МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ХАРЧОВИХ ТЕХНОЛОГІЙ

Інститут (факультет) автоматизації і комп'ютерних систем імені проф. І.В. Ельперіна
Кафедра інформаційних технологій, штучного інтелекту і кібербезпеки

«До захисту в ЕК»	**«До захисту допущено»**
Директор інституту(декан факультету)	Завідувач кафедри
_____ Андрій ФОРСЮК	_____ Сергій ГРИБКОВ
(підпис) (ім'я та прізвище)	(підпис) (ім'я та прізвище)
«08» грудня 2025р.	«08» грудня 2025р.

КВАЛІФІКАЦІЙНА РОБОТА
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

зі спеціальності 122 Комп'ютерні науки
(код та назва спеціальності)
освітньо-професійної програми Управління інформацією та аналітика даних
на тему: Експертна система підбору графічних процесорів для ТОВ "ТІСЕР"

Виконав: здобувач 2 курсу, групи КН-2-2М

Булій Даніл Юрійович _____
(прізвище, ім'я, по батькові повністю) (підпис)

Керівник Харкянен Олена Валеріївна _____
(прізвище , ім'я та по батькові повністю) (підпис)

Консультанти _____ _____
(ім'я та прізвище) (підпис)

_____ _____
(ім'я та прізвище) (підпис)

_____ _____
(ім'я та прізвище) (підпис)

Рецензент _____ _____
(ім'я та прізвище) (підпис)

Я як здобувач(ка) Національного університету харчових технологій розумію і підтримую політику університету з академічної доброчесності. Я не надавав(-ла) і не одержував(-ла) недозволеної допомоги під час підготовки цієї роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Здобувач _____
(підпис)

Київ – 2025р

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ХАРЧОВИХ ТЕХНОЛОГІЙ

Інститут (факультет) автоматизації і комп'ютерних систем імені проф. І.В. Ельперіна
Кафедра інформаційних технологій, штучного інтелекту і кібербезпеки
Освітній ступінь магістр
Спеціальність 122 Комп'ютерні науки
(код і назва)
Освітньо-професійна програма Управління інформацією та аналітика даних
(назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри інформаційних технологій, штучного інтелекту і кібербезпеки

_____ **Сергій ГРИБКОВ**
«05» листопада 2025 року

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА

Булія Даніла Юрійовича
(прізвище, ім'я, по батькові)

1. Тема роботи Експертна система підбору графічних процесорів для ТОВ«ТІСЕР»
керівник роботи Харкянен Олена Валеріївна,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом закладу вищої освіти від «05» листопада 2025 року №906-кс
2. Строк подання здобувачем роботи: 01 грудня 2025 р.
3. Вихідні дані до роботи: аналітичні та статистичні матеріали щодо характеристик GPU, технічна література з проєктування інформаційних систем і баз даних, методичні рекомендації з реалізації веб-додатків, а також джерела з програмування на Python та фреймворку Django.
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):
Розділ 1. Дослідження предметної області та постановка задачі
Розділ 2. Дослідження та обґрунтування технологій, методів, алгоритмів розробки експертної системи
Розділ 3. Дослідження та розробка експертної системи підбору графічних процесорів
5. Перелік графічного матеріалу:
Організаційна структура підприємства, порівняння існуючих технологій підбору обладнання для ПК, порівняння вимог до відеокарт за сферами їх застосування, опис таблиць бази даних експертної системи, схема бази даних експертної системи, схема сховища даних експертної системи, інтерфейс експертної системи

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Харкянен О. В., доцент НУХТ	01. 10. 2025	08. 10. 2025
2	Харкянен О. В., доцент НУХТ	09. 10. 2025	14. 10. 2025
3	Харкянен О. В., доцент НУХТ	15. 10. 2025	28. 11. 2025
4	Харкянен О. В., доцент НУХТ	28. 11. 2025	01. 12. 2025

7. Дата видачі завдання: 01.10.2025

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів виконання кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної області, збір вихідних даних, формування мети та завдань роботи	01.10.2025-10.10.2025	Виконав
2	Дослідження літературних джерел, існуючих рішень та методів аналізу GPU	10.10.2025-13.10.2025	Виконав
3	Формування вимог до системи та опис постановки задачі	14.10.2025-16.10.2025	Виконав
4	Вибір та обґрунтування технологій для розробки експертної системи	16.10.2025-20.10.2025	Виконав
5	Розробка структури бази даних, проектування схеми сховища та моделей	20.10.2025-25.10.2025	Виконав
6	Розробка експертної системи	25.10.2025-15.11.2025	Виконав
7	Тестування системи, оптимізація продуктивності та коригування модулів	15.11.2025-19.11.2025	Виконав
8	Оформлення кваліфікаційної роботи	20.11.2025-01.12.2025	Виконав

Здобувач _____ ___Даніл БУЛІЙ _____
(підпис) (ім'я та прізвище)

Керівник роботи _____ ___Олена ХАРКЯНЕН _____
(підпис) (ім'я та прізвище)

АНОТАЦІЯ

Кваліфікаційна робота присвячена створенню експертної системи підбору графічних процесорів для ТОВ "ТІСЕР", що забезпечує автоматизований підбір відеокарт залежно від потреб користувача — для ігор, графіки, машинного навчання, рендерингу чи майнінгу.

Розроблена експертна система автоматичного підбору графічних процесорів базується на впровадженні комбінованого підходу з використанням алгоритмів кластеризації K-Means та DBSCAN. Це дозволяє одночасно виділяти основні групи відеокарт та коректно обробляти моделі з нетиповими характеристиками.

У системі реалізовано аналітичний модуль, який виконує статистичний аналіз, порівняння продуктивності та енергоефективності процесорів GPU на основі даних з відкритих джерел.

Запропоноване рішення поєднує веб-інтерфейс та аналітичний механізм, що дозволяє користувачеві отримати рекомендації у зручній формі.

Впровадження розробленої системи сприятиме оптимізації процесу вибору відеокарти, зменшенню витрат часу на пошук технічної інформації та підвищенню ефективності прийняття рішень при закупівлі або апгрейді комп’ютерного обладнання.

Робота включає аналіз сучасних підходів до оцінювання продуктивності GPU, порівняння API (CUDA, Metal, OpenCL, Vulkan), опис архітектури системи, реалізацію веб-інтерфейсу та модулів аналітики, а також тестування результатів підбору.

Ключові слова: ВІДЕОКАРТА, ГРАФІЧНИЙ ПРОЦЕСОР, МАШИННЕ НАВЧАННЯ, АНАЛІЗ ДАНИХ, ІНФОРМАЦІЙНО-АНАЛІТИЧНА СИСТЕМА.

SUMMARY

The qualification work is devoted to the creation of an expert system for selecting graphics processors for LLC "TISER", which provides automated selection of video cards depending on the user's needs - for games, graphics, machine learning, rendering or mining.

The developed expert system for automatic selection of graphics processors is based on the implementation of a combined approach using the K-Means and DBSCAN clustering algorithms. This allows you to simultaneously identify the main groups of video cards and correctly process models with atypical characteristics.

The system implements an analytical module that performs statistical analysis, comparison of performance and energy efficiency of GPU processors based on data from open sources.

The proposed solution combines a web interface and an analytical mechanism, which allows the user to receive recommendations in a convenient form.

The implementation of the developed system will help optimize the process of choosing a video card, reduce time spent searching for technical information, and increase the efficiency of decision-making when purchasing or upgrading computer equipment.

The work includes an analysis of modern approaches to assessing GPU performance, comparing APIs (CUDA, Metal, OpenCL, Vulkan), describing the system architecture, implementing a web interface and analytics modules, and testing the selection results.

Keywords: VIDEO CARD, GRAPHICS PROCESSOR, MACHINE LEARNING, DATA ANALYSIS, INFORMATION AND ANALYTICAL SYSTEM.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

AI – штучний інтелект

ML – машинне навчання

GPU – Graphics Processing Unit, графічний процесор, спеціалізований чип для паралельної обробки графічних та обчислювальних задач

API — Application Programming Interface, інтерфейс, що дозволяє програмам взаємодіяти з апаратним або програмним забезпеченням

БД – база даних

СД – сховище даних

ПЗ – програмне забезпечення

ВСТУП

Сучасний розвиток комп’ютерних технологій супроводжується стрімким зростанням обчислювальної потужності графічних процесорів GPU та розширенням сфер їх застосування. Відеокарти сьогодні використовуються не лише для відтворення графіки в іграх, а й у наукових обчисленнях, штучному інтелекті, 3D-рендерингу, роботі з великими даними, машинному навчанні та блокчейн-технологіях. Таке широке застосування GPU визначає потребу у створенні інтелектуальних систем, здатних аналізувати характеристики відеокарт та надавати рекомендації щодо оптимального вибору GPU для конкретних завдань користувача. У сучасних умовах ринок графічних процесорів постійно поповнюється новими моделями, що ускладнює процес вибору для кінцевого користувача, оскільки традиційні методи порівняння, що базуються на ручному аналізі характеристик, стають недостатньо ефективними. Таким чином, виникає необхідність створення систем, які не лише агрегують технічні дані, а й здійснюють їхній комплексний аналіз, враховуючи продуктивність, енергоефективність, ціну та підтримку різних графічних API, таких як CUDA, Metal, OpenCL та Vulkan.

Обґрунтування вибору теми дослідження полягає у тому, що на сучасному ринку користувачі стикаються з надлишком інформації та відсутністю інтегрованих систем, які дозволяють оцінювати відеокарти за комплексними критеріями. Користувачам важливо мати доступ до аналітичних інструментів, які допомагають визначити оптимальну модель GPU не тільки з погляду продуктивності, а й з урахуванням співвідношення ціни та ефективності, енергоспоживання та підтримки різних графічних API. Попередні дослідження у галузі аналізу GPU зосереджувалися переважно на рейтингуванні та порівнянні окремих характеристик, таких як тактова частота, об’єм пам’яті або кількість ядер. Проте такі підходи не дозволяють повністю врахувати взаємозв’язок багатьох параметрів та їхній вплив на реальну продуктивність у різних сценаріях використання. Тому актуальність теми обумовлена потребою у створенні

комплексної системи підбору GPU, що поєднує методи аналітики, статистики та машинного навчання для формування об’єктивних рекомендацій.

Актуальність та практична значущість теми полягає у необхідності оптимізації процесу вибору графічних процесорів, особливо у контексті зростаючого попиту на високопродуктивні обчислювальні системи у сфері штучного інтелекту, наукових розрахунків та 3D-моделювання. Традиційні підходи, засновані на ручному порівнянні характеристик або на суб’єктивних відгуках користувачів, значно уповільнюють процес вибору та не гарантують об’єктивної оцінки. Розробка автоматизованої системи підбору GPU дозволяє усунути дані недоліки, забезпечуючи швидку обробку великих обсягів даних, виділення закономірностей, групування відеокарт за рівнем продуктивності та ефективності, а також надання рекомендацій з урахуванням специфічних завдань користувача. Така система є не лише інструментом практичного використання, але й науково обґрунтованим підходом до аналізу сучасного ринку GPU та методів класифікації технічних характеристик.

Мета дослідження полягає у розробці експертної системи для автоматизованого підбору відеокарт із використанням аналітичних методів обробки та аналізу даних.

Завдання дослідження включають:

- проведення аналізу сучасних підходів до оцінювання та порівняння GPU;
- виявлення переваг та недоліків існуючих методів;
- дослідження ключових показників продуктивності відеокарт, їхніх взаємозв’язків та впливу на ефективність роботи системи;
- розробку архітектури системи підбору GPU з модульною структурою та можливістю інтеграції нових даних;
- реалізацію аналітичного модуля для обробки та візуалізації даних у складі експертної системи;
- тестування експертної системи та оцінку точності наданих рекомендацій.

Об’єкт дослідження – це процес вибору графічних процесорів відповідно до технічних та експлуатаційних характеристик, який включає оцінку

продуктивності, енергоефективності та відповідності конкретним сценаріям використання. Процес вибору є багатофакторним і потребує інтеграції різних джерел даних, включаючи технічні специфікації виробників, тести продуктивності (PassMark, Geekbench), відгуки користувачів та аналітичні показники GPU.

Предметом дослідження є моделі, методи та алгоритми аналітичної обробки даних, що використовуються для автоматизованого підбору графічних процесорів. Зокрема, увагу зосереджено на алгоритмах кластеризації, методах визначення схожості та групування технічних характеристик GPU, статистичному аналізі їх продуктивності, а також підходах до візуалізації результатів, які забезпечують інформативне та зручне представлення даних кінцевому користувачу.

Методи дослідження включають аналітичний метод, що застосований для вивчення сучасних тенденцій розвитку GPU та підходів до їх класифікації; методи статистичного аналізу, що дозволили дослідити кореляції між технічними характеристиками відеокарт, виявити сильні та слабкі взаємозв’язки, що впливають на ефективність рекомендацій; методи машинного навчання та кластеризації, що застосовано для групування відеокарт за рівнем продуктивності та ефективності, забезпечуючи точність і гнучкість системи; методи моделювання, розробки та тестування експертних систем на основі використання реальних даних.

Наукова новизна розроблено метод підбору графічних процесорів (GPU), який на відміну від існуючих підходів поєднує кореляційний аналіз технічних характеристик, ранжування та кластеризацію відеокарт за показниками продуктивності, енергоефективності, співвідношенням «ціна-якість» і підтримкою графічних API, що дозволяє виділити закономірності у взаємодії технічних параметрів та визначити оптимальні моделі графічних процесорів для потреб користувача.

Практичне значення одержаних результатів: полягає у можливості впровадження розробленої експертної системи для автоматизованого підбору GPU у виробничі процеси компаній, що займаються підбором, продажем або

обслуговуванням комп’ютерної техніки, а також у науково-дослідних центрах, де необхідна оптимізація обчислювальних ресурсів для виконання специфічних задач.

Рекомендації, сформовані на основі дослідження, дозволяють зменшити час на вибір оптимальної моделі GPU, підвищити ефективність використання ресурсів та забезпечити об’єктивну оцінку продуктивності у комплексі технічних показників.

Особистий внесок здобувача. Усі основні положення і результати дисертаційної роботи, що захищаються, одержані автором самостійно.

Апробація здійснювалась шляхом практичного застосування розробленої експертної системи для автоматизованого підбору графічних процесорів на реальних тестових даних. Для проведення апробації було використано відкритий тренувальний набір даних з платформи Kaggle, який містить інформацію про різні моделі GPU, їхні технічні характеристики, результати продуктивності та підтримку графічних API. Даний набір даних включає числові показники (такі як G3DMark, G2DMark, кількість, енергоспоживання, ціна), а також категорійні ознаки (виробник, модель, тип використання), що дозволяє комплексно оцінювати відеокарти за багатьма параметрами одночасно.

Наукові та практичні результати кваліфікаційної роботи доповідались та обговорювались на XII Міжнародній науково-технічній Internet-конференції «Сучасні методи, інформаційне, програмне та технічне забезпечення систем керування організаційно-технічними та технологічними комплексами» (м. Київ, Україна, 2025 р.) та Другій міжнародній науково-практичній конференції «Штучний інтелект та інформаційні технології» (Київ, Україна, 2025).

Публікації.

1. Булій Д.Ю. Експертна система підбору графічних процесорів/ Булій Д.Ю., Харкянен О.В. // XII Міжнародна науково-технічна Internet-конференції «Сучасні методи, інформаційне, програмне та технічне забезпечення систем керування організаційно-технічними та технологічними комплексами», 27 листопада 2025 р (Київ, Україна). К. : НУХТ, 2025.

2. Булій Д.Ю., Харкянен О.В. Використання бенчмарків для оцінювання продуктивності відеокарт / Булій Д. Ю., Харкянен О. В. // Наукові праці Другої міжнар. наук.-практ. конф. «Штучний інтелект та інформаційні технології» (AIIT-2025), 3–4 червня 2025 р. (Київ, Україна). К. : НУХТ, 2025. 352 с.

Зв'язок роботи з науковими програмами, планами, темами кафедри, університету, іншої наукової установи. Кваліфікаційна робота виконувалась згідно з планом науково-дослідних робіт кафедри інформаційних технологій, штучного інтелекту і кібербезпеки Національного університету харчових технологій: № 0125U003889. Дослідження та впровадження сучасних методів аналізу даних у харчову промисловість.

Кваліфікаційна робота «Експертна система підбору графічних процесорів для ТОВ "ТІСЕР"» складається з 116 сторінок, 16 рисунків, 9 таблиць і 33 джерел.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальна характеристика ТОВ «ТІСЕР»

ТОВ «ТІСЕР» — це товариство з обмеженою відповідальністю, яке займається інжинірингом, розробкою, виробництвом та постачанням високотехнологічної продукції для різних галузей промисловості, зокрема енергетичної, хімічної, цукрової та спиртової. ТОВ «ТІСЕР» було зареєстровано 23 грудня 2005 року.

Основний вид діяльності компанії – інжинірингова діяльність, розробка, виробництво, постачання і монтаж високотехнологічного обладнання, що застосовується в енергетичній, хімічній, цукровій та інших галузях промисловості. Крім того, ТОВ «ТІСЕР» пропонує послуги з автоматизації технологічних процесів, розробки програмного забезпечення, а також металообробки: згинання, газо-плазмове та лазерне різання, вальцювання, рубку листового металу та токарну обробку.

Компанія володіє Білоцерківським машинобудівним заводом «ТІСЕР», який є її виробничою базою. На заводі є парк металообробного обладнання, що дозволяє виконувати різноманітні роботи з виготовлення деталей та виробів машинобудування. Організаційну схему підприємства наведена на рисунку А.1.

На підприємстві працює біля 102-х працівників, які залучені у наступних напрямках:

1. Цукрова промисловість – розробка та виробництво обладнання для цукрового виробництва.
2. Спиртова промисловість – розробка технологічних схем та виготовлення обладнання для виробництва спирту.
3. Системи автоматизації технологічних процесів – розробка, виготовлення та програмування систем автоматизації виробництв (АСУ ТП).
4. Теплосилове господарство – проєктування, розробка та виготовлення обладнання для теплосилових установок.

5. Виробництво конвеєрів – виробництво стрічкових, пластинчастих, грабельних, скребкових і гвинтових (шнекових) транспортерів.
6. Інші вироби та обладнання – проєктування і виготовлення виробів різного призначення для харчової промисловості.

ТОВ «ТІСЕР» завдяки своєму багатогалузевому підходу та високій якості послуг є одним із провідних підприємств у сфері машинобудування та автоматизації в Україні.

Керівництво компанії:

- Директор — відповідає за загальне управління підприємством, стратегічне планування, прийняття ключових рішень.
- Заступник директора з кадрових та соціальних питань — займається розробкою та реалізацією кадрової політики, управлінням персоналом, прогнозуванням потреби у трудових ресурсах, створенням сприятливих умов праці та мотивацією працівників, а також вирішенням соціальних питань у колективі та забезпеченням дотримання трудового законодавства.
- Заступник директора з комерційних питань — відповідає за комерційну діяльність, зокрема маркетинг, продажі та зв’язки з клієнтами.
- Фінансовий директор — контролює фінансові потоки, бюджетування, управління витратами.
- Технічний директор — здійснює контроль над технічними процесами, виробництвом та інноваційними проектами.

Функціональні відділи:

- Відділ постачання — відповідає за закупівлю матеріалів та ресурсів.
- Відділ збуту — займається продажем готової продукції.
- Транспортний відділ — забезпечує логістику та перевезення продукції.
- Відділ реклами — організовує маркетингові кампанії та просування продукції.

Технічна підтримка та інженерія:

- Головний інженер — координує діяльність технічних підрозділів.
- Головний бухгалтер — займається бухгалтерським обліком та звітністю.
- Технологічний відділ — відповідає за розробку та впровадження технологій.

- Тепло-енергетичний відділ — займається забезпеченням енергетичних потреб.
- Відділ програмування — розробляє програмні рішення.
- Конструкторський відділ — створює технічну документацію та креслення.
- АСУ ТП — автоматизує системи управління технологічними процесами.
- Відділ стандартизації та сертифікації — контролює відповідність продукції компанії чинним стандартам.
- Виробничий та монтажний відділи — займаються виробництвом і монтажем готової продукції.

Планування та економіка:

- Планово-економічний відділ — займається плануванням і аналізом економічної діяльності.
- Фінансовий відділ — розробляє фінансові стратегії.
- Відділ заробітної плати — забезпечує своєчасну оплату праці.
- Відділ маркетингу — аналізує ринок і визначає стратегії продажу.
- Відділ логістики — організовує ланцюги постачання.
- Виробничі та операційні підрозділи:
- Майстри дільниць — відповідають за оперативне управління виробничими процесами.
- Провідні інженери та спеціалісти — займаються розробкою, впровадженням та контролем технологій.
- Оператори верстатів з програмним керуванням — забезпечують точність виготовлення деталей.
- Інженери-конструктори та технологи — проектують та впроваджують рішення для покращення продукції.

Служби підтримки:

- Головний енергетик — контролює енергозабезпечення.
- Інженер з охорони праці — забезпечує дотримання норм безпеки.
- Завідувач складу — контролює збереження матеріалів та готової продукції.
- Лабораторія з контролю якості — перевіряє відповідність продукції стандартам.

На сучасному етапі розвитку підприємства ефективне управління виробничими та адміністративними процесами неможливе без належного рівня комп’ютеризації та автоматизації. ТОВ «ТІСЕР» активно впроваджує інформаційні та автоматизовані системи для підтримки виробничих процесів, обліку обладнання та контролю технологічних операцій. Це дозволяє підвищити точність обробки даних, скоротити час на виконання рутинних завдань та забезпечити оперативний доступ до ключової інформації.

У даному розділі здійснюється аналіз наявних автоматизованих систем підприємства, рівня комп’ютеризації різних напрямків діяльності та ефективності їх використання для підтримки управлінських та виробничих рішень. Такий аналіз дає змогу виявити наявні проблеми в обробці даних та обґрунтувати необхідність впровадження сучасних інструментів аналітики для оптимізації процесів.

На підприємстві ТОВ «ТІСЕР» впроваджено широкий спектр систем автоматизації, що охоплюють ключові виробничі процеси. Інтеграція IoT-рішень дозволяє підприємству впроваджувати розумні сенсори, які моніторять стан обладнання в реальному часі, сприяючи ранньому виявленню несправностей.

До основних автоматизованих систем належать:

- система автоматичного управління відділенням кристалізації;
- система управління продуктовим відділенням;
- система управління вакуум-апаратами;
- система управління клеровальним відділенням;
- система управління вакуум-конденсаційною установкою;
- система управління відводом конденсату;
- система управління випарною станцією;
- система управління станціями фільтрації;
- система управління станцією сокоочистки;
- система управління вапняним відділенням;
- система управління віджигальною піччю;
- система управління дифузійною установкою;
- система управління жомосушильним та мийним відділенням;

Крім того, реалізовано комплексну автоматизацію промислового виробництва, використовується як апаратне, так і програмне забезпечення для керування цими процесами.

На підприємстві ТОВ «ТІСЕР» система автоматизації технологічних процесів (АСУ ТП) виконує ключову роль у контролі, моніторингу та оптимізації виробничих операцій. АСУ ТП забезпечує збір даних із сенсорів та обладнання в реальному часі, контроль параметрів технологічних процесів, сигналізацію про відхилення та автоматичне регулювання роботи машин і устаткування.

Документообіг у межах підрозділу АСУ ТП та між взаємодіючими відділами організовано наступним чином: відділ АСУ ТП отримує технічні завдання від технологічного відділу та конструкторського підрозділу, фіксує результати автоматичного моніторингу, передає аналітичні дані в планово-економічний відділ та відділ контролю якості для обробки, а також надає інформацію в виробничий та монтажний підрозділ для коригування технологічних операцій. Крім того, АСУ ТП інтегрована з відділом програмування для налаштування та модернізації програмних алгоритмів управління.

Взаємодія між підрозділами відбувається через електронний документообіг, внутрішні звіти, автоматизовані протоколи контролю та системи моніторингу. Це забезпечує збереження актуальності даних, мінімізує людський фактор та скорочує час на передачу інформації між відділами. Взаємодію підрозділу АСУ ТП із підрозділами підприємства наведено у таблиці 1.1

Таблиця 1.1 - Взаємодія підрозділу АСУ ТП із іншими підрозділами підприємства ТОВ «ТІСЕР»

Відділ / Підрозділ	**Тип інформації для АСУ ТП**	**Вихідна інформація від АСУ ТП**	**Мета взаємодії**
Технологічний відділ	Технічні завдання, параметри процесів	Звіти про контроль параметрів, сигнали відхилень	Коригування технологічних операцій, оптимізація процесів
Конструкторський відділ	Проєктна документація, специфікації обладнання	Звіти про ефективність роботи обладнання	Перевірка відповідності конструкцій фактичним параметрам
Виробничий та монтажний	Робочі плани, графіки виробництва	Інформація про стан обладнання та виробничих процесів	Підтримка безперебійного виробництва, своєчасне обслуговування
Відділ програмування	Налаштування алгоритмів, оновлення ПЗ	Дані про ефективність алгоритмів управління	Модернізація та оптимізація програмного забезпечення
Відділ контролю якості	Стандарти якості, норми	Звіти про відповідність процесів та продукції	Контроль якості та відповідності стандартам
Планово-економічний відділ	Виробничі дані, обсяги робіт	Аналітичні дані про ефективність використання ресурсів	Оптимізація витрат та планування виробництва

1.2. Огляд існуючих на підприємстві інформаційних систем

На підприємстві впроваджено комплекс автоматизованих систем та інформаційних технологій, що зображено в таблиці 1.2.

