20, alpha=0.6)

# Налаштовуємо підписи осей
self.ax.set_xlabel('X', color='white')
self.ax.set_ylabel('Y', color='white')
self.ax.set_zlabel('Z', color='white')

# Встановлюємо заголовок
self.ax.set_title(f'Хмара точок ({len(self.point_cloud)} точок)',
                  color='white', pad=20)

# Автоматично підбираємо межі осей
self.ax.auto_scale_xyz(x, y, z)

# Встановлюємо початковий кут огляду
self.ax.view_init(elev=20, azim=45)
try:
    if hasattr(self, '_colorbar') and self._colorbar is not None:
        self._colorbar.remove()
except:
    pass

self._colorbar = self.fig.colorbar(scatter, ax=self.ax, shrink=0.5, aspect=20)
self._colorbar.ax.tick_params(axis='y', colors='white')

# Оновлюємо canvas
```

```
            self.canvas.draw()
        except Exception as e:
            self.add_log(f"Помилка візуалізації: {str(e)}")
            messagebox.showerror("Помилка", f"Помилка візуалізації: {str(e)}")

    def show_info(self):
        """Показує інформацію про застосунок"""
        info_text = """
        3D Point Cloud Classifier v1.0

        Застосунок для класифікації множини 3D-точок

        Функціональність:
        • Завантаження файлів хмар точок (OFF, PLY, XYZ, PCD, OBJ)
        • Навчання моделей ML (Random Forest, KNN)
        • Класифікація нових хмар точок
        • 3D-візуалізація
        • Аналіз навченої моделі
        • Збереження моделі
        • Завантаження моделі

        Інструкції:
        1. Завантажте файл для візуалізації
        2. Завантажте папку з навчальними даними
          (підпапки = класи, файли = зразки)
        3. Навчіть модель
        4. Класифікуйте нові хмари точок

