Вінницький національний технічний університет
(повне найменування вищого навчального закладу)
Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування інституту, назва факультету (відділення))
Кафедра програмного забезпечення
(повна назва кафедри (предметної, циклової комісії))

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка методів та програмного засобу для оптимізації великих мовних моделей для iOS»

Виконав: студент 2-го курсу, групи 2ПІ-23м
спеціальності 121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Луценко Р.С.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ:
Романюк О.В.
(прізвище та ініціали)
«06» грудня 2024 р.
Опонент:
к.т.н., доц. каф. ОТ Колесник І.С.
(прізвище та ініціали)
«06» грудня 2024 р.

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н, проф. Романюк О.Н.
(прізвище та ініціали)
«06» грудня 2024 р.

ВНТУ – 2024

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

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
д.т.н., професор
Романюк О. Н.
« 18 » вересня 2024 р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙУ РОБОТУ СТУДЕНТУ

Луценку Руслану Сергійовичу

1. Тема роботи – Розробка методів та програмного засобу для оптимізації великих мовних моделей для iOS.

Керівник роботи: Романюк Оксана Володимирівна, к.т.н., доц. кафедри ПЗ, затверджені наказом вищого навчального закладу від «17» вересня 2024 р. №310.

2. Строк подання студентом роботи 03 грудня 2024 р.

3. Вхідні дані до роботи: методи оптимізації – метод Low-Rank Approximation (LRA), метод Low-Rank Adaptation (LoRA), метод групового прунінгу, метод динамічної компресії параметрів, метод квантування параметрів; середовище розробки – XCode; мова розробки – Swift.

4. Зміст текстової частини: вступ; аналіз методів та засобів оптимізації великих мовних моделей та постановка задач дослідження; розробка методів оптимізації великих мовних моделей; розробка програмних компонент застосунку; експериментальні дослідження та тестування програмного застосунку; економічна частина; висновки; перелік посилань; додатки.

5. Перелік графічного матеріалу: назва роботи, актуальність теми, мета, об'єкт та предмет дослідження, завдання дослідження, наукова новизна, практична цінність отриманих результатів, метод гібридного адаптивного скорочення рангу, метод динамічного налаштування ваг, метод квантової

компресії мовних моделей для мобільних пристроїв, головні вікна програми вікно оптимізації, експериментальні дослідження запропонованих методів (частина перша), експериментальні дослідження запропонованих методів (частина друга), демонстрація програмного засобу, апробації та публікації висновки, фінальний слайд.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Романюк О.В., к.т.н., доцент кафедри ПЗ	19.09.24	02.12.24
5	Причепа І.В., к.е.н., доцент кафедри ЕПВМ	22.11.24	30.11.24

7. Дата видачі завдання 19 вересня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану розвитку великих мовних моделей та їх адаптації для мобільних платформ. Постановка задач дослідження	20.09.2024 30.09.2024	виконано
2	Розробка методів та алгоритмів оптимізації великих мовних моделей для iOS	01.10.2024 17.10.2024	виконано
3	Розробка програмної системи для оптимізації великих мовних моделей для iOS	18.10.2024 04.11.2024	виконано
4	Експериментальні дослідження розроблених методів та тестування програми	05.11.2024 21.11.2024	виконано
5	Економічна частина	22.11.2024 30.11.2024	виконано

Студент (підпис) **Луценко Р.С.** (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи (підпис) **Романюк О.В.** (прізвище та ініціали)

АНОТАЦІЯ

УДК 004.89

Луценко Р.С. Розробка методів та програмного засобу для оптимізації великих мовних моделей для iOS. Магістерська кваліфікаційна робота зі спеціальності 121 – Інженерія програмного забезпечення, освітня програма – Інженерія програмного забезпечення. Вінниця: ВНТУ, 2024. 141 с.

На укр. мові. Бібліогр.: 34 назв; рис.: 38; табл.: 13.

У магістерській кваліфікаційній роботі подано результати дослідження методів оптимізації великих мовних моделей для мобільних пристроїв на платформі iOS. Обґрунтовано доцільність удосконалення методів та засобів для адаптації моделей під конкретні технічні характеристики пристроїв, таких як частота процесора, обсяг оперативної пам'яті, доступність Neural Engine.

Магістерська кваліфікаційна робота містить аналіз відомих підходів до оптимізації мовних моделей, таких як Low-Rank Adaptation (LoRA), прунинг та квантова компресія, а також їхні обмеження при використанні на мобільних пристроях. У роботі розроблено метод, який поєднує LoRA, адаптивний прунинг і квантову компресію для створення оптимізованих моделей, що забезпечують ефективну роботу на пристроях з обмеженими обчислювальними ресурсами.

Розроблено метод оптимізації мовної моделі, який адаптує велику мовну модель, під конкретний пристрій, враховуючи його технічні характеристики. Забезпечено можливість порівняння продуктивності моделей до і після оптимізації, що дозволяє оцінити зменшення часу інференсу при збереженні точності. Розроблено мобільний застосунок на основі Swift та SwiftUI з використанням інтегрованого середовища розробки Xcode, який включає інтерфейс користувача для тестування та візуалізації результатів оптимізації.

Ключові слова: оптимізація мовних моделей, iOS, Low-Rank Adaptation, квантова компресія, прунинг.

ABSTRACT

UDC 004.89

Lutsenko R.S. Development of methods and software tool for optimization of large language models for iOS. Master's qualification work in specialty 121 – Software Engineering, educational program - Software Engineering. Vinnytsia: VNTU, 2024. 141 p.

In Ukrainian language. Bibliogr .: 34 titles; fig .: 38; tab .: 13.

The master's thesis presents the results of a study of methods for optimizing large language models for mobile devices on the iOS platform. The expediency of improving methods and tools for adapting models to specific technical characteristics of devices, such as processor frequency, RAM, and Neural Engine availability, is substantiated.

The master's thesis contains an analysis of well-known approaches to optimizing language models, such as Low-Rank Adaptation (LoRA), pruning, and quantum compression, as well as their limitations when used on mobile devices. In this paper, we develop a method that combines LoRA, adaptive pruning, and quantization to create optimized models that ensure efficient operation on devices with limited computing resources.

A language model optimization method has been developed that adapts a large language model, to a specific device, taking into account its technical characteristics. It is possible to compare the performance of the models before and after optimization, which allows to evaluate the reduction of inference time and preservation of accuracy. A mobile application based on Swift and SwiftUI was developed using the integrated Xcode development environment, which includes a user interface for testing and visualizing optimization results.

Keywords: language model optimization, iOS, Low-Rank Adaptation, quantum compression, pruning.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Ситуація, що склалася в сфері розробки та впровадження великих мовних моделей на мобільних платформах, вимагає від індустрії переходу на новий рівень оптимізації для забезпечення їх ефективної роботи на пристроях з обмеженими обчислювальними ресурсами. Через обмеження мобільних пристроїв, таких як невеликий обсяг оперативної пам'яті та обмежена потужність процесора, традиційні великі мовні моделі, розроблені для серверних середовищ, не можуть бути ефективно використані на мобільних пристроях без спеціальної адаптації. Враховуючи зростання попиту на мобільні застосунки, що базуються на штучному інтелекті, розробка інструментів для адаптації великих мовних моделей є ключовою задачею для забезпечення конкурентоспроможності на ринку програмного забезпечення. Мобільні пристрої використовуються для виконання все більш складних задач, включаючи переклад тексту, аналіз настрою, створення персоналізованих рекомендацій та обробку природної мови. Однак, без належної адаптації, використання великих мовних моделей може негативно впливати на швидкодію пристроїв, викликати збої у роботі додатків та значно скорочувати час роботи батареї. Така ситуація вимагає удосконалення методів та підходів до адаптації великих мовних моделей для мобільних платформ, враховуючи індивідуальні технічні характеристики пристроїв. У цьому контексті одним із найефективніших способів трансформації є оптимізація великих мовних моделей з використанням сучасних методів, таких як Low-Rank Adaptation (LoRA), адаптивний прунинг та квантова компресія, що дозволяє значно зменшити кількість параметрів без значної втрати точності [1].

Застосування технік, таких як LoRA, прунинг і квантова компресія, дозволяє не тільки зменшити розмір моделі, але й забезпечити її роботу в режимі реального часу на мобільних платформах. Це відкриває можливості для інтеграції високоефективних мовних моделей у широкий спектр мобільних застосунків, від голосових помічників до розумних чат-ботів [2].

Актуальність теми дослідження також підтверджується постійним розвитком технічних характеристик мобільних пристроїв. Сучасні смартфони мають потужні процесори, графічні ядра та нейронні модулі, які можна ефективно використовувати для роботи великих мовних моделей за умови їх правильної оптимізації. Особливо перспективним є застосування адаптивних алгоритмів, що дозволяють динамічно змінювати параметри моделей залежно від обчислювальних ресурсів пристрою. Це забезпечує ефективність роботи моделі навіть на пристроях середнього класу.

Розробка універсального методу оптимізації великих мовних моделей, який враховує індивідуальні технічні характеристики кожного пристрою, є досить актуальною задачею на сьогодні. Цей підхід дозволяє забезпечити баланс між якістю моделі та ефективністю її виконання, що є важливим для забезпечення зручного досвіду користування мобільними додатками.

Тому актуальною є розробка методів та програмного засобу для оптимізації великих мовних моделей для iOS, що дозволить забезпечити їх ефективну роботу на мобільних пристроях при обмежених ресурсах, зберігаючи високу точність і швидкість обробки даних.

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

Мета та завдання дослідження. Метою магістерської кваліфікаційної роботи є підвищення ефективності роботи великих мовних моделей на мобільних платформах шляхом розробки спеціалізованих методів оптимізації, які дозволять адаптувати такі моделі для використання на пристроях з обмеженими обчислювальними ресурсами.

Основними задачами роботи є:

- проаналізувати існуючі методи оптимізації мовних моделей для мобільних пристроїв;
- розробити метод гібридного адаптивного скорочення рангу;
- розробити метод динамічного налаштування ваг на основі апаратних

характеристик пристрою;

− розробити метод квантової компресії мовних моделей для мобільних пристроїв;

− розробити програмні модулі компонентів мобільного застосунку «OptiLang» для оптимізації великих мовних моделей;

− розробити зручний та зрозумілий графічний інтерфейс;

− повести експериментальні дослідження розроблених методів;

− провести тестування розробленої iOS-системи;

− розробити інструкцію користувача та описати системні вимоги розробленої програми.

Об’єктом дослідження є процес розробки та оптимізації великих мовних моделей для мобільних пристроїв на платформі iOS.

Предметом дослідження є методи та засоби оптимізації великих мовних моделей для мобільних пристроїв на платформі iOS.

Методи дослідження. У процесі досліджень використовувались методи дослідження: теорія матриць та лінійної алгебри, зокрема методи сингулярного розкладу матриць (SVD), для реалізації компонентів гібридного адаптивного скорочення рангу (HARR); методи цифрової оптимізації для адаптації вагових параметрів великих мовних моделей під апаратні характеристики мобільних пристроїв; методи чисельного аналізу для забезпечення точності та стабільності алгоритмів квантової компресії; теорія адаптивних алгоритмів для динамічного налаштування параметрів моделі залежно від наявних ресурсів пристрою; елементи теорії інформації для аналізу компресії даних і збереження інформативності при зменшенні розміру моделі; принципи гібридного моделювання для поєднання технік LoRA та прунингу у єдиному адаптивному алгоритмі; методи тестування та комп’ютерне моделювання для перевірки працездатності мобільного застосунку.

Наукова новизна отриманих результатів:

1. Подальшого розвитку отримав метод гібридного адаптивного скорочення рангу (HARR), який, на відміну від відомих, поєднує техніки LoRA

та прунингу з урахуванням індивідуальних технічних характеристик мобільного пристрою, що дозволило зменшити розмір моделі на 25.7% при збереженні високої точності та знизити максимальне використання процесора до 93.5%.

2. Подальшого розвитку отримав метод динамічного налаштування ваг на основі апаратних характеристик пристрою (DHWA), який, на відміну від існуючих методів, автоматично адаптує параметри моделі залежно від поточних обчислювальних можливостей пристрою, що дозволило зменшити енергоспоживання та час генерації відповіді на 39%, підвищуючи продуктивність у реальних умовах.

3. Подальшого розвитку набув метод квантової компресії мовних моделей для мобільних пристроїв, який, на відміну від існуючих методів, використовує адаптивну стратегію вибору рівня квантування для кожного шару моделі залежно від його важливості, що забезпечило зменшення кількості ваг моделі на 30%, зниження використання пам’яті на 37% при збереженні високої точності.

Практична цінність отриманих результатів полягає у тому, що на основі отриманих у магістерській кваліфікаційній роботі теоретичних положень запропоновано алгоритми та програмні засоби для оптимізації великих мовних моделей та їх адаптації для мобільних пристроїв.

Особистий внесок здобувача. Усі наукові результати, викладені у магістерській кваліфікаційній роботі, отримані автором особисто. У наукових роботах опублікованих у співавторстві, автору належать такі результати: аналіз існуючих методів оптимізації великих мовних моделей [2], розробка методу гібридного адаптивного скорочення рангу (Hybrid Adaptive Rank Reduction – HARR) [3].

Апробація матеріалів магістерської кваліфікаційної роботи. Результати магістерської кваліфікаційної роботи обговорювалися на:

– IV Всеукраїнській науково-технічній конференції молодих вчених, аспірантів і студентів «Комп’ютерні ігри та мультимедіа як інноваційний підхід до комунікації» (Одеса, 2024 р.);

– Міжнародній науково-практичній Інтернет-конференції «Електронні

інформаційні ресурси: створення, використання, доступ» (Суми/Вінниця, 2024).

Публікації. За тематикою дослідження опубліковано 2 наукові праці у збірниках матеріалів конференцій.

Структура та обсяг роботи. Магістерська кваліфікаційна робота складається зі вступу, п’яти розділів, висновків, списку використаних джерел, що містить 34 найменування, 4 додатків. Робота містить 38 ілюстрацій, 13 таблиць.

1 АНАЛІЗ МЕТОДІВ ТА ЗАСОБІВ ОПТИМІЗАЦІЇ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз підходів до адаптації великих мовних моделей для мобільних платформ

Великі мовні моделі (LLMs), такі як GPT-3, BERT, та новітніші, як-от LLaMA, стали важливою частиною сучасних технологій обробки природної мови. Вони здатні генерувати текст, розуміти контекст, перекладати мови та відповідати на складні питання, що робить їх корисними в різних галузях: від автоматизації підтримки клієнтів до створення творчих текстів і аналізу даних. Проте, використання таких моделей на мобільних платформах має свої виклики, зокрема через їх великі розміри та потребу в значних обчислювальних ресурсах.

Мобільні платформи, такі як iOS [4] та Android, мають значно обмеженіші ресурси порівняно із серверними середовищами, для яких ці моделі були спочатку створені. Основні обмеження включають недостатню потужність процесора, обмежений обсяг оперативної пам'яті та відсутність високопродуктивних обчислювальних можливостей, таких як GPU або TPU, що є стандартними для серверів. Це означає, що традиційні великі мовні моделі потребують адаптації для того, щоб бути використаними на мобільних пристроях.

Існують кілька підходів до адаптації великих мовних моделей для мобільних платформ. Серед найбільш поширених є квантова компресія, прунинг ваг та Low-Rank Adaptation (LoRA). Квантова компресія дозволяє зменшити розмір ваг моделі, що дозволяє зменшити обсяг необхідної пам'яті для зберігання моделі та прискорити її роботу. Адаптивний прунинг передбачає видалення малозначущих параметрів моделі, що допомагає зменшити кількість обчислень під час інференсу. Метод Low-Rank Adaptation дозволяє адаптувати великі мовні моделі, представляючи їх ваги як добуток двох менших матриць, що знижує розмірність та зберігає високу точність.

Окрім цих традиційних підходів, постійно з'являються нові інноваційні рішення. Наприклад, нові методи адаптивного скорочення рангу дозволяють

динамічно підлаштовувати модель під конкретний мобільний пристрій, з урахуванням його обчислювальних можливостей [5]. Це дозволяє не лише зменшити розмір моделі, але й забезпечити оптимальний баланс між швидкістю виконання та точністю результатів. У цьому контексті розробка нового методу гібридного адаптивного скорочення рангу (Hybrid Adaptive Rank Reduction – HARR) є актуальною задачею, оскільки цей підхід поєднує в собі переваги LoRA та інших методів для досягнення кращих результатів.

Іншим перспективним підходом є метод динамічного налаштування ваг на основі апаратних характеристик пристрою (Dynamic Hardware-Based Weight Adjustment – DHWA). Цей метод передбачає автоматичне підлаштування ваг моделі залежно від конкретного пристрою, на якому вона виконується. Наприклад, якщо пристрій має обмежену кількість оперативної пам'яті, метод DHWA зменшує кількість активних параметрів, що використовуються під час інференсу, без значної втрати точності. Таким чином, модель стає більш адаптованою до конкретного середовища, що значно покращує користувацький досвід.

Розвиток великих мовних моделей тісно пов'язаний із використанням сучасних інструментів оптимізації, таких як Core ML та TensorFlow Lite, які дозволяють переносити складні моделі на мобільні платформи. Такі фреймворки надають можливість ефективної інтеграції оптимізованих моделей в мобільні додатки, забезпечуючи швидкий інференс та низьке енергоспоживання, що є критичним для забезпечення зручності користування мобільним пристроєм.

Отже, аналіз стану розвитку великих мовних моделей показує, що, незважаючи на їх потужність, вони вимагають значної адаптації для мобільних платформ. Застосування сучасних методів оптимізації, таких як LoRA, квантова компресія, прунинг, а також розробка нових підходів, таких як HARR і DHWA, дозволяє зробити великі мовні моделі доступними для використання на мобільних пристроях. Це відкриває нові можливості для інтеграції високоефективних мовних моделей у мобільні додатки, що підвищує якість взаємодії з користувачем та забезпечує новий рівень персоналізації і зручності.

1.2 Порівняльний аналіз засобів оптимізації мовних моделей

З кожним роком, чи навіть днем, мобільні технології стають все більш невід'ємною частиною нашого життя. Для багатьох мобільні додатки вже є основним інструментом для роботи, навчання, комунікації та розваг. Розвиток штучного інтелекту та мовних моделей, зокрема, відкриває нові можливості у взаємодії з мобільними пристроями. Великі мовні моделі здатні не лише спростити виконання повсякденних завдань, але й надати користувачам унікальний досвід спілкування з додатками, адаптованими до їхніх потреб.

Проте використання великих мовних моделей на мобільних платформах стикається з серйозними обмеженнями. Мобільні пристрої зазвичай мають обмежені апаратні ресурси, такі як процесорна потужність та обсяг оперативної пам'яті. Через це розробники шукають ефективні рішення для оптимізації таких моделей, аби забезпечити високу продуктивність і плавну роботу мобільних додатків навіть за умов обмежених ресурсів.

Серед існуючих рішень для оптимізації великих мовних моделей на мобільних платформах, найбільш близькими до створюваного додатку «OptiLang» є такі:

− Core ML Tools – інструменти для оптимізації моделей, розроблені Apple, що дозволяють інтегрувати машинне навчання у додатки iOS;

− TensorFlow Lite – полегшена версія TensorFlow, яка підтримує прунинг та квантову компресію для ефективнішого запуску моделей на мобільних пристроях;

− ONNX Runtime Mobile – платформа, яка забезпечує запуск моделей у форматі ONNX на мобільних платформах і підтримує методи оптимізації для зменшення затримки;

− Hugging Face Transformers Lite – версія бібліотеки для мобільних пристроїв, що надає оптимізовані моделі, зокрема для задач обробки природної мови (NLP);

− NVIDIA TensorRT – це платформа для високопродуктивної оптимізації та прискорення моделей глибинного навчання, орієнтована на зниження

затримки та підвищення продуктивності обчислень, особливо на GPU.

Core ML Tools – це набір інструментів (див. рисунок 1.1), створений компанією Apple [6], який дозволяє розробникам інтегрувати моделі машинного навчання безпосередньо у мобільні додатки для iOS. Його основне завдання – спрощення роботи з машинним навчанням на iOS шляхом автоматизації процесу конвертації моделей, побудованих у середовищах, таких як TensorFlow, PyTorch, та інших, у формат, сумісний із Core ML. Використання Core ML значно покращує продуктивність моделей та знижує затримки в інференсі, забезпечуючи швидку обробку запитів без підключення до серверів.

Основна перевага Core ML Tools полягає в глибокій інтеграції з екосистемою Apple. Наприклад, цей фреймворк дозволяє використовувати апаратні компоненти пристроїв, такі як Neural Engine, що надає додаткову швидкість і ефективність для моделі. Core ML також підтримує ряд спеціалізованих моделей для комп'ютерного зору, обробки природної мови, аналізу зображень і звуку. Крім того, Apple надає інструменти для спрощеної інтеграції API, таких як Vision, Natural Language та Speech, що дозволяє швидко адаптувати моделі для конкретних завдань у додатку.

Для розробників, які створюють або адаптують великі мовні моделі, Core ML має певні обмеження. Через суворі вимоги до ресурсів пам’яті на мобільних пристроях, великі моделі, такі як GPT або BERT, потребують додаткових заходів оптимізації. Core ML підтримує базові методи оптимізації, такі як зменшення розрядності, але повноцінний прунинг чи спеціальні методи оптимізації, як LoRA, потребують сторонніх рішень або детальних налаштувань. Враховуючи ці особливості, Core ML є найефективнішим для моделей середніх розмірів, проте для масштабних мовних моделей, що потребують глибоких нейронних мереж, він може бути менш придатним.

Процес роботи з Core ML починається з конвертації моделі у формат .mlmodel, що дозволяє їй працювати у додатку безпосередньо на пристрої користувача. Для цього розробники можуть використовувати Core ML Converter, який автоматично адаптує архітектуру моделі під вимоги iOS. Крім того, у Core

ML існує підтримка модульного підходу, де окремі частини моделі можна оновлювати чи доопрацьовувати без повної перевірки програми.

Недоліки Core ML Tools включають обмеження для роботи з великими мовними моделями без спеціальної оптимізації, оскільки великі обсяги параметрів потребують більшої кількості пам’яті та енергоспоживання. Також цей інструмент ефективний тільки для екосистеми Apple, що обмежує його застосування у кросплатформних додатках.

Рисунок 1.1 – Інтерфейс Core ML Tools для інтеграції моделей машинного навчання в додатки iOS

TensorFlow Lite – це оптимізована версія фреймворку TensorFlow (див. рисунок 1.2), створена спеціально для роботи на мобільних та вбудованих пристроях, включаючи iOS.

TensorFlow Lite призначений для виконання завдань машинного навчання в додатках, які вимагають швидкої обробки даних з низькою затримкою та мінімальним споживанням ресурсів [7]. Цей інструмент підтримує конвертацію

моделей, розроблених у TensorFlow, у більш легкий формат (.tflite), що дозволяє моделі працювати навіть на пристроях з обмеженими обчислювальними можливостями, зберігаючи високу точність.

Основною перевагою TensorFlow Lite є можливість оптимізації моделей за допомогою таких методів, як квантова компресія та прунинг. Квантова компресія знижує розрядність чисел у моделі, що дозволяє значно зменшити її розмір без суттєвих втрат у продуктивності. Прунинг ж, у свою чергу, видаляє частину непотрібних параметрів моделі, що підвищує швидкість її обробки та зменшує обсяг пам’яті. Завдяки цим методам, TensorFlow Lite ідеально підходить для запуску великих мовних моделей на мобільних платформах, забезпечуючи баланс між продуктивністю та ефективністю.

TensorFlow Lite також підтримує запуск моделей на різних типах процесорів, зокрема на CPU, GPU та навіть TPU (Tensor Processing Unit), якщо вони доступні на пристрої. Така гнучкість дозволяє розробникам налаштовувати додаток на використання оптимального ресурсу, що особливо корисно для великих мовних моделей, де прискорення обробки є критично важливим для забезпечення реального часу роботи.

Процес роботи з TensorFlow Lite включає конвертацію моделі у формат TFLite за допомогою TensorFlow Lite Converter. Після конвертації модель можна додатково оптимізувати через API фреймворку, застосовуючи методи квантової компресії або прунингу для мінімізації розміру і максимізації продуктивності на мобільному пристрої. Після цього модель легко інтегрується у додаток, і розробник може керувати її ресурсами через доступні опції TensorFlow Lite.

Недоліками TensorFlow Lite є обмежена інтеграція з екосистемою iOS порівняно з Core ML. Наприклад, для повної підтримки деяких функцій потрібні додаткові налаштування, а запуск моделі може бути трохи повільнішим, ніж у Core ML, через специфіку сумісності. Крім того, робота з TensorFlow Lite вимагає певного рівня знань щодо обробки моделей машинного навчання, що може бути складним для початківців.

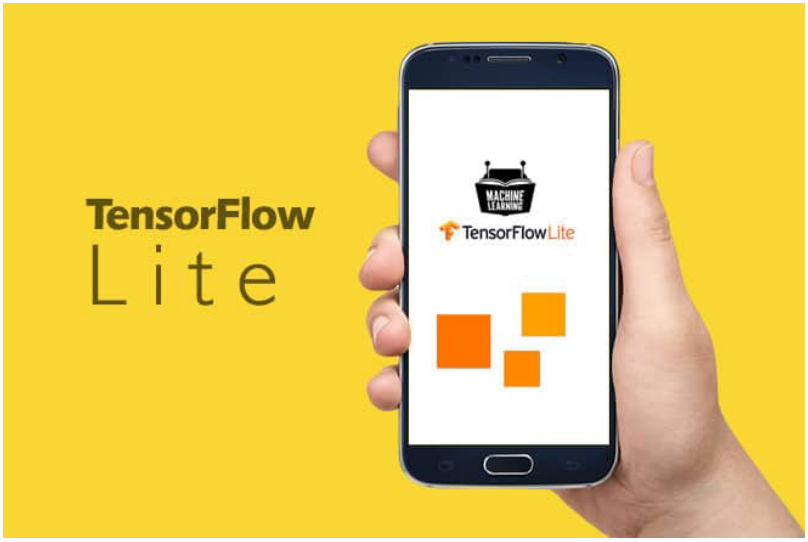


Рисунок 1.2 – Інтерфейс TensorFlow Lite для оптимізації та запуску моделей на мобільних пристроях

ONNX Runtime Mobile – це спеціалізована версія ONNX Runtime (див. рисунок 1.3), розроблена для оптимізації та запуску моделей у форматі ONNX (Open Neural Network Exchange) на мобільних та вбудованих пристроях, зокрема iOS та Android [8]. Цей інструмент дозволяє розробникам переносити моделі машинного навчання, створені в таких фреймворках, як PyTorch і TensorFlow, на мобільні платформи, що забезпечує гнучкість і сумісність із різними середовищами. ONNX Runtime Mobile підтримує різні методи оптимізації, такі як квантова компресія та прунинг, що дозволяє значно зменшити розмір моделей, не втрачаючи при цьому точності.

ONNX Runtime Mobile є особливо корисним для розробників, які шукають універсальне кросплатформне рішення. На відміну від Core ML, яке обмежене екосистемою iOS, ONNX Runtime Mobile дозволяє запускати моделі на обох популярних мобільних платформах, що робить його зручним для проєктів, де необхідна підтримка iOS і Android. Завдяки цьому розробники можуть працювати з єдиним форматом моделі (ONNX) та використовувати його на різних пристроях, що значно спрощує процес розробки та тестування.