Таблиця 1.2 - Існуючі інформаційні системи підприємства ТОВ «TICEP»

Система/Технологія	Призначення	Використання
АСУ ТП відділення кристалізації	Автоматичне управління процесами кристалізації	Технологічний контроль, моніторинг
АСУ ТП продуктових відділень	Управління виробничими процесами	Контроль якості, планування
SCADA	Моніторинг та управління виробничим обладнанням	Всі виробничі дільниці
PostgreSQL	Зберігання даних про обладнання та моніторинг	Аналітика, звітність
Excel	Ручна обробка даних	Підготовка звітів та аналіз
AutoCAD, SolidWorks	Проєктування та моделювання	Конструкторський та технологічний відділи
IoT (MQTT, OPC UA, Arduino, Raspberry Pi)	Моніторинг та передача даних	Відділ автоматизації

1.3. Виявлені задачі та проблеми

Аналіз наявних інформаційних систем та процесів збору, зберігання й обробки даних на підприємстві ТОВ «TICEP» дозволив визначити ряд проблем і задач, що впливають на ефективність управління виробничими та адміністративними процесами.

Найсуттєвішою проблемою є відсутність єдиної аналітичної платформи, яка б дозволяла централізовано збирати, зберігати та обробляти дані про технічне обладнання підприємства. Незважаючи на наявність баз даних реалізованих у СУБД PostgreSQL та MySQL, обробка інформації здебільшого здійснюється

вручну за допомогою Excel, що значно уповільнює роботу і підвищує ризик помилок.

Основні виявлені недоліки:

1. Відсутність аналітичного програмного забезпечення, яке могло б автоматично вести звітність щодо продуктивності, надійності та вартості обладнання, а також формувати дані для прийняття управлінських рішень.
2. Відсутність автоматизованих засобів підбору технічних пристроїв (комп’ютери, відеокарти, контролери) для потреб працівників, що ускладнює оперативну адаптацію обладнання під конкретні завдання виробництва.
3. Ручний процес закупівлі технічних засобів, при якому рішення про вибір обладнання приймаються інженерами на власний розсуд. Це займає багато часу і може призводити до помилок, а також до нераціональних витрат бюджету.

Використання сучасних методів аналізу даних (наприклад, інтеграція з аналітичними платформами типу Microsoft Power BI або Python-аналітики) дозволяє виявляти ці проблеми системно: централізувати дані з різних джерел, автоматизувати підготовку звітів, аналіз продуктивності та ефективності обладнання, а також оптимізувати процеси закупівлі.

Таким чином, ключовими завданнями для подальшого дослідження є:

- створення єдиної платформи для збору та аналізу даних;
- автоматизація підбору технічних пристроїв;
- оптимізація процесу закупівлі та зменшення ризику помилок у прийнятті рішень.

1.4. Огляд наукових підходів до аналітики та автоматизації підбору обладнання

Аналітичний огляд літературних джерел спрямований на узагальнення сучасних наукових та практичних підходів до автоматизації процесів підбору технічного обладнання та використання аналітичних систем для підтримки

прийняття управлінських рішень на підприємствах. Основна увага приділяється питанням інтеграції даних, аналітики технічних характеристик обладнання, оптимізації закупівель та вибору комп'ютерних ресурсів, зокрема відеокарт.

Сучасні наукові роботи підкреслюють важливість застосування аналітичних методів і алгоритмів оцінки продуктивності обладнання, наприклад:

- статистичний аналіз продуктивності дозволяє порівнювати різні моделі відеокарт за показниками обчислювальної потужності (G3DMark, API Score, TDP, price/performance).
- машинне навчання та кластеризація використовуються для групування відеокарт за класом продуктивності, виявлення аномалій та оптимального підбору компонентів для конкретних завдань.

Дослідження також показують, що централізація даних з різних джерел (Excel, бази даних PostgreSQL та MySQL, SCADA) є критичною для автоматизації підбору та аналітики. Розрізнені дані ускладнюють їх обробку і підвищують ймовірність помилок при прийнятті рішень.

Попри наявність програмних платформ для підбору комп'ютерних компонентів, існують наступні обмеження:

1. неповна аналітика продуктивності. Хоча деякі сучасні рішення дозволяють оцінювати продуктивність компонентів, вони не завжди забезпечують комплексне порівняння відеокарт за критеріями ціни, ефективності та відповідності конкретним тестовим сценаріям підприємства.
2. ручний відбір обладнання. Обмежена можливість отримання інтегрованого інструменту, який автоматично підбирає оптимальні моделі відеокарт за результатами тестових даних, змушує інженерів виконувати підбір вручну. Це займає додатковий час і створює ризик помилок або нераціонального використання бюджету.

Таким чином, існує потреба у створенні інформаційної, експертної системи підбору відеокарт з елементами аналітики, що дозволить автоматизувати процес вибору та спростити аналіз тестових даних.

1.5. Постановка завдання на дослідження

Метою дослідження є розробка експертної системи для автоматизованого підбору відеокарт на підприємстві ТОВ «ТІСЕР» із використанням аналітики тестових даних продуктивності. Така система покликана забезпечити швидкий та обґрунтований вибір оптимальних моделей відеокарт для різних виробничих сценаріїв, зокрема для застосування в графічних завданнях, обчисленнях штучного інтелекту та 3D-рендерингу.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

1. Аналіз предметної області та існуючих ІС. Оцінити наявні програмні рішення для підбору комп'ютерних компонентів та їх обмеження у контексті виробничих потреб підприємства.
2. Систематизація та обробка даних. Зібрати інформацію про тестові характеристики відеокарт в надійній базі даних та підготувати її для аналітичної обробки.
3. Застосування методів статистичного аналізу та машинного навчання. Використати алгоритми кластеризації (K-Means, DBSCAN) для групування відеокарт за класами продуктивності та виявлення аномальних або некоректних даних.
4. Розробка логіки підбору. Створити алгоритм, який на основі тестових даних обирає оптимальні відеокарти з урахуванням продуктивності, ціни та відповідності виробничим завданням.
5. Проєктування експертної системи. Визначити структуру системи, модулі для введення, обробки та відображення аналітики, інтерфейс користувача та інтеграцію з існуючими джерелами даних.
6. Тестування та валідація. Перевірити коректність роботи системи на прикладі реальних даних та оцінити її ефективність порівняно з ручним підбором відеокарт.

Очікуваним результатом дослідження є створення експертної системи, яка дозволить:

- автоматизувати процес підбору відеокарт для різних виробничих завдань;
- скоротити час прийняття рішень;
- підвищити точність та обґрунтованість вибору обладнання;
- інтегрувати аналітичні дані з різних джерел в єдину платформу для прийняття управлінських рішень.
-

1.6. Висновок до першого розділу

У першому розділі здійснено системний аналіз об’єкта дослідження — графічних процесорів (GPU) та процесу їх оцінювання й підбору для різних сфер застосування. Обґрунтовано актуальність створення експертної системи, оскільки наявні підходи до вибору відеокарт потребують значних витрат часу, а також не завжди забезпечують достатній рівень точності та об’єктивності при оцінюванні продуктивності.

Проаналізовано діяльність ТОВ «ТІСЕР», визначено основні напрями роботи підприємства та роль окремих відділів у процесі обліку й закупівлі технічного обладнання. Розглянуто сучасні інформаційні технології, що використовуються на підприємстві, а також їхні програмні аналоги, які стали підґрунтям для формування концепції та архітектури майбутньої експертної системи. Визначено ключові етапи підвищення рівня автоматизації та узагальнено проблемні аспекти, пов’язані з відсутністю єдиної інтегрованої платформи для аналітики GPU.

Таким чином, перший розділ сформував теоретичну та методологічну основу подальшої розробки експертної системи, визначив практичну значущість її впровадження та окреслив напрямки формалізації, структурування й організації даних, необхідних для побудови аналітичних механізмів у системі.

РОЗДІЛ 2. ДОСЛІДЖЕННЯ ТА ОБҐРУНТУВАННЯ ТЕХНОЛОГІЙ, МЕТОДІВ, АЛГОРИТМІВ РОЗРОБКИ ЕКСПЕРТНОЇ СИСТЕМИ

2.1. Відеокарти та їх функціональне призначення

Графічний процесор або відеокарта — електронний пристрій, частина комп'ютера, призначена для генерації та обробки зображень з подальшим їхнім виведенням на екран периферійного пристрою [9].

Сучасні відеокарти не обмежуються лише звичайним виведенням зображень, вони мають вбудований графічний мікропроцесор, котрий може здійснювати додаткову їх обробку, звільняючи від цих задач центральний процесор. Завдяки високопродуктивним графічним мікропроцесорам (GPU), вони здатні виконувати паралельні обчислення, суттєво розвантажуючи центральний процесор та забезпечуючи виконання задач, що потребують великої кількості паралельних операцій [9].

З розвитком технологій область застосування відеокарт значно розширилася. Сьогодні GPU використовуються не лише у класичних графічних задачах, але й у машинному навчанні, штучному інтелекті, наукових моделях, рендерингу, ігровій індустрії, аналізі даних та інженерних розрахунках. Це зумовлює необхідність правильного і обґрунтованого вибору графічного процесора під конкретні потреби користувача чи підприємства.

Сучасні графічні процесори (GPU) використовуються у широкому спектрі задач, що виходять далеко за межі традиційного рендерингу зображень. Завдяки високій паралельності обчислень та архітектурі, оптимізованій для масивних математичних операцій, GPU стали ключовим апаратним компонентом у багатьох галузях — від офісних систем до високопродуктивних обчислень. Розглянемо основні напрямки застосування відеокарт та їх функціональні особливості.

1) **Відеокарти для офісних і побутових задач.** У повсякденному використанні — для роботи з документами, веббраузингом, відеозв'язком та офісними застосунками — вимоги до GPU мінімальні. Основні операції, які виконуються в таких сценаріях, не потребують значних графічних ресурсів, а тому

більшість офісних систем функціонує на інтегрованій графіці (Intel UHD/ Iris Xe, AMD Radeon Vega).

Ключові характеристики таких GPU:

- невеликий тепловий пакет (TDP);
- висока енергоефективність;
- можливість роботи без додаткового живлення;
- мінімальний рівень шуму та охолодження.

Основні задачі GPU та навантаження в офісних задачах:

- виведення зображення на монітор;
- декодування відео (YouTube, Zoom, онлайн-стріми);
- базові 2D-операції та рендеринг інтерфейсу;
- робота з кількома моніторами;
- апаратне прискорення браузера.

2) Ігрові відеокарти. Ігри — одна з найпоширеніших сфер застосування відеокарт. Сучасні ігрові проєкти використовують складну 3D-графіку, високі текстури, системи світлотіней, трасування променів та фізичні симуляції. Обсяги обчислень у таких процесах надзвичайно великі, тому GPU є критично важливим компонентом.

Ключові характеристики таких GPU:

- висока продуктивність у синтетичних ігрових тестах (G3DMark);
- великий обсяг відеопам’яті (від 6–12 GB і більше);
- підтримка технологій DirectX 12, Vulkan;
- висока пропускна здатність пам’яті;
- підтримка DLSS/FSR/XeSS (апскейлінг та AI-рендеринг).

Основні задачі GPU у геймінгу:

- рендеринг кадрів з високою частотою (FPS);
- обробка шейдерів, текстур, ефектів постобробки;
- виконання складних математичних обчислень для фізичних моделей;
- забезпечення можливості гри бути у високій роздільності (1440p, 4K) та VR.

3) Відеокарти для штучного інтелекту та машинного навчання (AI/ML). Це найбільш технологічно вимоглива категорія. В останні роки GPU стали критично важливими для тренування нейронних мереж і виконання моделей глибинного навчання, оскільки здатні паралельно обробляти мільйони операцій над матрицями.

Ключові характеристики AI-орієнтованих GPU:

- високі показники обчислень у CUDA/Metal/OpenCL;
- велика кількість CUDA-ядер / потокових процесорів;
- підтримка FP16, BF16, Tensor Cores (для Nvidia);
- великий обсяг пам’яті (12–48 GB);
- висока пропускна здатність (GDDR6X, HBM).

Основні задачі GPU в AI/ML:

- тренування нейронних мереж (CNN, RNN, трансформери);
- прискорення інференсу моделей;
- виконання лінійної алгебри (матриці, тензори);
- робота з фреймворками TensorFlow, PyTorch, JAX.

4) Відеокарти для графіки, 3D-моделювання та рендерингу. Рендеринг і графічне виробництво — це професійні сфери, які інтенсивно використовують GPU для обчислень. Це включає 3D-анімацію, роботу у Blender, Maya, 3ds Max, Cinema4D, а також відеомонтаж у Adobe Premiere Pro, DaVinci Resolve.

Ключові характеристики таких GPU:

- архітектура, оптимізована під рендеринг (RT-ядра, Tensor cores);
- велика відеопам’ять (16+ GB бажано);
- стабільність драйверів та сертифікація під Autodesk, Adobe;
- високі значення G3DMark і API-тестів.

Основні задачі GPU у задачах графіки та рендерингу:

- прискорення трасування променів (ray tracing);
- виконання GPU-рендерингу (Cycles, Redshift, Octane);
- обробка відео в реальному часі;
- прискорення експорту відео;

- обробка складних сцен з великою кількістю полігонів.

5) Відеокарти для майнінгу криптовалют. Майнінг — це процес обчислення криптографічних задач для підтвердження блоків у блокчейні. Для деяких алгоритмів (наприклад, Ethash, KawPow) GPU є найефективнішим інструментом.

Ключові характеристики таких GPU:

- велика пропускна здатність пам'яті (важливіше, ніж ядра);
- низьке споживання електроенергії на 1 MH/s;
- ефективне охолодження;
- стабільність при тривалому навантаженні.

Основні задачі GPU в процесі майнінгу:

- виконання великої кількості хеш-функцій;
- обробка потоків даних із максимальною пропускною здатністю;
- безперервна робота 24/7.

Цінові сегменти GPU. Вибір відеокарти для певної задачі часто визначається не лише її технічними характеристиками, а й ціною. GPU можна умовно розділити на бюджетні, середнього рівня та високопродуктивні моделі.

Бюджетні GPU зазвичай призначені для офісних задач або легких ігор. Вони мають невелику відеопам'ять (2–4 GB), низьке енергоспоживання та обмежену кількість обчислювальних ядер. Ці карти є економічно вигідними для користувачів, які не потребують високої продуктивності, і становлять нижній ціновий сегмент на ринку.

Середній сегмент GPU включає карти, які підходять для ігор середньої та високої складності, а також для базових задач рендерингу. Вони мають більшу відеопам'ять (6–12 GB), середнє енергоспоживання і достатню кількість обчислювальних ядер. Такі відеокарти забезпечують оптимальний баланс між продуктивністю та ціною.

Високопродуктивні GPU призначені для задач AI/ML, професійного рендерингу та складних обчислювальних сценаріїв. Вони мають великий обсяг відеопам'яті (16–48 GB), високу енергоспоживаність та максимально можливу кількість обчислювальних ядер. Ці карти належать до преміум-сегменту і мають

високу ціну, проте забезпечують максимальну ефективність та надійність при інтенсивних обчисленнях.

Порівняння вимог до GPU за сферами застосування із ціновим сегментом наведено в таблиці Г.1.

2.2. Математичне моделювання та формалізація характеристик GPU

Моделювання основних задач експертної системи підбору графічних процесорів для ТОВ "ТІСЕР" полягає у побудові математичних залежностей, алгоритмічних процедур і логічних взаємозв'язків, які забезпечують об'єктивну оцінку, класифікацію та підбір графічних процесорів (GPU) за сукупністю їх технічних та експлуатаційних характеристик. У рамках цього етапу здійснюється формалізація вхідних даних, створення моделей інтегральної оцінки продуктивності, визначення коефіцієнта ефективності та підготовка ознак для подальшої кластеризації й аналітичної обробки.

Основна мета моделювання полягає у забезпеченні можливості системи автоматично інтерпретувати кількісні показники, що надходять з відкритих джерел (Kaggle, PassMark, Geekbench тощо), і перетворювати їх у зручну для користувача експертну інформацію.

На початковому етапі дослідження було сформовано множину даних $G = \{g_1, g_2, g_3 \ldots, g_n\}$, де кожен елемент g_i відповідає конкретній відеокарті. Для кожного GPU визначається вектор характеристик g_i *(2.1):*

$$g_i = \left(g3dmark_i, g2dmark_i, price_i, gpu\_value_i, tdp_i\ price_i, performance\_index_i, avg\_api\_score_i, min\_s\right), \quad (2.1)$$

Вектор g_i — це багатовимірна точка у простору ознак, де кожна компонента описує певну властивість графічного процесора. Таке формальне подання дає можливість застосовувати методи математичного аналізу, машинного навчання та алгоритми кластеризації.

Вектор характеристик репрезентує сукупність усіх графічних процесорів, що беруть участь у моделюванні, аналізі та подальшій кластеризації. Кожен елемент цієї множини — це формалізований опис відеокарти, представлений як вектор числових ознак.

Для оцінювання загальної продуктивності використано зважену модель інтегрального показника (2.2). На основі емпіричних досліджень і порівняння результатів тестів визначено оптимальні коефіцієнти ваги для найважливіших характеристик.

$$S_{int}(g_i) = 0.4 \times g3dmark_i + 0.2 \times g2dmark_i + 0.1 \times gpu\_value_i + 0.2 \times avg\_api\_score_i \quad (2.2)$$

$S_{int}(g_i)$ — інтегральний показник продуктивності конкретного GPU.

Отримане значення є базовим індикатором для порівняння відеокарт між собою незалежно від виробника чи архітектури. Ця формула враховує збалансований вплив тривимірної продуктивності (G3DMark), двовимірної графіки (G2DMark), ефективності ціни (GPU Value) та середніх API-результатів, які відображають рівень оптимізації для різних програмних середовищ — CUDA, OpenCL, Vulkan, Metal. Таким чином, модель охоплює як обчислювальний, так і прикладний потенціал GPU.

Другим важливим параметром є ефективність відеокарти (2.3), яка характеризує співвідношення продуктивності до вартості.

$$E(g_i) = \frac{S_{int}(g_i)}{price_i}, \quad (2.3)$$

де $E(g_i)$ — коефіцієнт ефективності графічного процесора. Якщо значення ціни відсутнє або дорівнює нулю, для уникнення помилок здійснюється нормалізація: $E(g_i) = 0$. Цей показник має ключове значення під час вибору GPU у межах обмеженого бюджету, оскільки дозволяє визначити, наскільки раціональним є вкладення коштів у конкретну модель.

Задля отримання достовірних результатів моделювання обов'язковим етапом є попередня підготовка та очищення даних. Оскільки дані збираються з відкритих джерел, вони можуть містити пропущені значення, дублікати, статистичні викиди або шумові спотворення. Такі аномалії здатні значно впливати на результати обчислень, зокрема під час кластеризації та оцінки подібності між моделями. Тому перед початком основного аналізу здійснюється стандартизація даних, заповнення пропусків, видалення некоректних рядків і масштабування ознак до єдиного діапазону.

Особлива увага приділяється виявленню аномалій та шумів, що можуть спотворювати структуру даних. Для цього використовуються алгоритми аналізу щільності, зокрема DBSCAN, який дозволяє виявляти об'єкти, що не належать до жодного кластера, та ізольовані випадки, котрі не відображають загальну тенденцію.

Такі елементи видаляються або обробляються окремо з метою підвищення точності наступних аналітичних процедур.

На основі цього набору параметрів проводиться подальше групування графічних процесорів за допомогою алгоритмів кластеризації, зокрема K-Means і DBSCAN, що дає можливість автоматично розділити GPU на сегменти використання — ігрові, професійні, AI/ML, офісні або енергозберігаючі. Метою цього етапу є не лише математичне групування, а й інтерпретація отриманих кластерів у термінах практичної корисності для кінцевого користувача.

Таким чином, на основі побудованих моделей формується єдина функція оптимізації, що поєднує продуктивність і ефективність (2.4):

$$F(g_i) = \alpha \times S_{int}\left(g_i\right) + \beta \times E\left(g_i\right), \quad (2.4)$$

де α, та β — вагові коефіцієнти, які задаються користувачем або системою за умовчанням. Їх зміна дозволяє адаптувати систему під різні сценарії використання: наприклад, пріоритет продуктивності для ігор або пріоритет ефективності для обмеженого бюджету. Цей підхід відображає гнучкість експертної системи та її здатність до самоадаптації під потреби користувача.

Усі наведені математичні співвідношення й етапи попередньої обробки формують базу для подальшої реалізації аналітичного модуля системи. Саме на цьому етапі відбувається інтеграція математичної моделі з алгоритмічними рішеннями, що дозволяє поєднати статистичну обробку, машинне навчання та експертну оцінку. У результаті формується інтелектуальна система, яка здатна самостійно аналізувати великі обсяги даних, виявляти закономірності та робити висновки щодо доцільності використання певних GPU у різних сферах.

2.3. Огляд та аналіз існуючих рішень для розв'язання виявлених задач

Розроблення експертної системи для аналітики та підбору графічних процесорів (GPU) вимагає детального аналізу вже наявних рішень, що функціонують у суміжних сферах — систем підбору комплектуючих, корпоративних рішень з управління технічними ресурсами, а також наукових платформ для аналізу продуктивності обладнання. Метою даного підрозділу є визначення наявних технологій, оцінка їхніх переваг і недоліків, а також виявлення функціональних обмежень, які виправдовують необхідність створення нового аналітичного інструменту з глибокою інтеграцією методів кластеризації та машинного навчання.

На сучасному ринку інформаційних технологій існує значна кількість інструментів, що дозволяють здійснювати підбір апаратного забезпечення. Найбільш відомими серед них є PCPartPicker, SpecOps Hardware Planner, HP Device

as a Service (DaaS), а також спеціалізовані рішення від виробників, такі як Nvidia System Advisor та AMD Radeon Configurator. Однак жоден із цих інструментів не забезпечує комплексного аналітичного підходу до оцінки саме графічних процесорів, а тим більше не використовує методи кластеризації для групування моделей за продуктивністю, енергоефективністю чи економічною доцільністю.

PCPartPicker. Платформа PCPartPicker є одним із найвідоміших рішень для підбору комплектуючих до персональних комп'ютерів. Вона надає користувачам зручний інтерфейс для вибору сумісних процесорів, відеокарт, материнських плат, блоків живлення тощо. Система автоматично перевіряє сумісність компонентів і виводить приблизну загальну вартість збірки. Основна перевага цього сервісу полягає в його масштабності та інтеграції з інтернет-магазинами — користувач може одразу порівняти ціни, переглянути наявність товарів і сформувати кошик. Водночас аналітична складова в PCPartPicker є мінімальною: система не аналізує продуктивність GPU на основі статистичних показників, не здійснює прогнозування ефективності чи довговічності, не визначає оптимальну модель під конкретне завдання (ігрове, графічне, обчислювальне).

Таким чином, PCPartPicker є типовим прикладом системи для сумісності компонентів, але не аналітики їхньої ефективності. Вона вирішує задачу логістики та організації вибору, проте не містить інтелектуальних модулів для прийняття рішень на основі багатовимірних даних. Це суттєве обмеження для користувачів, які хочуть підібрати GPU не за формальними параметрами, а за співвідношенням ціни, продуктивності та цільового використання.

SpecOps Hardware Planner. Інше рішення, SpecOps Hardware Planner, орієнтоване на корпоративний сегмент. Його головна мета — планування апгрейдів, ведення обліку робочих станцій, моніторинг стану комп'ютерної інфраструктури. Система дозволяє прогнозувати, коли пристрої потрібно замінити або модернізувати, а також оптимізує бюджет на закупівлі. Цей інструмент можна вважати кроком до аналітичних систем, оскільки він використовує елементи прогнозування. Проте SpecOps Hardware Planner не враховує специфіку GPU як окремого об'єкта дослідження. Всі показники

узагальнені, без урахування таких критичних метрик, як API-продуктивність, споживання енергії, співвідношення G3Dmark/G2Dmark або ефективність при різних типах навантаження.

Таким чином, SpecOps вирішує задачу управління технічними ресурсами підприємства, але не забезпечує глибокої експертної оцінки продуктивності або кластеризації моделей. Це обмежує можливість застосування системи для вибору GPU під специфічні цілі, наприклад, обчислення у сфері штучного інтелекту (AI/ML) або рендеринг відео.

HP Device as a Service (DaaS). Сервіс HP DaaS поєднує апаратну та аналітичну складові. Його особливість полягає в тому, що система збирає телеметричні дані з пристроїв користувачів (навантаження, продуктивність, частоту звернень до компонентів) і на основі цього формує рекомендації щодо модернізації обладнання.
HP DaaS є прикладом експертної системи корпоративного типу, проте вона не призначена для аналізу GPU у відкритому контексті. Її моделі замкнені в екосистемі HP, що робить її непридатною для універсального використання. Крім того, користувач не має доступу до алгоритмів аналітики, а система не виконує кластеризацію або багатовимірне групування даних — вона працює з попередньо визначеними правилами.

Таким чином, HP DaaS можна розглядати як ілюстрацію інтегрованого підходу до керування обладнанням, однак її функціональність є обмеженою і закритою, що суперечить концепції відкритої аналітичної системи.

Проведений аналіз показує, що наявні системи вирішують окремі задачі, але не забезпечують комплексного підходу до аналітики графічних процесорів. Вони не враховують багатофакторність проблеми вибору GPU, де одночасно впливають технічні параметри, енергоспоживання, ціна, API-підтримка та сфера застосування. Таблиця порівняння даних інструментів наведена в таблиці Б.1.