        Розробник: Гуменюк О. В. 4ПІ-21Б
        """
        messagebox.showinfo("Про застосунок", info_text)
    def run(self):
        """Запускає головний цикл застосунку"""
        self.root.mainloop()
# Точка входу
if __name__ == "__main__":
    app = PointCloudClassifier()
    app.run()
```

Додаток Г – Презентація

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

“Розробка програмного застосунку для класифікації множини 3D точок”

СТУДЕНТ: ст. гр. 4ПІ-21б

_____

КЕРІВНИК: к.т.н., доцент
Рейда О.М.

Вінниця 2025

Рисунок Г.1 – Слайд 1

Актуальність

Розвиток методів машинного навчання, зокрема керованих і некерованих алгоритмів класифікації, відкриває нові можливості для ефективної обробки множин 3D-точок. Такі методи дозволяють автоматизувати аналіз даних, підвищувати точність класифікації та оптимізувати обчислювальні ресурси. Проте більшість сучасних рішень для класифікації 3D-точок мають обмеження, пов’язані з недостатньою інтерактивністю, складністю інтеграції в прикладні системи або відсутністю зручних засобів візуалізації результатів. Це зумовлює потребу в розробці ефективних, автоматизованих і масштабованих програмних рішень для класифікації множин 3D-точок, які відповідають вимогам швидкості, точності та зручності використання.

Мета і завданнядослідження

Метою роботи є підвищення точності і якості класифікації множини 3D-точок у процесі обробки хмари тривимірних даних з використанням методів машинного навчання.

Предмет дослідження – методи та засоби для класифікації множин 3D-точок з використанням алгоритмів машинного навчання.

Об’єктомдослідження – процес розробки програмного застосунку для класифікації множини 3D-точок.

Рисунок Г.2 – Слайд 2

Задачі дослідження

- провести аналіз існуючих методів і засобів класифікації множин 3D-точок для визначення їх переваг та недоліків;
- модифікувати метод класифікації множин 3D-точок для підвищення точності класифікації;
- розробити алгоритми класифікації множин 3D-точок на основі методів машинного навчання;
- розробити програмний застосунок з інтуїтивно зрозумілим інтерфейсом для обробки та класифікації 3D-точок;
- провести експериментальне тестування розробленого програмного застосунку для оцінки його ефективності та точності.