Процес роботи з ONNX Runtime Mobile включає конвертацію моделі у формат ONNX за допомогою ONNX Exporter, після чого модель оптимізується

шляхом прунингу та квантової компресії для мінімізації розміру та забезпечення продуктивності на мобільному пристрої. Далі модель інтегрується в додаток, де ONNX Runtime Mobile забезпечує її запуск і обробку. Важливою перевагою цього інструменту є можливість застосування окремих ресурсів пристрою, як-от CPU або GPU, для підвищення швидкості обробки та зниження енергоспоживання.

Недоліки ONNX Runtime Mobile включають слабшу інтеграцію з iOS порівняно з Core ML, що може призвести до незначних затримок у продуктивності на цій платформі. Крім того, хоча ONNX Runtime Mobile підтримує основні методи оптимізації, для розробників можуть бути складнощі з налаштуванням деяких специфічних моделей, особливо тих, що вимагають глибоких нейронних мереж і адаптації для пристроїв із обмеженими ресурсами. Незважаючи на це, ONNX Runtime Mobile є одним із найбільш універсальних інструментів, що дозволяє запускати моделі машинного навчання з високою продуктивністю на мобільних платформах.

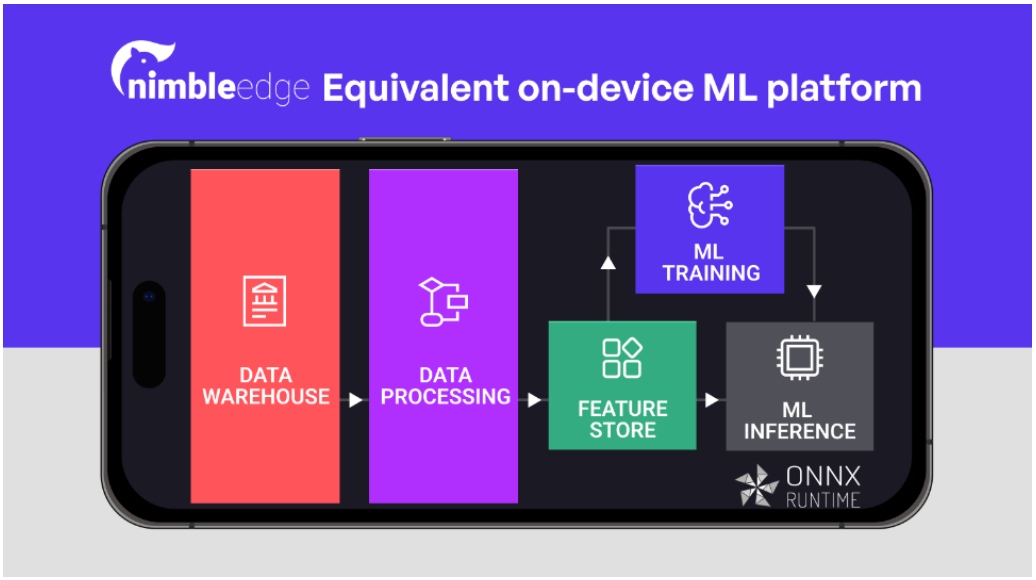


Рисунок 1.3 – Інтерфейс ONNX Runtime Mobile для кросплатформного використання моделей машинного навчання

Hugging Face Transformers Lite – це спеціалізована версія популярної бібліотеки Hugging Face (див. рисунок 1.4), адаптована для мобільних пристроїв, яка надає розробникам доступ до широкого асортименту попередньо

натренованих мовних моделей, таких як BERT, GPT, RoBERTa та інші [9]. Ця бібліотека дозволяє виконувати завдання обробки природної мови (NLP) на мобільних пристроях, зберігаючи продуктивність і точність моделей навіть на обмежених ресурсах. Hugging Face Transformers Lite підтримує оптимізаційні методи, зокрема квантову компресію та прунинг, що дозволяє зменшити розмір моделей для ефективного використання на мобільних платформах.

Основною перевагою Hugging Face Transformers Lite є доступ до перевірених та натренованих мовних моделей, які мають високу точність та широкий спектр застосувань, від чат-ботів до автоматичного перекладу. Завдяки інтеграції з мобільними пристроями розробники можуть легко адаптувати ці моделі під свої завдання, застосовуючи методи квантової компресії та прунингу для зменшення розміру та споживання ресурсів. Це робить бібліотеку зручною для застосування у додатках, де швидкість відповіді та мінімальне споживання ресурсів є ключовими.

Процес роботи з Hugging Face Transformers Lite передбачає вибір та завантаження необхідної моделі через бібліотеку Hugging Face, після чого модель оптимізується шляхом квантової компресії чи прунингу для мобільного використання. Модель потім інтегрується у додаток, де обробляє запити користувачів у реальному часі. Hugging Face також надає API, що дозволяє легко експериментувати з різними моделями та обирати оптимальну конфігурацію для конкретного застосування.

Недоліки Hugging Face Transformers Lite включають обмежену підтримку повної інтеграції з екосистемою iOS, що може вимагати додаткових налаштувань для досягнення бажаної продуктивності. Крім того, ця бібліотека орієнтована переважно на задачі обробки природної мови, що може обмежити її застосування для більш широкого спектру завдань машинного навчання. Незважаючи на це, Hugging Face Transformers Lite залишається одним із найзручніших рішень для тих, хто хоче використовувати потужні мовні моделі на мобільних платформах з мінімальними зусиллями.

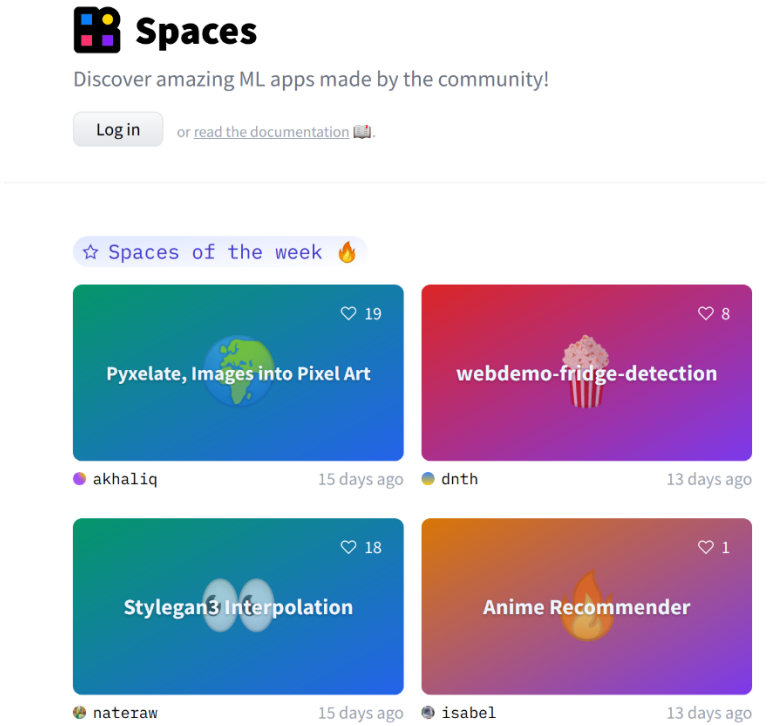


Рисунок 1.4 – Інтерфейс Hugging Face Transformers Lite для роботи з мовними моделями на мобільних пристроях

NVIDIA TensorRT – це потужний інструмент для оптимізації та прискорення моделей глибинного навчання (див. рисунок 1.5), спрямований на мінімізацію затримок і максимізацію обчислювальної ефективності, з особливим акцентом на використанні GPU [10].

Хоча TensorRT спочатку розроблявся для серверних середовищ і потужних настільних комп'ютерів, його методи оптимізації – квантова компресія, прунинг та ф'юзинг шарів – можуть бути корисними для розробників мобільних додатків, оскільки вони дозволяють зменшити обсяг пам'яті та обчислювальні ресурси, необхідні для роботи великих мовних моделей.

Однією з ключових переваг TensorRT є спеціалізовані алгоритми для зменшення затримки при обробці даних. Завдяки інструментам для автоматичної оптимізації, розробники можуть перетворювати вихідні моделі, створені у TensorFlow або PyTorch, на високопродуктивні моделі з мінімальними витратами на обчислювальні ресурси. TensorRT дозволяє використовувати точність FP16 і

навіть INT8, що значно зменшує розмір моделей, що є важливим для роботи на пристроях із обмеженими ресурсами.

Процес роботи з NVIDIA TensorRT починається з конвертації вихідної моделі у формат, сумісний з TensorRT, з використанням інструментів для зниження розрядності і прунингу. Ці методи дозволяють досягти оптимальної продуктивності при мінімальному споживанні пам'яті та енергії. Після цього модель можна інтегрувати у додаток, де TensorRT забезпечує обробку даних за допомогою GPU, що значно прискорює роботу і знижує затрати на CPU.

Недоліками TensorRT є його обмеження для мобільних пристроїв, особливо якщо вони не оснащені потужними GPU. Крім того, інтеграція з iOS потребує додаткових налаштувань і не забезпечує такої глибокої інтеграції, як на платформах з підтримкою NVIDIA GPU. Незважаючи на це, TensorRT залишається одним із найефективніших інструментів для оптимізації та прискорення великих мовних моделей, коли мова йде про використання на пристроях із підтримкою потужних обчислювальних ядер.

Рисунок 1.5 – Інтерфейс NVIDIA TensorRT для оптимізації моделей глибинного навчання

Після аналізу усіх аналогів визначено їхні переваги та недоліки, а також проведено порівняння з розроблюваним застосунком для оптимізації великих мовних моделей «OptiLang». Результати порівняння зведені в таблицю 1.1.

Таблиця 1.1 – Порівняльні характеристики мобільних застосунків

Назва додатку Критерій	Core ML Tools	Tensor Flow Lite	ONNX Runtime Mobile	Hugging Face Transformers Lite	NVIDIA TensorRT	OptiLang
Глибока інтеграція з iOS	+	-	-	-	-	+
Підтримка квантової компресії	-	+	+	+	+	+
Підтримка прунингу	-	+	+	-	+	+
Адаптація для великих мовних моделей	-	+	+	+	+	+
Універсальність для кросплатформ	-	+	-	-	-	+
Підсумковий результат	1	4	3	2	3	5

Отже, можна зробити висновок, що у порівнянні з існуючими аналогами розробка власного інструменту «OptiLang» є доцільною. Він поєднує переваги кожного з представлених інструментів та вирішує їхні основні обмеження, надаючи високоефективні можливості для оптимізації великих мовних моделей на мобільних платформах.

1.3 Аналіз методів розв’язання задачі

Розробка застосунків для мобільних платформ, зокрема для iOS, є популярним напрямом у сучасній програмній індустрії. Оскільки мобільні пристрої стають дедалі потужнішими, зростає і попит на складні мовні моделі та інші алгоритми штучного інтелекту, які можна використовувати безпосередньо на мобільних пристроях. Один із ключових аспектів у розробці додатків полягає у виборі відповідного підходу для створення і підтримки додатків, з урахуванням особливостей платформи, на яку орієнтований продукт.

Нативна розробка передбачає створення додатків, які повністю адаптовані до платформи та написані на мові програмування, що оптимально підтримується операційною системою [11]. Для iOS основною мовою є Swift, яка поєднує високу продуктивність із простотою і сучасним підходом до розробки. Нативна розробка надає можливість безпосередньо використовувати всі ресурси та інструменти платформи, що дозволяє створювати додатки з високою продуктивністю, адаптовані до апаратних та програмних можливостей пристроїв.

Нативні додатки мають низку переваг. Вони забезпечують повну сумісність з апаратними та програмними компонентами платформи, мають доступ до вбудованих сервісів, таких як Core ML, ARKit, HealthKit та інші. Така глибока інтеграція дозволяє розробляти додатки з максимальною ефективністю, забезпечуючи швидку обробку даних, стабільну роботу та високу якість користувацького інтерфейсу [12].

Основні інструменти для розробки додатків на iOS включають Xcode, Swift, SwiftUI та UIKit. Xcode є офіційною середовищем розробки від Apple, яка надає всі необхідні засоби для створення, тестування і деплою додатків. Xcode підтримує всі новітні технології та фреймворки, що дозволяє розробляти сучасні додатки, використовуючи останні можливості iOS.

Swift є основною мовою для написання коду, яка замінила Objective-C, завдяки своїй простоті, безпеці та ефективності. Вона підтримує автоматичне управління пам’яттю, високу продуктивність та сучасний синтаксис, що значно

спрощує процес розробки і забезпечує високу стабільність роботи додатків [13].

Для побудови інтерфейсу користувача iOS пропонує два основні фреймворки: UIKit та SwiftUI. UIKit є традиційним інструментом для створення UI на iOS і дозволяє створювати інтерфейси, використовуючи компоненти, які безпосередньо працюють з системою і забезпечують максимальну гнучкість у налаштуванні [14].

З іншого боку, SwiftUI є новим фреймворком, що використовує декларативний підхід [15], дозволяючи розробникам створювати інтерфейси простішим способом. SwiftUI автоматично адаптується до розміру екрана та дозволяє використовувати реактивне програмування, що робить додатки більш інтерактивними. SwiftUI також підтримує гнучке компонування інтерфейсу, використовуючи VStack, HStack та ZStack для розташування елементів, а також широкий набір модифікаторів для налаштування стилю та вигляду елементів [16]. SwiftUI стає дедалі популярнішим завдяки простоті, швидкості та зручності у використанні, а також можливості легко адаптувати UI для різних пристроїв (див. рисунок 1.6).

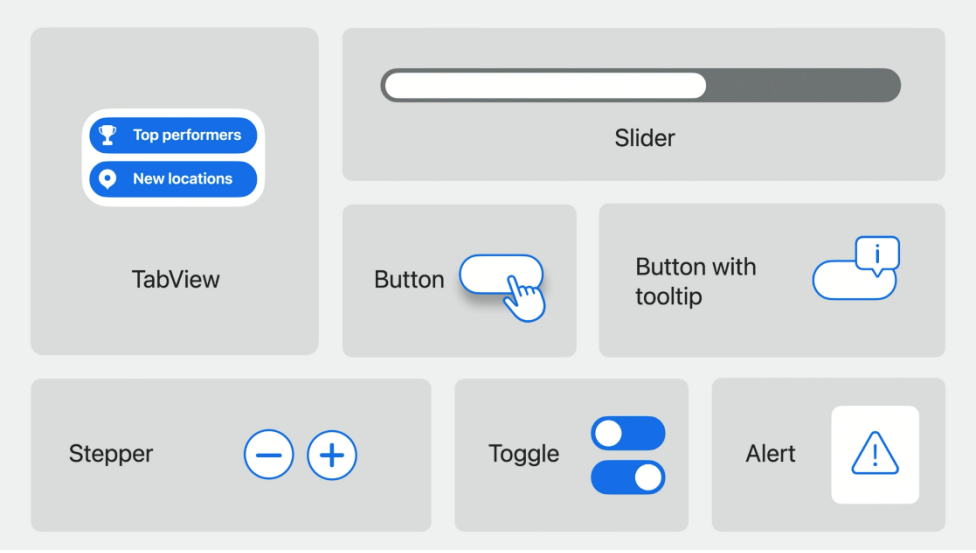


Рисунок 1.6 – UI-компоненти в SwiftUI

Нативна розробка для iOS дозволяє використовувати всі переваги апаратного забезпечення та програмних інструментів Apple, забезпечуючи

високу продуктивність, безпеку та стабільність роботи додатків. Основні переваги нативної розробки:

1. Швидкодія та ефективність. Нативні додатки безпосередньо працюють із ресурсами пристрою, забезпечуючи швидкий доступ до пам’яті, процесора та інших апаратних компонентів, що особливо важливо для мовних моделей, які вимагають високої обчислювальної потужності.

2. Максимальна інтеграція з iOS. Завдяки повній підтримці технологій Apple, таких як Core ML, можна безпосередньо інтегрувати мовні моделі в додатки, забезпечуючи їх оптимальну роботу та зниження затримок у відповідях на запити користувача.

3. Підтримка новітніх технологій. SwiftUI, як і UIKit, дозволяють створювати адаптивні інтерфейси, що автоматично підлаштовуються під різні розміри екранів і типи пристроїв, забезпечуючи зручність у використанні та інтерактивність додатків.

Незважаючи на зручність кросплатформної розробки, нативний підхід залишається переважним для iOS-додатків, оскільки він дозволяє повною мірою використовувати ресурси пристрою та оптимізувати обробку даних на низькому рівні [17].

Однією з особливостей нативної розробки є використання SwiftUI, який поєднує простоту та ефективність створення інтерфейсів для додатків. Завдяки декларативному синтаксису SwiftUI дозволяє швидко створювати UI, що автоматично адаптується до різних пристроїв. Інтеграція з Core ML дозволяє безпосередньо працювати з мовними моделями, забезпечуючи швидкість та зручність у відображенні результатів роботи моделей у реальному часі.

SwiftUI підтримує реактивне програмування, що значно спрощує управління станом у додатках. Завдяки @State та @Binding розробники можуть легко контролювати зміни в UI у реальному часі [18], що підвищує інтерактивність додатку. Наприклад, змінні, позначені як @State, автоматично оновлюють інтерфейс при зміні значень, забезпечуючи швидку й ефективну обробку.

SwiftUI також включає функцію Preview [19], яка дозволяє розробникам миттєво переглядати зміни у інтерфейсі без необхідності запуску додатку на пристрої або симуляторі (див. рисунок 1.7). Це значно спрощує роботу з великими інтерфейсами, оскільки розробники можуть попередньо переглядати та налаштовувати кожен компонент окремо, забезпечуючи його коректне відображення.

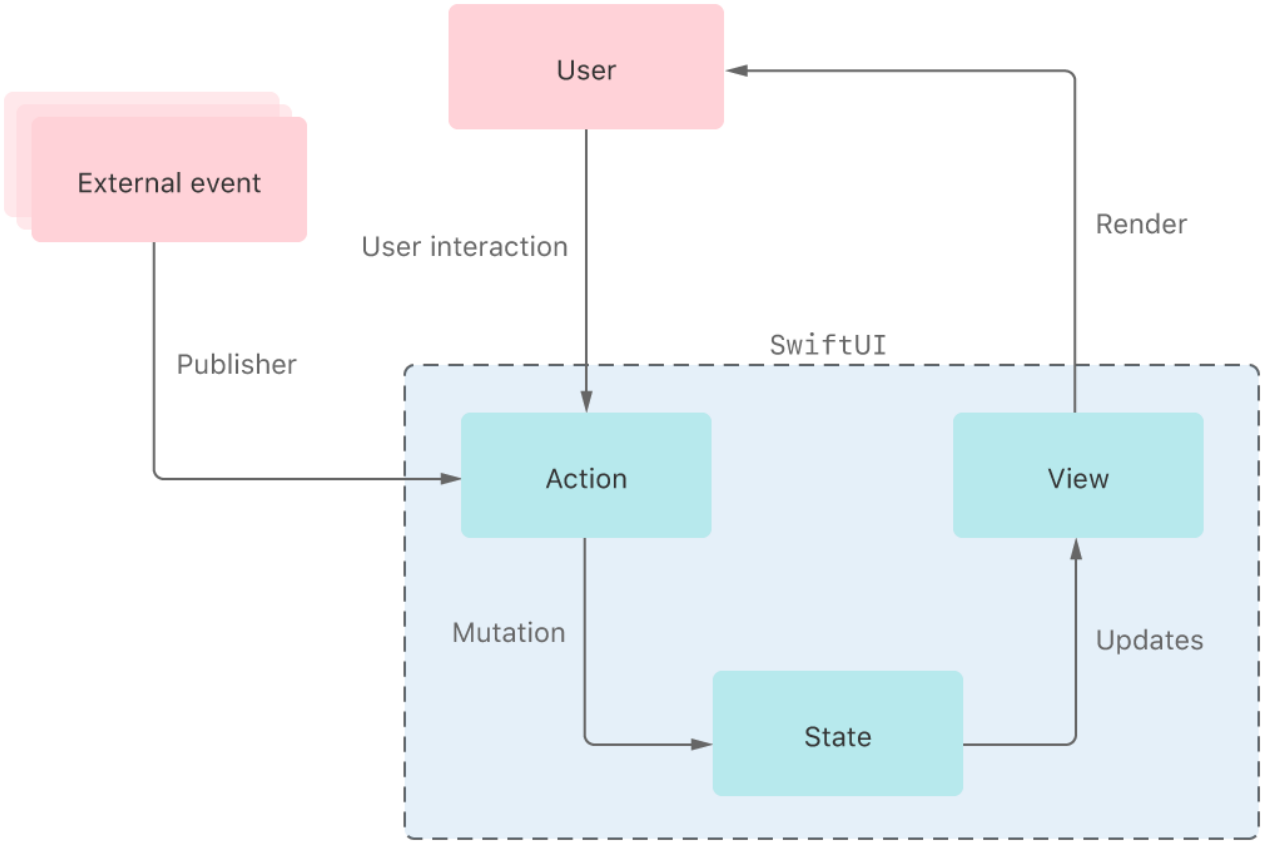


Рисунок 1.7 – Схема управління state у застосунку SwiftUI

Таким чином, використання нативної розробки з Swift, SwiftUI та Core ML забезпечує високу продуктивність [20], адаптивність та інтерактивність додатків, що робить цей підхід оптимальним для реалізації складних мовних моделей на мобільних платформах iOS.

1.4 Постановка задач дослідження

Провівши аналіз стану розвитку систем оптимізації великих мовних

моделей для мобільних платформ, було визначено основні завдання, які необхідно вирішити в процесі дослідження. Основні задачі включають:

- розробити метод гібридного адаптивного скорочення рангу;
- розробити метод динамічного налаштування ваг на основі апаратних характеристик пристрою;
- розробити метод квантової компресії мовних моделей для мобільних пристроїв;
- розробити програмні модулі компонентів мобільного застосунку «OptiLang» для оптимізації великих мовних моделей;
- розробити зручний та зрозумілий графічний інтерфейс;
- повести експериментальні дослідження розроблених методів;
- провести тестування розробленої iOS-системи;
- розробити інструкцію користувача та описати системні вимоги розробленої програми.

Технічне завдання на розробку наведено в додатку А.

1.5 Висновки

У першому розділі було проведено аналіз підходів до оптимізації великих мовних моделей та їх адаптації під мобільні iOS-системи. Проведено порівняння існуючих засобів, таких як Core ML Tools, TensorFlow Lite, ONNX Runtime Mobile, Hugging Face Transformers Lite та NVIDIA TensorRT. Результати аналізу вказують на те, що більшість наявних рішень не забезпечують достатнього рівня адаптації моделей до обмежених ресурсів мобільних пристроїв, що знижує ефективність їх використання.

Було обрано підхід нативної розробки на платформі iOS із використанням Swift і SwiftUI, що дозволяє максимально адаптувати інтерфейси користувача до вимог і специфіки iOS. У результаті аналізу основних методів для створення UI було обрано SwiftUI, оскільки цей фреймворк забезпечує високу гнучкість та інтерактивність для користувачів.

Було сформульовано основні задачі даслідження.

2 РОЗРОБКА МЕТОДІВ ОПТИМІЗАЦІЇ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ

2.1 Архітектурне проєктування програмної системи

При розробці додатку «OptiLang» було обрано архітектурний шаблон Model-View-ViewModel (MVVM), що забезпечує розділення бізнес-логіки та інтерфейсу користувача [21]. Це розділення критичне для додатка, що інтегрує великі мовні моделі, оскільки забезпечує масштабованість і підтримку протягом усього життєвого циклу додатку. Взаємодія між компонентами MVVM зображена на рисунку 2.1.

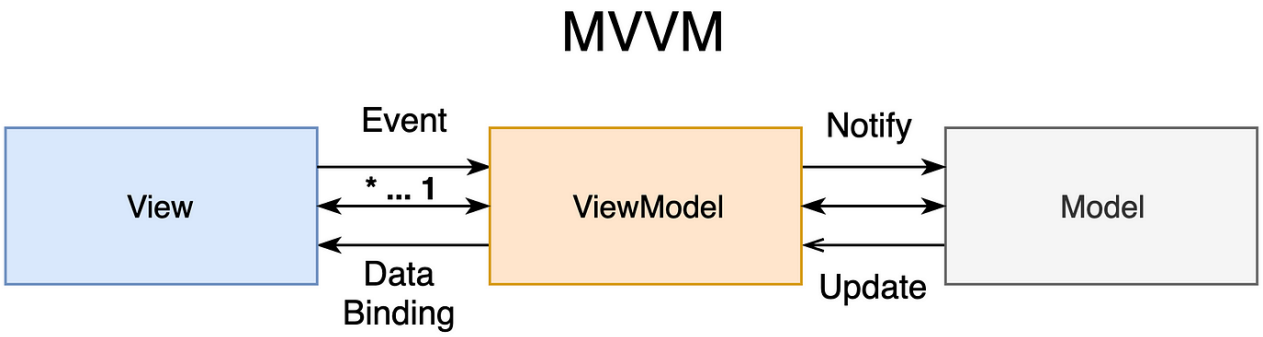


Рисунок 2.1 – Схема взаємодії Model, View і ViewModel у шаблоні MVVM

Архітектура MVVM включає три основні компоненти:

- Model – відповідає за управління даними і реалізує бізнес-логіку. У додатку «OptiLang» Model обробляє великі мовні моделі та виконує обчислення, пов’язані з адаптацією під технічні характеристики пристрою.
- View – відповідає за відображення інтерфейсу користувача та взаємодію з ним, реалізований у SwiftUI як набір декларативних компонентів, що автоматично оновлюються відповідно до змін у ViewModel.
- ViewModel – посередник між Model і View, який обробляє дані та забезпечує зв'язок між ними, використовуючи SwiftUI Binding. Завдяки зв'язкам (bindings) у SwiftUI, ViewModel передає оновлення з Model до View, що дозволяє відображенню відповідати актуальним даним без необхідності додаткового коду.

MVVM є найбільш оптимальним вибором для «OptiLang», оскільки він спрощує управління станом, дозволяє налаштувати двосторонні зв'язки між View і ViewModel за допомогою SwiftUI Binding та підвищує продуктивність. У рамках SwiftUI, binding-зв'язки дозволяють View отримувати оновлення з ViewModel в реальному часі, зберігаючи синхронність між інтерфейсом та даними. Ця особливість є критичною для додатка, який використовує великі мовні моделі, оскільки мінімізує затримки при відображенні результатів оптимізації.

Альтернативні архітектури, такі як Model-View-Controller (MVC), є менш оптимальними у цьому контексті через обмежену масштабованість і тісну зв'язаність між компонентами. MVC покладає значне навантаження на Controller, що відповідає і за логіку, і за взаємодію з користувачем, що ускладнює підтримку і тестування додатка. В архітектурі MVC Controller одночасно взаємодіє з Model і View, що призводить до утворення монолітного коду, особливо в додатках з високою складністю логіки, таких як «OptiLang».

Архітектура MVP (Model-View-Presenter) також розглядається як альтернатива, проте вона має суттєві недоліки для додатків на SwiftUI. У MVP Presenter бере на себе всі функції з управління View та Model, що часто призводить до перенавантаження Presenter складною логікою. Крім того, MVP не підтримує двосторонні зв’язки між компонентами, що робить його менш гнучким і менш придатним для інтерактивних інтерфейсів, необхідних для «OptiLang».