Крім того, більшість платформ використовує статичні методи — просте порівняння числових показників без елементів машинного навчання, адаптації чи прогнозування.

У той час, як методи кластеризації (зокрема K-Means, DBSCAN, Agglomerative Clustering і т. д.) дають можливість виявляти приховані закономірності у даних і формувати сегменти GPU за рівнем продуктивності, жодна з розглянутих систем не реалізує цього підходу. Це відкриває перспективу для створення нової системи, яка не лише аналізує, але й інтерпретує результати, надаючи користувачеві конкретні рекомендації.

2.4. Огляд алгоритмів кластеризації як методів прийняття рішень

Розглянемо алгоритми кластеризації для підбору графічних процесорів експертної системи підприємства.

«Кластеризація — це автоматичне розбиття елементів деякої множини на групи залежно від їх схожості. Елементами множини можуть бути будь-які об’єкти, зокрема дані або вектори характеристик. Самі групи прийнято називати кластерами» [4].

Кластеризація є методом, який часто використовується для рекомендаційних систем, оскільки вона дозволяє ідентифікувати групи користувачів зі схожими смаками. Це сприяє більш цілеспрямованим рекомендаціям, використовуючи переваги отримання інформації вже визначених вподобань клієнтів. В роботі було показано, що кластеризація може допомогти подолати такі проблеми, як розрідженість даних і масштабованість, які часто виникають при формуванні рекомендацій, і, таким чином, сприяти побудові більш точних і різноманітних рекомендацій.

У межах кластеризації виділяють два фундаментальні підходи — чітку (жорстку) кластеризацію та нечітку (fuzzy) кластеризацію, які визначають спосіб належності об’єктів до кластерів.

Чітка кластеризація — це підхід до групування об’єктів, за яким кожен елемент набору даних належить тільки до одного кластеру. У цьому методі для кожного об’єкта визначається лише один кластер, і немає перекриття між групами.

Основна ідея чіткої кластеризації полягає у поділі даних на взаємовиключні підмножини, де елементи одного кластеру мають високу схожість між собою і значно відрізняються від елементів інших кластерів. Цей підхід використовується, коли необхідно отримати однозначну класифікацію об’єктів, наприклад, у випадку сегментації продуктів за споживчими характеристиками або поділу графічних процесорів за рівнем продуктивності. До найбільш відомих алгоритмів чіткої кластеризації належить K-Means, де кожен об’єкт приписується до найближчого центроїду, та алгоритми кластеризації на основі центру мас (centroid-based clustering).

Нечітка кластеризація — це підхід, при якому об’єкт може одночасно належати до кількох кластерів з різними ступенями приналежності. Кожному елементу присвоюється коефіцієнт належності до кожного кластера, що дозволяє враховувати неповну визначеність та плавні межі між групами даних. Такий метод особливо ефективний у випадках, коли класи об’єктів перекриваються або межі між ними нечіткі. Нечітка кластеризація застосовується у задачах, де традиційна чітка класифікація може призводити до втрати інформації, наприклад, при аналізі складних характеристик GPU, де одна відеокарта може бути оптимальною одночасно для декількох сфер використання. Відомим прикладом є Fuzzy C-Means (FCM), де обчислюється матриця приналежності елементів до кластерів і центроїди оновлюються з урахуванням ваги кожного елемента.

Перевагою чіткої кластеризації є простота та зрозумілість результатів, а також висока швидкість обчислень для великих наборів даних. Її недолік — нездатність враховувати складні, перекриті структури даних. Нечітка кластеризація, навпаки, більш гнучка і дозволяє моделювати невизначеність, але потребує більш складних обчислень і часто налаштувань параметрів для досягнення стабільних результатів.

У сучасних аналітичних системах часто застосовують гібридні підходи до кластеризації, які поєднують принципи чіткої та нечіткої кластеризації. Такий підхід дозволяє поєднати простоту та швидкість чітких алгоритмів із гнучкістю нечітких методів. Наприклад, на першому етапі дані можуть бути попередньо

розбиті на великі чіткі кластери за базовими характеристиками (наприклад, рівень продуктивності або ціновий сегмент GPU), після чого кожен кластер деталізується за допомогою нечітких методів, що дозволяють визначити ступінь приналежності кожного об’єкта до підгруп із більш складними ознаками (наприклад, одночасна придатність для AI/ML і рендерингу).

Гібридна кластеризація особливо корисна для багатовимірних і неоднорідних даних, де присутні перекриття між групами та значна різниця в масштабах параметрів. Вона забезпечує більш точну сегментацію та гнучке управління кластерними структурами, дозволяючи аналітичній системі надавати користувачеві детальні та адаптивні рекомендації. У випадку GPU-аналітики гібридний підхід дозволяє не лише класифікувати відеокарти за традиційними показниками продуктивності та ціни, але й враховувати додаткові характеристики, такі як підтримка різних API, енергоефективність і потенційне призначення для різних сценаріїв використання.

Кластеризація на основі центроїдів. Розглянемо групу алгоритмів кластеризації, робота яких заснована на визначенні центрів кластерів. Принцип роботи алгоритму побудований на використанні групування подібних точок властивостей товарів у кластери шляхом встановлення репрезентативних точок, які є центроїдами. Відомим алгоритмом на основі центроїда є k-means, який працює шляхом виділення подібних точок даних, наприклад, користувачів або елементів у кластери, що представлені центроїдами. Головним недоліком цього алгоритму у персоналізованих системах рекомендацій є чутливість до стартового вибору центроїдів. Якщо початкові центроїди не вибрані ретельно, алгоритм може сходитися до локального оптимуму, що призведе до неоптимальної кластеризації. Існує покращена версія алгоритму, що має назву k-means++ – це варіант k-means, що усуває його основний недолік, тобто чутливість до початкових умов. Центр першого кластеру обрається випадково, а центр наступного кластера обирається як найбільш віддалений від попереднього. Це усуває основний недолік алгоритму, пов’язаний з чутливістю до вибору початкових даних [1].

До даного виду кластеризації відноситься ще покращений алгоритм k-means - Mini Batch K-means. Він сприяє швидкому зближенню, обробляючи 'міні-пакети' наборів даних в послідовних ітераціях, поєднуючи швидкість з точністю - комбінація, яка часто відсутня у стандартному алгоритмі K-means при роботі з великими наборами даних. Mini-Batch K-means – це оптимізований варіант його традиційного аналога, який спеціально розроблений для підвищення швидкості без істотного зниження якості кластерів. Замість використання всього набору даних на кожній ітерації, алгоритм Mini-batch K-means працює на випадкових підмножинах або "міні-пакетах" даних [5].

Ієрархічна кластеризація. Ієрархічна кластеризація базується на підході аналізу елементів, що більш пов’язані з елементами поряд, а ніж з тими, що розташовані далі. Цей тип алгоритмів створює деревоподібні структури з метою генерації кластерів. Основним підходом для створення кластера є підбір елементів зі структури, за відстанню, тоді створений фрагмент даних характеризується максимальною відстанню, для групування частин кластера [1].

Цікавим прикладом є алгоритм HDBSCAN, що ієрархічним алгоритм кластеризації, який особливо підтвердив свою ефективність у використанні для персоналізованих систем рекомендацій. Цей алгоритм кластеризації базується на щільності розподілу, що означає, що він групує точки датасету разом на основі їх щільності в просторі даних. Це робить HDBSCAN більш стійким до викидів і шуму, ніж інші алгоритми кластеризації. Додатковою перевагою визначення кількості кластерів в процесі роботи алгоритму, н відміну від k-means, і можливість ідентифікувати кластери різної форми [1].

До даного виду кластеризації відноситься ще така назки алгоритмів кластеризації:

1) Агломеративна кластеризація (Agglomerative Clustering) – принцип полягає у постійному обчисленні відстаней між кластерами. Для цього можна використовувати різні метрики, наприклад, євклідову, манхеттенську чи косинусну відстань. На кожному кроці алгоритм об’єднує два кластери, які знаходяться найближче один до одного, і цей процес повторюється до досягнення критерію

зупинки. Об’єднання кластерів може здійснюватися за різними стратегіями, серед яких single linkage, complete linkage, average linkage та Ward’s method, що дозволяє контролювати спосіб формування нових кластерів.

Результатом роботи алгоритму є дендрограма — деревоподібна структура, яка наочно відображає процес об’єднання кластерів і дозволяє аналізувати дані на різних рівнях деталізації. Ця особливість робить Agglomerative Clustering зручним для вивчення структури даних та визначення оптимальної кількості кластерів. Метод добре підходить для невеликих та середніх наборів даних, а також дозволяє формувати кластери складної форми, що не обов’язково мають сферичну структуру.

Серед переваг агломеративної кластеризації варто відзначити наочність результатів завдяки дендрограмі, можливість аналізу ієрархії кластерів та відсутність необхідності задавати кількість кластерів на початку роботи алгоритму. Однак алгоритм має і певні недоліки. Він є обчислювально витратним, що ускладнює його застосування на великих наборах даних. Крім того, результати сильно залежать від обраної метрики відстані та стратегії об’єднання, а також алгоритм недостатньо стійкий до шуму та викидів, якщо не застосовуються додаткові методи очищення даних.

Agglomerative Clustering знаходить широке застосування у різних галузях. Його використовують для сегментації користувачів у маркетингових дослідженнях, для класифікації біологічних даних, таких як гени або клітини, а також для аналізу технічних характеристик обладнання, включаючи графічні процесори. Завдяки цьому алгоритму можна виділяти групи GPU з подібною продуктивністю та енергоефективністю, що сприяє більш точному і структурованому підбору обладнання.

2) Divisive Clustering - це тип ієрархічної кластеризації, що використовує підхід «згори донизу». Вона починається з поміщення всіх точок даних в один великий кластер, а потім рекурсивно розбиває цей кластер на дрібніші, ґрунтуючись на відмінностях або відстанях між точками. Цей процес триває доти, доки кожен кластер не міститиме лише схожі точки даних або не досягне умови

зупинки. Це протилежність агломеративної кластеризації, яка будує кластери знизу нагору. Вона корисна, коли хочемо розбити широку категорію більш дрібні, змістовні групи [6].

Неієрархічна кластеризація — це підхід, за якого кластери формуються без побудови ієрархічної структури. На відміну від ієрархічних методів, що поступово об’єднують або розділяють групи об’єктів, неієрархічні алгоритми одразу поділяють множину даних на певну кількість кластерів на основі заздалегідь заданих критеріїв. Найчастіше ці методи потребують визначення кількості кластерів k до початку роботи алгоритму, що робить їх ефективними для великих обсягів даних і завдань, де структура груп відома або може бути оцінена.

Основна ідея неієрархічної кластеризації полягає в тому, що кожен об’єкт належить до одного кластера, який має певний центр або представника, що відображає середнє положення елементів у цьому кластері. Після початкового розподілу даних алгоритм поступово коригує центри кластерів і повторно призначає об’єкти, доки не буде досягнуто стабільності — тобто, коли зміни у складі груп стають мінімальними.

Найвідомішим прикладом неієрархічного методу є K-Means, який шукає такі центри кластерів (центроїди), щоб мінімізувати суму квадратів відстаней між об’єктами та їхніми центрами. До цієї ж групи належать варіації, як-от K-Means++, що оптимізує початковий вибір центрів, і Mini-Batch K-Means, який прискорює процес за рахунок роботи з підвибірками даних.

Неієрархічні методи є гнучкими та масштабованими, тому вони широко застосовуються для великих наборів даних, зокрема у сфері аналітики, машинного навчання та дослідження ринкових сегментів. Вони добре підходять тоді, коли кластери мають приблизно сферичну форму та однакову густину розподілу точок.

Серед недоліків неієрархічної кластеризації варто відзначити необхідність попереднього вибору кількості кластерів і чутливість до випадкового початкового розподілу центрів. Також такі алгоритми погано працюють із даними, що містять шуми або кластери складної форми, оскільки вони передбачають, що всі групи мають приблизно однакові розміри та структуру.

Кластеризація на основі щільності. Кластеризація на базі щільності використовує ідею ідентифікації груп елементів в даних через припущення, що кластер у просторі даних є безперервною областю високої щільності, відокремленої від інших таких кластерів суміжними областями низької точки щільності. Точки даних у роздільних областях із низькою щільністю точок зазвичай вважаються шумом чи викидами. Розглянемо цей тип кластеризації на прикладі алгоритму Mean shift. [1].

Перший крок передбачає оцінку базової функції щільності ймовірності розподілу точок даних. Зазвичай це робиться за допомогою оцінки щільності ядра, де кожна точка даних представлена функцією ядра з центром у цій точці. Функція ядра визначає вагу, призначену кожній точці даних у процесі оцінки щільності [1].

Наступним кроком алгоритм ітеративно переміщує точки даних до областей з вищою щільністю на основі векторів середнього зсуву, обчислених як зважене середнє значення відмінностей між кожною точкою та її сусідами. Ітерації тривають до конвергенції, що вказується векторами мінімального середнього зсуву. Кінцеве розташування кожної точки даних позначає центр кластера, що дозволяє призначити найближчий центр і ідентифікувати окремі кластери в даних. На відміну від добре відомого підходу кластеризації k-means, Mean shift не потребує припущень щодо кількості кластерів і форми розподілу, але його продуктивність залежить від вибору параметрів масштабу. Пропускна здатність є єдиним параметром для налаштування, тому для одновимірного випадку це відносно проста процедура, але в багатовимірному випадку це може викликати певні складнощі [1].

До даного виду кластеризації відноситься DBSCAN — алгоритм кластеризації даних, який запропонували Мартін Естер у 1996 році. Для заданої множини точок у деякому просторі він відносить в одну групу точки, які розташовані найбільш щільно (точки з багатьма сусідами) та розмічає точки, які лежать в областях з невеликою щільністю (чиї сусіди розташовані занадто далеко) як викиди. DBSCAN є одним з найпоширеніших алгоритмів кластеризації, а також найбільш цитованим у науковій літературі [7].

Кластеризація на основі графів. У цьому підході дані представляються у вигляді графа: вузли відповідають об’єктам, а ребра між ними характеризують зв’язки або схожість між цими об’єктами. Основна ідея полягає у формуванні кластерів на основі щільності та сили зв’язків між вузлами, що дозволяє ідентифікувати групи об’єктів із високою внутрішньою схожістю і слабкою зовнішньою взаємодією з іншими групами.

Одним із популярних алгоритмів графової кластеризації є Spectral Clustering. Він базується на використанні спектральних властивостей матриці суміжності або матриці Лапласа графа. Алгоритм спочатку будує матрицю подібності між об’єктами, потім визначає власні значення і власні вектори цієї матриці, що дозволяє знизити розмірність даних і виділити природні групи. Після цього стандартні алгоритми кластеризації, наприклад K-Means, застосовуються до отриманих власних векторів для формування фінальних кластерів. Spectral Clustering особливо ефективний у випадках, коли кластери мають складну форму або не є компактними в звичайному просторі ознак.

Ще одним методом є Louvain. Цей алгоритм застосовується для виявлення спільнот у великих мережах і ґрунтується на максимізації так званої модульності. Модульність характеризує, наскільки вузли всередині кластера більше пов’язані між собою, ніж із вузлами інших кластерів. Louvain працює ітеративно, поєднуючи вузли у малі спільноти, а потім об’єднуючи ці спільноти на наступному рівні, що дозволяє будувати багаторівневу ієрархічну структуру. Алгоритм показує високу ефективність для великих графів, наприклад, соціальних мереж, біологічних взаємодій чи мереж комунікацій.

Алгоритм Girvan-Newman також належить до графових методів кластеризації і заснований на концепції “центральності ребер”. Основна ідея полягає в тому, щоб послідовно видаляти ребра, які з’єднують різні кластери, оцінюючи їх за показником центральності між вузлами. При кожному кроці мережа розпадається на менші компоненти, що дозволяє поступово виділяти спільноти у графі. Girvan-Newman є ефективним для виявлення чітко відокремлених груп, проте може бути

обчислювально затратним для великих мереж через необхідність багаторазового перерахунку центральності ребер.

Використання графових алгоритмів кластеризації дає змогу вирішувати специфічні завдання, які важко піддаються традиційним методам, таким як K-Means або DBSCAN. Вони добре працюють на даних із складною структурою зв’язків, де важлива не тільки відстань між точками у просторі ознак, а й топологія взаємозв’язків між об’єктами. Graph-based clustering дозволяє виявляти природні групи у соціальних мережах, рекомендаційних системах, біологічних мережах і навіть у складних IT-інфраструктурах для кластеризації апаратних компонентів за взаємозв’язками.

Модельна кластеризація. Модельна кластеризація ґрунтується на припущенні, що дані створені за певною статистичною моделлю, яка описує розподіл спостережень у просторі ознак. На відміну від алгоритмів, що визначають кластери на основі відстаней або щільності, модельна кластеризація намагається знайти найімовірнішу модель, яка найкраще пояснює структуру даних. Кожен кластер при цьому розглядається як випадкова змінна, що підпорядковується певному розподілу ймовірностей [8].

Основна ідея полягає в тому, щоб представити всю множину даних як суміш декількох імовірнісних моделей (distribution components), кожна з яких відповідає певному кластеру. Таким чином, замість жорсткого розподілу об’єктів по кластерах (як у чіткій кластеризації), модельна кластеризація дозволяє оцінювати ймовірність належності кожного об’єкта до кожного з можливих кластерів. Це робить підхід більш гнучким і точним, особливо для складних, неперервних або неоднорідних даних [8].

Одним з найвідоміших алгоритмів у межах цього підходу є Gaussian Mixture Models (GMM). Цей метод передбачає, що дані утворені сумішшю кількох нормальних (гаусових) розподілів, кожен з яких описує один кластер. GMM не вимагає, щоб кластери були сферичними або мали однаковий розмір, на відміну від K-Means, що робить його більш універсальним для реальних даних. Для навчання такої моделі найчастіше застосовується алгоритм очікування-

максимізації (Expectation-Maximization, EM), який ітераційно оцінює параметри кожного розподілу — середнє значення, дисперсію та вагу (ймовірність приналежності до кластера) [8].

Перевагою модельної кластеризації є можливість роботи з перекривними кластерами та оцінювання ймовірностей замість жорстких рішень. Це дозволяє визначати об’єкти, що належать до кількох кластерів одночасно з різним ступенем впевненості. Однак недоліком цього підходу є його висока обчислювальна складність і залежність від припущення про форму розподілу, що може не завжди відповідати реальним даним [8].

Завдяки своїй статистичній природі, модельна кластеризація часто застосовується в аналізі зображень, біоінформатиці, маркетинговій сегментації, а також у системах прогнозування, де важливо оцінити імовірність віднесення спостереження до певного класу [8].

Таблиця порівняння алгоритмів кластеризації зображено в таблиці В.1.

2.5. Обґрунтування вибору технологій, методів, алгоритмів для створення експертної системи

Для створення алгоритму підбору графічних процесорів у межах експертної системи було достатньо використати дві основні методики кластеризації – DBSCAN та K-Means. Цей вибір обґрунтований як технічними особливостями алгоритмів, так і специфікою даних, з якими працює система.

K-Means був обраний як базовий алгоритм для кластеризації GPU за основними характеристиками, такими як продуктивність, ціна та показник GPU Value. Головною перевагою цього алгоритму є простота та швидкість обчислень, що дозволяє ефективно обробляти великі набори даних і формувати однорідні групи GPU. Алгоритм добре підходить для створення сегментів обладнання з подібними характеристиками, що дає змогу користувачу швидко отримувати рекомендації щодо оптимальних варіантів у межах конкретного цінового або продуктивного сегменту. Крім того, K-Means легко інтегрується у систему та

забезпечує чіткий, зрозумілий результат кластеризації, що важливо для аналітичного інтерфейсу системи.

DBSCAN використовується як додатковий алгоритм, що дозволяє враховувати специфіку даних із нерівномірним розподілом характеристик. Основною перевагою DBSCAN є здатність ідентифікувати кластери будь-якої форми та щільності, а також ігнорувати шумові точки або аномальні GPU, які не відповідають основним групам. Це особливо важливо у випадку аналізу великих наборів графічних процесорів, де деякі моделі мають нестандартні параметри або надзвичайно високу/низьку продуктивність. DBSCAN забезпечує більш гнучкий підхід, що дозволяє виділяти природні групи GPU без необхідності попередньо задавати кількість кластерів, на відміну від K-Means.

Використання комбінації K-Means та DBSCAN дозволяє поєднати переваги обох підходів. K-Means забезпечує швидку та зрозумілу класифікацію основних груп GPU, тоді як DBSCAN дає змогу виділяти нетипові або спеціалізовані кластери, а також виключати аномальні дані. Такий підхід дозволяє створити алгоритм, який здатен адаптуватися до різних сценаріїв використання GPU, враховувати багатофакторність показників і надавати користувачеві точні та практично корисні рекомендації.

Таким чином, для цілей підбору графічних процесорів достатньо було використати саме ці два алгоритми: вони забезпечують ефективність, гнучкість та надійність системи, а також дозволяють обробляти великі та неоднорідні набори даних без зайвих ускладнень.

2.6. Висновок до другого розділу

У другому розділі проведено детальний аналіз існуючих підходів та методів для розв’язання задач класифікації та підбору графічних процесорів. Розглянуто поняття кластеризації як одного з актуальних методів аналізу даних, її роль у побудові рекомендаційних систем та можливість виділення груп об’єктів за схожістю характеристик.

Розглянуто основні види алгоритмів кластеризації. Для кожного алгоритму визначено принцип роботи, переваги та недоліки, а також можливість застосування для аналізу відеокарт та рекомендаційних задач. На основі проведеного аналізу та обґрунтування прийнято рішення для реалізації майбутньої експертної системи використовувати алгоритми K-Means та DBSCAN для гнучної і швидкої обробки даних.

РОЗДІЛ 3. ДОСЛІДЖЕННЯ ТА РОЗРОБКА ЕКСПЕРТНОЇ СИСТЕМИ ПІДБОРУ ГРАФІЧНИХ ПРОЦЕСОРІВ

3.1. Опис масиву даних для аналізу GPU

Для реалізації експертної системи підбору графічних процесорів (GPU) був використаний тестовий набір даних з відкритого ресурсу Kaggle (https://www.kaggle.com/datasets/alanjo/gpu-benchmarks). Даний набір містить результати тестування продуктивності великої кількості сучасних відеокарт та дозволяє порівнювати їх між собою за стандартними метриками. Дані актуальні станом на 8 травня 2022 року та постійно оновлюються з джерела PassMark. Датасет складається з двох основних таблиць – Benchmarks, яка описана в таблиці 3.1 та API scores – в таблиці 3.2.

Таблиця 3.1 - Опис таблиці Benchmarks

Назва поля	Тип даних	Опис
gpuName	Текст	Назва графічного процесора
G3Dmark	Числовий	Інтегральний показник 3D-продуктивності
G2Dmark	Числовий	Показник 2D-продуктивності
price	Числовий	Ціна GPU у доларах США
gpuValue	Числовий	Індекс співвідношення продуктивність/ціна
TDP	Числовий	Тепловий пакет (Вт), що відображає енергоспоживання
powerPerformance	Числовий	Показник енергоефективності
testDate	Числовий	Рік проведення тесту
category	Текст	Категорія GPU (ігрова, професійна, офісна тощо)

Бенчмарки (Benchmarks) – це стандартизовані тестові програми, призначені для оцінки продуктивності графічних процесорів у різних задачах. Коли запускається тест, програма послідовно виконує серію обчислень або рендерингових операцій, які імітують реальні сценарії використання GPU: 3D-графіку, обробку текстур, моделювання фізики або прості 2D-операції.

Для підрахунку G3Dmark і G2Dmark програма взаємодіє безпосередньо з фізичною відеокартою через драйвери та системні API (наприклад, DirectX або OpenGL). Тест надсилає GPU команди на обробку певних графічних об’єктів і вимірює час виконання, пропускну здатність і кількість кадрів за секунду. Результати обробляються в бали, які дозволяють порівнювати різні моделі за єдиною шкалою.

Додаткові показники, такі як TDP (енергоспоживання) та powerPerformance, відображають ефективність використання ресурсів відеокарти.

Таблиця 3.2 - Опис таблиці API scores

Назва поля	**Тип даних**	**Опис**
Manufacturer	Текст	Виробник графічного процесора
Device	Текст	Модель графічного процесора
CUDA	Числовий	Оцінка продуктивності для обчислень через CUDA
Metal	Числовий	Оцінка продуктивності для API Metal
OpenCL	Числовий	Оцінка продуктивності для OpenCL
Vulkan	Числовий	Оцінка продуктивності для Vulkan

Графічні API (CUDA, Metal, OpenCL, Vulkan) – це інтерфейси, які дозволяють програмам запускати обчислення на GPU, використовуючи його паралельні обчислювальні блоки. Наприклад:

- CUDA використовується на відеокартах Nvidia і дає доступ до тисяч обчислювальних ядер для паралельної обробки даних.
- OpenCL і Vulkan підтримуються на різних виробниках і дозволяють виконувати складні обчислення та рендеринг на GPU незалежно від платформи.
- Metal – аналогічний інтерфейс для пристроїв Apple.

Після запуску API-тесту програма компілює код, що описує обчислення або рендеринг, у формат, зрозумілий GPU. Тест надсилає цей код відеокарті, яка паралельно обробляє тисячі потоків даних, після чого результати вимірюються і агрегуються в бали. Це дозволяє оцінити, наскільки ефективно конкретна модель GPU працює з різними типами обчислень та графічних операцій.