Рисунок Г.3 – Слайд 3

Наукова новизна

Модифіковано метод класифікації множин 3D-точок, який на відміну від існуючого, поєднує метод статистичної фільтрації для усунення «викидів» точок та машинне навчання за допомогою моделі «K-NN» та «RandomForest» для підвищення точності класифікації.

Рисунок Г.4 – Слайд 4

Практичне значення отриманих результатів

Розроблений програмний застосунок може бути використаний у різних галузях, таких як робототехніка, автономні системи, медична діагностика та геоінформаційні системи, для автоматизації обробки та класифікації множин 3D-точок. Застосунок забезпечує зручний інтерфейс, високу точність та швидкість обробки даних, що сприяє оптимізації робочих процесів та підвищенню ефективності аналізу тривимірних даних.

Рисунок Г.5 – Слайд 5

Аналіз аналогів

	CloudCompare	Open3D	PDAL	Potree	Власна розробка
Попередня обробка точок	1	1	1	1	1
Навчання моделей ML	0	1	0	0	1
Підтримка форматів off, ply, xyz, pcd, obj	1	1	1	1	1
3D-візуалізація	1	1	0	1	1
Аналіз навченої моделі	0	0	0	0	1
Збереження ML-моделі	0	1	0	0	1
Завантаження ML-моделі	0	1	0	0	1
Сумарний коефіцієнт	42%	85%	28%	42%	100%

Рисунок Г.6 – Слайд 6

Методи та засоби розробки

У процесі виконання роботи застосовувалися методи машинного навчання, зокрема алгоритми некерованої класифікації; методи обробки даних, такі як фільтрація та нормалізація; програмування для реалізації програмного застосунку; а також методи експериментального тестування для оцінки ефективності розроблених алгоритмів.

7

Рисунок Г.7 – Слайд 7

Метод K-NN та алгоритм Random Forest

У рамках бакалаврської кваліфікаційної роботи для розв'язання задачі класифікації множини 3D-точок були обрані два алгоритми машинного навчання — k-NN та Random Forest. Обидва методи реалізовані з використанням бібліотеки scikit-learn у середовищі Python. Кожна 3D-точка вхідної множини описується за допомогою координат (x, y, z) та, за потреби, додаткових ознак, таких як нормалі, інтенсивність або інші геометричні характеристики.

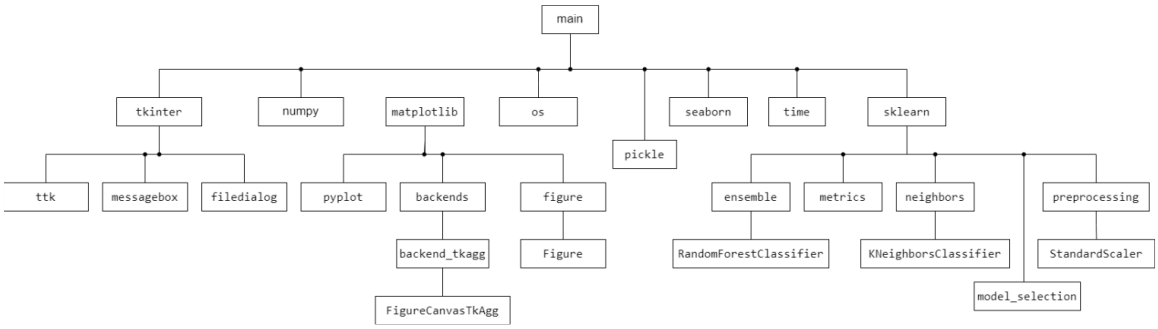
Метод K-NN ґрунтується на принципі пошуку найближчих точок у навчальній множині. Класифікація здійснюється шляхом голосування серед k сусідів, найближчих до точки, яку потрібно класифікувати. Цей метод простий у реалізації, не потребує попереднього навчання моделі, проте його продуктивність залежить від обсягу даних та вибору параметра k.