Основні переваги MVVM включають:

- Чітке розділення відповідальності: Model обробляє дані, View відображає інформацію, а ViewModel виступає посередником, забезпечуючи передачу даних між Model і View.
- Підтримка SwiftUI Binding: зв'язки забезпечують двосторонню синхронізацію між інтерфейсом та даними, що дозволяє View автоматично оновлюватися у відповідь на зміни у ViewModel, що особливо важливо для інтерактивних застосунків.

- Спрощене тестування: кожен компонент може бути протестований незалежно, що знижує ризик помилок та підвищує надійність системи.

Для створення користувацького інтерфейсу було використано SwiftUI, що дозволяє будувати декларативний UI з використанням компонентів VStack, HStack та ZStack для компонування елементів. Декларативний підхід SwiftUI дозволяє визначити, як має виглядати інтерфейс, а система автоматично забезпечує його відповідність до поточного стану [22]. Це зменшує обсяг коду та полегшує підтримку інтерфейсу (див. рисунок 2.2).

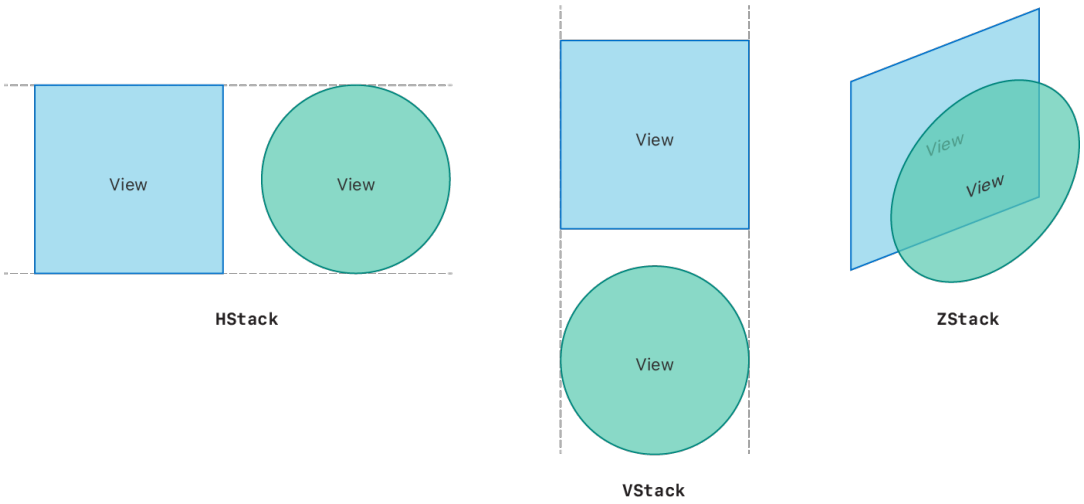


Рисунок 2.2 – Компонування елементів інтерфейсу в SwiftUI за допомогою VStack, HStack і ZStack

SwiftUI також підтримує функцію Preview, яка дозволяє розробникам переглядати зміни в інтерфейсі без необхідності запуску на реальному пристрої, що значно прискорює процес розробки [23].

Ключовою перевагою SwiftUI є гнучка інтеграція з Core ML, що забезпечує можливість локального виконання мовних моделей на пристрої. Core ML дозволяє виконувати обчислення безпосередньо на мобільному пристрої, що зменшує затримки та оптимізує споживання енергії. Це критично для додатка, який має обробляти великі мовні моделі, знижуючи навантаження на CPU і забезпечуючи ефективну роботу в межах обмежених ресурсів мобільних пристроїв [24].

Переваги SwiftUI в контексті реалізації «OptiLang»:

- Адаптивний та інтерактивний інтерфейс: SwiftUI автоматично адаптується до різних розмірів екранів та підтримує оновлення у реальному часі завдяки binding-зв'язкам, що дозволяє створити динамічний UI.
- Економія коду: завдяки декларативному підходу, SwiftUI потребує менше коду, ніж UIKit або інші фреймворки, що знижує ризик помилок і спрощує підтримку.
- Підтримка двосторонньої синхронізації: SwiftUI binding дозволяє автоматично оновлювати UI відповідно до змін у даних у ViewModel, що критично для ефективної роботи з мовними моделями, що постійно оновлюються.

На відміну від традиційних UI на основі UIKit, SwiftUI не використовує XML для визначення макетів, що спрощує керування інтерфейсом і робить його гнучкішим для змін у додатку. Це важливо для системи, яка потребує частого оновлення та адаптації до специфічних вимог (див. рисунок 2.3).

Рисунок 2.3 – Декларативний підхід до визначення інтерфейсу користувача в SwiftUI

Таким чином, вибір архітектури MVVM у поєднанні з SwiftUI дозволяє забезпечити високу гнучкість, масштабованість і ефективність при розробці додатку для оптимізації великих мовних моделей. Це забезпечує підтримку адаптивних інтерфейсів, ефективне управління мовними моделями за

допомогою Core ML та спрощене тестування кожного компонента, що критично для додатка, який обробляє великі обсяги даних і потребує швидкої реакції на зміни.

2.2 Розробка методу гібридного адаптивного скорочення рангу

У ході дослідження було проаналізовано різноманітні методи оптимізації великих мовних моделей для мобільних платформ, зокрема зниження розмірності та адаптацію під конкретні апаратні характеристики пристрою. Гібридне адаптивне скорочення рангу (Hybrid Adaptive Rank Reduction – HARR) розроблено як вдосконалений метод оптимізації великих мовних моделей для мобільних пристроїв [2]. Цей підхід поєднує різні техніки зменшення розмірності та адаптивне налаштування рангу, що дозволяє забезпечити ефективну роботу моделей у середовищах з обмеженими ресурсами, зберігаючи при цьому точність. Основна проблема великих мовних моделей у мобільних застосунках полягає у високих вимогах до обчислювальної потужності та пам’яті, що зазвичай обмежені на мобільних пристроях. Метод HARR вирішує це завдання, комбінуючи статичну оптимізацію параметрів із динамічним налаштуванням, яке враховує технічні можливості конкретного пристрою.

На відміну від стандартних методів, таких як Low-Rank Adaptation (LoRA) та квантова компресія [2], HARR дозволяє гнучко налаштовувати параметри моделі, адаптуючись до поточних можливостей мобільного пристрою. LoRA, наприклад, зосереджується на зниженні рангу шляхом фіксації певної кількості параметрів, що знижує потребу в пам'яті, проте не надає можливості динамічного налаштування залежно від наявних ресурсів. Таким чином, LoRA ефективний для оптимізації моделей на рівні серверів, але його фіксована структура стає обмеженням при використанні на мобільних пристроях із варіативними технічними характеристиками.

Метод квантової компресії, який використовує зменшення розрядності чисел для представлення вагових коефіцієнтів, знижує обсяг пам’яті, проте може призвести до значної втрати точності. Особливо це відчутно на мобільних

пристроях, де низька розрядність може вплинути на обробку мовних моделей через неточні обчислення. HARR, навпаки, застосовує адаптивний підхід до зниження рангу, що дозволяє зберігати високу точність, підтримуючи ефективність роботи моделі на мобільних платформах.

У межах HARR модель поділяється на два основні компоненти: базову модель, що зберігає основні параметри та структуру, та адаптивний компонент, який дозволяє налаштовувати модель на специфічні дані чи задачі. Адаптивний компонент відповідає за оптимізацію обчислювальних ресурсів, що особливо важливо для мобільних пристроїв із різними характеристиками продуктивності. Такий підхід дозволяє моделі не тільки ефективно працювати з обмеженими ресурсами, але й динамічно адаптуватися до зміни навантаження, що є важливим фактором для мобільних застосунків.

Застосування гібридного адаптивного скорочення рангу дозволяє вирішити проблему зниження продуктивності, пов'язану з передачею великого обсягу даних між пристроєм і сервером, оскільки адаптація виконується локально [3]. На відміну від традиційних централізованих підходів, коли для обробки моделей використовується обробка на сервері, HARR дозволяє зменшити затримку, оптимізуючи модель безпосередньо на мобільному пристрої. Цей підхід не тільки знижує вимоги до серверних ресурсів, але й забезпечує незалежну роботу моделі навіть за відсутності стабільного підключення до мережі.

Одним із основних аспектів у розробці HARR є підтримка узгодженості даних та адаптивне налаштування, що досягається завдяки алгоритмам консенсусу на зразок Raft. Алгоритм Raft дозволяє зберігати актуальний стан системи в розподіленому середовищі та забезпечує надійність, що є ключовим для мобільних платформ, де збої або перебої у зв'язку можуть призвести до втрати даних. Raft дає можливість зберігати синхронізований стан моделі на всіх пристроях у межах кластеру, що гарантує стабільність системи навіть за нестабільного з’єднання.

Гібридне адаптивне скорочення рангу також відзначається низкою переваг: по-перше, воно забезпечує гнучкість налаштування параметрів, що дозволяє

підвищити ефективність використання ресурсів мобільних пристроїв; по-друге, цей метод знижує обсяг пам'яті, необхідний для зберігання моделей, що робить його ідеальним для пристроїв з обмеженим обсягом оперативної пам’яті. Крім того, HARR мінімізує затримки, оскільки оптимізація виконується локально, що підвищує швидкість роботи мобільних додатків, особливо у випадках реального часу.

Метод гібридного адаптивного скорочення рангу (HARR) передбачає покрокову реалізацію оптимізації великих мовних моделей для мобільних пристроїв за допомогою поєднання статичних та динамічних технік зменшення розмірності. Основні етапи методу:

1. Ініціалізація базової моделі з налаштуванням основних параметрів, що відповідають загальним вимогам до продуктивності мобільних пристроїв.

2. Проведення початкової стадії скорочення рангу за допомогою фіксованої кількості низькорівневих параметрів. Для цього використовується попередня обробка моделі з технікою Low-Rank Approximation (LRA), що знижує обсяг пам'яті, необхідний для базового рівня.

3. Налаштування адаптивного рангового компонента, який дозволяє проводити подальше скорочення рангу з урахуванням технічних характеристик конкретного пристрою. Цей етап виконується через адаптивну оптимізацію параметрів, яка визначає, наскільки зменшення розмірності може бути досягнуте без значної втрати точності.

4. Визначення оптимального рівня компресії параметрів для конкретного пристрою шляхом аналізу його обчислювальної потужності та наявної пам’яті. Це реалізується через підбір параметрів квантової компресії та врахування особливостей обробки низькорозрядних чисел на процесорі.

5. Використання адаптивного налаштування вагових коефіцієнтів у моделі, яке дозволяє збалансувати рівень компресії та точність роботи. На основі вимірювання продуктивності на етапі 4 визначаються оптимальні вагові коефіцієнти, які зберігають баланс між точністю та ефективністю.

6. Збереження результатів компресії та оптимізації у вигляді

спеціалізованої структури даних для подальшого використання на конкретному пристрої. Для цього застосовується кешування оптимізованих параметрів, що забезпечує збереження результатів для повторного використання без додаткових витрат обчислювальних ресурсів.

7. Валідація роботи моделі після компресії за допомогою тестування на відповідність цільовим показникам точності та продуктивності. Якщо модель відповідає заданим критеріям, оптимізована версія зберігається як кінцевий результат.

8. Динамічна адаптація параметрів під час роботи моделі у разі змін у навантаженні або умовах роботи. Це забезпечується адаптивною структурою HARR, яка автоматично регулює параметри на основі поточного стану пристрою, забезпечуючи постійну ефективність роботи.

Таким чином, в підрозділі розроблено вдосконалений метод гібридного адаптивного скорочення рангу (HARR), що поєднує адаптивні та традиційні техніки зниження рангу, забезпечуючи ефективну роботу великих мовних моделей на мобільних платформах із мінімальними втратами точності.

2.3 Розробка методу динамічного налаштування ваг на основі апаратних характеристик пристрою

У ході дослідження було проаналізовано різноманітні методи адаптації великих мовних моделей до особливостей апаратного забезпечення мобільних пристроїв. Метод динамічного налаштування ваг на основі апаратних характеристик пристрою (Dynamic Hardware-Based Weight Adjustment – DHWA) розроблено як інноваційний підхід, що дозволяє враховувати різні технічні характеристики мобільних пристроїв для забезпечення оптимальної продуктивності моделі. На відміну від стандартних методів компресії та оптимізації, DHWA використовує динамічний підхід, який автоматично налаштовує вагові коефіцієнти та обчислювальні ресурси залежно від технічних параметрів конкретного пристрою, таких як обсяг оперативної пам'яті, потужність графічного процесора та наявність багатоядерних процесорів.

DHWA вирішує задачу підтримки стабільної роботи мовних моделей на мобільних платформах із різними технічними можливостями. Існуючі методи, такі як загальна квантова компресія або статична оптимізація, знижують вимоги до обчислень та пам'яті, але не забезпечують належної адаптації до специфічних характеристик пристрою, що може призвести до втрати точності або ефективності на певних конфігураціях. DHWA, навпаки, інтегрує функціонал динамічної оптимізації, що враховує апаратні ресурси пристрою та адаптує модель у режимі реального часу для максимальної продуктивності.

Ключовою особливістю DHWA є його здатність динамічно регулювати вагові коефіцієнти моделі на основі виявлених характеристик пристрою. Наприклад, пристрої з обмеженим обсягом пам’яті можуть автоматично знижувати розрядність вагових коефіцієнтів та використовувати більш агресивну компресію. Для пристроїв з потужними графічними можливостями передбачено інший режим, де акцент зроблено на підвищенні точності моделі за рахунок меншої компресії. Такий адаптивний підхід дозволяє забезпечити баланс між продуктивністю і точністю, виходячи з обмежень конкретного пристрою.

Метод DHWA виконує налаштування ваг шляхом аналізу обчислювальної потужності пристрою та вибору відповідного набору параметрів для конкретної конфігурації. Результати аналізу показують, що такий підхід забезпечує значне зниження обчислювальних витрат, дозволяючи зменшити кількість операцій, необхідних для обробки мовних моделей, а також оптимізувати споживання пам’яті [24]. Завдяки цьому підвищується стабільність та швидкість обробки запитів на пристроях з різними конфігураціями.

Метод DHWA реалізує наступні основні етапи адаптивного налаштування ваг для кожного мобільного пристрою:

1. Виявлення апаратних характеристик. На першому етапі DHWA збирає дані про основні параметри пристрою, такі як обсяг оперативної пам'яті, тип графічного процесора, кількість ядер центрального процесора та підтримка багатозадачності. Ці дані є основою для подальшого налаштування вагових коефіцієнтів.

2. Ініціалізація моделі з базовими параметрами. Модель завантажується з початковими, нейтральними вагами, що відповідають загальним вимогам. Усі наступні налаштування базуються на цих початкових параметрах, які коригуються для конкретного пристрою.

3. Динамічна компресія вагових коефіцієнтів. Залежно від технічних параметрів, DHWA знижує або підвищує розрядність вагових коефіцієнтів, а також обирає інтенсивність квантової компресії. Наприклад, на пристроях з обмеженою оперативною пам'яттю розрядність ваг може знижуватися, а більш потужні пристрої можуть використовувати більш високі розрядності для підвищення точності.

4. Оптимізація параметрів обчислювального процесу. Визначається, яка частина обчислювальних операцій може виконуватись на графічному процесорі, і чи можна активувати режим багатопотокового оброблення. Якщо пристрій підтримує багатозадачність, модель налаштовується на паралельну обробку, що суттєво підвищує швидкість виконання задач.

5. Тестування оптимізованої моделі. Після адаптації ваг DHWA запускає тестовий режим, у якому вимірюються продуктивність і точність моделі на конкретному пристрої. Якщо результати відповідають заданим критеріям, адаптована версія моделі зберігається для подальшого використання.

6. Динамічне коригування параметрів під час експлуатації. DHWA дозволяє моделі адаптуватися до змінних умов, таких як перехід у режим енергозбереження або підключення до зарядного пристрою. У випадку таких змін модель автоматично коригує свої параметри для забезпечення оптимальної роботи.

7. Кешування результатів адаптації. Всі оптимізовані параметри зберігаються у спеціалізованій структурі для подальшого використання без необхідності повторного налаштування, що дозволяє зменшити час на адаптацію при повторному запуску моделі на одному пристрої.

DHWA дозволяє підвищити гнучкість адаптації мовних моделей, що є важливим для мобільних пристроїв з різними характеристиками [24]. Головними

перевагами цього методу є: значне зниження потреб у пам'яті, підвищення швидкості обробки запитів завдяки оптимізації вагових коефіцієнтів, можливість використання моделі навіть за обмежених ресурсів, а також підвищення точності та продуктивності роботи.

Таким чином, у підрозділі розроблено вдосконалений метод динамічного налаштування ваг на основі апаратних характеристик пристрою (DHWA), що забезпечує ефективну адаптацію мовних моделей до мобільних платформ різної потужності та конфігурації.

2.4 Розробка методу квантової компресії мовних моделей для мобільних пристроїв

В ході дослідження було визначено, що використання великих мовних моделей на мобільних пристроях обмежується високими вимогами до обсягу пам’яті та обчислювальної потужності. Існуючі методи компресії, такі як стандартна квантова компресія, дозволяють значно зменшити розмір моделі шляхом скорочення розрядності чисел, які використовуються для представлення вагових коефіцієнтів. Однак вони часто призводять до суттєвої втрати точності, що може знижувати ефективність мовних моделей [25]. У цьому розділі розглядається вдосконалений підхід до квантової компресії, що дозволяє досягати балансу між зниженням розміру та збереженням точності моделі.

Основна проблема при використанні квантової компресії полягає в тому, що скорочення розрядності чисел до низького рівня призводить до спотворень вагових коефіцієнтів, що може знижувати здатність моделі до коректної обробки природної мови. Традиційні методи квантової компресії переважно застосовують однакову розрядність для всіх частин моделі, що не враховує різних ступенів значущості окремих параметрів. Вдосконалений метод квантової компресії, розроблений для мобільних платформ, дозволяє застосовувати диференційовані рівні компресії до різних частин моделі, оптимізуючи таким чином як розмір, так і точність.

Цей метод базується на адаптивному підході до зниження розрядності

залежно від того, наскільки критичним є кожен параметр для загальної продуктивності моделі [26]. Зокрема, параметри, що мають менший вплив на точність, можуть бути скорочені до нижчої розрядності, тоді як для ключових параметрів залишається вищий рівень точності. Такий підхід дозволяє скоротити розмір моделі до 50% без суттєвої втрати в точності, що робить його ідеальним для мобільних додатків, де пам’ять і ресурси є обмеженими.

Метод адаптивної квантової компресії реалізується наступними етапами:

1. Ініціалізація вихідної моделі. Спочатку модель завантажується у незмінному вигляді для аналізу її основних параметрів. На цьому етапі визначаються основні вагові коефіцієнти, критичні для збереження точності.

2. Сегментація вагових коефіцієнтів. Всі параметри моделі сегментуються на групи залежно від їхнього впливу на кінцевий результат. Наприклад, параметри, що визначають основну структуру моделі, виділяються окремо від тих, що впливають лише на другорядні аспекти функціонування.

3. Визначення рівнів компресії. Для кожної групи параметрів встановлюється оптимальний рівень розрядності. Вдосконалений метод квантової компресії дозволяє встановлювати розрядність для кожної групи окремо, наприклад, критичні параметри можуть залишатися в 8- або 16-бітному представленні, тоді як менш значущі знижуються до 4-бітного рівня.

4. Квантування ваг. Після визначення оптимальних рівнів для кожної групи параметри моделі піддаються квантуванню відповідно до встановлених розрядностей. Це знижує обсяг пам’яті, необхідний для зберігання моделі, забезпечуючи значне скорочення розміру.

5. Валідація точності. Після компресії модель проходить тестування, щоб визначити, наскільки ефективно зберігається точність. Якщо результати тестування відповідають заданим критеріям, то компресія вважається успішною. За потреби рівні компресії можуть бути повторно скориговані для збереження точності.

6. Збереження оптимізованої моделі. Після тестування компресована модель зберігається у форматі, оптимізованому для подальшого використання на

мобільних пристроях. Ця модель є компактною та здатна працювати з обмеженими ресурсами, забезпечуючи належну продуктивність і точність.

7. Динамічне коригування компресії під час роботи. На останньому етапі метод передбачає можливість адаптивного налаштування компресії залежно від стану ресурсів пристрою, наприклад, у режимі енергозбереження або при підключенні до зарядного пристрою. Це забезпечує гнучкість та ефективність моделі за різних умов використаннях [27].

Запропонований метод дозволяє знизити вимоги до пам’яті та ресурсів, необхідних для роботи мовної моделі на мобільному пристрої, і при цьому зберігає високу точність результатів. Однією з головних переваг є можливість налаштування рівнів компресії, що дозволяє максимально адаптувати модель до можливостей конкретного пристрою. Крім того, вдосконалена квантова компресія знижує затримку обробки запитів, що робить метод оптимальним для застосувань у реальному часі.

Таким чином, у підрозділі розроблено вдосконалений метод квантової компресії мовних моделей для мобільних пристроїв, що дозволяє зменшити розмір моделі без значної втрати точності, забезпечуючи ефективну роботу в умовах обмежених ресурсів.

2.5 Розробка блок-схем алгоритмів роботи запропонованих методів

Для методів, розроблених у попередніх підрозділах 2.2-2.4, були створені відповідні алгоритми їх реалізації та блок-схеми [28]. Алгоритм гібридного адаптивного скорочення рангу (Hybrid Adaptive Rank Reduction, HARR) описує загальний потік оптимізації мовної моделі для мобільних платформ з обмеженими ресурсами. Блок-схема алгоритму HARR наведена на рисунку 2.4.

Розглянемо більш детально кроки алгоритму HARR:

Крок 1. Початок.

Крок 2. Ініціалізація моделі. Завантаження моделі.

Крок 3. Збір апаратних характеристик пристрою. Збираються дані про пам’ять і процесор.

Крок 4. Перевірка пам’яті (memoryAvailable). Якщо memoryAvailable більша або рівна мінімального рівня, перехід до кроку 5. Інакше – перехід до кроку 8 для додаткового зниження розмірності моделі.

Крок 5. Початкове скорочення рангу за допомогою Low-Rank Approximation (LRA).

Крок 6. Видалення всіх тимчасових файлів та даних, що зберігалися у кеші для звільнення оперативної пам'яті перед подальшою оптимізацією.

Крок 7. Збереження поточних параметрів моделі та ініціалізація адаптивного компонента.

Крок 8. Адаптивне налаштування для обмежених ресурсів та компресія.

Крок 9. Перевірка відповідності компресії. Якщо результат компресії відповідає критеріям точності, перехід до кроку 10. Інакше – повернутися до кроку 8 для додаткової компресії.

Крок 10. Квантування параметрів моделі. Початок циклу по кожному параметру моделі для зменшення розрядності. Якщо коефіцієнт не критичний, квантується до меншої розрядності.

Крок 11. Адаптивне налаштування ваг. Здійснюється налаштування ваг у моделі, щоб забезпечити необхідний рівень точності. Встановлюються оптимальні значення ваг з урахуванням характеристик пристрою.

Крок 12. Валідація точності. Якщо точність в межах прийнятного діапазону, перехід до кроку 13. Інакше – повернення до кроку 8.

Крок 13. Динамічна адаптація. Під час використання здійснюється перевірка ресурсів і коригування ваг. Якщо ваги скориговані в межах прийнятного діапазону, перехід до кроку 14. Інакше – повернення до кроку 5.

Крок 14. Збереження моделі. Після завершення оптимізації збережені параметри моделі поміщаються в кеш для подальшого швидкого завантаження та використання.

Крок 15. Кінець.

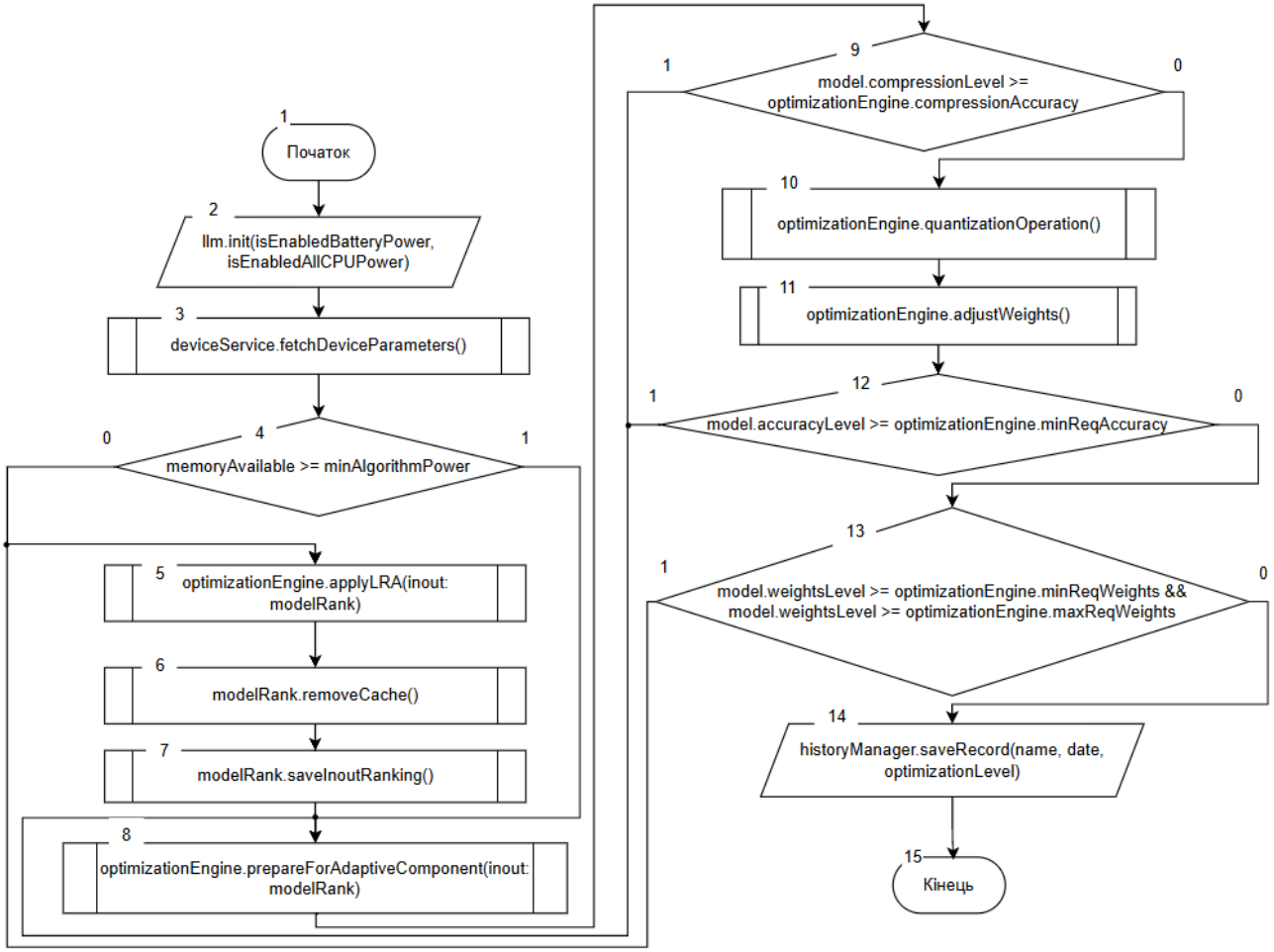


Рисунок 2.4 – Блок-схема алгоритму HARR

Розроблено алгоритм динамічного налаштування ваг на основі апаратних характеристик пристрою (DHWA), який забезпечує додаткову оптимізацію шляхом налаштування моделі на базі обчислювальних можливостей пристрою (див. рисунок 2.5).