Із даними з обох таблиць датасету можливо виконати широкий спектр аналітичних операцій та розрахунків, що безпосередньо використовуються в експертній системі підбору GPU. Наприклад, на основі показників продуктивності (G3Dmark, G2Dmark, CUDA, Metal, OpenCL, Vulkan) можна розрахувати інтегральний бал GPU, що відображає загальну потужність та ефективність відеокарти для різних сфер застосування. Дані про ціну та індекс GPU Value дозволяють визначити співвідношення продуктивність/вартість та класифікувати карти за економічною доцільністю придбання. Показники TDP та powerPerformance дають змогу оцінити енергоефективність і теплові характеристики карт, що важливо для вибору оптимальних рішень у професійних робочих станціях або серверних середовищах. Крім того, інформація про тестову дату та категорію GPU дозволяє проводити аналіз трендів розвитку графічних процесорів та автоматично рекомендувати актуальні моделі. Використовуючи ці дані, експертна система може здійснювати кластеризацію відеокарт за продуктивністю, ціною, енергоефективністю та сумісністю з різними графічними API, а також генерувати персоналізовані рекомендації для користувача.

3.2. Обґрунтування вибору СУБД

Для реалізації бази даних в аналітичному сховищі даних (data warehouse) у межах експертної системи підбору GPU було обрано СУБД **PostgreSQL**.

PostgreSQL - це потужна, відкрита об’єктно-реляційна система управління базами даних (СУБД), призначена для зберігання, обробки та керування великими обсягами даних. Розроблений з акцентом на розширюваність та сумісність зі стандартом SQL, PostgreSQL підтримує як реляційні, так і деякі NoSQL-функції, такі як зберігання JSON-даних та ключ-значення. Її основне призначення - надавати безпечне та надійне рішення для зберігання даних, яке дозволяє користувачам ефективно керувати складними транзакціями та виконувати аналіз даних у режимі реального часу [10] .

Основними перевагами СУБД є:

1) PostgreSQL — це вільна та відкрита СУБД, що дозволяє уникнути ліцензійних витрат — важлива перевага для академічного проєкту або корпоративного рішення з обмеженим бюджетом.

2) Висока сумісність із SQL. PostgreSQL підтримує практично всі функції стандартного SQL, що робить його зручним для розробки складних запитів та транзакцій [10].

3) Надійність та відповідність ACID-стандартам. Підтримка ACID (атомарність, узгодженість, ізоляція, довговічність) робить PostgreSQL стійким та передбачуваним при обробці критичних даних [10].

4) Розширюваність. PostgreSQL дозволяє користувачам додавати свої функції, що корисно для специфічних завдань та налаштування під унікальні вимоги

5) Підтримка JSON і NoSQL. Крім реляційних даних, PostgreSQL також підтримує JSON-дані, що робить його зручним для гібридних додатків, де потрібні як реляційні, так і документно-орієнтовані дані [10].

6) PostgreSQL підтримує горизонтальне розбиття даних (partitioning), що спрощує роботу з великими обсягами аналітичних даних.

7) PostgreSQL добре справляється з аналітичними (OLAP) запитами: підтримує багатовимірний аналіз, агреговані запити, статистичні моделі, і має механізми для опрацювання великих обсягів даних.

8) PostgreSQL дозволяє створювати власні типи даних, функції, процедури — це дає змогу адаптувати СУБД під спеціальні аналітичні задачі або специфіку доменної області

Використання PostgreSQL є доцільним у тих випадках, коли проєкту потрібна надійна й масштабована система зберігання даних, що забезпечує потужні можливості роботи зі складними SQL-запитами та здатна гнучко опрацьовувати різнорідні типи інформації.

3.3. Імпорт і нормалізація набору даних СУБД

Оскільки прямий імпорт даних з платформи Kaggle до розробленої бази даних є неможливим, інформацію було попередньо збережено у форматі .csv (GPU_scores_graphicsAPIs.csv та GPU_benchmarks_v7.csv), після чого виконано її подальше завантаження та обробку в системі.

Розглянемо код імпорту даних із файлу GPU_scores_graphicsAPIs.csv, що містить дані про API-показники (CUDA, Metal, OpenCL, Vulkan):

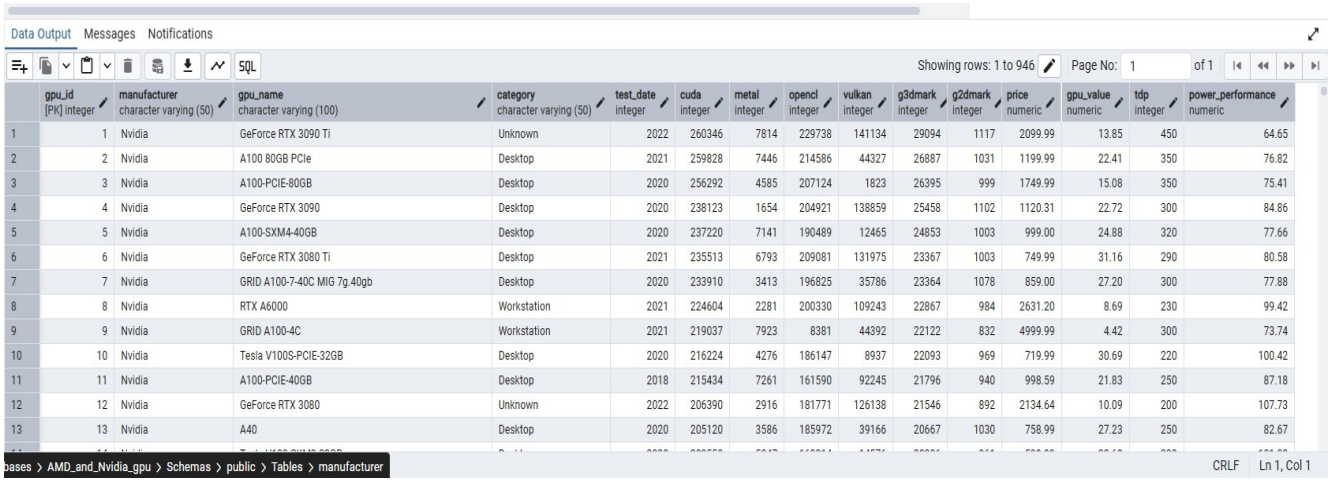
```
\copy tmp_gpu_api_perm (manufacturer, gpu_name, cuda, metal, opencl, vulkan)
FROM 'C:/Users/Danil/Documents/GPU_scores_graphicsAPIs.csv'
DELIMITER ';' CSV HEADER;
```

Розглянемо код імпорту даних із файлу GPU_benchmarks_v7.csv, що містить дані про результати бенчмарків, цінових показників та технічних характеристик:

```
\copy tmp_gpu_benchmarks_perm (gpu_name, g3dmark, g2dmark, price, gpu_value, tdp, power_performance, test_date, category)
FROM 'C:/Users/Danil/Documents/GPU_benchmarks_v7.csv'
DELIMITER ';' CSV HEADER;
```

Після завантаження даних у тимчасові таблиці створено одну об’єднану таблицю gpu_full, яка містила усі ключові атрибути відеокарт: назву моделі, виробника, категорію, дату тестування, API показники (CUDA, Metal, OpenCL, Vulkan), результати бенчмарків (G3Dmark, G2Dmark), ціну, значення GPU value, TDP та показник ефективності енергоспоживання.

Дані в таблицю gpu_full, наведену на рисунку 3.1, потрапляли безпосередньо з тимчасово створених таблиць tmp_gpu_benchmarks_perm і tmp_gpu_api_perm



Data Output | Messages | Notifications

Showing rows: 1 to 946 | Page No: 1 of 1

	gpu_id [PK] integer	manufacturer character varying (50)	gpu_name character varying (100)	category character varying (50)	test_date integer	cuda integer	metal integer	opencl integer	vulkan integer	g3dmark integer	g2dmark integer	price numeric	gpu_value numeric	tdp integer	power_performance numeric
1	1	Nvidia	GeForce RTX 3090 Ti	Unknown	2022	260346	7814	229738	141134	29094	1117	2099.99	13.85	450	64.65
2	2	Nvidia	A100 80GB PCIe	Desktop	2021	259828	7446	214586	44327	26887	1031	1199.99	22.41	350	76.82
3	3	Nvidia	A100-PCIE-80GB	Desktop	2020	256292	4585	207124	1823	26395	999	1749.99	15.08	350	75.41
4	4	Nvidia	GeForce RTX 3090	Desktop	2020	238123	1654	204921	138859	25458	1102	1120.31	22.72	300	84.86
5	5	Nvidia	A100-SXM4-40GB	Desktop	2020	237220	7141	190489	12465	24853	1003	999.00	24.88	320	77.66
6	6	Nvidia	GeForce RTX 3080 Ti	Desktop	2021	235513	6793	209081	131975	23367	1003	749.99	31.16	290	80.58
7	7	Nvidia	GRID A100-7-40C MIG 7g.40gb	Desktop	2020	233910	3413	196825	35786	23364	1078	859.00	27.20	300	77.88
8	8	Nvidia	RTX A6000	Workstation	2021	224604	2281	200330	109243	22867	984	2631.20	8.69	230	99.42
9	9	Nvidia	GRID A100-4C	Workstation	2021	219037	7923	8381	44392	22122	832	4999.99	4.42	300	73.74
10	10	Nvidia	Tesla V100S-PCIE-32GB	Desktop	2020	216224	4276	186147	8937	22093	969	719.99	30.69	220	100.42
11	11	Nvidia	A100-PCIE-40GB	Desktop	2018	215434	7261	161590	92245	21796	940	998.59	21.83	250	87.18
12	12	Nvidia	GeForce RTX 3080	Unknown	2022	206390	2916	181771	126138	21546	892	2134.64	10.09	200	107.73
13	13	Nvidia	A40	Desktop	2020	205120	3586	185972	39166	20667	1030	758.99	27.23	250	82.67

bases > AMD_and_Nvidia_gpu > Schemas > public > Tables > manufacturer

CRLF | Ln 1, Col 1

Рисунок 3.1 – Таблиця gpu_full яка містить усі дані про відеокарти

Зберігання всіх даних в одній таблиці створює ризики дублювання інформації, ускладнює внесення змін і обмежує можливості ефективного аналізу. Наприклад, при зміні назви виробника або корекції характеристик API довелося б редагувати кілька рядків одночасно, що підвищує ймовірність помилок.

Тому було прийнято рішення нормалізувати базу даних до третьої нормальної форми (3НФ). Це дозволило:

- уникнути повторення інформації про виробника і характеристики відеокарт;
- забезпечити чітку залежність атрибутів від ключових полів;
- підвищити цілісність даних та полегшити оновлення таблиць;
- підготувати структуру бази даних до подальшого ефективного використання в аналітичних завданнях

Для досягнення 1НФ була забезпечена атомарність всіх полів у таблицях. Це означає, що кожна колонка зберігає лише одне значення і не містить масивів чи повторюваних груп.

Для досягнення 2НФ одну таблицю було розділено на чотири окремі таблиці:

1. manufacturer – зберігає інформацію про виробників відеокарт (id, name);

```
CREATE TABLE manufacturer (
    id SERIAL PRIMARY KEY,
    name VARCHAR(50) UNIQUE NOT NULL
);
```

2. gpu – містить базові дані про відеокарти: gpu_id, manufacturer_id, gpu_name, category, test_date;

```
CREATE TABLE gpu (
    id SERIAL PRIMARY KEY,
    manufacturer_id INT REFERENCES manufacturer(id),
    gpu_name VARCHAR(100) UNIQUE NOT NULL,
    category VARCHAR(50),
    test_date INT
);
```

3. gpu_api_scores – містить результати тестів графічних API: score_id, gpu_id, cuda_score, metal_score, opencl_score, vulkan_score;

```
CREATE TABLE gpu_api_scores (
    score_id SERIAL PRIMARY KEY,
    gpu_id INT REFERENCES gpu(id) ON DELETE CASCADE,
    cuda_score INT,
    metal_score INT,
    opencl_score INT,
    vulkan_score INT
);
```

4. gpu_benchmarks – зберігає показники продуктивності та параметри відеокарт: benchmark_id, gpu_id, g3dmark, g2dmark, price, gpu_value, tdp, power_performance.

```
CREATE TABLE gpu_benchmarks (
    benchmark_id SERIAL PRIMARY KEY,
    gpu_id INT REFERENCES gpu(id) ON DELETE CASCADE,
    g3dmark INT,
    g2dmark INT,
    price NUMERIC,
    gpu_value NUMERIC,
    tdp INT,
```

```
    power_performance NUMERIC
);
```

Для 3НФ були усунуті транзитивні залежності, тобто ситуації, коли неключові атрибути залежать один від одного через інші неключові атрибути. У БД ціна відеокарти, показники продуктивності та API-бали тепер зберігаються в окремих таблицях, що залежать лише від gpu_id.

Наступним кроком було додавання даних у створені таблиці, які виокремлювались з таблиці gpu_full.

Код додавання даних до таблиць наведено у таблиці 3.1

Для заповнення таблиці manufacturer використано наступний SQL-код:

```
INSERT INTO manufacturer (name)
SELECT DISTINCT manufacturer
FROM gpu_full;
```

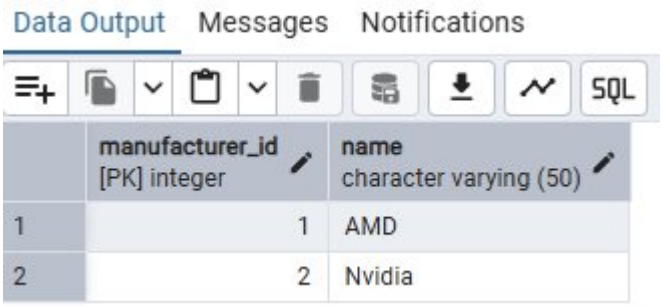


Рисунок 3.2 – Таблиця manufacturer, яка містить назви компаній AMD, Nvidia

Даний запит вибирає унікальні назви виробників з таблиці gpu_full і додає їх до нової таблиці виробників.

Для заповнення таблиці gpu застосовувався наступний код:

```
INSERT INTO gpu (manufacturer_id, gpu_name, category, test_date)
SELECT
    m.id AS manufacturer_id,
    f.gpu_name,
    f.category,
```

```
    f.test_date
FROM gpu_full f
JOIN manufacturer m
    ON f.manufacturer = m.name;
```

Наведений SQL-запит переносить дані з таблиці gpu_full у таблицю gpu, замінюючи текстову назву виробника на його унікальний ідентифікатор з таблиці manufacturer

	gpu_id [PK] integer	manufacturer_id integer	gpu_name character varying (100)	test_date integer	category character varying (50)
1	1	2	GeForce RTX 3090 Ti	2022	Unknown
2	2	2	A100 80GB PCIe	2021	Desktop
3	3	2	A100-PCIE-80GB	2020	Desktop
4	4	2	GeForce RTX 3090	2020	Desktop
5	5	2	A100-SXM4-40GB	2020	Desktop
6	6	2	GeForce RTX 3080 Ti	2021	Desktop
7	7	2	GRID A100-7-40C MIG 7g.40gb	2020	Desktop
8	8	2	RTX A6000	2021	Workstation
9	9	2	GRID A100-4C	2021	Workstation
10	10	2	Tesla V100S-PCIE-32GB	2020	Desktop
11	11	2	A100-PCIE-40GB	2018	Desktop
12	12	2	GeForce RTX 3080	2022	Unknown
13	13	2	A40	2020	Desktop

Total rows: 946 Query complete 00:00:00.095

Рисунок 3.3 – Таблиця gpu, яка містить основну інформацію про кожну відеокарту

Для заповнення таблиці gpu_api_scores було використано наступний SQL-код:

```
INSERT INTO gpu_api_scores (gpu_id, cuda_score, metal_score, opencl_score, vulkan_score)
    SELECT
        g.id AS gpu_id,
        f.cuda,
        f.metal,
        f.opencl,
        f.vulkan
    FROM gpu_full f
```

JOIN gpu g

ON f.gpu_name = g.gpu_name;

	score_id [PK] integer	gpu_id integer	cuda_score integer	metal_score integer	opencl_score integer	vulkan_score integer
1	1	1	260346	7814	229738	141134
2	2	2	259828	7446	214586	44327
3	3	3	256292	4585	207124	1823
4	4	4	238123	1654	204921	138859
5	5	5	237220	7141	190489	12465
6	6	6	235513	6793	209081	131975
7	7	7	233910	3413	196825	35786
8	8	8	224604	2281	200330	109243
9	9	9	219037	7923	8381	44392
10	10	10	216224	4276	186147	8937
11	11	11	215434	7261	161590	92245
12	12	12	206390	2916	181771	126138
13	13	13	205120	3586	185972	39166

Total rows: 946 | Query complete 00:00:00.233

Рисунок 3.4 – Таблиця gpu_api_scores, яка містить дані про продуктивність кожної відеокарти у різних графічних та обчислювальних API

Для заповнення таблиці gpu_benchmarks використовувався наступний код:

INSERT INTO gpu_benchmarks (gpu_id, g3dmark, g2dmark, price, gpu_value, tdp, power_performance)

SELECT

g.id AS gpu_id,

f.g3dmark,

f.g2dmark,

f.price,

f.gpu_value,

f.tdp,

f.power_performance

FROM gpu_full f

JOIN gpu g

ON f.gpu_name = g.gpu_name;

Даний запит вибирає дані про продуктивність і характеристики відеокарт із таблиці gpu_full і додає до них id відеокарти з таблиці gpu. За допомогою JOIN співставляються назви відеокарт (gpu_name), щоб отримати gpu_id, який потім можна використовувати для вставки цих даних у таблицю gpu_benchmarks.

	benchmark_id [PK] integer	gpu_id integer	g3dmark integer	g2dmark integer	price numeric (10,2)	gpu_value numeric (10,2)	tdp integer	power_performance numeric (10,2)
1	1	1	29094	1117	2099.99	13.85	450	64.65
2	2	2	26887	1031	1199.99	22.41	350	76.82
3	3	3	26395	999	1749.99	15.08	350	75.41
4	4	4	25458	1102	1120.31	22.72	300	84.86
5	5	5	24853	1003	999.00	24.88	320	77.66
6	6	6	23367	1003	749.99	31.16	290	80.58
7	7	7	23364	1078	859.00	27.20	300	77.88
8	8	8	22867	984	2631.20	8.69	230	99.42
9	9	9	22122	832	4999.99	4.42	300	73.74
10	10	10	22093	969	719.99	30.69	220	100.42
11	11	11	21796	940	998.59	21.83	250	87.18
12	12	12	21546	892	2134.64	10.09	200	107.73
13	13	13	20667	1030	758.99	27.23	250	82.67

Total rows: 946 Query complete 00:00:00.110

Рисунок 3.5 – Таблиця gpu_benchmarks, яка містить дані про продуктивність кожної відеокарти за оцінками бенчмарків

На виході сформовано 4 основні таблиці. Схема БД наведена на рисунку Д.1.

3.4. Розробка структури сховища даних

Сховище даних — предметно орієнтований, інтегрований, незмінний набір даних, що підтримує хронологію і здатний бути комплексним джерелом достовірної інформації для оперативного аналізу та прийняття рішень [11].

У межах експертної системи підбору графічних процесорів для ТОВ «ТІСЕР» створення сховища даних є ключовим етапом, оскільки воно забезпечує аналітичну узгодженість показників продуктивності, дає змогу виконувати розрахунки,

агрегувати дані та підтримувати процес кластеризації й інтегрального оцінювання GPU. Основою сховища даних є модель «зірка» (Star Schema), що містить фактову таблицю та декілька вимірних таблиць.

Вимір (dimension) — це таблиця довідкових або описових атрибутів, які дають контекст числовим показникам у факт-таблиці. Виміри використовуються для фільтрації, групування та підписування звітів. У класичній схемі «зірка» або «сніжинка» вимірні таблиці містять детальну інформацію про об’єкти аналітики (час, продукт, користувач, географія тощо).

Код SQL-запитів для створення таблиць-вимірів для нашої експертної системи:

```
CREATE TABLE dim_manufacturer (
    manufacturer_id SERIAL PRIMARY KEY,
    manufacturer_name TEXT UNIQUE
);
CREATE TABLE dim_category (
    category_id SERIAL PRIMARY KEY,
    category_name TEXT UNIQUE
);
CREATE TABLE dim_usage (
    usage_id SERIAL PRIMARY KEY,
   usage_name TEXT UNIQUE
);


CREATE TABLE dim_time (
    time_id SERIAL PRIMARY KEY,
    year INT,
);
CREATE TABLE dim_gpu (
    gpu_id INT PRIMARY KEY,
```

```
    manufacturer_id INT REFERENCES dim_manufacturer(manufacturer_id),
    gpu_name TEXT,
    test_date INT,
    category_id INT REFERENCES dim_category(category_id)
);
```

Таблиці-виміри сховища даних із описом полів наведені у таблиці 3.2.

Таблиця 3.3 - Опис таблиць-вимірів сховища даних

Таблиця	**Поле**	**Тип даних**	**Опис / Для чого**
dim_manufacturer	manufacturer_id	SERIAL (PK)	Унікальний ідентифікатор виробника. Використовується для зв'язку з GPU.
	manufacturer_name	TEXT	Назва компанії-виробника (AMD, Nvidia, Intel). Дає змогу групувати GPU за брендами.
dim_category	category_id	SERIAL (PK)	Унікальний ключ категорії GPU.
	category_name	Char	Класифікація GPU: Desktop, Mobile, Workstation. Дозволяє будувати аналітику за типами пристроїв.
	description	TEXT	Опис використання відеокарти
dim_time	time_id	SERIAL (PK)	Унікальний ключ часу тестування.
	year	INT	Рік проведення тесту або виходу GPU. Для трендового аналізу.
dim_usage	usage_id	SERIAL (PK)	Первинний ключ цільового призначення.
	usage_name	TEXT	Тип використання: AI/ML, Gaming, Rendering, Mining, Office. Використовується при сегментації моделей.

Продовження таблиці

dim_gpu	gpu_id	INT (PK)	Ідентифікатор GPU (успадкований з оперативної БД).
	manufacturer_id	INT (FK)	Посилання на dim_manufacturer, вказує бренд GPU.
	gpu_name	TEXT	Модель GPU (наприклад, GeForce RTX 4070). Основний атрибут для аналітики.
	category_id		Вказує категорію GPU: Desktop, Mobile, Workstation тощо. Дозволяє сегментувати відеокарти за типом пристрою, у якому вони використовуються.
	usage_id		Визначає основну сферу застосування GPU: AI/ML, Gaming, Rendering, Office, Mining
	segment_id		Вказує ціновий сегмент GPU: бюджетний, середній, преміальний
	category_id	INT (FK)	Зв’язок із категорією пристрою (desktop/mobile).

Таблиця фактів fact_gpu_metrics є центральним елементом сховища даних і містить числові, аналітичні та розрахункові показники, що відображають реальну продуктивність, енергоефективність та інші технічні характеристики відеокарт. На відміну від вимірних таблиць вимірів, які містять описові атрибути (виробника, модель, призначення, ціновий сегмент тощо), таблиця фактів акумулює метрики, що можуть бути агреговані, порівнювані та використовувані у машинному аналізі.

Основним призначенням таблиці фактів є організація даних у формі, оптимальній для виконання аналітичних запитів, побудови моделей, кластеризації та обчислення інтегральних показників. Таблиця концентрує об'єктивні числові значення, отримані з бенчмарків PassMark та Geekbench, а також розраховані на їх основі додаткові індекси. Зокрема, у таблиці зберігаються ключові поля продуктивності GPU: G3DMark, G2DMark, середній API-скор, індекс продуктивності, енергоефективність, мінімальні та максимальні показники, стандартне відхилення результатів тестів, а також кластерна мітка, що формується після виконання алгоритмів машинного навчання. Таблиця фактів сховища даних із описом полів наведена у таблиці 3.3.

Код SQL-запиту для створення таблиці фактів експертної системи:

```
CREATE TABLE fact_gpu_metrics (
    fact_id SERIAL PRIMARY KEY
    gpu_id INT REFERENCES dim_gpu(gpu_id),
    g3dmark NUMERIC,
    g2dmark NUMERIC,
    price NUMERIC,
    gpu_value NUMERIC,
    tdp NUMERIC,
    performance_index NUMERIC,
    avg_api_score NUMERIC,
    avg_benchmark_score NUMERIC,
    min_score NUMERIC,
    max_score NUMERIC,
    score_stddev NUMERIC,
    power_performance_ratio NUMERIC,
    benchmark_category VARCHAR(20),
    cluster_label INT
);
```

В таблиці 3.4 наведено таблиці фактів сховища даних.

Таблиця 3.4 - Опис таблиці фактів сховища даних

Поле	Тип даних	Опис / Для чого
fact_id	SERIAL (PK)	Унікальний рядок фактичних даних.
gpu_id	INT (FK)	Унікальний ID відеокарти
g3dmark	NUMERIC	Результат 3DMark, характеризує продуктивність GPU у 3D сценах.
g2dmark	NUMERIC	Результат тесту 2D графіки.
price	NUMERIC	Актуальна або середня ринкова ціна GPU.
gpu_value	NUMERIC	Показник «ціна / продуктивність». Дозволяє шукати найвигідніші моделі.