Алгоритм Random Forest є ансамблевим методом, який формує велику кількість дерев рішень, побудованих на випадкових підмножинах ознак і об'єктів. Кінцева класифікація визначається голосуванням дерев. Цей метод є стійким до шумів у даних, забезпечує високу точність та добре масштабується на складні вибірки.

8

Рисунок Г.8 – Слайд 8

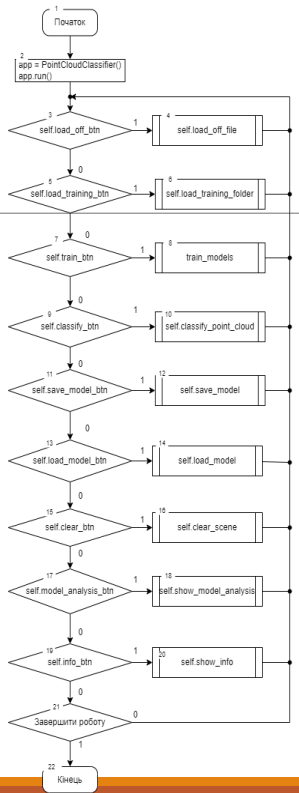
Структурна схема застосунку



9

Рисунок Г.9 – Слайд 9

Блок-схема застосунку



10

Рисунок Г.10 – Слайд 10

Алгоритм отримання ознак множини 3D-точок

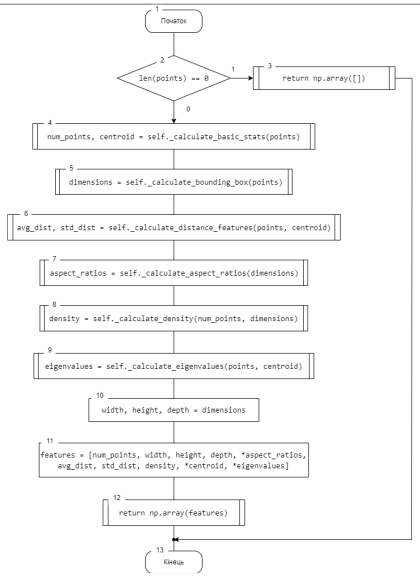


Рисунок Г.11 – Слайд 11

Алгоритм навчання моделей

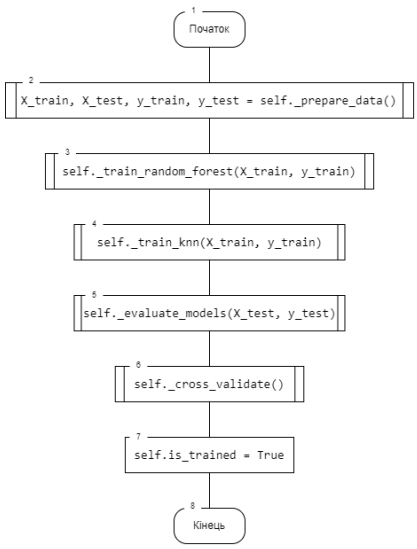
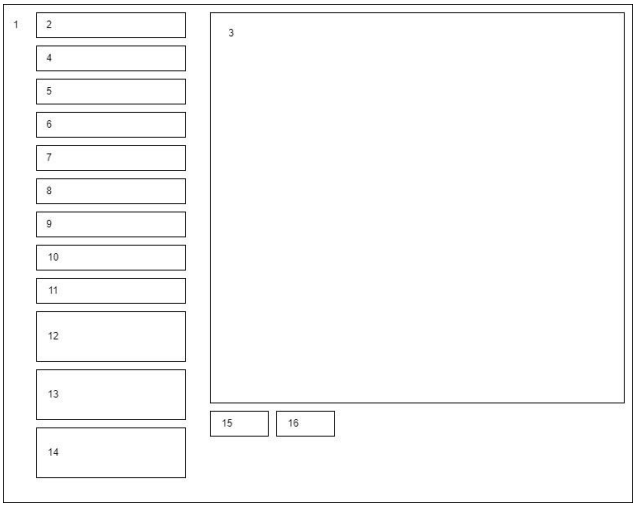


Рисунок Г.12 – Слайд 12

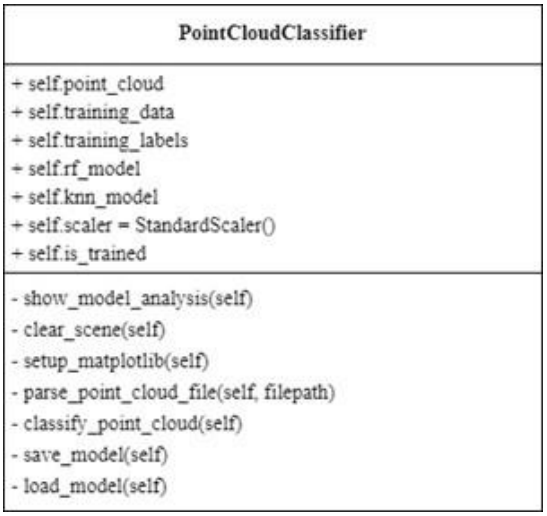
Графічна схема інтерфейсу



13

Рисунок Г.13 – Слайд 13

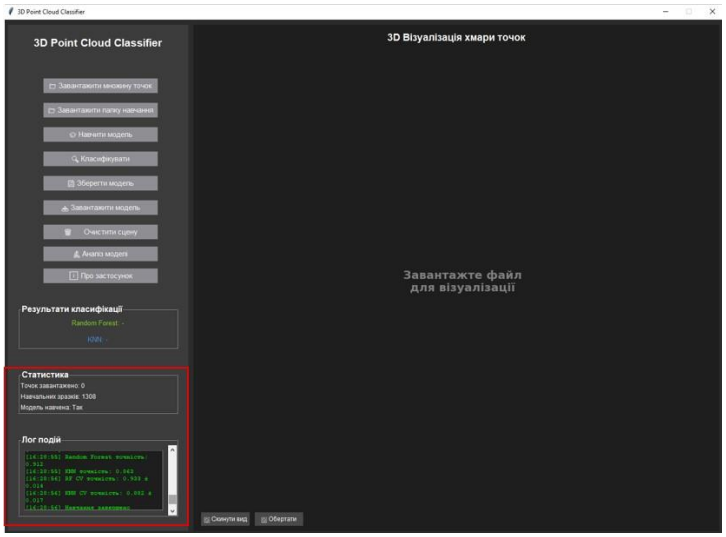
Діаграму класу «PointCloudClassifier»



14

Рисунок Г.14 – Слайд 14

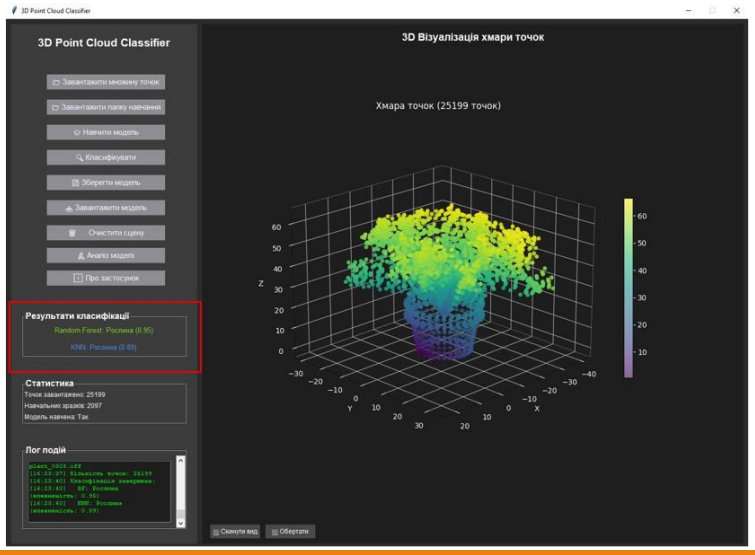
Тестування навчання



15

Рисунок Г.15 – Слайд 15

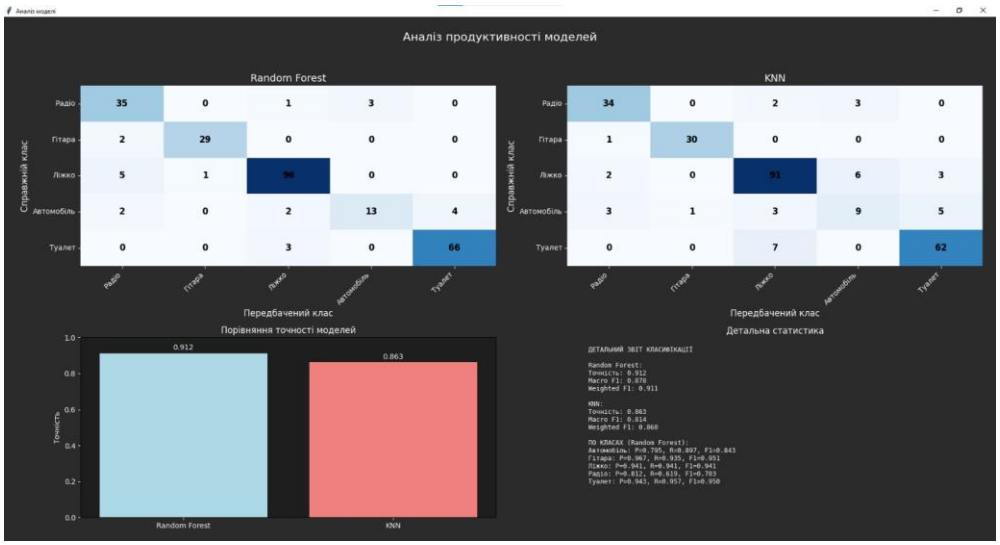
Тестування класифікації та візуалізації



16

Рисунок Г.16 – Слайд 16

Тестування аналізу моделі



17

Рисунок Г.17 – Слайд 17

Висновки

У результаті бакалаврської кваліфікаційної роботи було розроблено застосунок для класифікації множини 3D-точок. Також було розроблено структурну схему застосунку, блок-схему роботи застосунку, графічну схему інтерфейсу, алгоритми для якісного навчання моделей, графічну схему інтерфейсу, діаграму основного класу та проведено порівняльний аналіз аналогів, що підтвердив актуальність розробленого застосунку.

Рисунок Г.18 – Слайд 18

Публікації й апробації

Апробація матеріалів бакалаврської кваліфікаційної роботи. Описані у дослідженні положення доповідались на конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)» та опубліковані в тезах доповіді.

Публікації. За тематикою дослідження опубліковано 1 наукову працю, що доповідалась на конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)» та опубліковано в тезах доповіді.

Рисунок Г.19 – Слайд 19

ДЯКУЮ ЗА УВАГУ

20

Рисунок Г.20 – Слайд 20