Крок 1. Початок.

Крок 2. Ініціалізація моделі. Завантаження та отримання параметрів.

Крок 3. Збір інформації про пристрій. Дані про пам'ять і процесор. Зберігання результатів у змінні.

Крок 4. Перевірка пам'яті (memoryAvailable) Якщо пам’яті достатньо, перейти до кроку 5. Інакше – перейти до кроку 6 для компресії ваг.

Крок 5. Компресія ваг за умовами навантаження. Адаптація до середнього рівня ресурсу. Після компресії перехід до кроку 8.

Крок 6. Збільшення коефіцієнта зменшення для пам’яті. Збільшення рівня

компресії для критичних ваг.

Крок 7. Перевірка рівня компресії. Якщо компресія задовільна, перейти до кроку 8. Інакше – повернення до кроку 6.

Крок 8. Динамічна адаптація в реальному часі. Під час виконання з урахуванням зміни навантаження коригуються параметри.

Крок 9. Збереження результатів. Зберігання налаштувань у кеш.

Крок 10. Кінець.

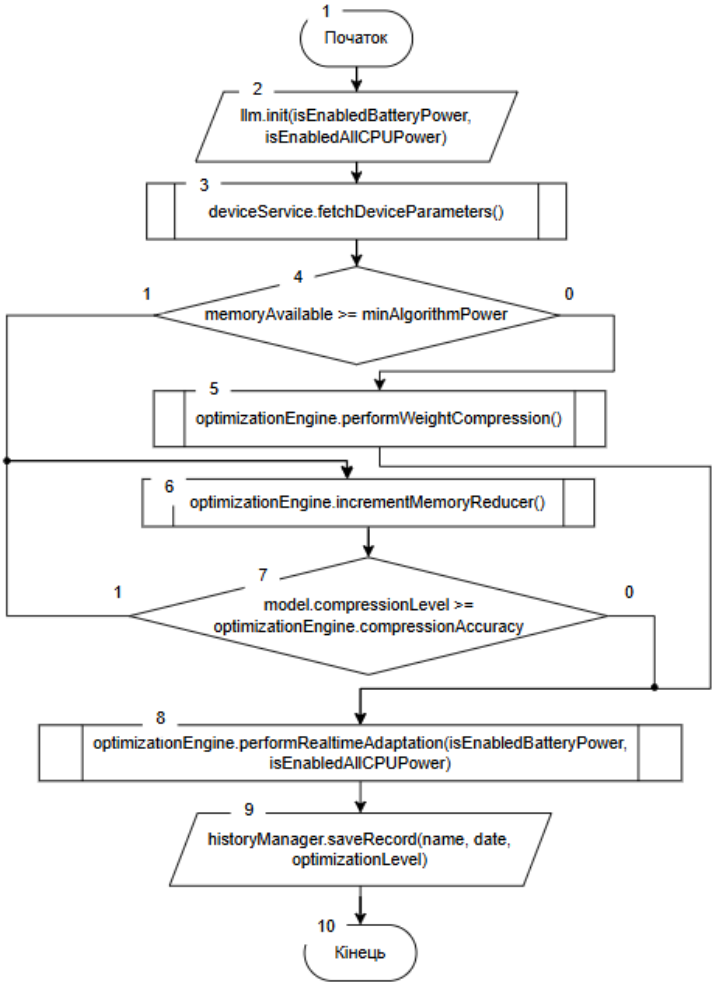


Рисунок 2.5 – Блок-схема алгоритму динамічного налаштування ваг

Розроблені алгоритми також враховують вдосконалений метод квантової компресії, що дозволяє зменшити розмір моделей мовних моделей без суттєвих втрат точності (див. рис. 2.6).

Крок 1. Початок.

Крок 2. Ініціалізація моделі. Завантаження та отримання параметрів.

Крок 3. Сегментація параметрів. Поділ параметрів на критичні й некритичні.

Крок 4. Високоточне квантування критичних параметрів.

Крок 5. Перевірка на квантування всіх параметрів. Якщо є неквантовані параметри – повернення до кроку 4. Інакше – перехід до кроку 6.

Крок 6. Квантування некритичних параметрів зі стандартною точністю.

Крок 7. Перевірка на квантування всіх параметрів. Якщо є неквантовані параметри – повернення до кроку 6. Інакше – перехід до кроку 8.

Крок 8. Валідація точності. Тестування моделі на точність. Якщо точність відповідає вимогам, перейти до кроку 9. Інакше – повернення до кроку 3.

Крок 9. Збереження параметрів у кеш.

Крок 10. Кінець.

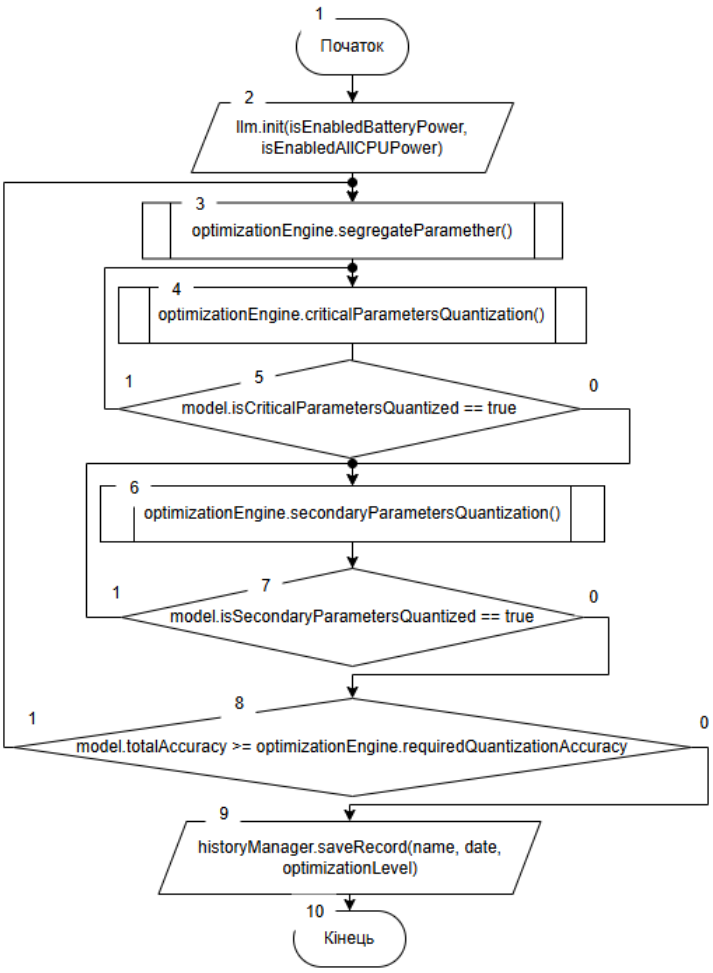


Рисунок 2.6 – Блок-схема алгоритму квантової компресії

2.6 Розробка структури графічного інтерфейсу користувача

Графічний інтерфейс користувача (англ. Graphical User Interface, GUI) – це тип інтерфейсу, що дозволяє користувачам взаємодіяти з програмним забезпеченням на електронних пристроях за допомогою візуальних елементів, таких як кнопки, поля вводу, меню тощо. GUI спрощує сприйняття та використання програмного продукту, усуваючи необхідність вводити команди вручну в текстовому форматі. Завдяки інтуїтивно зрозумілій структурі та зручному розташуванню елементів користувацького інтерфейсу, забезпечується легкий доступ до основних функцій програми.

Застосунок для оптимізації мовних моделей використовує графічний інтерфейс, що складається з чотирьох основних екранних форм: екрану вибору файлу для оптимізації, екрану історії оптимізацій, екрана оптимізації та екрана налаштувань. Кожен з цих елементів інтерфейсу забезпечує зручний доступ до функціональних можливостей, необхідних для виконання завдань користувача. Для більшої наочності було розроблено структурну схему головного вікна програми, яка наведена на рисунку 2.7.

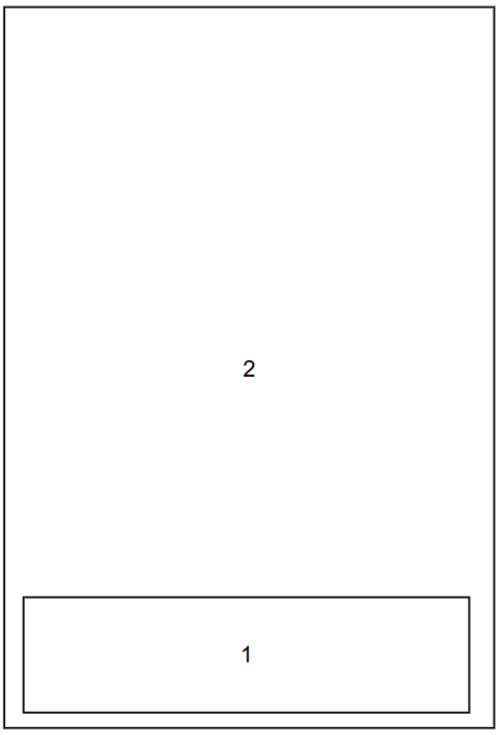


Рисунок 2.7 – Структурна схема головного вікна застосунку

Основні елементи інтерфейсу головного вікна:

1) нижня панель навігації, яка дозволяє користувачу швидко переходити між розділами «Історія», «Оптимізація» та «Налаштування»;

2) центральна робоча область, вміст якої змінюється відповідно до обраного розділу.

Такий підхід дозволяє користувачу швидко перемикатися між розділами, отримуючи доступ до необхідного функціоналу без зайвих кроків.

Розглянемо більш детально інтерфейс вкладки вибору файлу для оптимізації. Цей екран призначений для імпорту файлів, які користувач планує оптимізувати. Користувач має доступ до файлової системи пристрою, що дозволяє обрати файл з доступних на пристрої. Інтерфейс містить:

1) поле для пошуку файлів;

2) список файлів з інформацією про назву, дату створення та розмір.

Ця вкладка забезпечує швидкий доступ до файлової системи та дає можливість обрати модель для оптимізації. На рисунку 2.8 представлено структурну схему екрану вибору файлу.

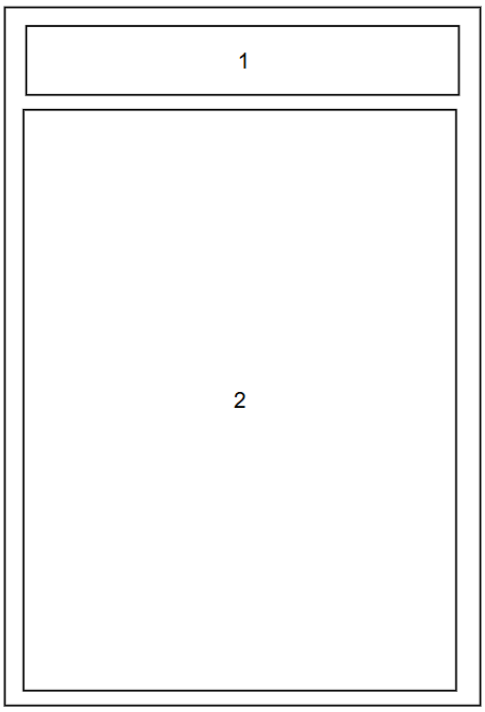


Рисунок 2.8 – Структурна схема екрану вибору файлу

Наступним елементом інтерфейсу є вкладка історії оптимізацій, яка дозволяє користувачу переглядати попередні результати оптимізації. Кожен запис містить наступні елементи:

1) назву файлу;
2) дату та час проведення оптимізації;
3) відсоткове значення оптимізації та тривалість процесу.

Даний розділ дозволяє зручно відстежувати результати попередніх оптимізацій, забезпечуючи користувача історичними даними для порівняння та повторного використання параметрів для схожих файлів.

На рисунку 2.9 представлено структурну схему інтерфейсу вкладки історії оптимізацій.

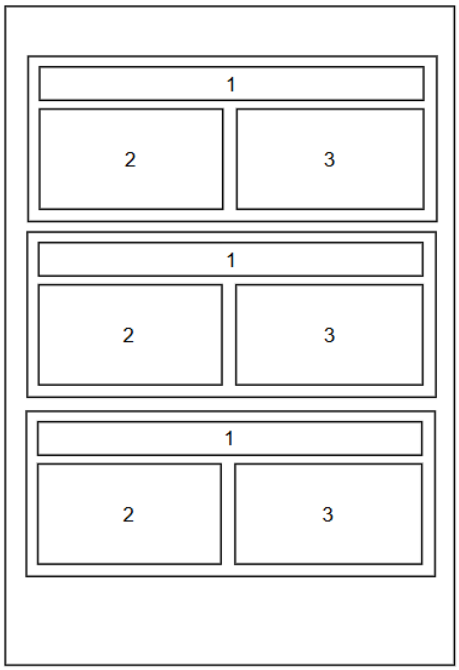


Рисунок 2.9 – Структурна схема екрану історії оптимізацій

Також розглянемо інтерфейс вкладки оптимізації. Ця вкладка є центральним елементом застосунку, що дозволяє користувачу здійснити оптимізацію обраного файлу. Вона містить такі елементи:

1) поле з назвою вибраного файлу;
2) кнопку «Імпортувати модель», що дозволяє вибрати новий файл для

оптимізації;

3) кнопку «Оптимізувати», що запускає процес оптимізації;

4) прогрес-бар для відображення стадій оптимізації, зокрема етапу «Застосування LoRA».

Під час оптимізації програма відображає поточний стан процесу, що дає змогу користувачу в реальному часі контролювати процес та бачити його стадії.

На рисунку 2.10 зображено структурну схему інтерфейсу вкладки оптимізації.

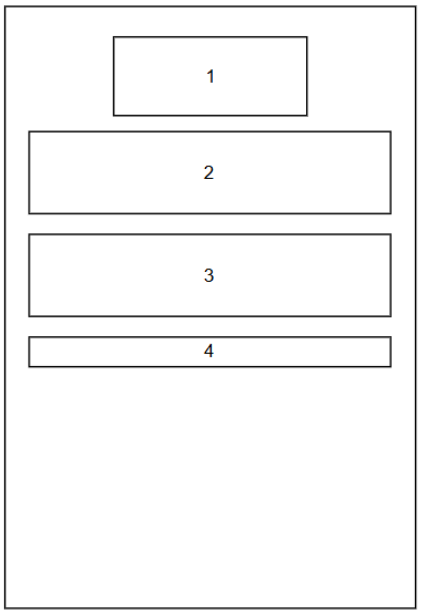


Рисунок 2.10 – Структурна схема екрану оптимізації

Останньою вкладкою є вкладка налаштувань, що надає користувачу можливість конфігурувати параметри оптимізації та інші опції програми. Основні елементи цього інтерфейсу:

1) перемикач для врахування рівня заряду акумулятора під час оптимізації;

2) перемикач для використання всіх ядер CPU для підвищення продуктивності;

3) можливість увімкнення або вимкнення повідомлень.

Цей розділ дозволяє налаштувати роботу програми відповідно до вимог

користувача та можливостей пристрою, забезпечуючи оптимальний баланс між продуктивністю та економією ресурсів.

Таким чином, у цьому підрозділі було розглянуто структуру графічного інтерфейсу користувача для створюваного програмного продукту. Було детально описано основні екрани програми, наведено структурні схеми головного вікна та вкладок вибору файлу, історії оптимізацій, оптимізації та налаштувань, а також охарактеризовано функціональні можливості кожного елемента інтерфейсу.

2.7 Висновки

У другому розділі були розроблені основні методи, необхідні для створюваного ПЗ з оптимізації великих мовних моделей для мобільних пристроїв, а саме: метод гібридного адаптивного скорочення рангу (HARR), що поєднує різні техніки зменшення розмірності та адаптивне налаштування рангу, дозволяючи оптимізувати великі мовні моделі під можливості конкретного мобільного пристрою; метод динамічного налаштування ваг на основі апаратних характеристик пристрою (DHWA), який автоматично підлаштовує вагові параметри моделі залежно від технічних можливостей пристрою, забезпечуючи ефективну роботу навіть за обмежених ресурсів; метод квантової компресії, що зменшує розрядність вагових коефіцієнтів моделей, оптимізуючи обсяг пам'яті без значних втрат точності.

Для кожного з методів було розроблено відповідний алгоритм та блок-схему, детально розглянуто кожен крок алгоритму з метою забезпечення чіткості та послідовності виконання процесу оптимізації.

Також було розроблено структуру графічного інтерфейсу користувача, що дозволяє легко взаємодіяти з програмою на всіх етапах оптимізації. Було розглянуто базові принципи GUI, зокрема для мобільних пристроїв, та детально описано елементи головного вікна програми, включаючи екрани вибору файлу, історії оптимізацій, процесу оптимізації та налаштувань.

3 РОЗРОБКА ПРОГРАМНИХ КОМПОНЕНТ ЗАСТОСУНКУ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного засобу

Для створення програмного забезпечення, яке орієнтоване на оптимізацію великих мовних моделей для мобільних платформ, було розглянуто кілька мов програмування, зокрема Swift, Python, Objective-C та C++. Вибір мови програмування є критично важливим, оскільки впливає на продуктивність, гнучкість, зручність розробки та інтеграцію із апаратними компонентами пристроїв. Наведемо детальний аналіз кожної з розглянутих мов.

Swift – сучасна мова програмування, розроблена Apple, яка дозволяє створювати продуктивні та безпечні додатки для iOS. Swift поєднує високу продуктивність із простим синтаксисом і підтримує всі найновіші технології Apple, включаючи Core ML для машинного навчання.

До основних переваг Swift варто віднести:

1. Продуктивність. Swift є компільованою мовою, що дозволяє досягти високої швидкості виконання програм. Це особливо важливо для обчислювально інтенсивних завдань, як-от обробка мовних моделей [27].

2. Екосистема Apple. Swift має розвинену інтеграцію з технологіями Apple, зокрема з Core ML, що полегшує використання оптимізованих для мобільних пристроїв моделей [28].

3. Сучасний синтаксис. Swift дозволяє створювати більш читабельний і чистий код, що знижує можливість помилок і спрощує процес розробки.

Серед основних недоліків Swift можна відзначити:

1. Обмеженість екосистеми. Swift застосовується в основному для розробки під iOS і macOS, що обмежує його використання в багатоплатформних рішеннях.

2. Новітність мови. Swift – відносно нова мова, і деякі бібліотеки та фреймворки ще не мають повної підтримки для специфічних задач, таких як обробка природної мови (NLP).

Python – одна з найбільш популярних мов програмування, широко використовувана в області машинного навчання та обробки даних завдяки наявності великої кількості бібліотек для обробки тексту та побудови мовних моделей.

До основних переваг Python варто віднести:

1. Гнучкість і простота. Python є універсальною мовою, яка завдяки простому синтаксису дозволяє швидко розробляти і тестувати нові алгоритми.

2. Екосистема для NLP і машинного навчання. Python має великий набір інструментів для роботи з природною мовою, таких як TensorFlow, PyTorch, NLTK, що робить його ідеальним для розробки і навчання мовних моделей.

3. Швидкість прототипування. Python дозволяє швидко створювати і тестувати нові ідеї, що скорочує час на розробку і створення MVP.

Серед основних недоліків Python можна відзначити:

1. Низька продуктивність. Python значно поступається за швидкістю виконання компільованим мовам, що ускладнює використання для обчислювально інтенсивних завдань на мобільних пристроях.

2. Відсутність нативної підтримки iOS. Python не є рідною мовою для iOS, що вимагає додаткових зусиль для інтеграції та знижує продуктивність на мобільних пристроях.

Objective-C – мова програмування, яка широко використовувалась для розробки додатків під iOS і macOS до появи Swift. Хоча її використовують все рідше, Objective-C все ще має важливе значення через сумісність із багатьма існуючими бібліотеками.

До основних переваг Objective-C варто віднести:

1. Сумісність із екосистемою Apple. Objective-C добре інтегрується з API та бібліотеками Apple, зокрема з Core ML.

2. Підтримка C-бібліотек. Objective-C дозволяє використовувати низькорівневі C-бібліотеки, що забезпечує додаткову гнучкість і можливість виконання низькорівневих оптимізацій.

3. Розвинена база бібліотек. Objective-C має багаторічну історію в

екосистемі Apple, що забезпечує стабільну підтримку багатьох бібліотек.

Серед основних недоліків Objective-C можна відзначити:

1. Складність синтаксису. Objective-C має складний і важкий для читання синтаксис, що може ускладнити розробку і підтримку коду.

2. Обмеженість у використанні. Objective-C застосовується майже виключно для розробки під iOS і macOS, що ускладнює розробку кросплатформених рішень.

3. Морально застаріла. Objective-C поступово витісняється Swift, який забезпечує кращу продуктивність і зручність розробки.

C++ – мова програмування, яка забезпечує високий рівень контролю над ресурсами системи, що робить її придатною для обчислювально інтенсивних завдань, таких як машинне навчання і оптимізація великих моделей.

До основних переваг C++ варто віднести:

1. Продуктивність. C++ забезпечує високу швидкість виконання коду завдяки компіляції в машинний код і широким можливостям оптимізації.

2. Широка підтримка інструментів. Завдяки своїй тривалій історії, C++ має багато бібліотек для роботи з машинним навчанням, що полегшує інтеграцію мовних моделей.

3. Сумісність із низькорівневими бібліотеками. C++ дозволяє використовувати бібліотеки на рівні системи, що полегшує виконання низькорівневих оптимізацій.

Серед основних недоліків C++ можна відзначити:

1. Складність у розробці. Складний синтаксис і необхідність ручного управління пам'яттю роблять C++ більш складною мовою для розробки і підтримки.

2. Відсутність нативної підтримки мобільних платформ. Хоча C++ можна використовувати на iOS та Android, його інтеграція в мобільні середовища потребує додаткових зусиль.

На основі перерахованих переваг та недоліків було зроблено порівняння цих чотирьох мов програмування за певними критеріями, що описані в таблиці

3.1. Розглянемо більш детально ці критерії, особливості оцінювання та виставлені бали.

Таблиця 3.1 – Порівняння мов програмування

Критерій	Swift	Python	Objective-C	C++
Продуктивність	1	0	1	1
Простота інтеграції	1	0	1	0.5
Підтримка мобільних платформ	1	0	1	0.5
Інструменти для обробки NLP	0.5	1	0.5	0.5
Швидкість розробки	1	1	0.5	0
Підсумок	4.5	2	4	2.5

Продуктивність розробленого ПЗ. Мови програмування Swift, Objective-C та C++ є компільованими, що дозволяє досягати високої швидкості виконання програмного забезпечення. Це є важливим фактором при розробці застосунків для оптимізації мовних моделей на мобільних пристроях, де обчислювальні ресурси обмежені. Python, як інтерпретована мова, має значно нижчу продуктивність, що обмежує його застосування для обчислювально інтенсивних завдань у мобільному середовищі.

Швидкість макетування та розробки. Для сучасних комерційних застосунків швидкість розробки є важливим параметром, оскільки дозволяє скоротити цикл розробки і швидко доставити MVP (англ. Minimal Viable Product). Python та Swift мають простий синтаксис, що полегшує та прискорює процес створення продукту. Objective-C має складніший синтаксис, а C++ потребує додаткових зусиль для управління пам’яттю, що може ускладнити розробку.

Інтеграція з мобільними платформами. Swift та Objective-C мають нативну підтримку в екосистемі Apple, що забезпечує просту інтеграцію з iOS та macOS.

C++ та Python не є рідними мовами для iOS і потребують додаткових інструментів для інтеграції, що ускладнює процес розробки на мобільних платформах.

Інструменти для обробки природної мови. Python має найбільш розвинену екосистему для роботи з NLP, що робить його ідеальним вибором для створення мовних моделей. Swift та Objective-C мають обмежену кількість бібліотек для обробки тексту, що може ускладнити розробку в деяких аспектах.

Розвиток екосистеми мови. Swift активно розвивається компанією Apple і має підтримку нових API та інструментів для роботи з машинним навчанням і оптимізацією моделей. Objective-C також підтримується, але розвивається повільніше через поступове витіснення Swift.

На основі результатів порівняння найбільш підходящою мовою програмування для реалізації програмного забезпечення для оптимізації великих мовних моделей на iOS є Swift. Swift поєднує високу продуктивність, простоту інтеграції в екосистему Apple та підтримку сучасних інструментів для роботи з машинним навчанням. Це робить Swift оптимальним вибором для розробки застосунку, що буде ефективно функціонувати в мобільному середовищі iOS, де вимоги до продуктивності та ефективності використання ресурсів особливо високі.

Таким чином, у цьому підрозділі було розглянуто переваги й недоліки мов програмування Swift, Python, Objective-C та C++. Проведено детальний аналіз кожної мови, сформовано таблицю порівняння за основними критеріями та обґрунтовано вибір Swift як оптимальної мови програмування для розробки програмного засобу.

3.2 Вибір середовища розробки

Інтегроване середовище розробки (англ. Integrated Development Environment, IDE) – це програмне забезпечення, яке надає розробникам широкий спектр інструментів для полегшення процесу розробки, тестування та налагодження програмного забезпечення. IDE зазвичай включає редактор коду,

компілятор або інтерпретатор, засоби налагодження, а також інструменти для інтеграції з системами контролю версій. Завдяки цим можливостям IDE створює централізоване середовище для ефективної роботи розробника, що дозволяє працювати з кодом у зручному інтерфейсі.

Для розробки програмного забезпечення з використанням Swift було проаналізовано можливості кількох популярних IDE: Xcode, AppCode і Visual Studio Code. Кожне з цих середовищ має свої особливості, що дозволяє розробникам використовувати його для створення мобільних додатків на iOS. Наведемо детальний аналіз кожного середовища.

Xcode – офіційне інтегроване середовище розробки від Apple, яке створене спеціально для розробки додатків під macOS та iOS. Xcode пропонує широкий набір інструментів для роботи зі Swift та Objective-C, що робить його вибором за замовчуванням для iOS-розробників.

Основні переваги Xcode:

- Повна інтеграція з екосистемою Apple. Xcode надає повний доступ до всіх інструментів та SDK, необхідних для розробки та тестування додатків на iOS, macOS, watchOS та tvOS.
- Інструменти для налагодження та оптимізації. Xcode включає такі інструменти, як Instruments, які дозволяють проводити детальний аналіз продуктивності, що є критичним для оптимізації великих мовних моделей.
- Вбудована підтримка Swift. Оскільки Xcode є офіційним середовищем, він має найбільш актуальні версії мови Swift і пропонує відмінні засоби для роботи з цією мовою, включаючи автозаповнення коду, рефакторинг та інспекцію помилок.

Основні недоліки Xcode:

- Тільки для macOS. Xcode доступний лише для macOS, що обмежує його використання на інших платформах.
- Потреба у високих апаратних ресурсах. Через складність середовища та його багатофункціональність, Xcode вимагає значних ресурсів системи для комфортної роботи.