Продовження таблиці

tdp	NUMERIC	Енергоспоживання (TDP). Використовується в аналізі енергоефективності.
performance_index	NUMERIC	Загальний індекс продуктивності (g3dmark + g2dmark).
avg_api_score	NUMERIC	Середній результат API (CUDA/Metal/OpenCL/Vulkan).
avg_benchmark_score	NUMERIC	Середній бал основних бенчмарків.
min_score	NUMERIC	Мінімальний результат серед усіх тестів.
max_score	NUMERIC	Максимальний результат серед усіх тестів.
score_stddev	NUMERIC	Варіативність (стандартне відхилення) між тестами.
power_performance_ratio	NUMERIC	Продуктивність на 1 Вт енергії — показник енергоефективності.
integrated_score	NUMERIC	Інтегральний показник продуктивності GPU
kmeans_label	NUMERIC	Номер кластера, присвоєний моделі K-Means у процесі машинного групування відеокар
efficiency	NUMERIC	Показник співвідношення
is_best_cluster	BOOLEAN	True/False - Кластер, отриманий з алгоритму ML (K-Means). Визначає чи відноситься відеокарта до ТОП

Схема сховища даних у СУБД PostgreSQL наведена на рисунку Д.2.

3.5. Вибір середовища та інструментів розробки

При розробці експертної системи підбору графічних процесорів для ТОВ «ТІСЕР» важливим етапом стало обрання відповідного середовища програмування та технологічного стеку. Основними критеріями вибору були: можливість інтеграції з базою даних та сховищем даних, підтримка роботи з великими обсягами даних, реалізація аналітичних алгоритмів та машинного

навчання, а також забезпечення гнучкого веб-інтерфейсу для користувачів системи.

Для розробки експертної системи було обрано мову Python як основну мову програмування завдяки її широкому функціоналу для аналітики, обробки даних та машинного навчання.

Також Python є одним з найкращих виборів для веброзробки завдяки здатності обробляти складні, мультипротокольні застосунки, зберігаючи код чистим і читабельним. Багато з найпопулярніших додатків сьогодні створені на Python [12].

Однією з основних причин популярності Python є його “жива” спільнота з відкритим вихідним кодом, яка пропонує багатий вибір повторно використовуваного коду, фреймворків і підтримки. Python має легкий для розуміння синтаксис, що робить його доступним для фахівців з невеликим досвідом програмування або взагалі без нього. Не дивно, що це найпопулярніша мова серед дата-сайєнтистів для аналізу та візуалізації даних [12].

Окрім цього Python забезпечує швидку розробку та тестування. Гнучкість дозволяє розробникам швидко створювати прототипи та ітерації свого коду, спрощуючи експерименти з новими ідеями та концепціями без турбот про оголошення типів і компіляцію [12].

Для побудови веб-інтерфейсу та інтеграції з базою даних використано Django — потужний веб-фреймворк, який забезпечує зручну роботу з ORM, дозволяє швидко реалізовувати моделі даних та забезпечує безпечну взаємодію з користувачем через веб-інтерфейс.

Для аналітичної обробки даних використані бібліотеки pandas та NumPy, що дозволяють ефективно опрацьовувати великі таблиці та обчислювати складні показники, такі як інтегральний індекс продуктивності GPU, коефіцієнт ефективності та статистичні метрики (мінімальні, максимальні значення, стандартне відхилення). Кластеризація та машинне навчання реалізовані за допомогою бібліотеки scikit-learn, що дозволяє застосовувати алгоритми K-Means,

DBSCAN та інші методи для групування відеокарт за характеристиками продуктивності.

Для візуалізації даних та підготовки аналітичних звітів застосовуються бібліотеки matplotlib та plotly, що дають змогу створювати інтерактивні графіки та діаграми для зручного аналізу продуктивності GPU у різних сегментах.

Таким чином, обраний стек технологій забезпечує повну інтеграцію між базою даних, сховищем даних та аналітичним модулем, дозволяє реалізувати автоматизовану кластеризацію та оцінку графічних процесорів, а також надає гнучкий інтерфейс для користувача, що значно підвищує ефективність підбору апаратного забезпечення для підприємства.

3.6. Реалізація експертної системи

Реалізація аналітичної інформаційної системи виконувалася у кілька етапів, починаючи з побудови моделей даних та закінчуючи створенням веб-інтерфейсу для користувача.

На початковому етапі було спроектовано та реалізовано моделі даних у Django для взаємодії з базою даних. Основні моделі включали:

- **Manufacturer** — зберігає інформацію про виробника відеокарти (ID та назву).
- **Category** — категорія GPU (наприклад, Desktop, Laptop, Workstation).
- **Usage** — призначення відеокарти, що відображає, для яких задач найбільш оптимальна карта (ігри, AI/ML, рендеринг, майнінг, офісні задачі).
- **Time** — рік випуску або тестування GPU.
- **Segment** — ціновий сегмент GPU (low, mid, high).
- **GPU** — основна модель, що об’єднує інформацію про GPU, його виробника, категорію, призначення, рік та ціновий сегмент.
- **FactGPUMetrics** — фактичні показники продуктивності GPU, включаючи результати тестів (G3DMark, G2DMark), ціну, ефективність, інтегральний індекс продуктивності, кластеризаційні мітки та додаткові аналітичні показники.

Моделі реалізовані з використанням ForeignKey для зв'язку між таблицями, що дозволяє Django ORM ефективно працювати з реляційною базою даних. Поля, пов'язані з аналітикою (наприклад, integrated_score, efficiency, kmeans_label, is_best_cluster), розраховуються програмно на основі даних GPU.

Код створення таблиць Segment та GPU:

```
class Segment(models.Model):
    segment_id = models.AutoField(primary_key=True)
    segment_name = models.CharField(max_length=50)

    class Meta:
        managed = False
        db_table = 'dim_segment'

class GPU(models.Model):
    gpu_id = models.IntegerField(primary_key=True)
    gpu_name = models.CharField(max_length=100)
    manufacturer = models.ForeignKey(Manufacturer, on_delete=models.CASCADE)
    category = models.ForeignKey(Category, on_delete=models.CASCADE)
    usage = models.ForeignKey(Usage, on_delete=models.SET_NULL, null=True,
blank=True)
    time = models.ForeignKey(Time, on_delete=models.CASCADE)
    segment = models.ForeignKey(Segment, on_delete=models.SET_NULL, null=True,
blank=True)

    class Meta:
        managed = False
        db_table = 'dim_gpu'
```

Повний код створення усіх моделей ORM наведено у Додатку Ж.

Для створення єдиного інтерфейсу вебсервісу було створено **базовий шаблон** base.html, який містить:

- Підключення CSS через Bootstrap для швидкої стилізації та адаптивного дизайну.
- Підключення JavaScript-бібліотек: jQuery, Plotly для візуалізації, DataTables для інтерактивних таблиць.

- Навігаційну панель з посиланнями на головну сторінку, підбір GPU та кореляційну матрицю.
- Контейнер для вмісту сторінок, який змінюється через блоки шаблонів Django.
- Футер із інформацією про систему та авторські права.

Код сторінки base.html наведено у додатку Ж.

Головна сторінка системи реалізована через шаблон analytics.html. Вона містить вступний опис системи, що пояснює її основні можливості, такі як аналіз, кластеризація та підбір графічних процесорів (GPU). Користувач одразу отримує уявлення про призначення та функціонал вебсервісу.

На сторінці розташовані візуальні елементи, які демонструють класифікацію відеокарт за призначенням та їхні цінові сегменти. Це дозволяє наочніше сприймати інформацію про категорії GPU і їхню цінову структуру. Інтерфейс сторінки «Головна» зображений (рис. Е.1).

Система пояснює категорії Usage, такі як Gaming, AI/ML, Graphics/Rendering, Mining та Office/Low-power, надаючи характеристики кожного типу GPU. Користувач може легко зрозуміти, які карти підходять для ігор, обчислень штучного інтелекту, рендерингу, майнінгу або базових офісних завдань.

Шапка профілю має можливості переходу до таких сторінок:

- «Кореляційна матриця»
- «Підбір GPU»
- «Головна»

Інтерактивна таблиця на головній сторінці відображає усі доступні GPU від виробників Nvidia та AMD. Вона дозволяє користувачеві обирати потрібні карти для аналізу та порівняння їхніх технічних показників і продуктивності.

Відображення таблиці з даними про графічні процесори реалізовано з використанням бібліотеки DataTables, що забезпечує інтерактивність та зручність роботи. Вона надає користувачу можливість сортування даних, швидкого пошуку та пагінації для перегляду великої кількості записів.

Користувач може вибирати декілька GPU через чекбокси, що дозволяє здійснювати порівняння їхніх метрик.

Код сторінки gpu_table_partial.html з відображенням таблиці даних:

```
{% load static %}
{% load custom_filters %}

<table id="gpu-table" class="table table-bordered table-hover table-sm text-center">
  <thead class="table-dark">
    <tr>
      <th></th>
      <th>Назва</th>
      <th>Виробник</th>
      <th>Категорія</th>
      <th>Призначення</th>
      <th>Ціна</th>
    </tr>
  </thead>
  <tbody>
  {% for g in gpus %}
    <tr>
      <td>
        <input type="checkbox" class="gpu-checkbox" value="{{ g.gpu_id }}">
      </td>
      <td>{{ g.gpu_name }}</td>
      <td>{{ g.manufacturer.manufacturer_name }}</td>
      <td>{{ g.category.category_name }}</td>
      <td>{{ g.usage.usage_name|default:"N/A" }}</td>
      {% with f=fact_dict|get_item:g.gpu_id %}
        {% if f %}
          <td>{{ f.price|floatformat:2 }}</td>
        {% else %}
          <td>N/A</td>
        {% endif %}
      {% endwith %}
    </tr>
  {% endfor %}
  </tbody>
</table>
```

```
<div class="text-center mt-3">
   <button id="show-metrics" class="btn btn-primary" disabled>Показати
метрики</button>
</div>
```

Таблиця даних з інформацією про відеокарти, наведена на рисунку 3.6 надає можливість обрати кількості записів, функції скролу та пошуку інформації.

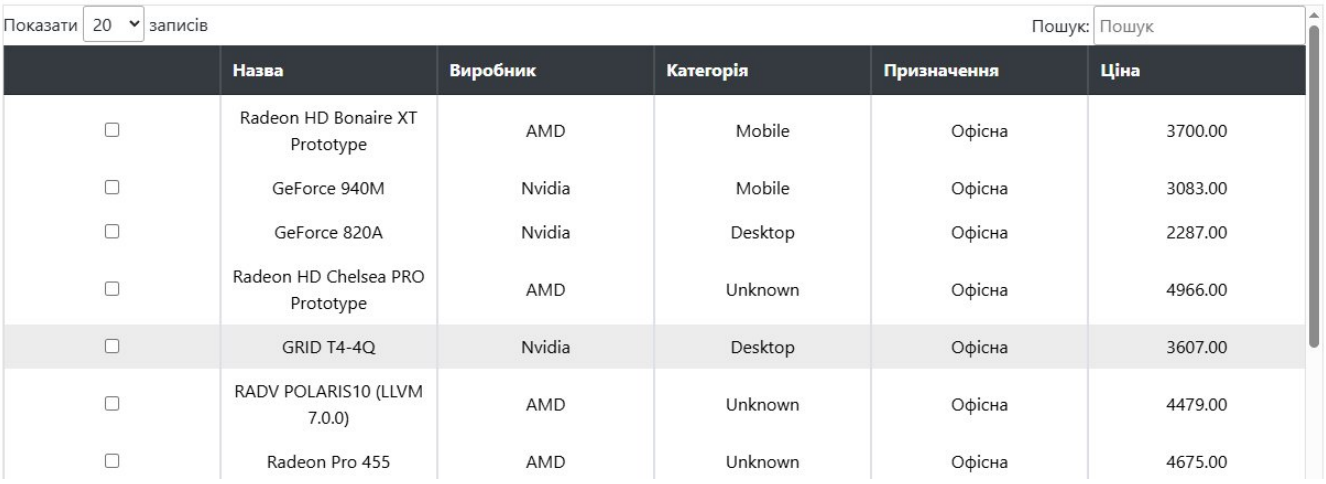


Рисунок 3.6 – Таблиця даних про відеокарти

Також у таблиці присутня кнопка "Показати метрики", яка формує запит до серверу і перенаправляє користувача на сторінку з детальними аналітичними показниками обраних GPU, забезпечуючи інтерактивний та швидкий аналіз. Таблиця має функцію перемикання сторінок. Даний функціонал зображений на рисунку 3.7.

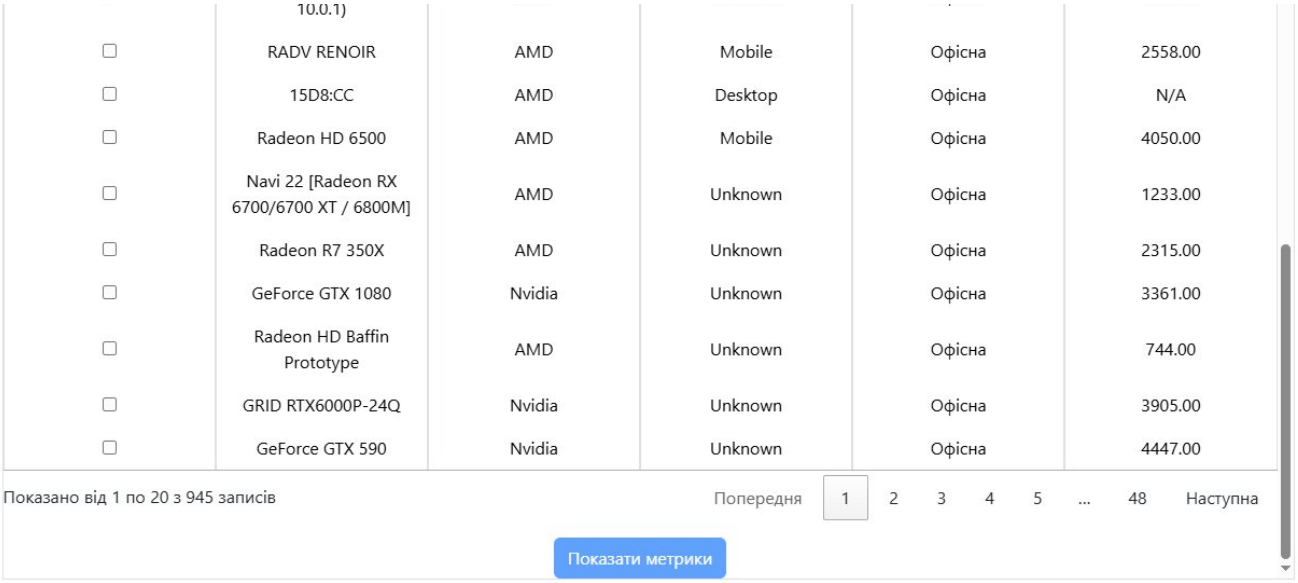


Рисунок 3.7 – Друга частина таблиці на сторінці підбору GPU із даними про відеокарти

Сторінка метрик наведена на рисунку 3 додатку Е.

Контролер main_view у Django відповідає за підготовку даних для головної сторінки та її відображення. Він отримує усі GPU та їхні фактичні метрики, використовуючи метод select_related, що мінімізує кількість SQL-запитів та оптимізує продуктивність системи.

Для швидкого доступу до показників кожного GPU створюється словник fact_dict, де ключем виступає gpu_id. Це дозволяє легко підставляти відповідні метрики у таблицю на фронтенді без додаткових запитів.

Дані передаються у шаблон через контекст, включно з параметром пошуку, що забезпечує можливість фільтрувати GPU за назвою або іншими характеристиками. Така архітектура контролера дозволяє ефективно поєднувати серверну логіку та відображення даних на вебсторінці.

Код контролеру main_view:

```
def main_view(request):
    gpus = GPU.objects.select_related('manufacturer', 'category', 'time', 'usage').all()
    facts = FactGPUMetrics.objects.select_related('gpu').all()
    fact_dict = {f.gpu_id: f for f in facts}

    context = {
```

```
        'gpus': gpus,
        'fact_dict': fact_dict,
        'search_query': request.GET.get('search', '')
    }
    return render(request, 'gpu_app/analytics.html', context)
```

Сторінка підбору графічних процесорів реалізована на основі шаблону pickup.html та призначена для інтелектуального вибору найоптимальніших GPU відповідно до потреб користувача. Вона забезпечує зручний інтерфейс для фільтрації відеокарт за ключовими параметрами: виробником (Nvidia або AMD), призначенням (ігрова, AI/ML, рендер, майнінг або офісна), а також ціновим сегментом (бюджетний, середній або високий). Форма фільтрів інтегрована у вигляді окремої картки, що забезпечує структурованість і комфортне використання.

Після застосування фільтрів система виводить інтерактивну таблицю, яка формує динамічний список рекомендованих GPU. Таблиця підтримує можливість вибору відеокарт через чекбокси, що дозволяє користувачу порівнювати декілька моделей або переходити до перегляду розширених метрик. Для запобігання перевантаженню користувача введено обмеження — одночасно можна вибрати не більше десяти GPU.

Кожний рядок таблиці містить ключові параметри: інтегральний бал продуктивності, ціну, показник GPU Value, індексацію найкращого кластера та обчислену ефективність (співвідношення продуктивності до вартості). Це дозволяє користувачу швидко оцінити, наскільки дана модель відповідає його бюджету та вимогам до продуктивності.

У нижній частині сторінки розміщена кнопка «Показати метрики», яка активується лише після вибору GPU. Кнопка перенаправляє користувача на сторінку з детальними аналітичними метриками, що покращує взаємодію між елементами системи та забезпечує логічну структуру переходу від загального перегляду до поглибленого аналізу.

Логіка роботи функції підбору графічних процесорів. Основна логіка розрахунку метрик та підбору GPU реалізована у методі calc(), код якого наведений

у додатку Ж, який обробляє усі доступні дані та формує інтегральні оцінки для кожного графічного процесора. На першому етапі виконується попередня підготовка даних: визначається призначення GPU (usage), обчислюються базові бенчмарки (performance_index, avg_api_score, avg_benchmark_score) та статистичні показники (min_score, max_score, score_stddev).

Окрему роль у системі відіграє інтегральний бал (integrated_score) — зважена оцінка GPU, що враховує продуктивність у 3D (G3DMark), 2D (G2DMark), загальну цінність GPU (GPU Value) та середні API-бали (CUDA/Metal/OpenCL/Vulkan). Ця метрика дозволяє комплексно оцінювати карту не за одним параметром, а за сукупністю її переваг.

Для глибшого аналізу та побудови рекомендацій використовується два методи кластеризації: DBSCAN та K-Means. Їх застосування дає змогу системі не лише аналізувати GPU, а й групувати їх за поведінковими характеристиками, формувати сегменти ринку та визначати найефективніші групи відеокарт.

Алгоритм DBSCAN використовується як інструмент для виявлення та виключення аномальних даних. Він визначає точки, які не належать жодному кластеру через надмірно відмінні значення (наприклад, GPU з некоректними або екстремальними цінами чи бенчмарками). Такі точки позначаються міткою –1 та не беруть участі у подальшій кластеризації.

Застосування DBSCAN дозволяє:

- уникати спотворення середніх значень у кластерах;
- формувати більш коректні групи відеокарт;
- зменшувати вплив "викидів", що могли виникнути через неточні дані або дефекти вимірювань.

Після очищення даних застосовується алгоритм **K-Means** для створення тематичних груп GPU, кожна з яких характеризується подібністю у продуктивності, ціні, енергоефективності та API-показниках. Алгоритм використовує такі ознаки, як:

- g3dmark, g2dmark
- price, gpu_value

- tdp
- performance_index
- avg_api_score
- показники варіативності та стабільності тестів
- power_performance_ratio

Після формування кластерів система обчислює середнє значення ефективності для кожного з них. Кластер з найвищою середньою ефективністю позначається як найкращий кластер, і відповідні GPU отримують індикатор is_best_cluster = True. Це дозволяє користувачу швидко визначати, які GPU належать до найоптимальнішої групи

Окремо застосовується K-Means з 3 кластерами для класифікації GPU за ціною. Він автоматично формує сегменти:

- low — бюджетні карти
- mid — середня цінова категорія
- high — преміум сегмент

Таким чином, система адаптується до реальних ціноутворень на ринку та дозволяє формувати рекомендації відповідно до бюджету користувача.
Сторінка «підбір GPU» зображена на рисунку 4 додатку Е, а її код наведено у додатку П.

Код контролеру, який формує сторінку pickup_view наведено у додатку Ж.

Сторінка «Метрики вибраних GPU» відображає аналітичну інформацію для відеокарт, які користувач попередньо вибрав у системі. Вона надає детальний огляд ключових технічних характеристик, індексів продуктивності та інтегральних метрик, а також візуальні графіки, що дозволяють порівняти GPU між собою. Інтерфейс даної сторінки зображений на рисунку 3.8. Код HTML-сторінки наведений в додатку Ж.

Код контролеру metrics_view, який формує сторінку:

```
def metrics_view(request):
    ids = request.GET.get('ids', '')
    gpu_ids = [int(i) for i in ids.split(',') if i.isdigit()]
```

```
    metrics_qs = FactGPUMetrics.objects.filter(gpu_id__in=gpu_ids).values(
        'gpu_id', 'g3dmark', 'g2dmark', 'price', 'gpu_value', 'tdp',
        'performance_index', 'avg_api_score', 'integrated_score', 'efficiency'
    )

    gpu_names_dict = {g['gpu_id']: g['gpu_name'] for g in
GPU.objects.filter(gpu_id__in=gpu_ids).values('gpu_id', 'gpu_name')}

    metrics_list = []
    for row in metrics_qs:
        new_row = {}
        for k, v in row.items():
            if isinstance(v, Decimal):
                new_row[k] = float(v)
            else:
                new_row[k] = v
        new_row['gpu_name'] = gpu_names_dict.get(row['gpu_id'], f'GPU
{row["gpu_id"]}')
        metrics_list.append(new_row)

    return render(request, 'gpu_app/metrics.html', {
        'metrics_json': json.dumps(metrics_list),
    })
```

Рисунок 3.8 – Інтерфейс сторінки метрик обраних GPU

Сторінка кореляційного аналізу є важливим інструментом системи аналітики GPU, оскільки дозволяє дослідити взаємозв’язки між основними кількісними показниками відеокарт. Вона призначена для статистичного порівняння характеристик, що впливають на продуктивність, енергоефективність, вартість та бенчмарк-показники. Завдяки кореляційному аналізу користувач може зрозуміти, які параметри тісно пов’язані між собою, а які — слабко або зовсім не корелюють.

Метод кореляції Пірсона зазвичай використовується як первинна перевірка зв'язку між двома змінними.

Коефіцієнт кореляції Пірсона є мірою сили лінійного зв’язку між змінними та обчислюється наступним чином (3.1 і 3.2):

$$r = \frac{Cov(x,y)}{\sigma_x \sigma_y} \quad (3.1)$$

де: r — коефіцієнт кореляції Пірсона (безрозмірна величина);

Cov(x, y) — коваріація між змінними x та y, розмірність збігається з добутком одиниць вимірювання змінних (наприклад, м², кг², с² — залежно від величин x і y);

σ_x — середньоквадратичне відхилення змінної xxx (у тих самих одиницях, що й x);

σ_y — середньоквадратичне відхилення змінної «y» (у тих самих одиницях, що й y);

x, y — досліджувані кількісні змінні, значення яких подаються у відповідних системі SI величинах.

$$\sigma_x = \sum (x - \bar{x})^2 \; i \; \sigma_y = \sum (y - \bar{y})^2 \quad (3.2)$$

де: σ_x — середньоквадратичне відхилення змінної xxx, міряється в тих самих одиницях, що і x;

σ_y — середньоквадратичне відхилення змінної «y», міряється в тих самих одиницях, що і y;

$x - \bar{x}$ — відхилення значення від середнього;

$\bar{x}$ — середнє значення змінної xxx;

$\sum$ — оператор підсумовування;

$(x - \bar{x})^2$ — квадрат відхилення.

При переході на сторінку система автоматично завантажує всі доступні метрики GPU з бази даних і формує DataFrame. Далі з нього відбираються лише числові показники, необхідні для обчислення кореляції Пірсона: бенчмарк-бали (G3DMark, G2DMark), ринкова ціна, GPU Value, TDP, середній API Score, індекс продуктивності, інтегральний бал, ефективність та інші агреговані значення. Якщо

дані відсутні або їх недостатньо, система інформує про неможливість побудови матриці.

Кореляційна матриця генерується за допомогою коефіцієнтів Пірсона, що вимірюють силу та напрямок лінійної залежності між параметрами. Отримана таблиця стилізується за допомогою градієнтного підсвічування: від яскраво-червоного для дуже сильних зв'язків ($|r| \geq 0.8$) до світлих тонів для слабких або відсутніх кореляцій. Такий підхід дозволяє користувачеві миттєво ідентифікувати найважливіші залежності, наприклад: чи зростає ціна разом із продуктивністю, наскільки ефективність пов'язана з TDP, чи впливає мінімальний бал API на інтегральний показник.

Візуально матриця розміщується у вигляді інтерактивної таблиці з горизонтальною та вертикальною прокруткою, що забезпечує зручність перегляду навіть при великій кількості параметрів. Діагональні елементи виділені окремим кольором, оскільки відображають абсолютну кореляцію показника із самим собою. Поруч на сторінці розташована легенда кольорів, що пояснює рівні сили кореляції, а також декодування скорочень назви параметрів — це спрощує інтерпретацію даних для користувача. Сторінка із кореляційною матрицею зображена на рисунку 5 додатку Е, а код сторінки наведений в додатку Ж.