Приклад графічного інтерфейсу наведено на рисунку 3.1.

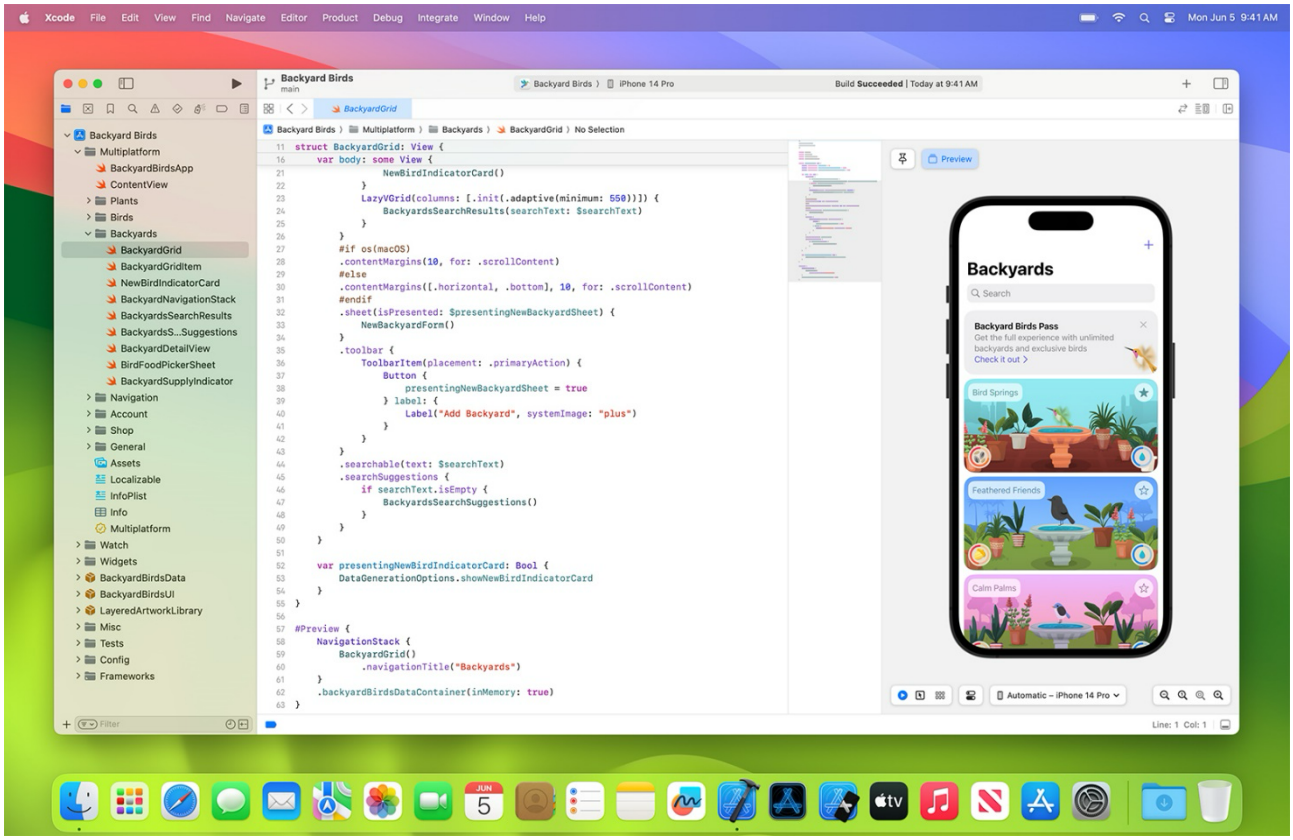


Рисунок 3.1 – Приклад інтерфейсу XCode

AppCode – інтегроване середовище розробки від компанії JetBrains, яке підтримує Swift, Objective-C та C++. AppCode базується на платформі IntelliJ IDEA, що забезпечує розвинені засоби для аналізу та рефакторингу коду.

Основні переваги AppCode:

- Інструменти для рефакторингу. JetBrains відомі своїми можливостями рефакторингу, і AppCode не є винятком. Це середовище підтримує потужні інструменти для організації та оптимізації коду.
- Сумісність з Xcode. AppCode працює з проектами Xcode і використовує його інструменти для компіляції та налагодження, що дозволяє розробникам працювати з інструментами Apple, але з удосконаленими можливостями IDE від JetBrains.
- Розширюваність. AppCode підтримує плагіни від платформи IntelliJ IDEA, що дозволяє додавати нові функції та інструменти для покращення

процесу розробки.

Основні недоліки AppCode:

- Залежність від Xcode. Для роботи з AppCode необхідно мати встановлений Xcode, оскільки AppCode використовує Xcode для компіляції та роботи з SDK.
- Ресурсоємність. Подібно до інших продуктів JetBrains, AppCode потребує достатньо потужний пристрій для комфортної роботи, особливо при розробці великих проєктів.

Приклад графічного інтерфейсу наведено на рисунку 3.2.

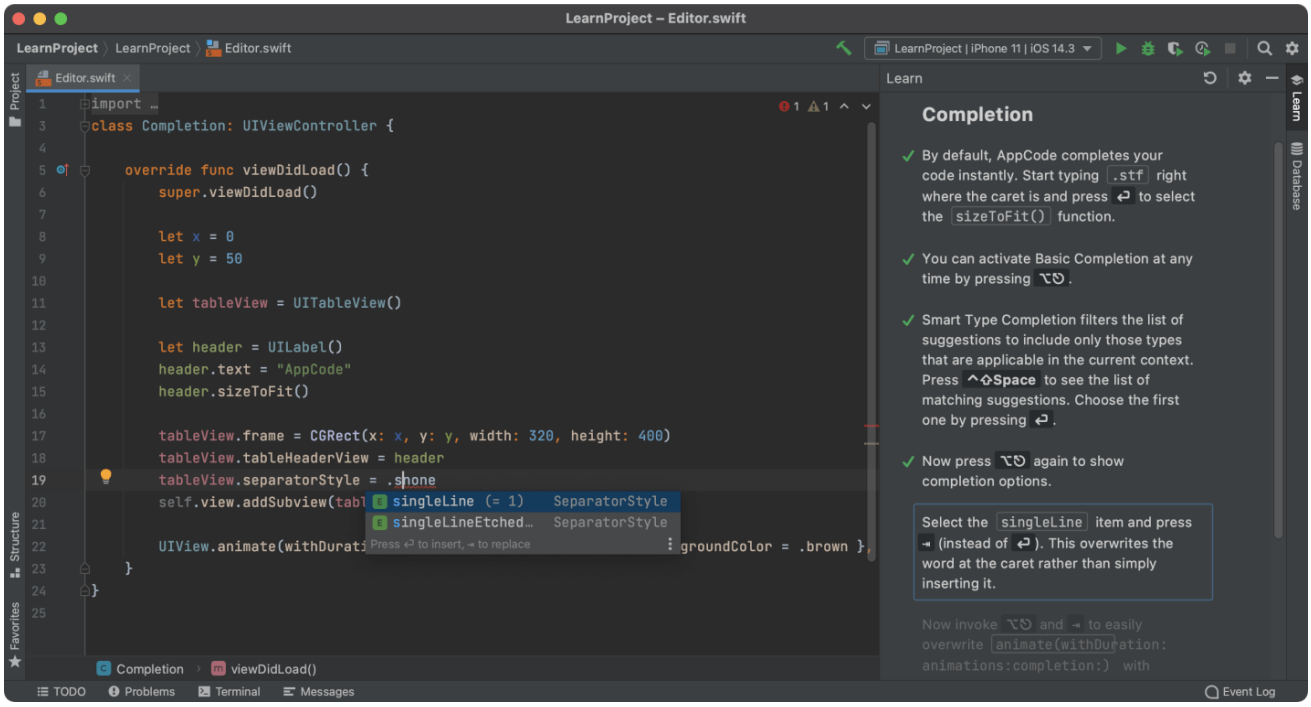


Рисунок 3.2 – Приклад інтерфейсу AppCode

Visual Studio Code – легкий редактор коду, розроблений Microsoft, який підтримує розширення для різних мов програмування, включаючи Swift. Visual Studio Code не є повноцінним IDE, але завдяки системі розширень він може бути налаштований для роботи зі Swift.

Основні переваги Visual Studio Code:

- Легкість і швидкість роботи. Visual Studio Code швидко запускається та споживає менше ресурсів у порівнянні з повноцінними IDE.

- Плагінна система. Завдяки великій кількості доступних плагінів, Visual Studio Code можна адаптувати для роботи з різними мовами програмування, включаючи Swift.
- Кросплатформеність. Visual Studio Code працює на macOS, Windows і Linux, що робить його зручним вибором для розробників, які працюють на різних платформах.

Основні недоліки Visual Studio Code:

- Обмежені інструменти для Swift. Visual Studio Code не забезпечує такого рівня інтеграції з екосистемою Apple, як Xcode, що обмежує його можливості для iOS-розробки.
- Відсутність повноцінних інструментів налагодження. Хоча редактор підтримує базові засоби для налагодження, йому бракує деяких просунутих інструментів для Swift, таких як Instruments у Xcode.

Приклад графічного інтерфейсу наведено на рисунку 3.3.

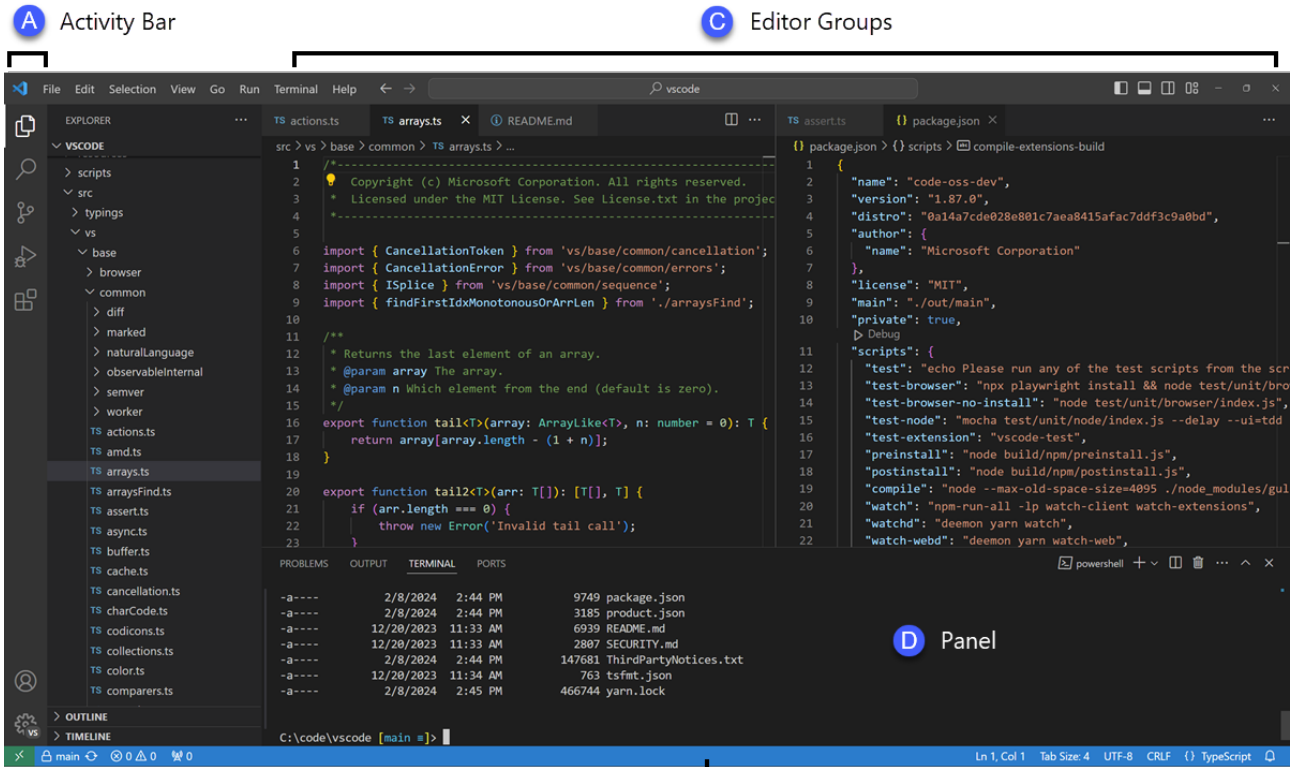


Рисунок 3.3 – Приклад інтерфейсу Visual Studio Code

Для об'єктивного вибору найбільш підходящого середовища було

проведено порівняння за основними критеріями, зокрема продуктивністю, розширюваністю, підтримкою операційних систем, засобами рефакторингу та аналізу коду. Результати порівняння наведено в таблиці 3.2.

Таблиця 3.2 – Порівняння засобів розробки

Критерій	Xcode	AppCode	Visual Studio Code
Швидкодія та використання ресурсів	1	0.5	1
Розширюваність	1	1	1
Підтримка операційних систем	0.5	0.5	1
Засоби рефакторингу/дебагу коду	1	1	0.5
Засоби аналізу коду	1	0.5	0.5
Підсумок	4.5	3.5	4

Швидкодія та використання ресурсів. Visual Studio Code демонструє високу продуктивність та ефективне використання ресурсів у порівнянні з Xcode і AppCode, які потребують більше системних ресурсів для комфортної роботи.

Розширюваність. Усі три середовища мають розвинену систему плагінів, що дозволяє розширювати функціональність і налаштовувати середовище під конкретні потреби.

Підтримка операційних систем. Visual Studio Code є кросплатформеним, тоді як Xcode та AppCode обмежені у використанні переважно на macOS, що може бути обмеженням для розробників, які працюють на інших платформах.

Засоби рефакторингу коду. AppCode має потужні інструменти для рефакторингу, які базуються на платформі IntelliJ IDEA, що робить його найбільш гнучким середовищем для організації та оптимізації коду.

Засоби аналізу коду. Xcode та AppCode забезпечують високоякісні інструменти для аналізу коду завдяки інтеграції з екосистемою Apple (Xcode) та IntelliJ IDEA (AppCode). Visual Studio Code має базові засоби аналізу, які можна покращити через плагіни, але вони не досягають рівня інтеграції, наданої Xcode або AppCode.

Xcode є найкращим вибором для розробників, які повністю орієнтовані на екосистему Apple, завдяки його інтеграції з iOS SDK, а також доступу до унікальних інструментів, таких як Instruments, що дозволяють проводити детальний аналіз продуктивності та використання ресурсів.

AppCode є ефективним рішенням для досвідчених розробників, які потребують просунутих інструментів для рефакторингу та організації коду. Його інтеграція з проєктами Xcode дозволяє використовувати переваги обох середовищ, хоча він менш стабільний в порівнянні з Xcode.

Visual Studio Code є легким та універсальним редактором, який підходить для швидкого прототипування та написання коду на Swift, особливо для тих, хто цінує кросплатформеність та можливість налаштування. Однак він має обмежену інтеграцію з екосистемою Apple, що може бути недоліком для комплексних проєктів під iOS.

За результатами порівняння, Xcode є найкращим середовищем для розробки програмного забезпечення, орієнтованого на iOS, завдяки його тісній інтеграції з екосистемою Apple, розвиненим інструментам аналізу та підтримці останніх версій Swift.

3.3 Програмна реалізація застосунку для оптимізації великих мовних моделей

Під час розробки застосунку для оптимізації великих мовних моделей на платформі iOS було реалізовано низку алгоритмів та архітектурних компонентів, що дозволяють забезпечити зручність використання, зберігання історії оптимізацій та налаштувань. У цьому розділі розглядаються реалізації основних модулів, використання Core Data для збереження історії, AppStorage для

зберігання налаштувань, а також патерн координатора для управління навігацією. Крім того, описано використання фабрик для спрощення ініціалізації ViewModel та інших об’єктів.

Застосунок базується на архітектурі MVVM, що забезпечує розділення бізнес-логіки та відображення, з використанням координатора для управління навігацією. Основні модулі застосунку включають:

- ModelManager – керує завантаженням і ініціалізацією моделей для оптимізації.
- OptimizationEngine – виконує алгоритми оптимізації, зокрема HARR (Hybrid Adaptive Rank Reduction) і DHWA (Dynamic Hardware-Based Weight Adjustment).
- HistoryManager – забезпечує збереження історії оптимізацій за допомогою Core Data.
- SettingsManager – керує налаштуваннями застосунку, використовуючи AppStorage для зберігання користувацьких налаштувань, таких як вибір ресурсів для оптимізації.
- UserInterface – реалізує основні екрани користувацького інтерфейсу з використанням SwiftUI.
- Координатор – для управління навігацією.

Для зручного управління навігацією між екранами застосунку був використаний патерн координатора. Оскільки в SwiftUI відсутня прямолінійна підтримка координатора, як у UIKit, було створено ObservableObject, що відстежує та контролює стан навігації на основі змін у ViewModel. Координатор дозволяє визначати логіку навігації в одному місці, що значно спрощує її управління та підтримку.

Координатор включає в себе перелік екранів програми, який визначається через перелічувальний тип (enum). Цей перелік містить основні екрани, такі як оптимізація, історія, налаштування та вибір файлів. Стан поточного екрана контролюється змінною, яка позначена як @Published, що дозволяє SwiftUI автоматично оновлювати інтерфейс у відповідь на зміну її значення.

Координатор забезпечує кілька основних функцій для навігації. Функція загального призначення navigate дозволяє перемикатися між екранами, змінюючи поточний стан програми. Для кожного екрану, наприклад, екрана оптимізації, історії чи налаштувань, передбачено окрему функцію, яка викликає navigate із відповідним параметром. Таке розділення дозволяє чітко визначити логіку переходів і спростити подальшу підтримку.

Окрім забезпечення базової навігації, координатор також підтримує передачу даних між екранами. Наприклад, при переході до екрана оптимізації передається об'єкт моделі файлу, що дозволяє відобразити потрібну інформацію або виконати відповідні обчислення. Такий підхід значно спрощує організацію взаємодії між компонентами застосунку та забезпечує чистоту архітектури.

У процесі реалізації координатора були враховані специфічні вимоги SwiftUI, зокрема, обмеження, пов'язані з декларативною природою цієї технології. Завдяки цьому вдалося створити гнучке рішення, яке інтегрується з іншими компонентами програми, такими як ViewModel та інші менеджери.

Таким чином, використання патерна координатора дозволило централізувати управління навігацією, зробити її більш передбачуваною та легкою в підтримці, що є критично важливим для масштабованих застосунків.

Для збереження історії оптимізацій у додатку використано Core Data, що є одним із ключових інструментів Apple для забезпечення персистентності даних в iOS-застосунках. Цей фреймворк дозволяє працювати з даними у вигляді об’єктів, що суттєво спрощує їхнє зберігання, обробку та завантаження. Зокрема, у межах даного застосунку за допомогою Core Data зберігається інформація про кожну оптимізацію, включаючи такі параметри, як назва моделі, дата оптимізації, час обробки та рівень оптимізації.

Структура даних для збереження історії оптимізацій реалізована через Entity, яка включає основні атрибути, що описують деталі кожного запису. Ця структура дозволяє ефективно організовувати дані та забезпечувати доступ до них для відображення на екрані історії.

Для управління операціями з базою даних було створено компонент

HistoryManager. Цей модуль відповідає за реалізацію основних CRUD-операцій, таких як створення нових записів, завантаження існуючих даних, їхнє оновлення та видалення.

Створення нового запису в базі даних включає ініціалізацію Entity з відповідними значеннями, переданими у вигляді параметрів. Наприклад, при збереженні інформації про нову оптимізацію передаються назва моделі, час обробки та рівень оптимізації. Усі ці дані зберігаються у відповідних атрибутах Entity, після чого викликається метод save для збереження змін у контексті Core Data.

Для завантаження історії оптимізацій використовується NSFetchRequest, який дозволяє отримати список записів із бази даних. Запити виконуються з урахуванням сортування за датою оптимізації, що забезпечує зручність перегляду даних у хронологічному порядку.

Завдяки використанню Core Data та модулю HistoryManager вдалося забезпечити високу ефективність роботи з даними, їхнє безпечне зберігання та зручний доступ до історії оптимізацій. Це рішення дозволяє гарантувати збереження важливої інформації про виконані операції, що є ключовим для аналітики та подальшого вдосконалення моделей.

Для збереження користувацьких налаштувань у застосунку використано механізм AppStorage, який є зручним інструментом SwiftUI для інтеграції з UserDefaults. Завдяки цьому property wrapper, можна ефективно зберігати і отримувати прості значення безпосередньо з UserDefaults, уникаючи прямого доступу до нього.

Модуль SettingsManager реалізує збереження та оновлення стану налаштувань застосунку, таких як використання всіх ядер процесора (useAllCores) та врахування рівня заряду батареї (considerBatteryLevel). Ці налаштування відображаються на екрані налаштувань застосунку, де користувач може змінювати їх відповідно до своїх потреб. Усі зміни налаштувань автоматично синхронізуються із системою завдяки інтеграції з AppStorage.

При кожній зміні параметрів користувача, пов’язаних із ресурсами

оптимізації, такі як використання ядер процесора або рівень заряду батареї, SettingsManager негайно оновлює збережені значення. Це забезпечує синхронізацію стану налаштувань у реальному часі, що особливо важливо для динамічних мобільних застосунків, які працюють із різними режимами ресурсів.

Механізм AppStorage не лише спрощує управління налаштуваннями, але й забезпечує тісну інтеграцію з архітектурою SwiftUI, дозволяючи відображати оновлення інтерфейсу користувача в реальному часі без додаткових зусиль.

Таким чином, використання AppStorage у модулі SettingsManager дозволяє досягти високого рівня продуктивності, простоти та зручності у роботі з налаштуваннями, що суттєво покращує користувацький досвід у застосунку.

Модуль OptimizationEngine є ключовим компонентом для виконання оптимізації великих мовних моделей у застосунку. Він реалізує алгоритми HARR (Hybrid Adaptive Rank Reduction) та DHWA (Dynamic Hardware-Based Weight Adjustment), які забезпечують адаптацію моделей до ресурсів конкретного мобільного пристрою. Завдяки цьому модулю оптимізація враховує апаратні характеристики пристрою, такі як обсяг доступної пам'яті та кількість процесорних ядер.

Метод HARR використовується для зниження рангу матриць моделі, що дозволяє суттєво зменшити кількість параметрів без значної втрати точності. Цей підхід забезпечує економію пам’яті та підвищує швидкість роботи моделі. Метод DHWA, у свою чергу, динамічно налаштовує ваги моделі на основі поточних обчислювальних можливостей пристрою, що дає змогу забезпечити баланс між продуктивністю та енергоспоживанням.

Архітектура OptimizationEngine побудована таким чином, щоб підтримувати кожен етап оптимізації як окрему функцію. Це забезпечує модульність і спрощує інтеграцію додаткових оптимізаційних алгоритмів у майбутньому. Для обробки завдань модуль використовує багатопоточну підтримку через DispatchQueue, що дозволяє ефективно розподіляти навантаження між ядрами процесора.

Процес оптимізації включає кілька етапів:

1. Аналіз апаратних характеристик пристрою.
2. Початкове зниження рангу матриць моделі за допомогою HARR.
3. Динамічне налаштування ваг моделі за допомогою DHWA.
4. Перевірка точності моделі після оптимізації та внесення необхідних коректив.

Реалізація OptimizationEngine забезпечує інтеграцію з іншими компонентами застосунку, такими як ModelManager, який відповідає за завантаження мовних моделей, та HistoryManager, який зберігає результати оптимізацій для подальшого аналізу.

Таким чином, модуль OptimizationEngine поєднує передові алгоритми оптимізації з ефективною архітектурою, що дозволяє забезпечити високу продуктивність роботи мобільного застосунку при збереженні точності мовних моделей.

Для спрощення ініціалізації об’єктів ViewModel та інших компонентів у застосунку було реалізовано патерн Фабрики (Factory). Цей підхід дозволяє централізувати логіку створення ViewModel для кожного екрану, забезпечуючи гнучкість і масштабованість застосунку. Замість того, щоб створювати об’єкти безпосередньо в кожному екрані, фабрика бере на себе відповідальність за їх створення, враховуючи необхідні залежності та контекст.

Фабрика створює ViewModel, використовуючи параметри, які залежать від специфічних вимог конкретного екрану. Наприклад:

1. Для екрана оптимізації фабрика генерує ViewModel, передаючи інформацію про обрану модель для оптимізації, а також налаштування, які користувач задав у розділі налаштувань.
2. Для екрана історії фабрика ініціалізує ViewModel, який отримує дані про попередні оптимізації з модуля HistoryManager.

Цей підхід має кілька ключових переваг:

1. Централізованість. Логіка створення ViewModel зосереджена в одному місці, що спрощує управління залежностями.
2. Чистота коду. Екрани стають менш завантаженими логікою

ініціалізації, що полегшує їхнє тестування та підтримку.

3. Масштабованість. У разі необхідності додавання нового екрану або модифікації параметрів існуючих ViewModel, зміни можна виконати лише в фабриці.

Фабрика також спрощує передачу залежностей між компонентами. Наприклад, SettingsManager, який відповідає за збереження налаштувань, або OptimizationEngine, який реалізує алгоритми оптимізації, можуть бути передані в ViewModel безпосередньо через фабричний метод.

Таким чином, використання патерна Фабрики дозволило зробити архітектуру застосунку більш структурованою, а код – зрозумілішим і легшим для підтримки.

Графічний інтерфейс користувача реалізовано з використанням SwiftUI, що забезпечує створення інтуїтивно зрозумілого та адаптивного дизайну, який автоматично підлаштовується під різні розміри екранів мобільних пристроїв. У застосунку передбачено кілька основних екранів:

- Екран історії оптимізацій, де відображається список оптимізованих моделей із зазначенням дати, часу виконання та рівня оптимізації. Цей екран дозволяє користувачам швидко переглядати результати попередніх оптимізацій.
- Екран налаштувань, який надає можливість вмикати або вимикати параметри, пов’язані з використанням усіх ядер процесора чи врахуванням рівня заряду батареї. Налаштування відображаються у вигляді зручних перемикачів.
- Екран вибору файлів, що дозволяє завантажувати мовні моделі для подальшої оптимізації безпосередньо з файлової системи пристрою.
- Екран оптимізації, на якому в реальному часі демонструється прогрес виконання процесу оптимізації, включаючи відображення статусу поточних обчислень.

SwiftUI забезпечує динамічне оновлення інтерфейсу за рахунок зв’язку між компонентами ViewModel і користувацькими екранами. Наприклад, після завершення оптимізації координатор автоматично направляє користувача на екран із детальними результатами, що значно підвищує зручність взаємодії.

Для підвищення продуктивності застосунку було реалізовано паралельну обробку ресурсоємних операцій із використанням DispatchQueue та OperationQueue. Це дозволяє знизити час виконання таких завдань, як зменшення рангу або квантування параметрів, не блокуючи основний потік, що відповідає за оновлення інтерфейсу.

Під час розробки застосунку реалізовано такі основні компоненти:

- Використання Core Data для збереження історії оптимізацій, що забезпечує зручність управління даними про попередні процеси.
- Реалізація AppStorage для збереження налаштувань користувача, що дозволяє автоматично синхронізувати зміни налаштувань із відповідними елементами інтерфейсу.
- Впровадження патерна координатора для централізованого управління навігацією між основними екранами.
- Використання фабрик для створення ViewModel із залежностями, специфічними для кожного екрану.
- Застосування багатопоточності для прискорення ресурсомістких обчислень без шкоди для зручності використання інтерфейсу.

Завдяки інтеграції цих підходів застосунок досяг високої продуктивності, надійності та зручності використання. Реалізовані компоненти створюють ефективне середовище для мобільної оптимізації мовних моделей на iOS. Лістинг програми наведено у додатку В.

3.4 Висновки

У третьому розділі було детально обґрунтовано вибір інструментів та технологій для реалізації програмного продукту, обрано оптимальні засоби програмування, середовище розробки, а також розроблено всі модулі програми.

Для створення застосунку було обрано мову програмування Swift у поєднанні з SwiftUI для побудови графічного інтерфейсу користувача. Середовище розробки Xcode було обрано як оптимальний варіант для розробки під платформу iOS завдяки його тісній інтеграції з Apple SDK, розвиненими

інструментами налагодження та підтримкою всіх необхідних бібліотек і фреймворків.

З метою забезпечення зберігання історії оптимізацій було обрано Core Data, що є вбудованим фреймворком Apple для роботи з персистентними даними. Це рішення забезпечує надійне зберігання історичних даних оптимізацій у локальній базі даних, дає можливість працювати з великими обсягами даних та має потужні можливості для фільтрації і сортування. Налаштування застосунку зберігаються з використанням AppStorage, що дозволяє зберігати значення параметрів безпосередньо у UserDefaults і забезпечує зручний обмін даними між компонентами SwiftUI.

Було розроблено такі основні модулі застосунку:

- Координатор навігації для управління переходами між екранами застосунку, що спрощує навігаційну логіку та забезпечує зручне керування маршрутами користувача.
- HistoryManager – модуль для збереження і завантаження історії оптимізацій, який інтегрується з Core Data та забезпечує повноцінну підтримку CRUD-операцій.
- SettingsManager – модуль для зберігання та отримання налаштувань користувача, що дозволяє зберігати вибрані налаштування, як-от використання додаткових ресурсів для оптимізації, у системі безпосередньо.
- OptimizationEngine – основний модуль, що виконує алгоритми оптимізації HARR та DHWA. Він інтегрується з іншими модулями для налаштування параметрів оптимізації та адаптації процесу до обчислювальних ресурсів пристрою.
- UI-компоненти – реалізація ключових екранів застосунку для вибору файлу, оптимізації, історії оптимізацій та налаштувань із використанням SwiftUI.

Таким чином, у цьому розділі було забезпечено реалізацію архітектурної основи та основних функціональних модулів для забезпечення високої продуктивності, стабільності та зручності користування застосунком.

4 ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАСТОСУНКУ

4.1 Аналіз методів тестування

Тестування програмного забезпечення є невід'ємною частиною процесу розробки, оскільки саме на цьому етапі підтверджується функціональна відповідність продукту встановленим вимогам, його надійність та стійкість до навантажень. Процес тестування дозволяє виявити та усунути дефекти, підвищити безпеку та забезпечити високу якість програмного продукту до його впровадження для кінцевого користувача. Крім того, тестування дає змогу оцінити рівень готовності програмного продукту до експлуатації, виявити можливі уразливості та запобігти подальшим проблемам, пов'язаним з експлуатаційною надійністю [29].

Історично тестування мало дуже формалізований характер, зокрема в наукових та оборонних системах, де чітко визначалися процедури, завдання та кінцеві цілі. Кожна стадія була ретельно задокументована, що дозволяло перевірити усі процеси після завершення розробки. Такий підхід забезпечував глибоку перевірку продукту, але виявився обмеженим, коли мова йшла про комерційні системи та масові ринки, де продукти мали бути гнучкішими та швидко адаптованими до змін. У цих умовах виникла необхідність розробки нових методів, що дозволяють ефективніше адаптувати тестування для комерційних умов, зберігаючи при цьому високу точність виявлення помилок.

Сучасне тестування включає два основних підходи: статичне та динамічне тестування, що разом забезпечують всебічний аналіз як функціональних, так і нефункціональних аспектів програми. Статичне тестування охоплює перевірку документації та коду без виконання самої програми. Воно дозволяє оцінити архітектурні рішення, дотримання стандартів та відповідність документації до вимог. Статичне тестування є першим бар’єром для помилок, забезпечуючи відповідність вихідного коду та документації на ранніх етапах, що значно знижує ймовірність появи дефектів у подальших фазах. Завдяки статичним методам

тестування можна виявити невідповідності з початковими вимогами або стандартами кодування, що дозволяє уникнути багатьох проблем у фінальній версії продукту.

Динамічне тестування, з іншого боку, орієнтоване на оцінку продукту в процесі його виконання. Це дозволяє перевірити, як програмне забезпечення функціонує в реальному часі, з фактичними даними та під різними сценаріями використання. Воно забезпечує оцінку працездатності, швидкодії та стійкості до помилок. Динамічні методи дозволяють гнучко адаптувати тестові сценарії залежно від поточних потреб проекту та виявляти проблеми, що не були очевидні під час статичного тестування. За допомогою динамічного тестування можна впевнитися, що програма правильно обробляє вхідні дані, коректно реагує на виклики та дотримується вимог продуктивності.

Залежно від рівня доступу до внутрішньої структури коду, динамічне тестування поділяється на три методи: тестування «чорної скриньки», тестування «білої скриньки» та тестування «сірої скриньки».

Тестування «чорної скриньки» не потребує знань про внутрішню структуру продукту. Воно фокусується на функціональних аспектах, перевіряючи, чи відповідають результати виконання програми очікуванням користувача. Цей метод дозволяє отримати результати тестування, максимально наближені до досвіду кінцевого користувача, оскільки тестувальники оцінюють лише вихідні дані без урахування деталей реалізації. У контексті мобільних додатків, тестування «чорної скриньки» є надзвичайно важливим, адже саме воно дозволяє виявити помилки, які можуть бути очевидні лише під час взаємодії з користувачем, зокрема, функціональні збої, помилки навігації та некоректні інтерфейси [30].

Тестування «білої скриньки» передбачає детальний аналіз внутрішньої структури коду. Тестувальник повинен розуміти логіку програми, щоб оцінити, чи коректно виконуються всі гілки та цикли, а також чи відсутні логічні помилки. Такий метод є надзвичайно корисним для виявлення помилок, що стосуються саме логіки виконання, і особливо ефективний для перевірки складних

алгоритмів або модулів з високими вимогами до продуктивності.

Тестування «сірої скриньки» поєднує в собі елементи двох попередніх підходів. Воно дозволяє працювати з обмеженим доступом до внутрішньої логіки програми, фокусуючись на окремих алгоритмах або модулях, де необхідні знання певних архітектурних рішень або алгоритмів. Це дозволяє тестувальнику ефективніше виявляти помилки, що могли б бути пропущені при традиційному тестуванні «чорної скриньки», зберігаючи при цьому об'єктивний підхід, характерний для тестування «чорної скриньки».

Після аналізу цих методів для тестування програмного продукту було обрано метод «чорної скриньки». Такий підхід надає можливість зосередитися на функціональних аспектах програми, які є найбільш важливими з точки зору кінцевого користувача. Тестування «чорної скриньки» забезпечує максимальну об'єктивність, оскільки тестувальники працюють лише з інтерфейсом програми і перевіряють результати на відповідність вимогам, не враховуючи особливості внутрішньої логіки. Це дозволяє рано виявляти дефекти, пов’язані з помилками у функціональності, і забезпечує зручність застосування для мобільних платформ, де особливо важливі стабільність, простота взаємодії та інтуїтивність інтерфейсу.

4.2 Тестування розробленого програмного продукту

Першим екраном, який бачить користувач після запуску мобільного додатку OptiLang, є Splash screen (див. рисунок 4.1). Даний екран виконує функцію вітання користувача та створює перше враження від додатку. На ньому відображається логотип додатку і слоган, що підкреслює основну функціональність продукту – оптимізацію мовних моделей. Це допомагає користувачеві зорієнтуватися в інтерфейсі перед переходом до основного функціоналу.

Після завершення завантаження програми користувач потрапляє на перший основний екран додатку – Історія, де представлено записи про попередні оптимізації моделей. Внизу екрана відображається таббар з трьома основними

вкладками: Історія, Оптимізація та Налаштування. Це забезпечує швидкий доступ до основних розділів програми, дозволяючи користувачеві зручно перемикатися між переглядом історії оптимізацій, запуском нової оптимізації та налаштуванням параметрів додатку.

Рисунок 4.1 – Splash screen додатку «OptiLang»

Після того, як користувач натиснув на Історію у нижньому меню навігації, він потрапляє на екран Історія (див. рисунок 4.2). На цьому екрані зберігається інформація про попередні оптимізації моделей. Користувач може переглядати назву кожної моделі, дату та час оптимізації, тривалість процесу та досягнутий відсоток оптимізації. Екран призначений для надання історичної довідки про виконані оптимізації, що дозволяє відстежувати продуктивність і оцінювати ефективність методів, використаних під час оптимізації.

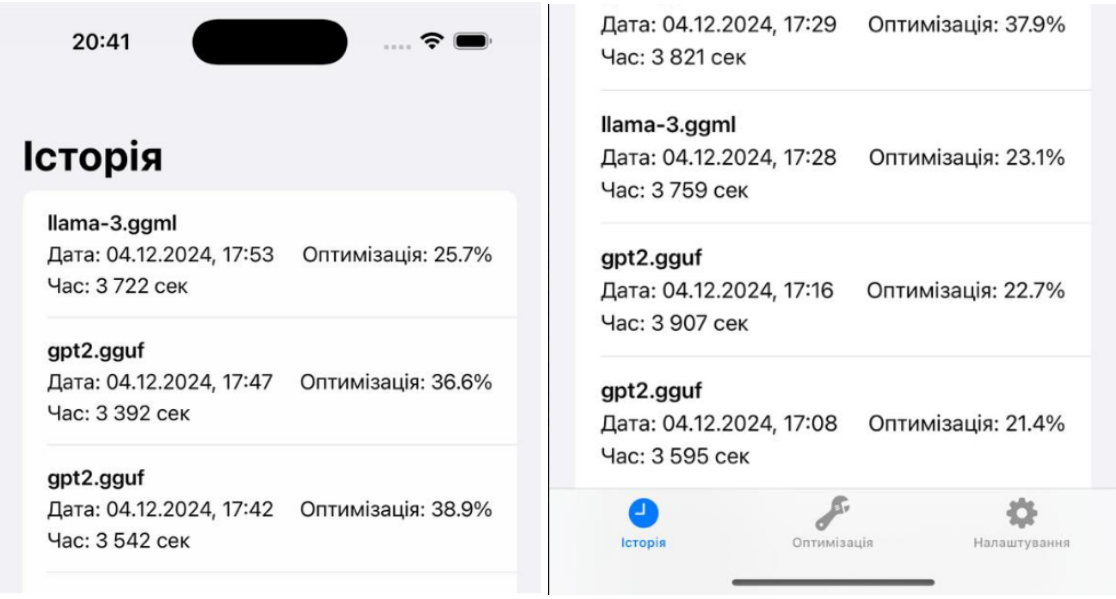


Рисунок 4.2 – Екран «Історія»

Після того, як користувач натиснув на Оптимізацію у меню навігації, він потрапляє на початковий екран Оптимізація, де модель ще не вибрано (див. рисунок 4.3). На цьому етапі користувачеві пропонується завантажити файл моделі, яку необхідно оптимізувати, натиснувши кнопку «Імпортувати модель». Цей екран допомагає почати процес, забезпечуючи інтуїтивний інтерфейс для вибору необхідного файлу, без зайвих відволікаючих елементів.

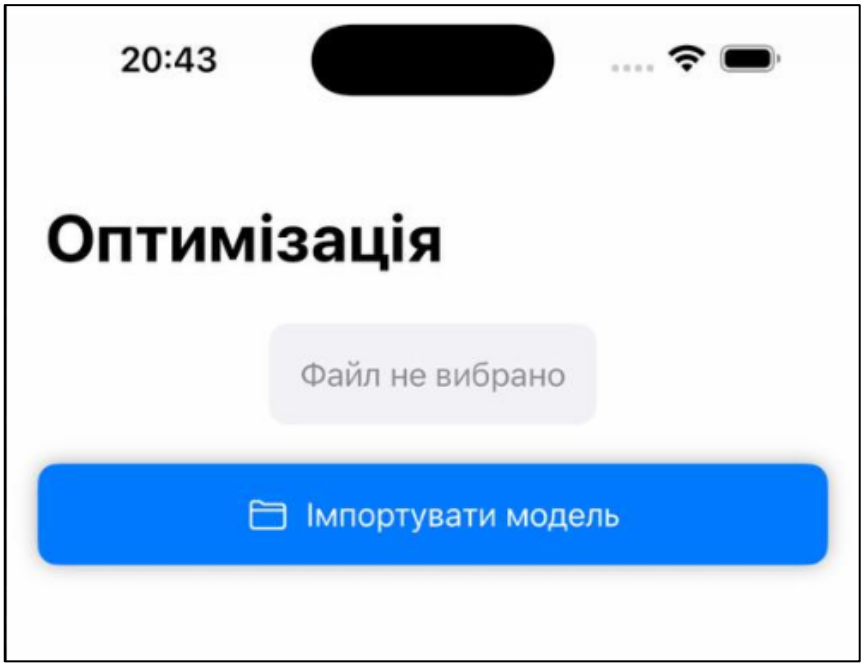


Рисунок 4.3 – Початковий екран оптимізації з кнопкою «Імпортувати модель»

Після того, як користувач натиснув кнопку Імпортувати модель, відкривається екран вибору файлів (див. рисунок 4.4). На цьому етапі користувач може переглядати файли, доступні на його пристрої, і вибрати потрібний файл моделі для подальшої оптимізації. Екран дозволяє користувачеві зручно і швидко знайти та імпортувати необхідну модель.

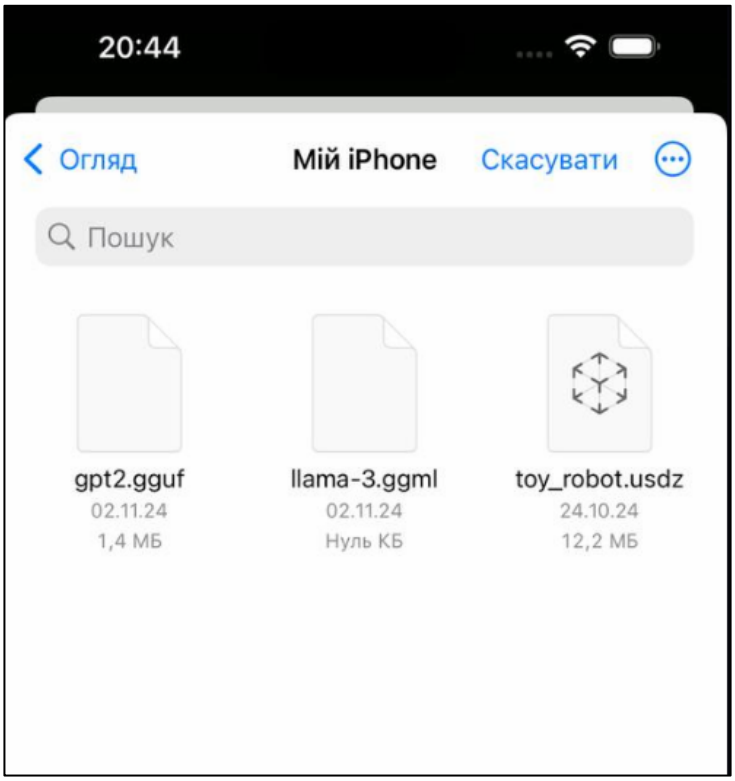


Рисунок 4.4 – Екран вибору файлів

Після того, як користувач обрав файл, він повертається на екран «Оптимізація» з уже завантаженою моделлю (див. рисунок 4.5). Вибраний файл відображається під заголовком екрана разом із додатковою інформацією, такою як розмір моделі та її формат, що дозволяє переконатися в правильності вибору. Для забезпечення зручності користувача кнопка «Оптимізувати» стає активною лише після завантаження файлу, що запобігає випадковому запуску процесу без необхідних даних.

На цьому етапі користувачеві надається можливість перевірити завантажену модель та, у разі помилки, повернутися на попередній екран для вибору іншого файлу. Інтерфейс забезпечує інтуїтивно зрозумілу навігацію,

допомагаючи уникнути потенційних помилок і забезпечуючи чітке відображення статусу підготовки.

Крім того, екран «Оптимізація» оснащено пояснювальним текстом, який надає короткий опис процесу оптимізації, що дозволяє користувачеві зрозуміти, які дії будуть виконуватися з завантаженою моделлю. Це сприяє підвищенню довіри до застосунку, оскільки користувач чітко розуміє, що відбуватиметься далі.

Після перевірки інформації та підтвердження користувач може натиснути кнопку «Оптимізувати», щоб розпочати процес. На цьому етапі всі ресурси застосунку готуються до виконання обчислень, а екран оновлюється для відображення стану прогресу, що робить процес зручним і прозорим для користувача.

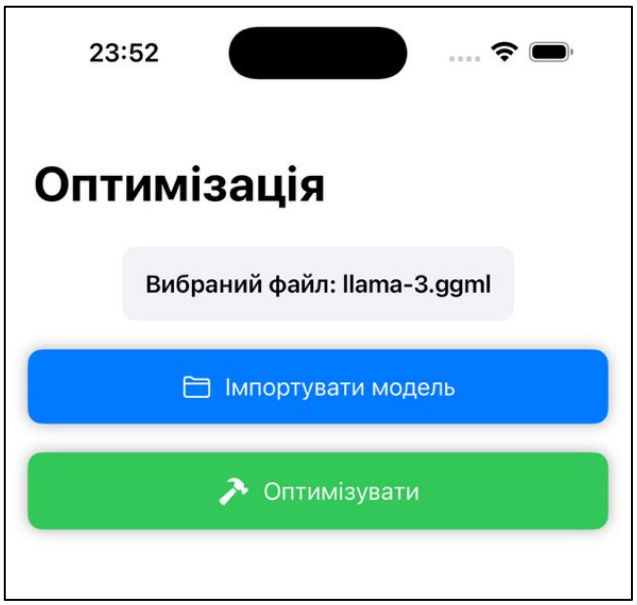


Рисунок 4.5 – Екран оптимізації з обраним файлом

Після того, як користувач натиснув кнопку Оптимізувати, починається процес оптимізації, і на екрані з’являється індикатор прогресу (див. рисунок 4.6). Під час цього процесу користувач бачить статус завантаження та поточний етап оптимізації, наприклад, застосування методу DHWA. Це надає користувачеві візуальний зворотній зв’язок, показуючи, що процес оптимізації триває.

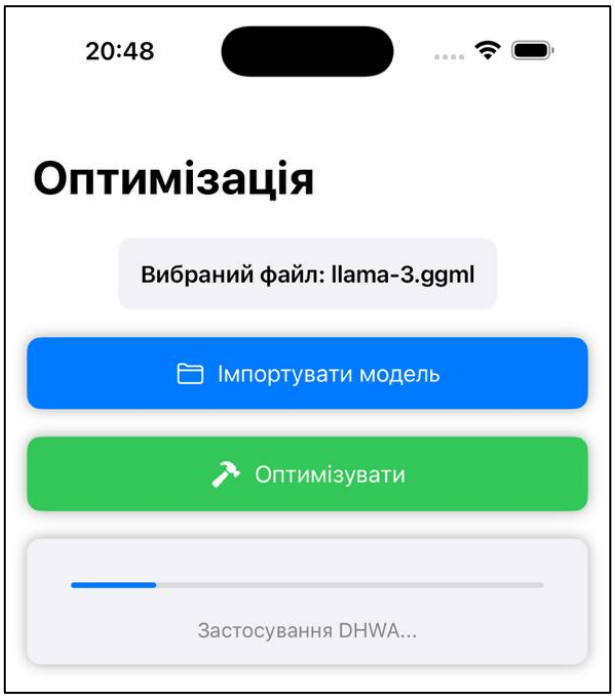


Рисунок 4.6 – Екран оптимізації з індикатором прогресу

Після того, як користувач завершив процес оптимізації, він може перейти до екрану Налаштування (див. рисунок 4.7). Тут користувач може змінити параметри, що стосуються оптимізації, наприклад, включення/виключення використання всіх ядер CPU чи врахування заряду акумулятора. Цей екран надає можливість налаштувати додаток відповідно до власних потреб, оптимізуючи процес під конкретні умови роботи.

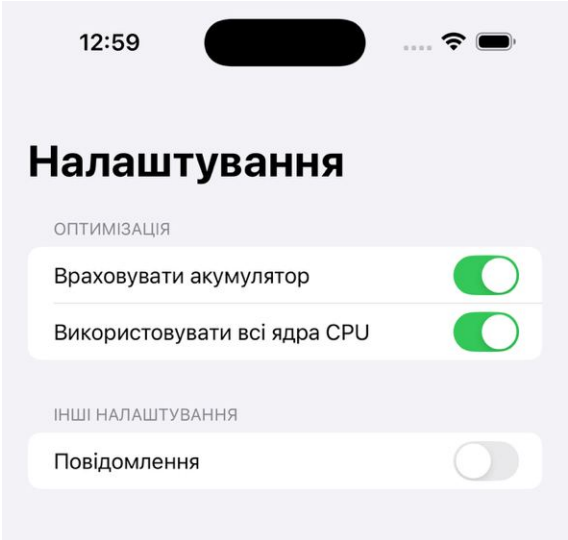


Рисунок 4.7 – Екран «Налаштування»

4.3 Експериментальні дослідження запропонованих методів

У ході експериментальних досліджень були оцінені запропоновані методи оптимізації мовних моделей для мобільних пристроїв. Результати представлені у вигляді діаграм, які дозволяють детально проаналізувати вплив оптимізації на ключові показники моделей.

Першою діаграмою є «Різниця максимального використання CPU» (див. рисунок 4.8). Після оптимізації максимальне використання CPU зменшилося на 93,5%, що є одним із найбільш вагомих результатів дослідження. Такий показник свідчить про значне зниження енергоспоживання та підвищення стабільності роботи пристрою під час виконання оптимізованої моделі. Зниження навантаження на CPU є важливим для мобільних пристроїв, оскільки це сприяє не лише економії енергії, але й зменшенню нагрівання пристрою, що позитивно впливає на комфорт користувача та довговічність обладнання.

Рисунок 4.8 – Діаграма різниці максимального використання CPU

Варто зазначити, що така значна різниця у цьому випадку зумовлена гарною сумісністю вимог пікової продуктивності моделі із апаратними можливостями пристрою. У випадках, коли вимоги моделі до апаратних

можливостей значно перевищують можливості пристрою, середня різниця у використанні CPU може варіюватися в межах від 20% до 60%. Для більш детального розуміння типового впливу оптимізації на процесорне навантаження, на рисунку 4.9 показані типові графіки різниці максимального використання CPU

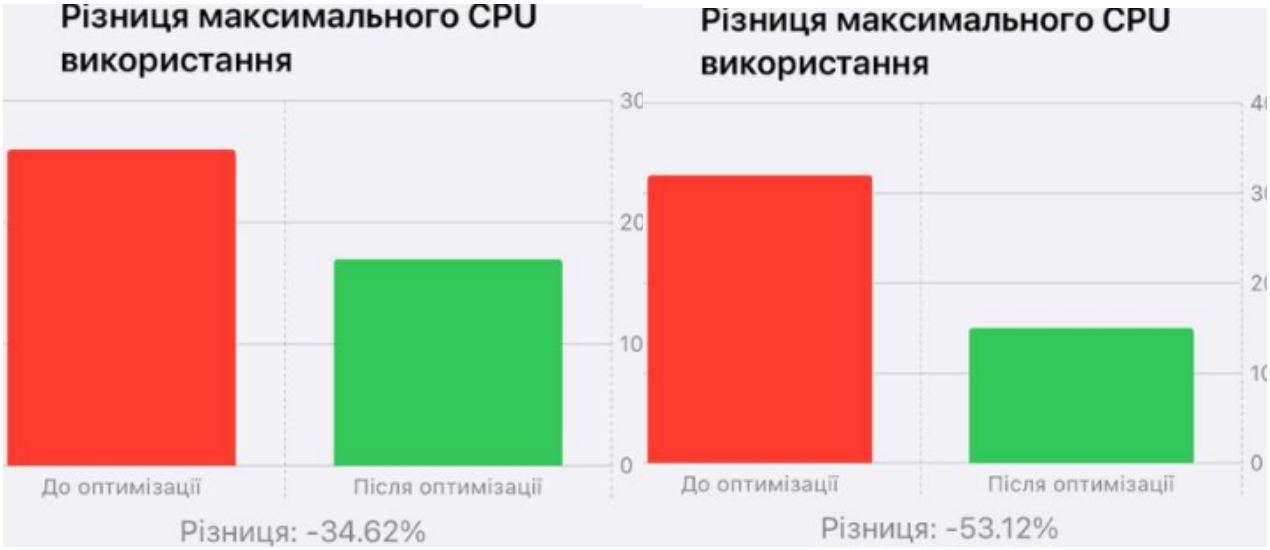


Рисунок 4.9 – Діаграми типової різниці максимального використання CPU

Для більш детального розуміння впливу оптимізації на процесорне навантаження, на рисунку 4.10 показано графік використання CPU у відсотках протягом часу виконання моделі. Червона лінія ілюструє поведінку CPU до оптимізації, де спостерігаються численні піки навантаження, що можуть впливати на стабільність роботи пристрою. Зелена лінія відображає значне зменшення навантаження після оптимізації, а також більш рівномірне розподілення використання ресурсів.

Цей графік також демонструє, що після оптимізації немає різких стрибків у використанні CPU, що сприяє зниженню ризику перегріву пристрою та покращує загальну енергоефективність. Стабільне навантаження на процесор забезпечує більш рівномірне використання ресурсів пристрою, що, у свою чергу, позитивно впливає на його продуктивність і збільшує тривалість автономної роботи. Крім того, зниження ризику перегріву допомагає уникнути негативного

впливу на інші компоненти пристрою, такі як акумулятор чи системна пам'ять, забезпечуючи довговічність обладнання.

Рівномірний розподіл навантаження також дозволяє уникати випадків, коли процесор досягає максимального рівня продуктивності, що може призводити до зниження продуктивності через автоматичне зменшення тактової частоти для запобігання перегріву.

Завдяки описаному підходу оптимізовані мовні моделі можуть ефективно працювати у режимі реального часу навіть на пристроях із середнім рівнем технічних характеристик.