Код контролеру, який відтворює сторінку:

```
def correlation_matrix_view(request):
    video_card = VideoCard()
    df = video_card.load_df()
    if df.empty:
        corr_html = "<p class='text-muted'>Нет данных для расчёта корреляции</p>"
    else:
        numeric_cols = [
            'g3dmark', 'g2dmark', 'price', 'gpu_value', 'tdp',
            'performance_index', 'avg_api_score', 'avg_benchmark_score',
            'min_score', 'max_score', 'score_stddev', 'power_performance_ratio',
            'integrated_score', 'efficiency'
        ]

        existing_cols = [col for col in numeric_cols if col in df.columns]
```

```
        corr = df[existing_cols].corr(method='pearson')

        def apply_styles(s):
            # s - это DataFrame с значениями корреляции
            styles = pd.DataFrame('', index=s.index, columns=s.columns)
            for i in range(len(s.index)):
                for j in range(len(s.columns)):
                    if i == j:  # диагональ
                        styles.iloc[i, j] = 'background-color: #f8f9fa; font-weight: bold; color:
#333;'
                    else:
                        val = s.iloc[i, j]
                        abs_val = abs(val)
                        if abs_val >= 0.8:
                            styles.iloc[i, j] = 'background-color: #dc3545; color: white; font-
weight: bold;'
                        elif abs_val >= 0.6:
                            styles.iloc[i, j] = 'background-color: #fd7e14; color: white;'
                        elif abs_val >= 0.4:
                            styles.iloc[i, j] = 'background-color: #ffc107; color: black;'
                        elif abs_val >= 0.2:
                            styles.iloc[i, j] = 'background-color: #d4edda; color: black;'
                        else:
                            styles.iloc[i, j] = 'background-color: #f8f9fa; color: #6c757d;'
            return styles

        styled = corr.style.format("{:.3f}") \
            .apply(apply_styles, axis=None) \
            .set_table_attributes('class="table table-bordered correlation-table"')

        corr_html = styled.to_html()

    return render(request, 'gpu_app/correlation_matrix.html', {
        'corr_matrix_html': corr_html
    })
```

3.7. Висновок до третього розділу

У третьому розділі детально розглянуто процес розроблення та реалізації експертної системи для аналітики відеокарт, починаючи зі створення структури бази даних і завершуючи побудовою інтерактивного веб-інтерфейсу. Було обґрунтовано вибір архітектури сховища даних, проаналізовано роль таблиць вимірів і фактів, а також показано, як вони забезпечують ефективне зберігання, агрегацію та подальшу аналітичну обробку технічних характеристик GPU.

Описано реалізацію ключових компонентів системи — контролера, моделей, шаблонів, а також логіку обробки даних у Django. Показано механізм взаємодії між базою даних та інтерфейсом користувача, включно з автоматичним завантаженням метрик, кластеризацією, візуалізацією та застосуванням фільтрів. Особливу увагу приділено створенню інструментів для порівняння GPU, перегляду метрик, відображення кореляційних зв’язків і побудови графіків, що дозволяє перетворити «сирі» дані на зрозумілу аналітику.

Представлені сторінки експертної системи демонструють комплексний підхід до формування дашборду: інтерактивної таблиці з інформацією про GPU, графічних візуалізацій, категоризації за типами використання, системи вибору моделей та перегляд їхніх детальних метрик. Реалізація кореляційної матриці додатково підсилює аналітичні можливості платформи, дозволяючи виявляти залежності між технічними параметрами відеокарт.

ВИСНОВКИ

У даній кваліфікаційній роботі проведено комплексне дослідження, аналіз і практичну реалізацію експертної системи для здійснення аналітичних задач, класифікації та підбору графічних процесорів (GPU). Метою роботи було створення рішення, яке дозволяє ефективно обробляти великі обсяги технічних даних, виконувати кластеризацію, порівняння та визначення оптимального призначення відеокарт для різних сфер використання.

У роботі реалізовано повноцінну веб-систему на основі фреймворку Django, яка забезпечує зручний та інтуїтивний інтерфейс для перегляду, дослідження та аналітики відеокарт.

Першим елементом є інтерактивна таблиця GPU, яка дозволяє здійснювати пошук, сортування, фільтрацію та порівняння відеокарт. Користувач може обирати кілька моделей для детального аналізу характеристик та продуктивності, використовуючи інтерактивні дашборди, що значно підвищує зручність аналізу і роботу з великими наборами даних.

Другим важливим блоком є модуль аналітики та машинного навчання, у якому реалізовано алгоритми кластеризації K-Means та DBSCAN. Вони використовуються для автоматичного групування відеокарт за подібністю технічних характеристик та поведінкових параметрів. Кластеризація дозволяє виявляти закономірності, сегменти продуктивності та виділяти типові групи GPU, що є основою для подальшого аналізу.

На основі результатів кластерного аналізу визначено до якого цінового сегменту відносяться відеокарти, а також побудовано систему сегментації Usage, яка автоматично визначає рекомендовану сферу застосування відеокарт: Gaming, AI/ML, Graphics/Rendering, Mining або Office.

Окремим модулем системи є побудова кореляційної матриці, яка дозволяє виявити статистичні залежності між основними характеристиками GPU — такими як продуктивність, енергоефективність, API-результати, теплоспоживання тощо.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Каразінський університет. Реалізація інформаційних систем та аналітичних моделей.
URL: https://periodicals.karazin.ua/mia/article/download/24349/22046/ (дата звернення: 21.11.2025).
2. ScienceDirect. Agglomerative Clustering — огляд методів та застосувань. URL: https://www.sciencedirect.com/topics/computer-science/agglomerative-clustering (дата звернення: 21.11.2025).
3. LibreTexts. Kellis M. Gene Regulation I — Clustering algorithms in computational biology.
URL: https://ukrayinska.libretexts.org/.../Алгоритми_кластеризації (дата звернення: 21.11.2025).
4. Донцов М. Д. Матеріали з інформаційних систем. Національний технічний університет України “КПІ ім. Ігоря Сікорського” URL: https://ela.kpi.ua/server/api/core/bitstreams/66da37a6-98b0-4832-8935-60615dc375de/content (дата звернення: 21.11.2025).

5. ДСТУ 3918:1999 (ISO/IEC 12207:2008). Інформаційні технології. Процеси життєвого циклу програмного забезпечення. – 57 с.

6. GeeksForGeeks. Divisive Clustering — концепції та приклади. URL: https://www.geeksforgeeks.org/artificial-intelligence/divisive-clustering/ (дата звернення: 21.11.2025).
7. Wikipedia. DBSCAN — алгоритм просторової кластеризації. URL: https://uk.wikipedia.org/wiki/DBSCAN (дата звернення: 21.11.2025).
8. GitConnected. Gaussian Mixture Models — Model-based clustering. URL: https://levelup.gitconnected.com/model-based-clustering-using-gmm-gaussian-mixture-models-d985ffead947 (дата звернення: 21.11.2025).
9. Wikipedia. Відеокарта — архітектура та класифікація. URL: https://uk.wikipedia.org/wiki/Відеокарта (дата звернення: 21.11.2025).

10. TheHost. Інструкція з установки PostgreSQL у Linux. URL: https://thehost.ua/ua/wiki/administration/database/postgresql-install (дата звернення: 21.11.2025).

11. Wikipedia. Сховище даних — концепції та призначення. URL: https://uk.wikipedia.org/wiki/Сховище_даних (дата звернення: 21.11.2025).

12. JavaRush. Для чого використовується Python: огляд застосувань. URL: https://javarush.com/ua/groups/posts/dlja-chogo-vikoristovutjhsja-python (дата звернення: 21.11.2025).

13. Mustafa H., Leal E., Gruenwald L. An Experimental Comparison of GPU Techniques for DBSCAN Clustering. University of Minnesota Duluth, University of Oklahoma, 2019. (день звернення: 21.11.2025) URL: https://www.cs.ou.edu/~database/HIGEST-DB/publications/BPOD%202019.pdf

14. Prokopenko A., Lebrun-Grandie D., Arndt D. Fast tree-based algorithms for DBSCAN for low-dimensional data on GPUs. arXiv, 2021. URL: https://arxiv.org/abs/2103.05162 (дата звернення: 21.11.2025)

15. Ersoy P., Erşahin M., Erşahin B. The Comparative Effects of Clustering Algorithms on CPU and GPU. Artificial Intelligence Theory and Applications, Vol. 2, Issue 2, 2022, pp. 19–27. URL: https://dergipark.org.tr/en/pub/aita/issue/72862/1141500 (дата звернення: 21.11.2025)

16. Baligodugula V. V., Amsaad F. Unsupervised Learning: Comparative Analysis of Clustering Techniques on High-Dimensional Data (K-Means, DBSCAN, Spectral). arXiv, 2025. URL: https://arxiv.org/abs/2503.23215 (дата звернення: 21.11.2025)

17. Kotelevets K. A. Дослідження методів кластеризації геоданих з використанням дискретизації (K-Means, Spectral, DBSCAN, OPTICS). NURE, 2024. URL: https://openarchive.nure.ua/entities/publication/8280dece-2024-4e5b-8119-cce9986ac199 (дата звернення: 21.11.2025)

18. Tkachenko A., Kyrychenko L., Radyvylova T. Кластеризація зашумлених часових рядів: методи K-Means і DBSCAN. Системні технології, 2019. DOI: https://doi.org/10.34185/1562-9945-3-122-2019-15 (дата звернення: 21.11.2025)

19. Бойко Б., Процик І. Використання алгоритмів кластеризації для сегментації персоналу компанії. Herald of Khmelnytskyi National University. Technical Sciences, 2024. DOI: https://doi.org/10.31891/2307-5732-2024-333-2-14 (дата звернення: 21.11.2025)

20. Гавриленко О., Чимшир В., Жаріков Е., Теленик С., Амонс О. Метод формування пакетів сервісів за допомогою алгоритмів кластеризації (зокрема DBSCAN). Адаптивні системи автоматичного управління, 2024. DOI: https://doi.org/10.20535/1560-8956.44.2024.302437 (дата звернення: 21.11.2025)

21. Gómez-Luna J., Guo Y., Brocard S. та ін. An Experimental Evaluation of Machine Learning Training on a Real Processing-in-Memory System (включно з K-Means на GPU). arXiv, 2022. URL: https://arxiv.org/abs/2207.07886 (дата звернення: 21.11.2025)

22. Palii V. Магістерська робота: порівняння K-Means та DBSCAN, ієрархічної та м'якої кластеризації. Національний технічний університет України, КПІ, 2024. URL: https://ela.kpi.ua/bitstream/123456789/57970/1/Paliy_magistr.pdf (дата звернення: 21.11.2025)

23. LeCun Y., Bottou L., Orr G. B., Müller K.-R. Efficient BackProp. Lecture Notes in Computer Science, 2012. DOI: https://doi.org/10.1007/978-3-642-35289-8_3 (дата звернення: 21.11.2025)

24. Sara Daoudi. Parallelization of the K-Means++ Clustering Algorithm (GPU реалізація). IIETA. URL: https://www.iieta.org/journals/isi/paper/10.18280/isi.260106 (дата звернення: 21.11.2025)

25. LevelUp GitConnected. Model-based clustering using GMM / Gaussian Mixture Models. 2023. URL: https://levelup.gitconnected.com/model-based-clustering-using-gmm-gaussian-mixture-models-d985ffead947 (дата звернення: 21.11.2025)

26. LNU (Львівський національний університет) / MMF. Машинне навчання простими словами: кластеризація (K-Means, DBSCAN). 2024. URL: https://mmf.lnu.edu.ua/ar/1739 (дата звернення: 21.11.2025)

27. MDPI. Evaluation of Clustering Algorithms on GPU-Based Edge Computing Platforms (K-Means, FCM на GPU). SENSORS, 2020. URL: https://www.mdpi.com/1424-8220/20/21/6335 (дата звернення: 21.11.2025)

28. КПІ. Паламдієв О., Лісовиченко О. Тривимірні нейронні мережі у задачах кластеризації. Адаптивні системи, 2024. DOI: https://doi.org/10.20535/1560-8956.44.2024.302431 (дата звернення: 21.11.2025)

29. ДСТУ ISO/IEC 12207:2014. Інженерія систем і програмного забезпечення. Процеси життєвого циклу програмного забезпечення.

30. ДСТУ ISO/IEC 15910:2012. Інформаційні технології. Документування програм. Документація користувача.

31. ДСТУ 2226:1993. Автоматизовані системи.

32. Poudel M., Gowanlock M. CUDA-DClust+: Revisiting Early GPU-Accelerated DBSCAN Clustering Designs. 2021. URL: https://jan.ucc.nau.edu/mg2745/publications/Poudel_Gowanlock_HiPC2021.pdf (дата звернення: 21.11.2025)

33. Nagarajan V., Kulkarni M. RT-DBSCAN: Accelerating DBSCAN using Ray Tracing Hardware. arXiv, 2023. URL: https://arxiv.org/abs/2303.09655 (дата звернення: 21.11.2025)

34. Wu G., Cao L., Tian H., Wang W. HY-DBSCAN: A hybrid parallel DBSCAN clustering algorithm scalable on distributed-memory computers. Journal of Parallel and Distributed Computing, 2022. DOI: 10.1016/j.jpdc.2022.06.005 (дата звернення: 21.11.2025)

35. Кіндзерський О. В. Методи та програмні засоби кластеризації даних на основі технології Nvidia CUDA. Магіст. дисертація, КПІ, 2018. (дата звернення: 21.11.2025) URL: https://ela.kpi.ua/handle/123456789/23820

ДОДАТКИ

Додаток А. Організаційна структура підприємства ТОВ «ТІСЕР»

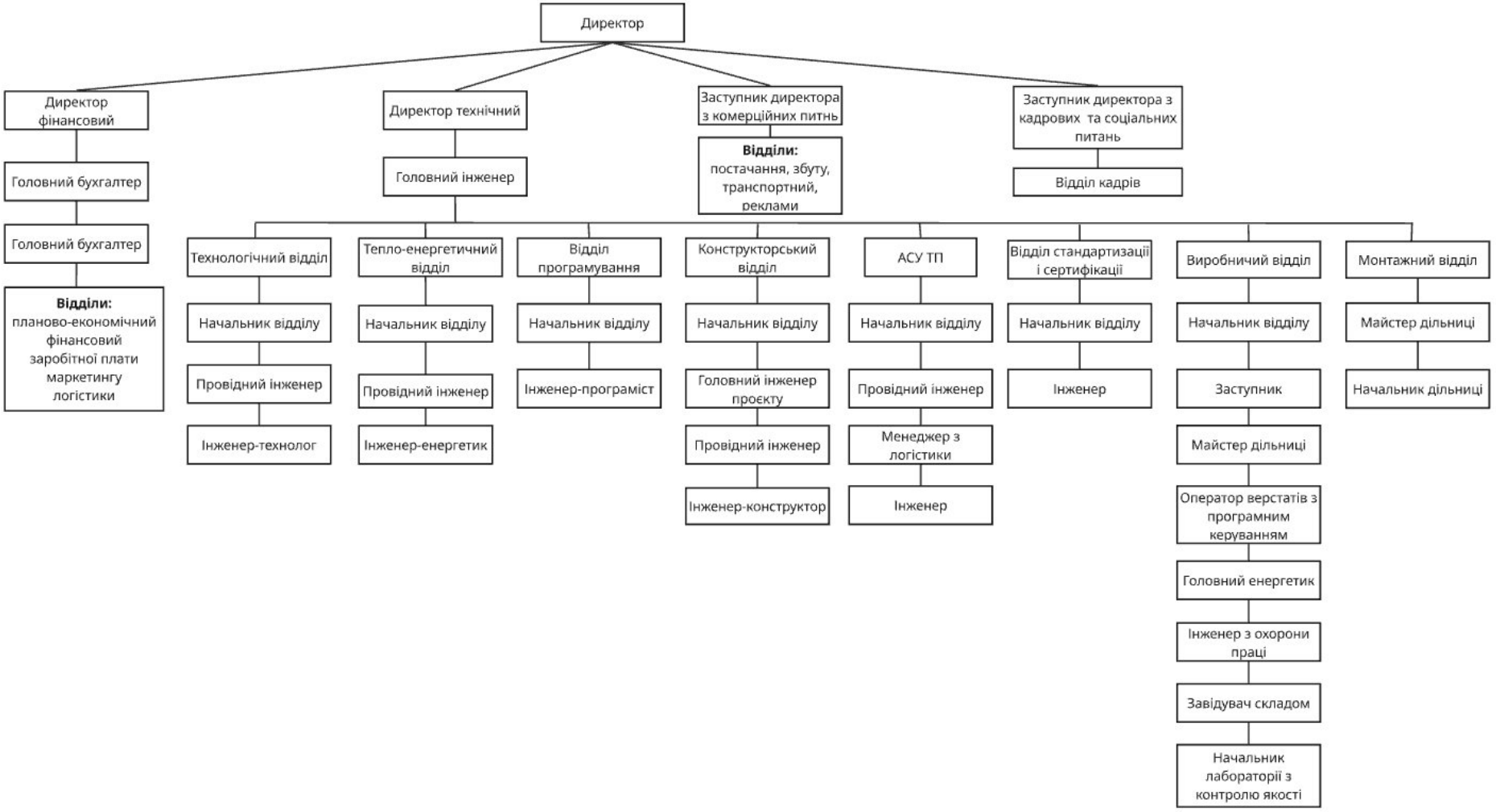


Рисунок А.1 – Організаційна структура підприємства ТОВ «ТІСЕР»

Додаток Б. Порівняння існуючих технологій підбору обладнання компонентів ПК

Таблиця Б.1 - Порівняння існуючих технологій підбору обладнання компонентів ПК

Система / Сервіс	Основні функції	Можливість інтеграції	Мова інтерфейсу	Особливості	Переваги	Недоліки
PCPartPicker	Підбір комплектуючих, перевірка сумісності, порівняння цін	Обмежена (API для магазинів)	Англійська	Орієнтована на індивідуальних користувачів	Простий інтерфейс, швидкий підбір сумісних комплектуючих, відстеження цін	Не підходить для корпоративного використання, відсутність інтеграції з внутрішніми системами, не враховує специфіку бізнес-потреб
SpecOps Hardware Planner	Облік обладнання, планування апгрейдів, формування бюджетів	Так, інтеграція з ERP	Англійська	Корпоративне рішення, підходить для великих компаній	Підтримка централізованого обліку обладнання, прогнозування потреб, інтеграція з ERP	Орієнтований на облік, а не на аналітику продуктивності та порівняння моделей, складне налаштування
HP DaaS	Рекомендації по заміні пристроїв, автоматичні замовлення	Так, інтеграція з сервісами HP	Англійська	SaaS-модель, підтримка лише пристроїв HP	Автоматизація замовлень, моніторинг стану пристроїв, спрощене управління корпоративним парком	Працює тільки з пристроями HP, не дозволяє аналізувати продуктивність різних відеокарт, обмежена гнучкість для корпоративних потреб

Додаток В. Порівняння існуючих методів кластеризації

Таблиця В.1 - Порівняння існуючих методів кластеризації

Алгоритм	Переваги	Недоліки	Застосування у системі підбору GPU
K-Means	Швидкий і зрозумілий результат; добре працює з компактними кластерами однакової форми	Чутливий до вибору початкових центроїдів; погано працює з перекритими кластерами	Сегментація GPU за продуктивністю, ціною, показником GPU Value
K-Means++	Усуває чутливість K-Means до початкових центроїдів	Більш складний старт, все одно потребує задання кількості кластерів	Точніше визначення центроїдів для GPU
Mini-Batch K-Means	Швидкий для великих наборів даних, економія пам’яті	Може знизити точність кластерів	Попередня кластеризація великих наборів GPU
Agglomerative Clustering	формування ієрархії кластерів, немає потреби задавати кількість кластерів	Обчислювально витратний, чутливий до шуму та метрики відстані	Вивчення ієрархії GPU, виділення підгруп продуктивності та енергоефективності
Divisive Clustering	Початковий великий кластер розбивається на підкласи, зручний для структурованих даних	Рідко використовується на великих наборах даних, обчислювально складний	Розбиття широких категорій GPU на детальні групи
DBSCAN	Виявляє кластери будь-якої форми, стійкий до шуму, не потребує заданої кількості кластерів	Погано працює з даними різної щільності, чутливий до параметрів eps і minPts	Виявлення груп GPU за високою щільністю характеристик, ігнорування викидів
HDBSCAN	Робота з різнорідними кластерами, автоматичне визначення їх кількості, стійкий до шуму	Складний у налаштуванні, повільний на дуже великих наборах	Аналіз складних структур GPU
Spectral Clustering	Виявлення складних форм кластерів, ефективний для даних з топологічними зв’язками	Вимагає побудови матриці суміжності, обчислювально витратний	Виявлення природних груп GPU за схожістю
Louvain	Виявлення спільнот у великих мережах, ефективний для багаторівневих структур	Обмежений застосунок поза графами	Аналіз зв’язків між GPU (наприклад, за сумісністю або API)
Girvan-Newman	Чітке відокремлення груп у графі, детальна ієрархія	Дуже обчислювально затратний для великих мереж	Виявлення груп GPU за специфічними зв’язками
Gaussian Mixture Models (GMM)	Гнучка модель, дозволяє класифікувати об’єкти з імовірностями належності, добре працює з перекритими кластерами	Чутливий до ініціалізації, може застрягати у локальних мінімумах, вимагає задання кількості компонент	Аналіз GPU із плавними переходами між групами продуктивності або ціновими сегментами

Додаток Г. Вимоги до відеокарт за сферами їх застосування

Таблиця Г.1 - Порівняння вимог до відеокарт за сферами їх застосування

Сфера застосування	Продуктивність (G3DMark, FPS)	Відеопам'ять	Енергоспоживання (TDP)	Паралельні обчислення / CUDA ядра	Ціновий сегмент	Особливості / додаткові функції
Офісні задачі	Низька	2–4 GB	Дуже низьке	Невелика кількість	Бюджетний	Інтегрована графіка або бюджетні GPU; підтримка декодування відео; енергоефективність
Ігрові	Середня–висока (для AAA ігор 60+ FPS)	6–12 GB	Середнє–високе	Висока кількість	Середній	Підтримка DirectX 12, Vulkan, апаратне трасування променів, високі частоти кадрів
AI / ML	Висока (обчислення тензорів, тренування нейронних мереж)	12–48 GB	Високе	Дуже висока кількість (Tensor Cores / CUDA ядра)	Високий / преміум	Паралельна обробка матриць, підтримка FP16/BF16, CUDA, OpenCL, Metal
Графіка / Рендеринг	Висока	16–24 GB	Високе	Висока кількість	Високий / преміум	Підтримка трасування променів, професійні драйвери, апаратне прискорення рендеру (RT / Tensor Cores), стабільність при довгих сесіях
Майнінг	Середня (ефективність хеш-обчислень)	6–12 GB	Середнє	Середня кількість	Середній	Оптимізація співвідношення продуктивність/енергія, ефективне охолодження, стабільна робота 24/7

Додаток Д. Схема бази і сховища даних експертної системи

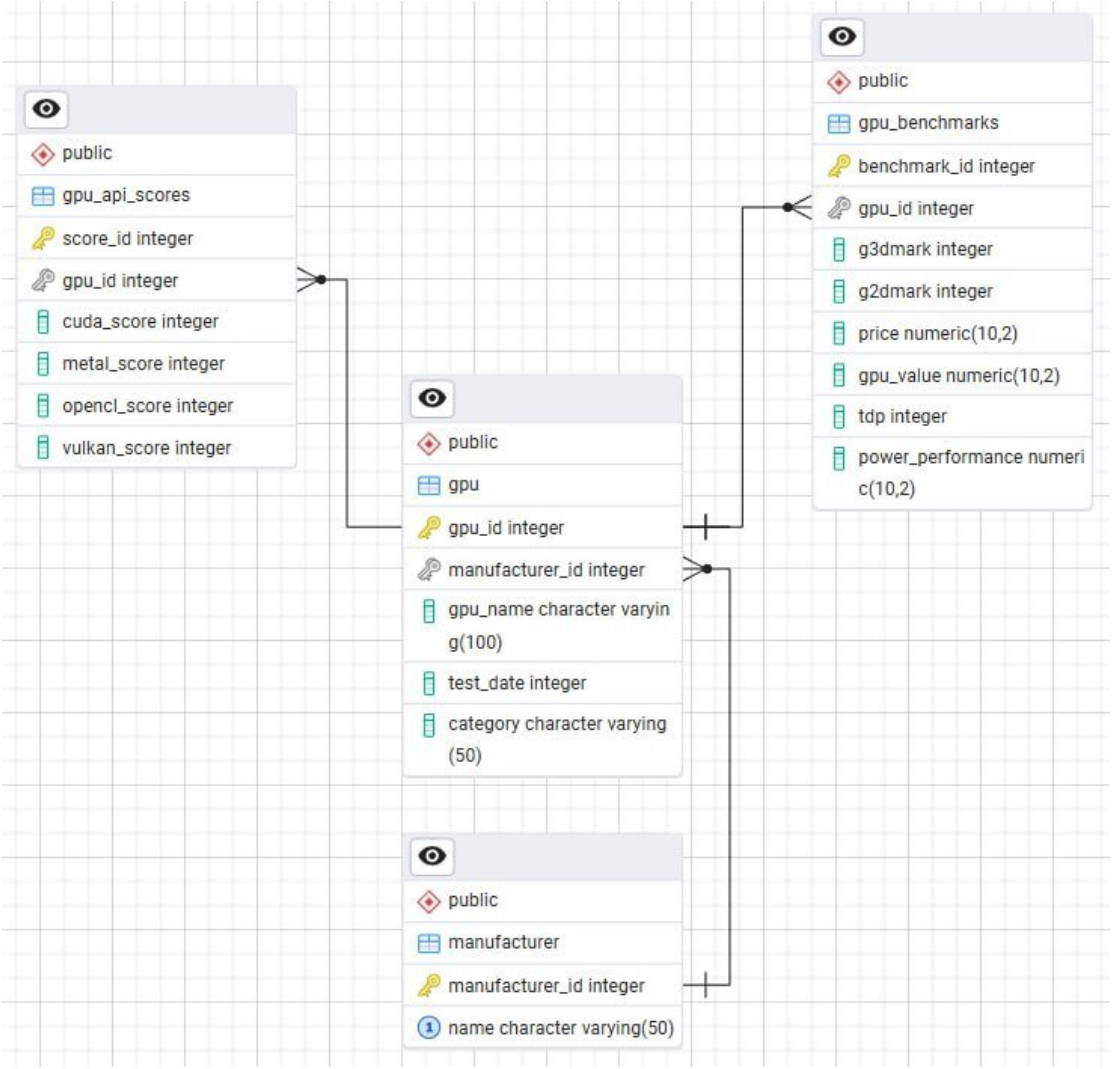


Рисунок Д.1 – Схема бази даних експертної системи

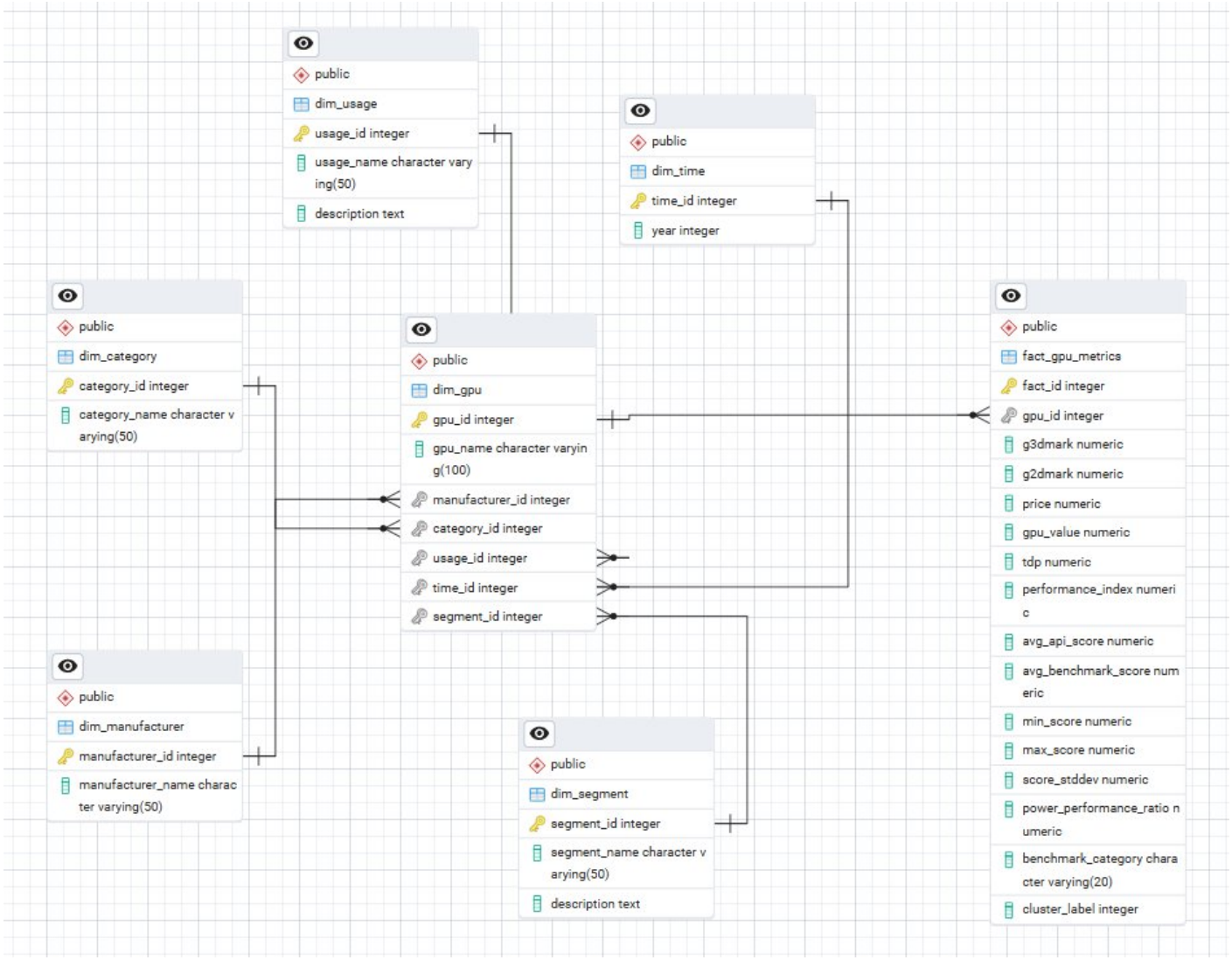


Рисунок Д.2 – Схема сховища даних експертної системи

Додаток Е. Інтерфейс експертної системи

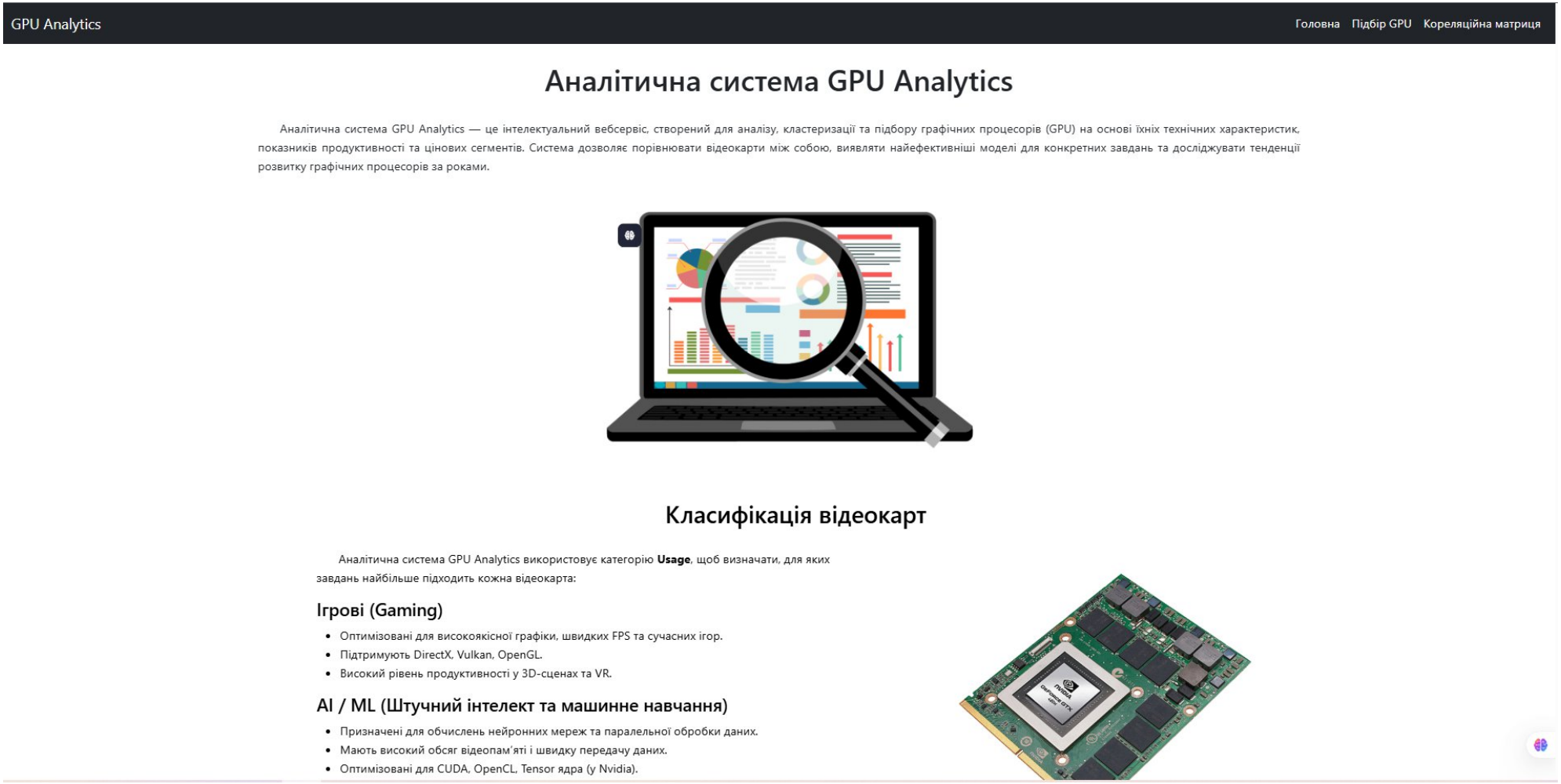


Рисунок Е.1 – Початок головної сторінки

AI / ML (Штучний інтелект та машинне навчання)

- Призначені для обчислень нейронних мереж та паралельної обробки даних.
- Мають високий обсяг відеопам'яті і швидку передачу даних.
- Оптимізовані для CUDA, OpenCL, Tensor ядра (у Nvidia).

Графіка / Рендер (Graphics / Rendering)

- Для роботи із CAD, 3D-моделюванням, рендерингом відео та анімації.
- Висока продуктивність у OpenCL, Metal та Vulkan.
- Підходять для професійних дизайнерів, архітекторів, студій анімації.

Майнінг (Mining)

- Оптимізовані для обчислень блокчейну та криптовалют.
- Має високу ефективність обчислень на одиницю споживаної енергії.
- Може не мати високої графічної продуктивності для ігор або рендерингу.

Офісні (Office / Low-power)

- Для базових завдань: офісні програми, браузери, відеоконференції.
- Низьке енергоспоживання, невисока продуктивність.
- В основному інтегровані GPU у CPU або бюджетні дискретні карти.

Цінові сегменти відеокарт

Крім класифікації за призначенням, відеокарти поділяються на різні цінові сегменти. У нашій системі можна легко визначити, які моделі бюджетні, середнього класу чи преміальні. Це дозволяє обрати GPU, що оптимально підходить за ціною та продуктивністю для ваших завдань.

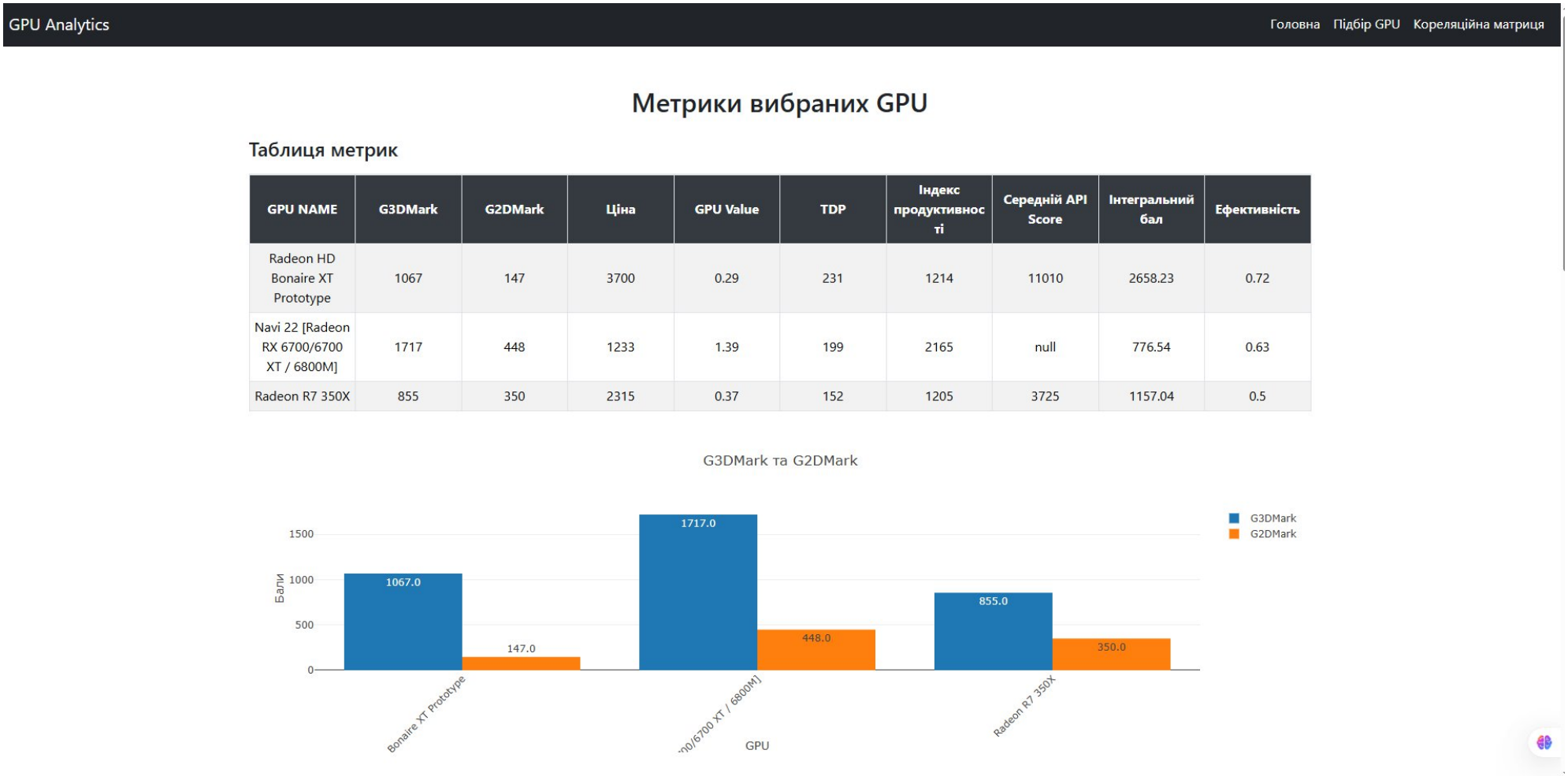
Від економних рішень до топових моделей – ми показуємо весь спектр можливостей.

Відеокарти (Nvidia and AMD)

Показати 20 записів Пошук: Пошук

	Назва	Виробник	Категорія	Призначення	Ціна
☐	Radeon HD Bonaire XT Prototype	AMD	Mobile	Офісна	3700.00
☐	GeForce 940M	Nvidia	Mobile	Офісна	3083.00
☐	GeForce 820A	Nvidia	Desktop	Офісна	2287.00
☐	Radeon HD Chelsea PRO Prototype	AMD	Unknown	Офісна	4966.00
☐	GRID T4-4Q	Nvidia	Desktop	Офісна	3607.00
☐	RADV POLARIS10 (LLVM 7.0.0)	AMD	Unknown	Офісна	4479.00
☐	Radeon Pro 455	AMD	Unknown	Офісна	4675.00
☐	Radeon Pro 5300	AMD	Unknown	Офісна	2897.00
☐	Radeon Pro W5500	AMD	Unknown	Офісна	1391.00
☐	Radeon HD Verde XT Prototype	AMD	Unknown	Офісна	3227.00

Рисунок Е.2 – Кінець головної сторінки



GPU NAME	G3DMark	G2DMark	Ціна	GPU Value	TDP	Індекс продуктивності	Середній API Score	Інтегральний бал	Ефективність
Radeon HD Bonaire XT Prototype	1067	147	3700	0.29	231	1214	11010	2658.23	0.72
Navi 22 [Radeon RX 6700/6700 XT / 6800M]	1717	448	1233	1.39	199	2165	null	776.54	0.63
Radeon R7 350X	855	350	2315	0.37	152	1205	3725	1157.04	0.5

Рисунок Е.3 – Сторінка метрик відеокарт

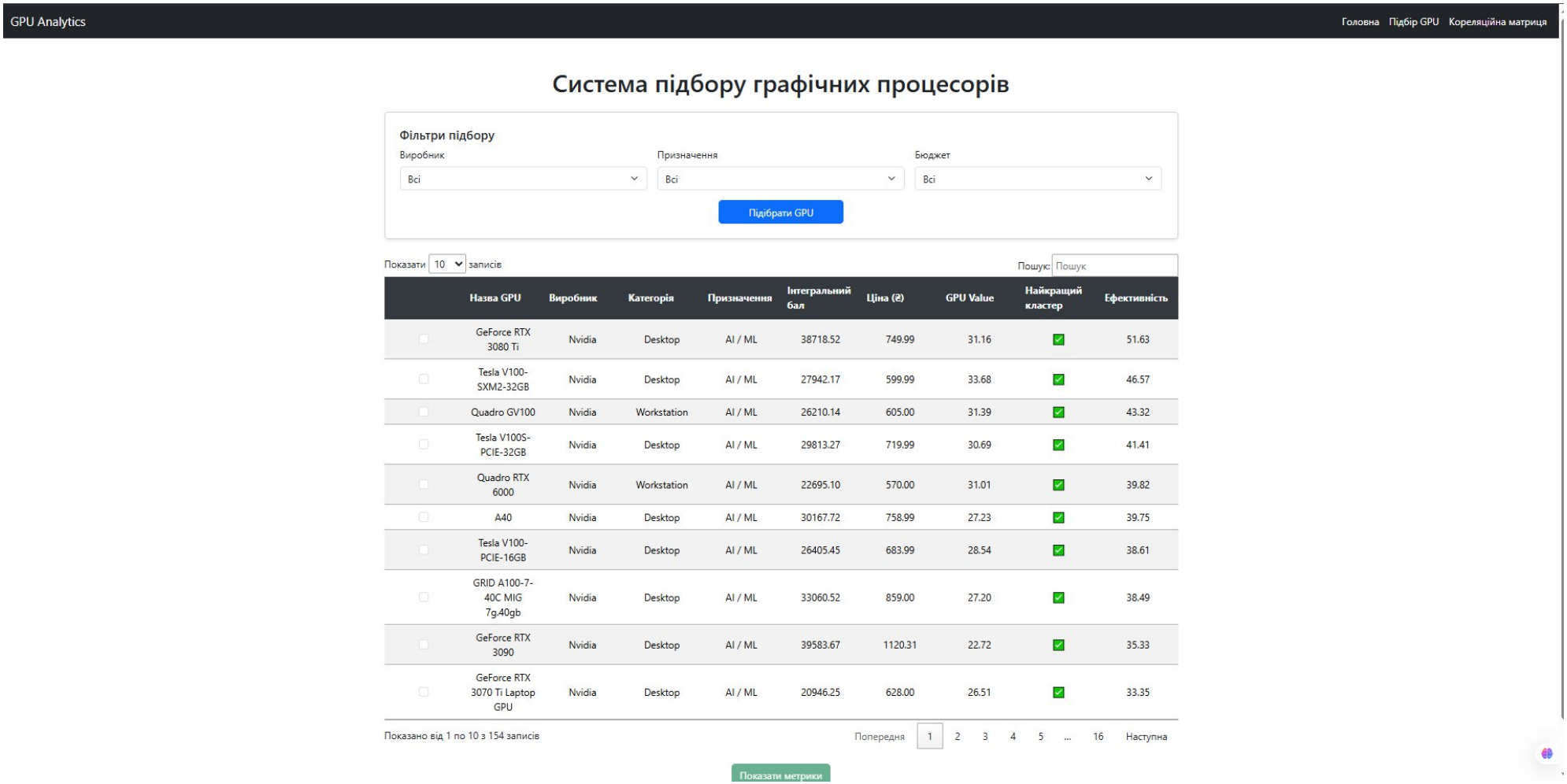


Рисунок Е.4 – Сторінка «підбір GPU»

	g3dmark	g2dmark	price	gpu_value	tdp	performan...	avg_api_sc...	avg_bench...	min_score	max_score	score_stddev	power_per...	integrated...	efficiency
g3dmark	1.000	0.290	-0.080	0.758	0.255	0.996	0.804	0.999	0.686	0.719	0.697	0.769	0.912	0.670
g2dmark	0.290	1.000	-0.040	0.303	0.020	0.214	0.213	0.322	0.339	0.133	0.124	0.179	0.272	0.190
price	-0.080	-0.040	1.000	-0.381	-0.038	-0.079	-0.077	-0.081	-0.065	-0.082	-0.083	-0.012	-0.082	-0.390
gpu_value	0.758	0.303	-0.381	1.000	0.246	0.747	0.680	0.760	0.517	0.621	0.612	0.502	0.742	0.943
tdp	0.255	0.020	-0.038	0.246	1.000	0.257	0.310	0.252	0.153	0.322	0.305	-0.224	0.304	0.263
performan...	0.996	0.214	-0.079	0.747	0.257	1.000	0.798	0.993	0.708	0.720	0.697	0.774	0.904	0.666
avg_api_sc...	0.804	0.213	-0.077	0.680	0.310	0.798	1.000	0.801	0.499	0.910	0.943	0.498	0.977	0.762
avg_bench...	0.999	0.322	-0.081	0.760	0.252	0.993	0.801	1.000	0.706	0.715	0.693	0.769	0.910	0.669
min_score	0.686	0.339	-0.065	0.517	0.153	0.708	0.499	0.706	1.000	0.453	0.423	0.588	0.592	0.449
max_score	0.719	0.133	-0.082	0.621	0.322	0.720	0.910	0.715	0.453	1.000	0.951	0.399	0.883	0.707
score_stddev	0.697	0.124	-0.083	0.612	0.305	0.697	0.943	0.693	0.423	0.951	1.000	0.395	0.899	0.718
power_per...	0.769	0.179	-0.012	0.502	-0.224	0.774	0.498	0.769	0.588	0.399	0.395	1.000	0.617	0.391
integrated...	0.912	0.272	-0.082	0.742	0.304	0.904	0.977	0.910	0.592	0.883	0.899	0.617	1.000	0.765
efficiency	0.670	0.190	-0.390	0.943	0.263	0.666	0.762	0.669	0.449	0.707	0.718	0.391	0.765	1.000

Рисунок Е.5 – Сторінка «Корелаційна матриця»

Додаток Ж. – Програмний код створення усіх моделей ORM

```
from django.db import models


class Manufacturer(models.Model):
    manufacturer_id = models.IntegerField(primary_key=True)
    manufacturer_name = models.CharField(max_length=50)

    class Meta:
        managed = False
        db_table = 'dim_manufacturer'

class Category(models.Model):
    category_id = models.IntegerField(primary_key=True)
    category_name = models.CharField(max_length=50)

    class Meta:
        managed = False
        db_table = 'dim_category'

class Usage(models.Model):
    usage_id = models.IntegerField(primary_key=True)
    usage_name = models.CharField(max_length=50)
    description = models.TextField(blank=True, null=True)

    class Meta:
        managed = False
        db_table = 'dim_usage'

class Time(models.Model):
    time_id = models.IntegerField(primary_key=True)
    year = models.IntegerField()

    class Meta:
        managed = False
        db_table = 'dim_time'

class Segment(models.Model):
    segment_id = models.AutoField(primary_key=True)
```

```
    segment_name = models.CharField(max_length=50)

    class Meta:
        managed = False
        db_table = 'dim_segment'

class GPU(models.Model):
    gpu_id = models.IntegerField(primary_key=True)
    gpu_name = models.CharField(max_length=100)
    manufacturer = models.ForeignKey(Manufacturer, on_delete=models.CASCADE)
    category = models.ForeignKey(Category, on_delete=models.CASCADE)
    usage = models.ForeignKey(Usage, on_delete=models.SET_NULL, null=True,
blank=True)
    time = models.ForeignKey(Time, on_delete=models.CASCADE)
    segment = models.ForeignKey(Segment, on_delete=models.SET_NULL, null=True,
blank=True)

    class Meta:
        managed = False
        db_table = 'dim_gpu'

class FactGPUMetrics(models.Model):
    fact_id = models.IntegerField(primary_key=True)
    gpu = models.ForeignKey(GPU, on_delete=models.CASCADE,
related_name='factgpumetrics_set')
    g3dmark = models.DecimalField(max_digits=10, decimal_places=2)
    g2dmark = models.DecimalField(max_digits=10, decimal_places=2)
    price = models.DecimalField(max_digits=10, decimal_places=2)
    gpu_value = models.DecimalField(max_digits=10, decimal_places=2)
    tdp = models.DecimalField(max_digits=10, decimal_places=2)
    performance_index = models.DecimalField(max_digits=10, decimal_places=2)
    avg_api_score = models.DecimalField(max_digits=10, decimal_places=2)
    avg_benchmark_score = models.DecimalField(max_digits=10, decimal_places=2)
    min_score = models.DecimalField(max_digits=10, decimal_places=2)
    max_score = models.DecimalField(max_digits=10, decimal_places=2)
    score_stddev = models.DecimalField(max_digits=10, decimal_places=2)
    power_performance_ratio = models.DecimalField(max_digits=10, decimal_places=2)
    integrated_score = models.DecimalField(max_digits=10, decimal_places=2)
```

```
    kmeans_label = models.IntegerField(null=True, blank=True)
    is_best_cluster = models.BooleanField(default=False)
    efficiency = models.FloatField(null=True, blank=True,
verbose_name="Ефективність")

    class Meta:
        managed = False
        db_table = 'fact_gpu_metrics'
```