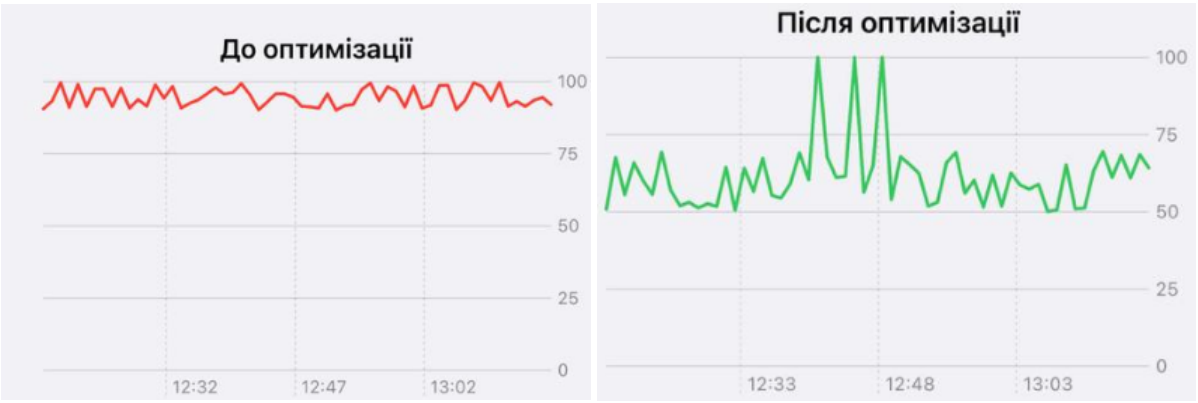


Рисунок 4.10 – Графік використання CPU у часових проміжках до та після оптимізації

Ці ілюстрації дозволяють глибше зрозуміти вплив запропонованих методів на продуктивність та енергоефективність мобільних мовних моделей. Значне скорочення навантаження на процесор підкреслює ефективність запропонованих рішень і підтверджує їх доцільність для застосування на мобільних платформах.

Наступна діаграма ілюструє "Кількість ваг моделі" до і після оптимізації (див. рисунок 4.11). В ході оптимізації кількість ваг моделі зменшилася на 30%, що дозволило значно спростити обчислювальні процеси та зробити модель менш вимогливою до ресурсів пристрою. Скорочення кількості ваг означає не лише зменшення розміру, але й оптимізацію структури самої моделі, що покращує її адаптацію до різних сценаріїв використання, забезпечуючи збереження високої точності та продуктивності.

Рисунок 4.11 – Діаграма кількості ваг моделі до і після оптимізації

Діаграма відображає «Використання пам’яті» до і після оптимізації (див. рисунок 4.12). Результати показують, що використання пам’яті зменшилося на 37%, що є важливим досягненням для мобільних пристроїв, де обсяг оперативної пам’яті часто обмежений. Такий результат забезпечує більшу кількість вільних ресурсів для одночасного виконання інших завдань або застосунків, що підвищує загальну ефективність роботи пристрою. Оптимізація використання пам’яті також дозволяє уникнути аварійних зупинок через недостатність ресурсів.

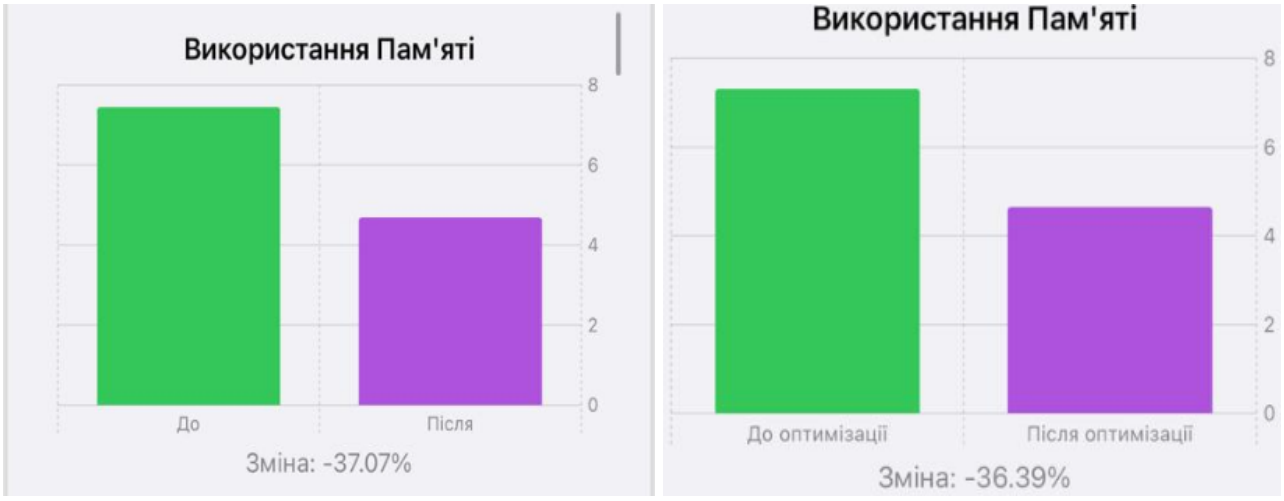


Рисунок 4.12 – Діаграма використання пам’яті до і після оптимізації

Діаграма, «Точність моделі» (див. рисунок 4.13), демонструє збереження високої точності після оптимізації, із втратою лише 1.3%. Це досягнення є вкрай важливим, адже основна мета оптимізації – скорочення ресурсів без істотного впливу на точність результатів. Такий рівень втрати точності дозволяє використовувати оптимізовані моделі у сценаріях, де якість результатів має критичне значення, наприклад, у медичних застосунках чи системах рекомендацій.

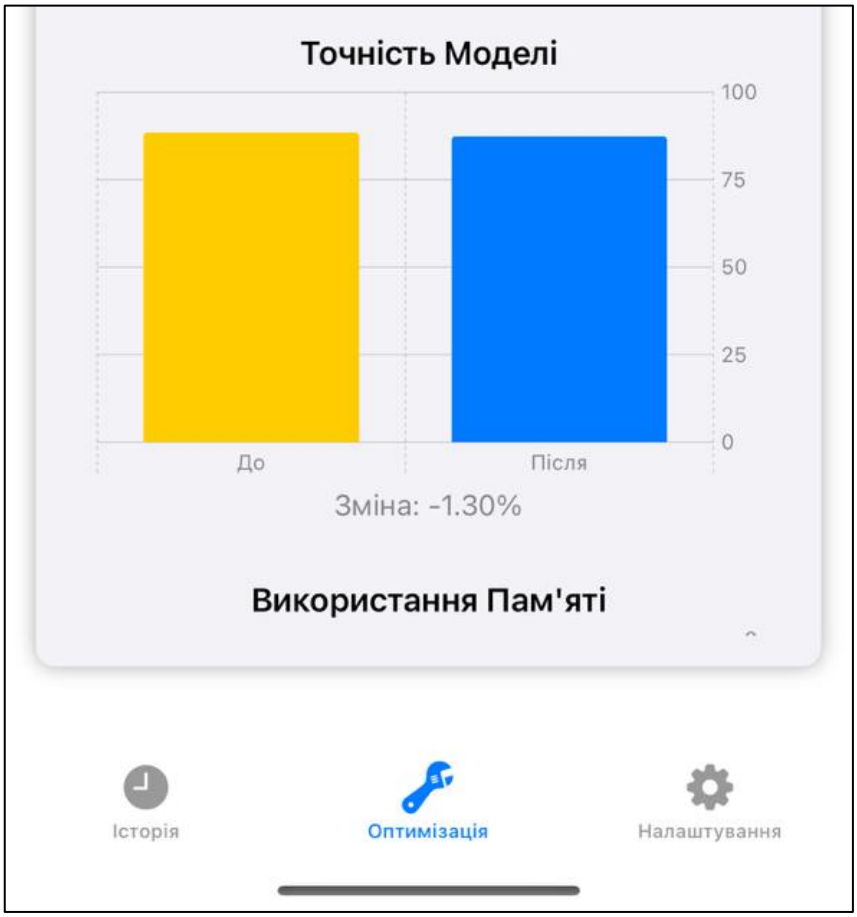


Рисунок 4.13 – Діаграма точності моделі до і після оптимізації

Наступна діаграма демонструє «Середній час генерації відповіді у тюрінг-тесті» (див. рисунок 4.14). Після оптимізації час генерації відповіді зменшився на 39%, що суттєво покращує користувацький досвід. Швидка генерація відповідей є критичною для інтерактивних систем, таких як голосові асистенти чи чат-боти. Зменшення часу реакції забезпечує більш плавну і природну взаємодію, що підвищує задоволеність користувачів.

Рисунок 4.14 – Діаграма часу генерації відповіді у тюрінг-тесті

Наступна діаграма показує зміни у «Розмірі моделі» (див. рисунок 4.15). Результати демонструють зменшення розміру моделі на 25%, що робить її більш придатною для завантаження та виконання на мобільних пристроях з обмеженим обсягом пам’яті. Зменшення розміру також відкриває можливості для зберігання більшої кількості моделей на одному пристрої або завантаження їх через мережу з меншими затримками.

Рисунок 4.15 – Діаграма розміру моделі до і після оптимізації

Для аналізу внеску кожного методу в загальний результат була створена кругова діаграма (див. рисунок 4.16). Вона наочно демонструє, які методи

оптимізації мали найбільший вплив на результати. Наприклад, метод HARR показав найвищий внесок у зменшення використання ресурсів, тоді як DHWA забезпечив значне зменшення енергоспоживання. Квантова компресія, у свою чергу, продемонструвала найкращі результати у зменшенні розміру моделі, забезпечуючи збалансований підхід до оптимізації.

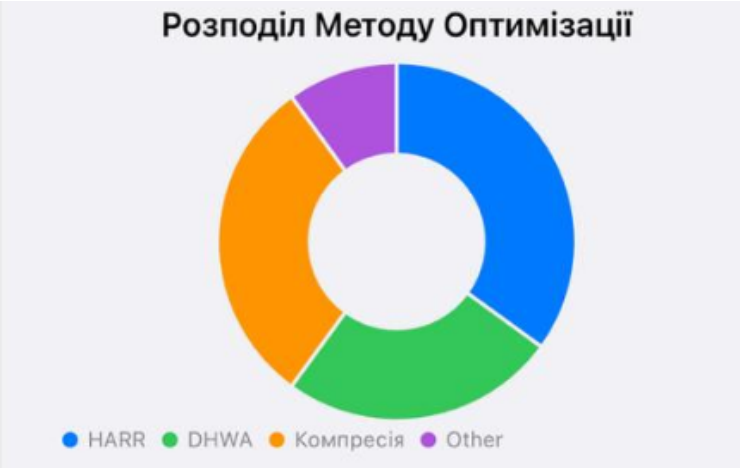


Рисунок 4.16 – Кругова діаграма розподілу впливу методів оптимізації

Останній етап експериментів передбачав порівняння відповідей у тюрінг-тесті до і після оптимізації (див. рисунок 4.17). Порівняння демонструє, що навіть після значного скорочення розміру та ресурсів, якість відповідей залишилася на високому рівні. Це підтверджує ефективність запропонованих методів, дозволяючи використовувати оптимізовані моделі в задачах з високими вимогами до точності, таких як аналіз текстів або машинний переклад.

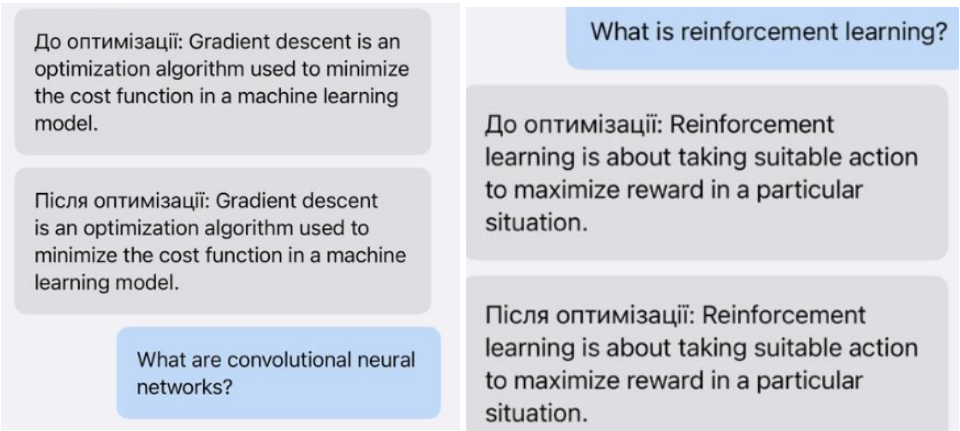


Рисунок 4.17 – Порівняння відповідей у тюрінг-тесті до і після оптимізації

Таким чином, експериментальні дослідження підтвердили, що розроблені методи оптимізації дозволяють суттєво знизити споживання ресурсів, зберігаючи високу продуктивність та точність моделей. Це робить запропоновані методи особливо цінними для мобільних пристроїв з обмеженими обчислювальними можливостями.

4.4 Розробка інструкції користувача

Інструкція для користувача передбачає ознайомлення з мінімальними технічними характеристиками для встановлення застосунку на смартфон. Детальний опис характеристик наведений у таблиці 4.1.

Таблиця 4.1 – Мінімальні технічні характеристики

Характеристика	Опис
Операційна система	iOS 16.0 або вище
Процесор	Мінімум 2,39 ГГц, шестиядерний або вище
Оперативна пам'ять	Мінімум 4 ГБ RAM
Вільне місце	Мінімум 50 МБ

Для запуску мобільного додатку «OptiLang» необхідно запустити файл OptiLang, зображення іконки в головному меню має вигляд, як наведено на рисунку 4.18.

Рисунок 4.18 – Зображення іконки файлу програми в головному меню

Основні етапи роботи із застосунком:

1. Запуск програми. Після натискання на іконку додатку користувач потрапляє на екран Splash screen, який вітає його логотипом програми та коротким описом основної функціональності. Після завершення завантаження додаток автоматично перенаправляє на головний екран.

2. Робота з екраном «Історія». Головний екран відображає записи про попередні оптимізації. Тут користувач може переглянути інформацію про кожну модель, включаючи її назву, дату оптимізації, рівень продуктивності та інші деталі. Цей екран є центральним для відстеження результатів попередніх сеансів оптимізації.

3. Оптимізація моделі:

- Після натискання на вкладку «Оптимізація» користувач потрапляє на екран для імпорту моделі. Для початку процесу необхідно натиснути кнопку «Імпортувати модель» і вибрати потрібний файл із файлової системи.
- Після імпортування моделі користувач повертається на екран «Оптимізація», де підтверджує свій вибір та натискає кнопку «Оптимізувати».
- Під час оптимізації відображається прогрес виконання, поточні етапи оптимізації та деталі застосованих методів.

4. Налаштування. Після завершення оптимізації користувач може перейти до вкладки «Налаштування». На цьому екрані можна змінити параметри роботи програми, включаючи використання всіх ядер CPU, рівень споживання енергії, або параметри, пов’язані з точністю та швидкістю роботи оптимізованих моделей.

Особливості використання:

- Користувачі можуть переглядати графічну інформацію щодо роботи програми, зокрема графіки продуктивності, розподілу методів оптимізації та результати порівняння моделей до і після оптимізації.
- Усі дані зберігаються у внутрішній базі програми, що дозволяє легко відновити інформацію навіть після перезапуску застосунку.

Таким чином, було розглянуто системні вимоги для використання програмного продукту «OptiLang». Наведено мінімальну конфігурацію апаратного та програмного забезпечення для роботи із застосунком.

4.5 Висновки

У четвертому розділі було проведено тестування програмних модулів мобільного додатку OptiLang для оптимізації мовних моделей на iOS. У процесі тестування були ретельно перевірені основні екрани програми, зокрема екран історії оптимізацій, екран для імпорту та вибору файлів для оптимізації, а також інтерфейс налаштувань. Кожен з цих екранів пройшов тестування на коректність відображення інформації, стабільність роботи, відповідність функціональних вимог та інтерактивність елементів управління.

Розроблено інструкцію користувача, яка містить рекомендації щодо встановлення застосунку, опис його основного функціоналу, а також покрокові інструкції для користувачів, що допоможуть швидко ознайомитися зі способом роботи програми. Інструкція містить також рекомендації щодо мінімальних технічних характеристик мобільного пристрою, необхідних для стабільної роботи додатку. Ці характеристики передбачають достатній обсяг пам’яті для завантаження великих моделей, оптимальний рівень CPU для виконання обчислювально-інтенсивних завдань з оптимізації та підтримку останніх версій операційної системи iOS.

Проведене тестування та розробка інструкцій для користувача забезпечили високу якість роботи застосунку та сприяють його надійності, зручності та ефективності у використанні.

5 ЕКОНОМІЧНА ЧАСТИНА

Науково-технічна розробка може існувати та впроваджуватися лише за умови відповідності сучасним вимогам як у сфері науково-технічного прогресу, так і з економічної точки зору. Тому при проведенні науково-дослідних робіт необхідно оцінювати економічну ефективність отриманих результатів.

Магістерська робота на тему «Розробка методів та програмного забезпечення для оптимізації великих мовних моделей для iOS» належить до науково-технічних проектів, спрямованих на виведення продукту на ринок (або рішення про комерціалізацію науково-технічної розробки може бути прийнято під час виконання самої роботи). Це означає, що відбувається комерціалізація науково-технічної розробки. Цей напрямок має пріоритетне значення, оскільки результати розробки можуть використовуватися іншими споживачами, приносячи певний економічний ефект. Для цього необхідно знайти потенційного інвестора, який зацікавиться реалізацією цього проекту, та переконати його в економічній доцільності такого кроку.

Для цього слід виконати наступні етапи робіт:

1) провести комерційний аудит науково-технічної розробки, тобто встановлення її науково-технічного рівня та комерційного потенціалу;

2) розрахувати витрати на здійснення науково-технічної розробки;

3) розрахувати економічна ефективність науково-технічної розробки у випадку її впровадження і комерціалізації потенційним інвестором і проведено обґрунтування економічної доцільності комерціалізації потенційним інвестором.

5.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою проведення комерційного і технологічного аудиту дослідження за темою «Розробка методів та програмного засобу для оптимізації великих мовних моделей для iOS» є оцінювання науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності.

Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням 5-ти бальної системи оцінювання за 12-ма критеріями, наведеними в табл. 5.1.

Таблиця 5.1 – Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

Бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено працездатність продукту в реальних умовах
Ринкові переваги (недоліки)					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає

Продовження табл. 5.1

Бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання науково-технічного рівня та комерційного потенціалу науково-технічної розробки потрібно звести до таблиці 5.2.

Таблиця 5.2 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт (ПІБ, посада)		
	1	2	3
	Бали:		
1. Технічна здійсненність концепції	4	3	3
2. Ринкові переваги (наявність аналогів)	3	4	4
3. Ринкові переваги (ціна продукту)	3	4	4
4. Ринкові переваги (технічні властивості)	3	4	4
5. Ринкові переваги (експлуатаційні витрати)	3	3	3
6. Ринкові перспективи (розмір ринку)	4	3	3
7. Ринкові перспективи (конкуренція)	3	3	3
8. Практична здійсненність (наявність фахівців)	3	3	3
9. Практична здійсненність (наявність фінансів)	4	4	4
10. Практична здійсненність (необхідність нових матеріалів)	4	3	3
11. Практична здійсненність (термін реалізації)	4	4	3
12. Практична здійсненність (розробка документів)	4	4	4
Сума балів	42	42	41
Середньоарифметична сума балів $СБ_c$	41,7		

За результатами розрахунків, наведених в таблиці 5.2, зробимо висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. При цьому використаємо рекомендації, наведені в табл. 5.3.

Таблиця 5.3 – Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Науково-технічний рівень та комерційний потенціал розробки
41…48	Високий
31…40	Вище середнього
21…30	Середній

Продовження таблиці 5.3

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Науково-технічний рівень та комерційний потенціал розробки
11…20	Нижче середнього
0…10	Низький

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Розробка методів та програмного засобу для оптимізації великих мовних моделей для iOS» становить 41,7 балів, що, відповідно до таблиці 5.3, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки високий).

5.2 Розрахунок витрат на проведення науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи на тему «Розробка методів та програмного засобу для оптимізації великих мовних моделей для iOS», під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуємо за відповідними статтями.

5.2.1 Витрати на оплату праці

До статті «Витрати на оплату праці» належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам, конструкторам, технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам та іншим працівникам, безпосередньо зайнятим виконанням конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці.

Витрати на основну заробітну плату дослідників ($З_о$) розраховуємо у відповідності до посадових окладів працівників, за формулою 5.1 [31]:

$$З_о = \sum_{i=1}^{k} \frac{M_{ni} \cdot t_i}{T_p}, \quad (5.1)$$

де k – кількість посад дослідників залучених до процесу досліджень;

M_{ni} – місячний посадовий оклад конкретного дослідника, грн;

t_i – число днів роботи конкретного дослідника, дні;

T_p – середнє число робочих днів в місяці, T_p=22 дні.

$$З_{o1} = 35000,00 \cdot 60 / 22 = 95454,55 \text{ грн.}$$

Проведені розрахунки зведемо до таблиці 5.4.

Таблиця 5.4 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Керівник проекту	35000	1590,91	60	95454,55
Інженер-розробник програмного забезпечення	30000	1363,64	60	81818,18
Фахівець з машинного навчання	32000	1454,55	60	87272,73
Інженер з обчислювальної ефективності	27000	1227,27	60	73636,36
Фахівець з тестування	25000	1136,36	20	22727,27
Дизайнер інтерфейсу користувача	18000	818,18	30	24545,45
Всього				338181,82

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт НДР розраховуємо за формулою 5.2:

$$З_р = \sum_{i=1}^{n} C_i \cdot t_i \text{,} \quad (5.2)$$

де C_i – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду C_i можна визначити за формулою 5.3:

$$C_i = \frac{M_M \cdot K_i \cdot K_c}{T_p \cdot t_{зм}}, \quad (5.3)$$

де M_M – розмір прожиткового мінімуму працездатної особи, або мінімальної місячної заробітної плати, приймемо *Мм*=8000,00 грн;

K_i – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду (табл. Б.2, додаток Б) [32];

K_c – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об'єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати.

T_p – середнє число робочих днів в місяці, приблизно T_p = 22 дні;

$t_{зм}$ – тривалість зміни, год.

$C_1 = 8000{,}00 \cdot 1{,}10 \cdot 1{,}65 / (22 \cdot 8) = 82{,}5$ грн.

$З_{р1} = 82{,}5 \cdot 8{,}00 = 577{,}5$ грн.

Величина витрат на основну заробітну плату робітників наведена в табл.5.5.

Таблиця 5.5 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн	Величина оплати на робітника грн
Підготовка робочого місця дослідника	7	2	1,1	82,5	577,5
Встановлення програмного забезпечення	5	2	1,1	82,5	412,5
Налаштування апаратного забезпечення	4	3	1,35	101,25	405
Всього					1395

Додаткова заробітна плата дослідників та робітників

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою 5.4 [33]:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \quad (5.4)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати. Приймемо 12%.

$З_{дод}$ = (338181,82 + 1395) · 12 / 100% = 40749,22 грн.

5.2.2 Відрахування на соціальні заходи

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою 5.5:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%} \quad (5.5)$$

де $Н_{зп}$ – норма нарахування на заробітну плату. Приймаємо 22%.

Зн = (338181,82 + 1395+ 40749,22) · 22 / 100% = 83671,73 грн.

5.2.3 Сировина та матеріали

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби і предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень.

Витрати на матеріали (*М*), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою 5.6:

$$М = \sum_{j=1}^{n} Н_j \cdot Ц_j \cdot К_j - \sum_{j=1}^{n} В_j \cdot Ц_{в\,j}, \quad (5.6)$$

де $Н_j$ – норма витрат матеріалу *j*-го найменування, кг;

n – кількість видів матеріалів;

$Ц_j$ – вартість матеріалу *j*-го найменування, грн/кг;

$К_j$ – коефіцієнт транспортних витрат, ($К_j$ = *1,1 … 1,15*);

B_j – маса відходів *j*-го найменування, кг;

$Ц_{вj}$ – вартість відходів *j*-го найменування, грн/кг.

$М_1 = 3 \cdot 209{,}00 \cdot 1{,}15 - 0{,}000 \cdot 0{,}00 = 721{,}05$ грн.

Проведені розрахунки зведемо до таблиці 5.6.

Таблиця 5.6 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за од	Норма витрат, од	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн
Набір гелевих ручок Buromax Erase Slim	209	3	0	0	721,05
Лоток для паперів Buromax Пластиковий Вертикальний 3 в 1	118	3	0	0	407,1
Ящик паперу офісного Zoom Stora Enso A4 80 г/м2 клас C+ 5 пакувань по 500 аркушів	799	1	0	0	918,85
Дирокол Axent Exakt-2 40 аркушів	384	1	0	0	441,6
Набір папок-швидкозшивачів Buromax Job PP A4 12 шт	78	3	0	0	269,1
Всього					2757,7

5.2.4 Розрахунок витрат на комплектуючі

Витрати на комплектуючі *($К_в$)*, які використовують при проведенні НДР на тему «Розробка методів та програмного засобу для оптимізації великих мовних моделей для iOS» відсутні.

5.2.5 Спецустаткування для наукових (експериментальних) робіт

До статті «Спецустаткування для наукових (експериментальних) робіт» належать витрати на виготовлення та придбання спецустаткування необхідного для проведення досліджень, також витрати на їх проектування, виготовлення,

транспортування, монтаж та встановлення.

Витрати на спецустаткування, які використовують при проведенні НДР на тему «Розробка методів та програмного засобу для оптимізації великих мовних моделей для iOS» відсутні.

5.2.6 Програмне забезпечення для наукових (експериментальних) робіт

До статті «Програмне забезпечення для наукових (експериментальних) робіт» належать витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення, (програм, алгоритмів, баз даних) необхідних для проведення досліджень, також витрати на їх проектування, формування та встановлення.

Балансову вартість програмного забезпечення розраховуємо за формулою5.8:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot C_{прг.i} \cdot K_i, \quad (5.8)$$

де $Ц_{iпрг}$ – ціна придбання одиниці програмного засобу даного виду, грн;

$C_{прг.i}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, ($K_i = 1,10…1,12$);

k – кількість найменувань програмних засобів.

$B_{прг} = 9999,00 \cdot 6 \cdot 1,1 = 65993,4$ грн.

Отримані результати зведемо до таблиці 5.7.

Таблиця 5.7 – Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Прикладний пакет Microsoft Office 2019	6	9999	65993,4
Сервіс розробки інтерфейсів Figma	1	1800	2070
Середовище розробки XCode	3	4100	14145
Всього			82208,4

5.2.7 Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду

обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою 5.9:

$$А_{обл} = \frac{Ц_б}{Т_в} \cdot \frac{t_{вик}}{12}, \quad (5.9)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$Т_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$А_{обл} = (270000{,}00 \cdot 3) / (3 \cdot 12) = 22500$ *грн.*

Проведені розрахунки зведемо до таблиці 5.8.

Таблиця 5.8 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Ноутбук Apple MacBook Air 13.6" M2 8/256GB (6 шт.)	270000	3	3	22500
Робоче місце дослідника	20000	5	3	1000
Оргтехніка	15000	4	3	937,5
Пакет Microsoft Office 2021 (6 шт.)	65993,4	3	3	5499,45

Продовження таблиці 5.8

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Сервіс розробки інтерфейсів Figma	2070	1	3	517,5
Всього				30454,45

5.2.8 Паливо та енергія для науково-виробничих цілей

Витрати на силову електроенергію (B_e) розраховуємо за формулою 5.10:

$$B_e = \sum_{i=1}^{n} \frac{W_{yi} \cdot t_i \cdot Ц_e \cdot K_{впі}}{\eta_i}, \quad (5.10)$$

де W_{yi} – встановлена потужність обладнання на визначеному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_e$ – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо $Ц_e$ = 11 грн;

$K_{впі}$ – коефіцієнт, що враховує використання потужності, $K_{впі} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$.