Програмний код, який є базовим для усіх сторінок

```
{% load static %}
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>{% block title %}Аналітична система GPU{% endblock %}</title>
  <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css">
  <link rel="stylesheet" href="{% static 'gpu_app/css/styles.css' %}">
  <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script src="https://cdn.datatables.net/1.13.6/js/jquery.dataTables.min.js"></script>
  <link rel="stylesheet"
href="https://cdn.datatables.net/1.13.6/css/jquery.dataTables.min.css">
</head>
<body>

<!-- Header -->
<nav class="navbar navbar-expand-lg navbar-dark bg-dark mb-4">
  <div class="container-fluid">
    <a class="navbar-brand" href="{% url 'main' %}">GPU Analytics</a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-
target="#navbarNav"
        aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle
navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarNav">
      <ul class="navbar-nav ms-auto">
        <li class="nav-item"><a class="nav-link" href="{% url
'main' %}">Головна</a></li>
        <li class="nav-item"><a class="nav-link" href="{% url 'pickup' %}">Підбір
GPU</a></li>
        <li class="nav-item"><a class="nav-link" href="{% url
'correlation_matrix' %}">Кореляційна матриця</a></li>
      </ul>
```

```
        </div>
    </div>
</nav>
<div class="container-fluid">
    {% block content %}{% endblock %}
</div>
<footer class="footer mt-auto py-3 bg-light text-center">
    <div class="container">
        <span class="text-muted">&copy; 2025 ТОВ "ТІСЕР" — Аналітична система підбору GPU</span>
    </div>
</footer>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"></script>
{% block scripts %}
<script>
$(document).ready(function() {
    // Инициализация DataTable
    if ($('#gpu-table').length && !$.fn.DataTable.isDataTable('#gpu-table')) {
        $('#gpu-table').DataTable({
            paging: true,
            pageLength: 20,
            lengthMenu: [10, 20, 50, 100],
            searching: true,
            ordering: false,
            info: true,
            language: {
                url: "//cdn.datatables.net/plug-ins/1.13.6/i18n/uk.json"
            }
        });
    }
    const showBtn = document.getElementById('show-metrics');
    if (showBtn) {
        $(document).on('change', '.gpu-checkbox', function() {
            const selected = $('.gpu-checkbox:checked');
            showBtn.disabled = selected.length === 0 || selected.length > 10;
```

```
            if (selected.length > 10) {
                alert("Можна вибрати максимум 10 відеокарт!");
                $(this).prop('checked', false);
            }

            console.log("Checked IDs:", selected.map((_, el) => el.value).get());
        });

        $('#show-metrics').click(function() {
            const selectedIds = $('.gpu-checkbox:checked').map(function() {
                return $(this).val();
            }).get();
            console.log("Sending IDs:", selectedIds);

            if (selectedIds.length > 0) {
                const url = "{% url 'metrics_view' %}?ids=" + selectedIds.join(',');
                window.location.href = url;
            }
        });
    }
});
</script>
{% endblock %}
</body>
</html>
```

Програмний код, який відповідає за головний механізм підбору відеокарт

```
def culc(self):
    df = self.load_df()
    if df.empty:
        return

    df['usage_name'] = df.apply(lambda row: self.determine_usage(row), axis=1)
    df['performance_index'] = df['g3dmark'] + df['g2dmark']
    df['avg_api_score'] = df[['cuda_score', 'metal_score', 'opencl_score',
'vulkan_score']].mean(axis=1)
    df['avg_benchmark_score'] = df[['g3dmark', 'g2dmark']].mean(axis=1)
    df['min_score'] = df[['g3dmark', 'g2dmark', 'cuda_score', 'metal_score', 'opencl_score',
'vulkan_score']].min(
        axis=1)
    df['max_score'] = df[['g3dmark', 'g2dmark', 'cuda_score', 'metal_score', 'opencl_score',
'vulkan_score']].max(
        axis=1)
    df['score_stddev'] = df[
        ['g3dmark', 'g2dmark', 'cuda_score', 'metal_score', 'opencl_score',
'vulkan_score']].std(axis=1)
    df['power_performance_ratio'] = df.apply(
        lambda row: (row['g3dmark'] + row['g2dmark']) / row['tdp'] if row['tdp'] else 0,
axis=1)
    # Інтегральний бал та ефективність
    df['integrated_score'] = df['g3dmark'] * 0.4 + df['g2dmark'] * 0.2 + df['gpu_value'] *
0.1 + df[
        'avg_api_score'] * 0.2
    df['efficiency'] = df['integrated_score'] / df['price'].replace(0, pd.NA)
    df['efficiency'] = df['efficiency'].fillna(0)
    if len(df) >= 3:
        kmeans_price = KMeans(n_clusters=3, random_state=42, n_init=10)
        df['price_cluster'] = kmeans_price.fit_predict(df[['price']])
        cluster_means = df.groupby('price_cluster')['price'].mean().sort_values()
        cluster_to_segment = {cluster_means.index[0]: 'low', cluster_means.index[1]: 'mid',
                              cluster_means.index[2]: 'high'}
        df['segment_name'] = df['price_cluster'].map(cluster_to_segment)
        df['segment_id'] = df['segment_name'].map(self.SEGMENT_MAP)
```

```
    else:
        df['segment_name'] = 'mid'
        df['segment_id'] = self.SEGMENT_MAP['mid']

    features = ['g3dmark', 'g2dmark', 'price', 'gpu_value', 'tdp',
                'performance_index', 'avg_api_score', 'avg_benchmark_score',
                'min_score', 'max_score', 'score_stddev', 'power_performance_ratio']
    if len(df) >= 3:
        scaler = StandardScaler()
        X_scaled = scaler.fit_transform(df[features])
        dbscan = DBSCAN(eps=1.0, min_samples=2)
        df['dbscan_label'] = dbscan.fit_predict(X_scaled)
        df_clean = df[df['dbscan_label'] != -1].copy()
        if len(df_clean) >= 3:
            n_clusters = min(3, len(df_clean))
            kmeans = KMeans(n_clusters=n_clusters, random_state=42, n_init=10)
            df_clean['kmeans_label'] =
kmeans.fit_predict(scaler.transform(df_clean[features]))
            cluster_eff = df_clean.groupby('kmeans_label')['efficiency'].mean()
            best_cluster = cluster_eff.idxmax() if not cluster_eff.empty else 0
            df_clean['is_best_cluster'] = df_clean['kmeans_label'] == best_cluster
            # Оновлюємо DataFrame
            for _, row in df_clean.iterrows():
                df.loc[df['gpu_id'] == row['gpu_id'], ['kmeans_label', 'is_best_cluster']] = [
                    int(row['kmeans_label']), bool(row['is_best_cluster'])]
        else:
            df['kmeans_label'] = 0
            df['is_best_cluster'] = False
    else:
        df['kmeans_label'] = 0
        df['is_best_cluster'] = False

    usage_map = {u.usage_name: u for u in Usage.objects.all()}
    facts_to_update = []
    for _, row in df.iterrows():
        gpu_obj = row['gpu']
        db_usage_name = self.USAGE_DB_MAP.get(row['usage_name'],
row['usage_name'])
```

```
    usage_obj = usage_map.get(db_usage_name)
    if usage_obj:
        gpu_obj.usage = usage_obj
    gpu_obj.segment_id = row['segment_id']
    gpu_obj.save(update_fields=['usage', 'segment_id'])

    fact, _ = FactGPUMetrics.objects.get_or_create(gpu=gpu_obj)
    fact.integrated_score = round(row['integrated_score'], 2)
    fact.efficiency = round(row['efficiency'], 2)
    fact.kmeans_label = int(row.get('kmeans_label', 0))
    fact.is_best_cluster = bool(row.get('is_best_cluster', False))
    fact.performance_index = row['performance_index']
    fact.avg_api_score = row['avg_api_score']
    fact.avg_benchmark_score = row['avg_benchmark_score']
    fact.min_score = row['min_score']
    fact.max_score = row['max_score']
    fact.score_stddev = row['score_stddev']
    fact.power_performance_ratio = row['power_performance_ratio']
    facts_to_update.append(fact)

FactGPUMetrics.objects.bulk_update(facts_to_update, [
    'integrated_score', 'efficiency', 'kmeans_label', 'is_best_cluster',
    'performance_index', 'avg_api_score', 'avg_benchmark_score',
    'min_score', 'max_score', 'score_stddev', 'power_performance_ratio'
])
```

Програмний код, який відповідає за формування сторінки підбору відеокарт

```
{% extends 'gpu_app/base.html' %}
{% block title %}Підбір GPU — GPU Analytics{% endblock %}
{% block content %}
<div class="container mt-4">
  <h1 class="text-center mb-4">Система підбору графічних процесорів</h1>
  <div class="card p-4 mb-4 shadow-sm">
    <h5>Фільтри підбору</h5>
    <form method="get" action="{% url 'pickup' %}">
      <div class="row g-3">
        <div class="col-md-4">
          <label for="manufacturer" class="form-label">Виробник</label>
          <select class="form-select" id="manufacturer" name="manufacturer">
            <option value="">Всі</option>
            <option value="Nvidia" {% if selected_manufacturer ==
"Nvidia" %}selected{% endif %}>Nvidia</option>
            <option value="AMD" {% if selected_manufacturer ==
"AMD" %}selected{% endif %}>AMD</option>
          </select>
        </div>
        <div class="col-md-4">
          <label for="usage" class="form-label">Призначення</label>
          <select class="form-select" id="usage" name="usage">
            <option value="">Всі</option>
            <option value="Ігрова" {% if selected_usage == "Ігрова" %}selected{%
endif %}>Ігрова</option>
            <option value="AI / ML" {% if selected_usage == "AI /
ML" %}selected{% endif %}>AI / ML</option>
            <option value="Графіка / Рендер" {% if selected_usage == "Графіка /
Рендер" %}selected{% endif %}>Графіка / Рендер</option>
            <option value="Майнінг" {% if selected_usage ==
"Майнінг" %}selected{% endif %}>Майнінг</option>
            <option value="Офісна" {% if selected_usage ==
"Офісна" %}selected{% endif %}>Офісна</option>
          </select>
        </div>
        <div class="col-md-4">
```

```
            <label for="budget" class="form-label">Бюджет</label>
            <select class="form-select" id="budget" name="budget">
              <option value="">Всі</option>
              <option value="low" {% if selected_budget == 'low' %}selected{%
endif %}>Бюджетний</option>
              <option value="mid" {% if selected_budget == 'mid' %}selected{%
endif %}>Середній</option>
              <option value="high" {% if selected_budget == 'high' %}selected{%
endif %}>Високий</option>
            </select>
          </div>
        </div>
        <div class="text-center mt-3">
          <button type="submit" class="btn btn-primary px-5">Підібрати
GPU</button>
        </div>
      </form>
    </div>
    <div class="table-responsive mb-4">
      <table id="gpu-table" class="table table-striped table-hover align-middle">
        <thead class="table-dark">
          <tr>
            <th></th>
            <th>Назва GPU</th>
            <th>Виробник</th>
            <th>Категорія</th>
            <th>Призначення</th>
            <th>Інтегральний бал</th>
            <th>Ціна (₴)</th>
            <th>GPU Value</th>
            <th>Найкращий кластер</th>
            <th>Ефективність</th>
          </tr>
        </thead>
        <tbody>
        {% if filtered_gpus %}
          {% for gpu in filtered_gpus %}
          <tr>
```

```
                <td>
                    <input type="checkbox" class="form-check-input gpu-checkbox"
value="{{ gpu.gpu_id }}">
                </td>
                <td>{{ gpu.gpu_name }}</td>
                <td>{{ gpu.manufacturer.manufacturer_name }}</td>
                <td>{{ gpu.category.category_name }}</td>
                <td>{{ gpu.usage.usage_name|default:"N/A" }}</td>
                <td>{{ gpu.fact.integrated_score|default:"N/A"|floatformat:2 }}</td>
                <td>{{ gpu.fact.price|default:"N/A"|floatformat:2 }}</td>
                <td>{{ gpu.fact.gpu_value|default:"N/A"|floatformat:2 }}</td>
                <td class="text-center">{% if gpu.fact.is_best_cluster %}✔{% else %}✖{%
endif %}</td>
                <td>{{ gpu.fact.efficiency|default:"N/A"|floatformat:2 }}</td>
              </tr>
              {% endfor %}
            {% else %}
              <tr>
                <td colspan="10" class="text-center text-muted">Немає результатів за
обраними фільтрами</td>
              </tr>
            {% endif %}
            </tbody>
        </table>
    </div>
    <div class="text-center mb-4">
        <button id="show-metrics" class="btn btn-success" disabled>Показати
метрики</button>
    </div>
</div>
{% endblock %}
```

Програмний код – представлення, яке відтворює HTML-сторінку підбору відеокарт із даними СД

```
def pickup_view(request):
    selected_manufacturer = request.GET.get('manufacturer', '').strip()
    selected_usage = request.GET.get('usage', '').strip()
    selected_budget = request.GET.get('budget', 'all').strip()

    video_card = VideoCard()

    need_recalc = (
            not FactGPUMetrics.objects.exists() or
            FactGPUMetrics.objects.filter(integrated_score__isnull=True).exists() or
            GPU.objects.filter(usage__isnull=True).exists() or
            GPU.objects.filter(segment__isnull=True).exists()
    )
    #video_card.culc()
    if need_recalc:
        video_card.culc()

    gpus = GPU.objects.select_related('manufacturer', 'category', 'usage', 'segment') \
        .prefetch_related('factgpumetrics_set')

    if selected_manufacturer:
        gpus = gpus.filter(manufacturer__manufacturer_name=selected_manufacturer)
    if selected_usage:
        gpus = gpus.filter(usage__usage_name=selected_usage)

    gpu_list = []
    for gpu in gpus:
        fact = gpu.factgpumetrics_set.first()
        if fact and fact.integrated_score is not None and fact.price is not None and
fact.price > 0:
            gpu.fact = fact
            fact.price = float(fact.price)
            fact.integrated_score = float(fact.integrated_score)
            fact.efficiency = float(fact.efficiency or 0)
            gpu_list.append(gpu)
```

```
if selected_budget in segment_map and segment_map[selected_budget] != 0:
    seg_id = segment_map[selected_budget]
    gpu_list = [g for g in gpu_list if g.segment_id == seg_id]
best_cluster_gpus = [g for g in gpu_list if getattr(g.fact, 'is_best_cluster', False)]
USAGE_ORDER = {
    'AI / ML': 1,
    'Графіка / Рендер': 2,
    'Майнінг': 3,
    'Ігрова': 4,
    'Офісна': 5
}
best_cluster_gpus.sort(key=lambda g: (-g.fact.efficiency,
    USAGE_ORDER.get(g.usage.usage_name if g.usage else '', 99),
))
#the_best_card = best_cluster_gpus[0]
context = {
    "filtered_gpus": best_cluster_gpus,
    "selected_manufacturer": selected_manufacturer,
    "selected_usage": selected_usage,
    "selected_budget": selected_budget,
}

return render(request, 'gpu_app/pickup.html', context)
```

Програмний код – представлення, яке формує HTML-сторінку «Метрики»

```
{% extends 'gpu_app/base.html' %}
{% block title %}Метрики GPU{% endblock %}

{% block content %}
<div class="container mt-4">
  <h2 class="mb-4 text-center">Метрики вибраних GPU</h2>

  <!-- Таблица сверху -->
  <h4 class="mb-3">Таблиця метрик</h4>
  <table id="metrics-table" class="table table-striped table-bordered table-sm">
    <thead>
      <tr>
        <th>GPU NAME</th>
        <th>G3DMark</th>
        <th>G2DMark</th>
        <th>Ціна</th>
        <th>GPU Value</th>
        <th>TDP</th>
        <th>Індекс продуктивності</th>
        <th>Середній API Score</th>
        <th>Інтегральний бал</th>
        <th>Ефективність</th>
      </tr>
    </thead>
    <tbody id="metrics-tbody"></tbody>
  </table>

  <!-- Графики -->
  <div id="chart-3d2d" class="mb-4" style="height:400px;"></div>
  <div id="chart-api-score" class="mb-4" style="height:400px;"></div>
  <div id="chart-value-efficiency" class="mb-4" style="height:400px;"></div>
  <div id="chart-price" class="mb-4" style="height:400px;"></div>
  <div id="chart-performance" class="mb-4" style="height:500px;"></div>
</div>
```

```
{% endblock %}

{% block scripts %}
{{ block.super }}
<script>
const metrics = {{ metrics_json|safe }};

// Генерация цветов для каждой видеокарты
const colors =
['#1f77b4','#ff7f0e','#2ca02c','#d62728','#9467bd','#8c564b','#e377c2','#7f7f7f','#bcbd22','
#17becf'];_

if (metrics.length > 0) {
    // --- Таблица ---
    const tbody = document.getElementById('metrics-tbody');
    metrics.forEach(row => {
        const tr = document.createElement('tr');
        tr.innerHTML = `
            <td>${row.gpu_name}</td>
            <td>${row.g3dmark}</td>
            <td>${row.g2dmark}</td>
            <td>${row.price}</td>
            <td>${row.gpu_value}</td>
            <td>${row.tdp}</td>
            <td>${row.performance_index}</td>
            <td>${row.avg_api_score}</td>
            <td>${row.integrated_score}</td>
            <td>${row.efficiency}</td>
        `;
        tbody.appendChild(tr);
    });

    // Создаем словарь GPU -> цвет
    const gpuColors = {};
    metrics.forEach((m, i) => gpuColors[m.gpu_name] = colors[i % colors.length]);

    // --- График G3DMark и G2DMark ---
    const traceG3D = {
```

```
    x: metrics.map(m => m.gpu_name),
    y: metrics.map(m => m.g3dmark),
    name: 'G3DMark',
    type: 'bar',
    marker: { color: '#1f77b4' },
    text: metrics.map(m => m.g3dmark.toFixed(1)),
    textposition: 'auto',
    hovertemplate: '<b>%{x}</b><br>G3DMark: %{y}<extra></extra>'
};
const traceG2D = {
    x: metrics.map(m => m.gpu_name),
    y: metrics.map(m => m.g2dmark),
    name: 'G2DMark',
    type: 'bar',
    marker: { color: '#ff7f0e' },
    text: metrics.map(m => m.g2dmark.toFixed(1)),
    textposition: 'auto',
    hovertemplate: '<b>%{x}</b><br>G2DMark: %{y}<extra></extra>'
};
Plotly.newPlot('chart-3d2d', [traceG3D, traceG2D], {
    title: 'G3DMark та G2DMark',
    barmode: 'group',
    xaxis: {
        title: 'GPU',
        tickangle: -45
    },
    yaxis: { title: 'Бали' },
    margin: { b: 100 }
});

// --- График API Score ---
const traceAPI = {
    x: metrics.map(m => m.gpu_name),
    y: metrics.map(m => m.avg_api_score),
    name: 'API Score',
    type: 'bar',
    marker: { color: '#9467bd' },
    text: metrics.map(m => m.avg_api_score.toFixed(1)),
```

```
    textposition: 'auto',
    hovertemplate: '<b>%{x}</b><br>API Score: %{y}<extra></extra>'
};
Plotly.newPlot('chart-api-score', [traceAPI], {
    title: 'Середній API Score',
    xaxis: {
        title: 'GPU',
        tickangle: -45
    },
    yaxis: { title: 'Бали' },
    margin: { b: 100 }
});

// --- График GPU Value и Efficiency ---
const traceValue = {
    x: metrics.map(m => m.gpu_name),
    y: metrics.map(m => m.gpu_value),
    name: 'GPU Value',
    type: 'scatter',
    mode: 'lines+markers',
    line: {
        color: '#2ca02c',
        width: 3
    },
    marker: {
        color: '#2ca02c',
        size: 8
    },
    hovertemplate: '<b>%{x}</b><br>GPU Value: %{y}<extra></extra>'
};
const traceEff = {
    x: metrics.map(m => m.gpu_name),
    y: metrics.map(m => m.efficiency),
    name: 'Efficiency',
    type: 'scatter',
    mode: 'lines+markers',
    line: {
        color: '#d62728',
```

```
        width: 3,
        dash: 'dash'
    },
    marker: {
        color: '#d62728',
        size: 8,
        symbol: 'diamond'
    },
    hovertemplate: '<b>%{x}</b><br>Efficiency: %{y}<extra></extra>'
};
Plotly.newPlot('chart-value-efficiency', [traceValue, traceEff], {
    title: 'GPU Value та Efficiency',
    xaxis: {
        title: 'GPU',
        tickangle: -45
    },
    yaxis: { title: 'Значення' },
    margin: { b: 100 }
});
// --- График Цены ---
const tracePrice = {
    x: metrics.map(m => m.gpu_name),
    y: metrics.map(m => m.price),
    type: 'bar',
    marker: {
        color: metrics.map(m => gpuColors[m.gpu_name]),
        opacity: 0.8
    },
    text: metrics.map(m => m.price.toFixed(1)),
    textposition: 'auto',
    name: 'Ціна',
    hovertemplate: '<b>%{x}</b><br>Ціна: %{y}<extra></extra>'
};
Plotly.newPlot('chart-price', [tracePrice], {
    title: 'Ціна відеокарт',
    xaxis: {
        title: 'GPU',
        tickangle: -45
```

```
        },
        yaxis: { title: 'Ціна' },
        margin: { b: 100 }
    });

    // --- Круговая диаграмма для сравнения производительности ---
    const performanceTrace = {
        labels: metrics.map(m => m.gpu_name),
        values: metrics.map(m => m.integrated_score),
        type: 'pie',
        hole: 0.4,
        marker: {
            colors: metrics.map(m => gpuColors[m.gpu_name])
        },
        textinfo: 'label+percent',
        textposition: 'outside',
        hovertemplate: '<b>%{label}</b><br>Інтегральний
бал: %{value}<br>Частка: %{percent}<extra></extra>'
    };
    Plotly.newPlot('chart-performance', [performanceTrace], {
        title: 'Порівняння загальної продуктивності (Інтегральний бал)',
        showlegend: true,
        legend: {
            orientation: 'h',
            y: -0.1
        },
        margin: { t: 50, b: 80, l: 50, r: 50 }
    });
}
</script>
{% endblock %}
```

Програмний код, який формує HTML-сторінку із кореляційною матрицею

```
{% extends 'gpu_app/base.html' %}
{% block title %}Кореляційна матриця — GPU Analytics{% endblock %}
{% block content %}
<div class="container mt-4">
  <h1 class="text-center mb-4">Кореляційна матриця (статистичний аналіз)</h1>

  <div class="card shadow-sm">
    <div class="card-header bg-dark text-white">
      <h5 class="mb-0">Матриця кореляції Пірсона</h5>
    </div>
    <div class="card-body p-0">
      <div class="table-responsive" style="max-height: 80vh; overflow: auto;">
        {{ corr_matrix_html|safe }}
      </div>
    </div>
  </div>

  <!-- Легенда кольорів -->
  <div class="mt-4">
    <h6>Легенда кореляції:</h6>
    <div class="d-flex flex-wrap gap-2">
      <span class="badge bg-danger">≥ 0.8 (дуже сильна)</span>
      <span class="badge bg-warning text-dark">≥ 0.6 (сильна)</span>
      <span class="badge bg-warning">≥ 0.4 (помірна)</span>
      <span class="badge bg-success">≥ 0.2 (слабка)</span>
      <span class="badge bg-secondary">< 0.2 (дуже слабка / відсутня)</span>
      <span class="badge bg-light text-dark">діагональ</span>
    </div>
  </div>

  <!-- Розшифрування скорочень -->
  <div class="mt-3">
    <h6>Розшифрування показників:</h6>
    <div class="row">
      <div class="col-md-6">
        <ul class="list-unstyled small">
```

```
                <li><strong>3D Mark</strong> — продуктивність 3D графіки</li>
                <li><strong>2D Mark</strong> — продуктивність 2D графіки</li>
                <li><strong>GPU Value</strong> — цінність відеокарти</li>
                <li><strong>TDP</strong> — тепловиділення</li>
                <li><strong>Perf Index</strong> — індекс продуктивності</li>
                <li><strong>API Score</strong> — середній бал API</li>
                <li><strong>Bench Score</strong> — середній бал бенчмарків</li>
            </ul>
        </div>
        <div class="col-md-6">
            <ul class="list-unstyled small">
                <li><strong>Min Score</strong> — мінімальний бал</li>
                <li><strong>Max Score</strong> — максимальний бал</li>
                <li><strong>Score Std</strong> — стандартне відхилення балів</li>
                <li><strong>Power/Perf</strong> — співвідношення потужності та
продуктивності</li>
                <li><strong>Int Score</strong> — інтегральний бал</li>
                <li><strong>Efficiency</strong> — ефективність</li>
            </ul>
        </div>
    </div>
  </div>

  <div class="mt-3">
    <small class="text-muted">
      * Значення кореляції варіюються від -1 до 1. Додатні значення вказують на
пряму залежність,
      від’ємні — на обернену залежність між параметрами.
    </small>
  </div>
</div>

<style>
.correlation-table {
  font-size: 12px;
  margin-bottom: 0;
}
```

```
.correlation-table th,
.correlation-table td {
    white-space: nowrap;
    min-width: 70px;
    max-width: 90px;
    text-overflow: ellipsis;
    overflow: hidden;
}

.correlation-table thead th {
    position: sticky;
    top: 0;
    z-index: 10;
}
.correlation-table tbody th {
    position: sticky;
    left: 0;
    background-color: #f8f9fa !important;
    z-index: 5;
    font-weight: bold;
}
.table-responsive {
    font-size: 0.875rem;
}
</style>
{% endblock %}
```