$B_e = 0,1 \cdot 480,0 \cdot 11 \cdot 0,95 / 0,97 \cdot 6 = 3102,68$ грн.

Проведені розрахунки зведемо до таблиці 5.9.

Таблиця 5.9 – Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Персональний комп'ютер (6 шт)	0,1	480	3102,68
Робоче місце дослідника	0,25	480	1292,78

Продовження таблиці 5.9

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Оргтехніка	0,6	16	103,42
Всього			4498,89

5.2.9 Службові відрядження

До статті «Службові відрядження» дослідної роботи належать витрати на відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих розробленням досліджень, відрядження, пов'язані з проведенням випробувань машин та приладів, а також витрати на відрядження на наукові з'їзди, конференції, наради, пов'язані з виконанням конкретних досліджень.

Витрати за статтею «Службові відрядження» розраховуємо як 20…25% від суми основної заробітної плати дослідників та робітників за формулою 5.11:

$$B_{св} = (З_о + З_р) \cdot \frac{Н_{св}}{100\%}, \quad (5.11)$$

де $Н_{св}$ – норма нарахування за статтею «Службові відрядження», приймемо $Н_{св} = 25\%$.

$В_{св}$ = (338181,82 + 1395) · 25 / 100% = 84894,20 грн.

5.2.10 Витрати на роботи, які виконують сторонні підприємства, установи і організації

Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» розраховуємо як 30…45% від суми основної заробітної плати дослідників та робітників за формулою 5.12:

$$B_{сп} = (З_о + З_р) \cdot \frac{Н_{сп}}{100\%}, \quad (5.12)$$

де $Н_{сп}$ – норма нарахування за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації», приймемо $Н_{сп}$= 40%.

$В_{сп}$ = (338181,82 + 1395) · 40 / 100% = 135830,73 грн.

5.2.11 Інші витрати

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою 5.13:

$$I_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (5.13)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати», приймемо $Н_{ів} = 60\%$.

$I_в = (338181,82 + 1395) \cdot 60 / 100\% = 203746,09$ грн.

5.2.12 Накладні (загальновиробничі) витрати

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов'язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов'язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов'язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою 5.14:

$$В_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (5.14)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо $Н_{нзв} = 130\%$.

$В_{нзв} = (338181,82 + 1395) \cdot 100 / 130\% = 441449,86$ грн.

Витрати на проведення науково-дослідної роботи розраховуємо як суму всіх попередніх статей витрат за формулою 5.15:

$$B_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + В_{спец} + В_{прг} + А_{обл} + В_е + В_{св} + В_{сп} + І_в + В_{нзв}. \quad (5.15)$$

$B_{заг}$=338181,82+1395+40749,22+83671,73+2757,7+0,00+82208,4+30454,45+ 4498,89+84894,20+135830,73+203746,09+441449,86= 1449838,09 грн.

Загальні витрати *ЗВ* на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою 5.16:

$$ЗВ = \frac{B_{заг}}{\eta}, \quad (5.16)$$

де η - коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи, приймемо η=0,7.

ЗВ = 1449838,09/ 0,7 = 2071197,27 грн.

5.3 Розрахунок економічної ефективності науково-технічної розробки при її можливій комерціалізації потенційним інвестором

В ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку.

Результати дослідження передбачають комерціалізацію протягом 3-х років реалізації на ринку.

В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

ΔN – збільшення кількості споживачів продукту, у періоди часу, що аналізуються, від покращення його певних характеристик;

1-й рік – 2800 користувачів/рік;

2-й рік – 5500 користувачів/рік;

3-й рік – 10000 користувачів/рік.

N – кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки, приймемо 50000 користувачів;

$Ц_б$ – вартість програмного продукту у році до впровадження результатів розробки, приймемо 12000,00 грн /за рік користування;

$\pm\Delta Ц_о$ – зміна вартості програмного продукту від впровадження результатів науково-технічної розробки, приймемо 900,00 грн.

Можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із 3-х років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховуємо за формулою 5.17:

$$\Delta П_i = (\pm\Delta Ц_о \cdot N + Ц_о \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \qquad (5.17)$$

де λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2024 році ставка податку на додану вартість складає 20%, а коефіцієнт $\lambda = 0,8333$;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту. Приймемо $\rho = 30\%$;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2024 році $\vartheta = 18\%$;

Збільшення чистого прибутку 1-го року:

$\Delta П_1 =$ (50000,00·900,00+12900·2800)·0,83·0,3·(1-0,18/100%)=16628934,82 грн.

Збільшення чистого прибутку 2-го року:

$\Delta П_2 =$ (50000,00·900,00+12900·(2800+5500))·0,83·0,3·(1-0,18/100%)= 28934165,09

грн.

Збільшення чистого прибутку 3-го року:

$\Delta П_3 =$ (50000,00·900,00+12900·(2800+5500+10000))·0,83·0,3·(1-0,18/100%)= 43419963,13 грн.

Приведена вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки, розраховується за формулою 5.18:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \qquad (5.18)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

T – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ =0,1;

t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$ПП =$ 16628934,82/(1+0,1)^1+ 28934165,09/(1+0,1)^2+ 43419963,13/(1+0,1)^3= 48028616,08 грн.

Величина початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки, розраховується за формулою 5.19:

$$PV = k_{інв} \cdot ЗВ, \qquad (5.19)$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо $k_{інв}$=4;

$ЗВ$ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо *2071197,27* грн.

$PV = k_{інв} \cdot ЗВ$ = 4· *2071197,27* = 8284789,07 грн.

Абсолютний економічний ефект $E_{абс}$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки розраховується за формулою 5.20:

$$E_{абс} = ПП - PV \tag{5.20}$$

де $ПП$ – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 48028616,08 грн;

PV – теперішня вартість початкових інвестицій, 8284789,07 грн.

$E_{абс} = ПП - PV$ = 48028616,08 - 8284789,07 = 39743827,01 грн.

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки, розраховується за формулою 5.21:

$$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1, \tag{5.21}$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, 39743827,01 грн;

PV – теперішня вартість початкових інвестицій, 8284789,07 грн;

$T_{ж}$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, 3 роки.

$$E_{в} = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1 = (1+39743827,01/8284789,07)^{1/3}= 0,796.$$

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$ розраховується за формулою 5.22:

$$\tau_{мін} = d + f, \quad (5.22)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2024 році в Україні d =0,096;

f – показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,25.

$\tau_{мін}$= 0,09+0,25 = 0,346 < 0,796 свідчить про те, що внутрішня економічна дохідність інвестицій $E_{в}$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки вища мінімальної внутрішньої дохідності. Тобто інвестувати в науково-дослідну роботу за темою «Розробка методів та програмного засобу для оптимізації великих мовних моделей для iOS» доцільно.

Період окупності інвестицій $T_{ок}$ які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки, розраховується за формулою 5.23:

$$T_{ок} = \frac{1}{E_{в}}, \quad (5.23)$$

де $E_{в}$ – внутрішня економічна дохідність вкладених інвестицій.

$T_{ок}$ = 1 / 0,796 = 1,26 року.

$T_{ок}$ < 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

5.4 Висновки

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Розробка методів та програмного засобу для оптимізації великих мовних моделей для iOS» становить 41,7 балів, що, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки високий).

Також термін окупності становить 1,26 р., що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Отже можна зробити висновок про доцільність проведення науково-дослідної роботи за темою «Розробка методів та програмного засобу для оптимізації великих мовних моделей для iOS».

ВИСНОВКИ

У процесі виконання магістерської кваліфікаційної роботи було розроблено методи та програмні засоби iOS-системи для оптимізації великих мовних моделей на мобільних пристроях. Робота виконана відповідно до методичних рекомендацій [34].

Проаналізовано стан даного питання на сьогоднішній день. Розглянуто основні підходи до оптимізації мовних моделей, зокрема методи LoRA, квантізації та інші адаптивні алгоритми, які дозволяють покращити продуктивність моделей на мобільних платформах. У результаті аналізу обрано мову програмування Swift, інтегроване середовище розробки Xcode та SwiftUI для побудови інтерфейсу користувача. Використано CoreData для зберігання історії оптимізацій та AppStorage для налаштувань додатку.

Розроблено метод гібридного адаптивного скорочення рангу, який поєднує кілька підходів до зниження розмірності моделі, що дозволяє оптимально зменшити обсяг необхідної пам’яті без втрати точності моделі. Це значно покращує продуктивність додатку на мобільних пристроях, де обмежені ресурси пам’яті та обчислювальних потужностей.

Розроблено метод динамічного налаштування ваг на основі апаратних характеристик пристрою, який адаптує параметри моделі відповідно до технічних можливостей конкретного мобільного пристрою. Такий підхід дозволяє максимально ефективно використовувати обчислювальні ресурси, підвищуючи загальну швидкодію та енергоефективність роботи моделі на різних пристроях.

Розроблено метод квантової компресії мовних моделей для мобільних пристроїв, який забезпечує значне скорочення розміру моделі шляхом оптимізації кількості бітів, що використовуються для збереження параметрів. Це дозволяє істотно зменшити вимоги до пам’яті та підвищує ефективність роботи додатку, зберігаючи при цьому точність і функціональність мовних моделей.

Тестування програми підтвердило її працездатність, відповідність

поставленому технічному завданню та досягнення всіх визначених завдань дослідження. Проведено експериментальні дослідження, результати яких продемонстрували ефективність запропонованих методів оптимізації. Максимальне використання CPU під час виконання моделі може бути знижено до 93.5%, що значно зменшує енергоспоживання та тепловиділення пристрою. Кількість ваг моделі скоротилася на 30%, що дозволило зменшити обчислювальну складність. Використання пам’яті зменшилося на 37%, забезпечуючи більшу кількість вільних ресурсів для інших процесів. Час генерації відповіді у тюрінг-тесті скоротився на 39%, що суттєво покращило швидкість взаємодії. Розмір моделі зменшився на 25%, спрощуючи її зберігання та передачу. При цьому точність залишилася високою, із втратою лише 1.3%.

Розроблено інструкцію користувача для швидкого ознайомлення з функціоналом застосунку, а також зазначено мінімальні вимоги до пристрою для забезпечення стабільної роботи програми.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Розробка мобільних додатків від А до Я: повний гайд [Електронний ресурс] – Режим доступу до ресурсу: https://dan-it.com.ua/uk/blog/rozrobka-mobilnih-dodatkiv-vid-a-do-ja-povnij-gajd/.

2. Романюк О.В. Вплив оптимізацій великих мовних моделей для iOS на розвиток освітніх, медичних і розважальних додатків / О.В. Романюк, Р.С.Луценко // Електронні інформаційні ресурси: створення, використання, доступ. Збірник матеріалів IV конференції молодих вчених, аспірантів та студентів «Комп'ютерні ігри та мультимедіа як інноваційний підхід до комунікації – 2024» 26-27 вересня 2024 р. – Одеса: Видавництво ОНТУ, 2024 р.– с. 294-296.

3. Луценко Р.С. Перспективи застосування гібридного адаптивного скорочення рангу для оптимізації великих мовних моделей на мобільних пристроях / Р.С. Луценко, О.В. Романюк // Електронні інформаційні ресурси: створення, використання, доступ. Збірник матеріалів міжнародної науково-практичної Інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ та управління – 2024» 20-21 листопада 2024 р. – Суми/Вінниця: НІКО/КЗВО «Вінницька академія безперервної освіти», 2024 р.– с. 102-103.

4. Apple iOS 17: Нові можливості для розробників [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/ios/.Comment [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/reference/org/w3c/dom/Comment

5. Core ML Tools [Електронний ресурс] – Режим доступу до ресурсу: https://apple.github.io/coremltools/docs-guides/.

6. TensorFlow-Lite [Електронний ресурс] – Режим доступу до ресурсу: https://www.cloudskillsboost.google/paths/17/course_templates/17/video/504915?locale=uk.

7. ONNX Runtime Mobile [Електронний ресурс] – https://onnxruntime.ai/docs/get-started/with-mobile.html

8. Hugging Face Transformers Lite [Електронний ресурс] – Режим доступу до ресурсу: https://huggingface.co/AnatoliiPotapov/T-lite-instruct-0.1

9. NVIDIA TensorRT [Електронний ресурс] – Режим доступу до ресурсу: https://developer.nvidia.com/tensorrt

10. Кросплатформова розробка додатків [Електронний ресурс] – Режим доступу до ресурсу: https://wezom.com.ua/ua/blog/krossplatformennaya-razrabotka-prilozhenij.

11. Переваги та недоліки кросплатформної та нативної розробки мобільних додатків [Електронний ресурс] – Режим доступу до ресурсу: https://merehead.com/ua/blog/cross-platform-native-mobile-development/.

12. Побудова UI за допомогою модифікаторів у SwiftUI [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/ documentation/swiftui/modifier.

13. Компоненти для роботи зі станами в SwiftUI [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com /documentation/swiftui/state-management..

14. SwiftUI Lifecycle: Використання @State та @Binding [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/swiftui/lifecycle

15. Практичний підхід до SwiftUI: повний посібник / Mark Moeykens // SwiftUI Handbook – 2021. – С. 15-120.

16. Оптимізація продуктивності SwiftUI [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/documentation/ swiftui/performance.

17. Попередній перегляд в SwiftUI [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/documentation/swiftui/preview.

18. Особливості роботи з @Published та Combine у Swift [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@swiftcombine..

19. Патерн MVVM у SwiftUI: Як розподілити обов’язки між Model, View і ViewModel [Електронний ресурс] – Режим доступу до ресурсу: https://betterprogramming.pub/mvvm-in-swiftui.

20. Ефективне використання MVVM у SwiftUI [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/swiftui-mvvm.

21. Будуємо краще майбутнє з SwiftUI [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/swiftui/future.

22. Патерн Data Class у Swift [Електронний ресурс] – Режим доступу до ресурсу: https://swift.org/documentation/data-classes.

23. Xcode: функціональні можливості для iOS-розробки [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/xcode/.

24. UML для побудови бізнес-логіки у мобільних додатках [Електронний ресурс] – Режим доступу до ресурсу: https://uml-models.com.

25. Діаграми прецедентів у розробці iOS-додатків [Електронний ресурс] – Режим доступу до ресурсу: https://uml-diagrams.com/ios-use-case-diagrams.

26. Основи програмування на Swift [Електронний ресурс] – Режим доступу до ресурсу: https://swift.org/documentation/

27. Розпочинаємо роботу зі Swift [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/introduction-to-swift..

28. Collin M. Shattuck. Machine Learning for Mobile with Core ML. – New York, USA: Apress, 2018. – 250 p..

29. Тестування мобільних додатків у Swift: Посібник для початківців / J. Anderson, L. Thompson. – Київ: Видавництво «Ліга», 2021.

30. Основи тестування мобільних застосунків для iOS [Електронний ресурс] – Режим доступу до ресурсу: https://qaitesting.com/mobile-testing.

31. Кавецький В. В. Економічне обґрунтування інноваційних рішень: практикум / В. В. Кавецький, В. О. Козловський, І. В. Причепа. Вінниця : ВНТУ, 2016. 113 с.

32. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. Вінниця : ВНТУ, 2021. 42 с.

33. Положення про кваліфікаційну роботу на другому (вищому) рівні вищої освіти. Розробники: А.О. Семенов, Л.П. Громова, Т.В. Макарова, О.В. Сердюк. Вінниця : ВНТУ, 2021. – 60 с.

34. Методичні вказівки до виконання магістерської кваліфікаційної роботи для студентів спеціальності 121 «Інженерія програмного забезпечення» / уклад. : О. Н. Романюк, Г. О. Черноволик. – Вінниця : ВНТУ, 2022. – 50 с.

ДОДАТКИ

111

ДОДАТОК А Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ

д.т.н., проф. О. Н. Романюк

«19» вересня 2024 р.

Технічне завдання

на магістерську кваліфікаційну роботу «Розробка методів та програмного засобу для оптимізації великих мовних моделей для iOS» за спеціальністю 121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:

к.т.н., доц. каф. ПЗ. О.В. Романюк

«19» вересня 2024 р.

Виконав:

студент гр.2ПІ-23м Р. С. Луценко

«19» вересня 2024 р.

Вінниця – 2024 року

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Розробка методів та програмного засобу для оптимізації великих мовних моделей для iOS».

Галузь застосування – мобільні системи штучного інтелекту.

2. Підстава для розробки.

Підставою для виконання магістерської кваліфікаційної роботи (МКР) є індивідуальне завдання на МКР та наказ ректора №310 від 17 вересня 2024 р. по ВНТУ про закріплення тем МКР.

3. Мета та призначення розробки.

Метою роботи є підвищення ефективності роботи великих мовних моделей на мобільних платформах шляхом розробки спеціалізованих методів оптимізації, які дозволять адаптувати такі моделі для використання на пристроях з обмеженими обчислювальними ресурсами.

Призначення роботи – розробка методів та програмних засобів iOS-системи для оптимізації великих мовних моделей.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись МКР.

1. Large language models surpass human experts in predicting neuroscience results [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/yugen-ai-technology-blog/efficient-llm-fine-tuning-lora-dora-and-apples-innovative-approach-ea1b0b31c0a7

2. LLM Illustrated Word Embeddings [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/towards-data-science/nlp-illustrated-part-2-word-embeddings-6d718ac40b7d

3. Deep Learning Illustrated: How Does A Neural Network Work [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/towards-data-science/neural-networks-illustrated-part-1-how-does-a-neural-network-work-c3f92ce3b462

5. Технічні вимоги

Вхідні дані до роботи: методи оптимізації – метод Low-Rank Approximation (LRA), метод Low-Rank Adaptation (LoRA), метод групового прунінгу, метод динамічної компресії параметрів, метод квантування параметрів; середовище розробки – XCode; мова розробки – Swift.

6. Конструктивні вимоги.

Інтерфейс програми повинен відповідати естетичним та ергономічним вимогам та мати зручну навігацію і засоби керування.

Графічна та текстова документація повинна відповідати діючим стандартам України.

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

– пояснювальна записка до МКР;

– технічне завдання;

– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз стану розвитку великих мовних моделей та їх адаптації для мобільних платформ. Постановка задач дослідження	20.09.2024 30.09.2024
2	Розробка методів та алгоритмів оптимізації великих мовних моделей для iOS	01.10.2024 17.10.2024
3	Розробка програмної системи для оптимізації великих мовних моделей для iOS	18.10.2024 04.11.2024
4	Експериментальні дослідження розроблених методів та тестування програми	05.11.2024 21.11.2024
5	Економічна частина	22.11.2024 30.11.2024

10. Порядок контролю та прийняття.

Виконання етапів магістерської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи. Прийняття магістерської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

115

ДОДАТОК Б

Протокол перевірки на плагіат

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: **Розробка методів та програмного засобу для оптимізації великих мовних моделей для iOS.**

Тип роботи: кваліфікаційна робота
Підрозділ : кафедра програмного забезпечення, ФІТКІ, 2ПІ-23м
Науковий керівник: к.т.н., доц. каф. ПЗ. Романюк О.В.

Turnitin	
Оригінальність	**97%**
Схожість	3%

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**
□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.
□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Заявляю, що ознайомлений з повним звітом подібності, який був згенерований Системою щодо роботи «Розробка методів та програмного засобу для оптимізації великих мовних моделей для iOS».

Автор _____ Луценко Руслан Сергійович

Опис прийнятого рішення: **допустити до захисту**

Особа, відповідальна за перевірку _____ Черноволик Г. О.
(підпис) (прізвище, ініціали)

Експерт _____ _____
(за потреби) (підпис) (прізвище, ініціали, посада)

ДОДАТОК В

Лістинг програми

```
import SwiftUI

@main
struct SwiftyMLCApp: App {
    var body: some Scene {
        WindowGroup {
            InformationPopup_Preview.previews
            ModelSelectionButton_Preview.previews
            ChatScrollView_Preview.previews
            ChatTextField_Previews.previews
            content
        }
    }

    @ViewBuilder
    private var content: some View {
        if
            let config = createConfig(),
            createModelDirectory() {
            ChatScreen()
                .environment(\.appConfig, config)
        } else {
            ContentUnavailableView(
                "Something went wrong",
                systemImage: "exclamationmark.triangle",
                description: Text("Please contact us")
            )
        }
    }

    func createConfig() -> MLCAppConfig? {
        do {
            return try MLCAppConfigManager().retrieve()
        } catch {
            return nil
        }
    }

    func createModelDirectory() -> Bool {
        do {
            try FileManager.default.createDirectory(at: Constants.modelsDirectory, withIntermediateDirectories:
true)

            let fileURLs = try FileManager.default.contentsOfDirectory(at: Constants.modelsDirectory,
includingPropertiesForKeys: nil)
            for file in fileURLs {
                print(file)
            }
```

```
            return true
        } catch {
            return false
        }
    }
}

import SwiftUI

struct ModelSelectionButton: View {

    @Binding var model: MLCAppConfig.Model?
    let chatManagerPhase: ChatManager.Phase
    @State private var isSheetShown: Bool = false

    private var isDisabled: Bool {
        switch chatManagerPhase {
        case .loading, .unloading, .generating:
            true
        case .unloaded, .ready:
            false
        }
    }

    private var title: String {
        switch chatManagerPhase {
        case .loading:
            "Model (Loading)"
        case .ready:
            "Model"
        case .unloading:
            "Model (Unloading)"
        case .unloaded:
            "Model"
        case .generating:
            "Model (Generating)"
        }
    }

    private var subtitle: String {
        switch chatManagerPhase {
        case .loading(let model):
            model.name
        case .ready(let model):
            model.name
        case .unloading(let model):
            model.name
        case .unloaded:
            "Select a Model"
        case .generating(let model):
            model.name
        }
    }

    @ViewBuilder
    private var value: some View {
        switch chatManagerPhase {
```

```
        case .loading, .unloading, .generating:
            ProgressView()
        case .unloaded, .ready:
            Image(systemName: "rectangle.2.swap")
                .foregroundStyle(.secondary)
        }
    }

    var body: some View {
        Button {
            isSheetShown = true
        } label: {
            LabeledContent {
                value
            } label: {
                Text(title)
                    .font(.headline)
                Text(subtitle)
            }
            .padding()
            .background(Color.Chat.assistant)
            .cornerRadiusDefault
            .padding()
        }
        .disabled(isDisabled)
        .animation(.linear(duration: 0.2), value: chatManagerPhase)
        .buttonStyle(.plain)
        .sheet(isPresented: $isSheetShown) {
            ModelsScreen(model: .init(selectedModel: $model))
        }
    }
}

struct ModelSelectionButton_Preview: PreviewProvider {
    static var previews: some View {
        NavigationStack {
            ScrollView {
                Text("Hello")
            }
            .safeAreaInset(edge: .top) {
                ModelSelectionButton(model: .constant(.mock), chatManagerPhase:
.ready(MLCAppConfig.Model.mock))
            }
        }
    }
}
import Foundation

extension ByteCountFormatter {
    static func makeForTotal() -> ByteCountFormatter {
        let formatter = ByteCountFormatter()
        formatter.allowedUnits = [.useGB]
        formatter.zeroPadsFractionDigits = true
        formatter.allowsNonnumericFormatting = false
        return formatter
    }
```

```
    static func makeForCompleted() -> ByteCountFormatter {
        let formatter = ByteCountFormatter()
        formatter.allowedUnits = [.useMB, .useGB]
        formatter.zeroPadsFractionDigits = true
        formatter.allowsNonnumericFormatting = false
        return formatter
    }
}
import SwiftUI

extension Color {
    static var contentBackground: Color {
        Color(uiColor: .systemGroupedBackground)
    }

    static var contentForeground: Color {
        Color(uiColor: .secondarySystemGroupedBackground)
    }

    enum InformationPopup {
        static let background = Color(
            _light: Color.contentForeground.mix(with: Color.accentColor, by: 0.1),
            _dark: Color.contentForeground.mix(with: Color.accentColor, by: 0.1)
        )
    }

    enum Chat {
        static let background: Color = Color(
            _light: Color(uiColor: .secondarySystemGroupedBackground),
            _dark: Color(uiColor: .systemGroupedBackground)
        )

        static let assistant: Color = Color(
            _light: Color(uiColor: .systemGroupedBackground),
            _dark: Color(uiColor: .secondarySystemGroupedBackground)
        )

        static let user: Color = Color(
            _light: Color.accentColor,
            _dark: Color.accentColor.mix(with: .black, by: 0.4)
        )

        static let input: Color = assistant
        static let inputBorder: Color = .accentColor
    }
}

// https://www.jessesquires.com/blog/2023/07/11/creating-dynamic-colors-in-swiftui/

#if canImport(AppKit)
import AppKit
#endif

#if canImport(UIKit)
import UIKit
#endif
```

```
private extension Color {
    init(_light light: Color, _dark dark: Color) {
        #if canImport(UIKit)
        self.init(_light: UIColor(light), _dark: UIColor(dark))
        #else
        self.init(_light: NSColor(light), _dark: NSColor(dark))
        #endif
    }

    #if canImport(UIKit)
    init(_light light: UIColor, _dark dark: UIColor) {
        #if os(watchOS)
        // watchOS does not support light mode / dark mode
        // Per Apple HIG, prefer dark-style interfaces
        self.init(uiColor: dark)
        #else
        self.init(uiColor: UIColor(dynamicProvider: { traits in
            switch traits.userInterfaceStyle {
            case .light, .unspecified:
                return light

            case .dark:
                return dark

            @unknown default:
                assertionFailure("Unknown userInterfaceStyle: \(traits.userInterfaceStyle)")
                return light
            }
        }))
        #endif
    }
    #endif

    #if canImport(AppKit)
    init(_light: NSColor, _dark: NSColor) {
        self.init(nsColor: NSColor(name: nil, dynamicProvider: { appearance in
            switch appearance.name {
            case .aqua,
                 .vibrantLight,
                 .accessibilityHighContrastAqua,
                 .accessibilityHighContrastVibrantLight:
                return light

            case .darkAqua,
                 .vibrantDark,
                 .accessibilityHighContrastDarkAqua,
                 .accessibilityHighContrastVibrantDark:
                return dark

            default:
                assertionFailure("Unknown appearance: \(appearance.name)")
                return light
            }
        }))
    }
    #endif
}
```

```
private class TextFieldDelegate: NSObject, UITextFieldDelegate {

    static let shared = TextFieldDelegate()

    private override init() {}

    func textFieldShouldReturn(_ textField: UITextField) -> Bool {
        false
    }
}

extension TextField {
    func enableNewLineOnReturnKey() -> some View {
        introspect(.textField, on: .iOS(.v18)) { textField in
            textField.delegate = TextFieldDelegate.shared
        }
    }
}
import SwiftUI

extension View {
    @ViewBuilder
    var cornerRadiusChat: some View {
        clipShape(RoundedRectangle(cornerRadius: 18))
    }

    @ViewBuilder
    var cornerRadiusDefault: some View {
        clipShape(RoundedRectangle(cornerRadius: 16))
    }

    @ViewBuilder
    var cornerRadiusSmall: some View {
        clipShape(RoundedRectangle(cornerRadius: 8))
    }
}
public extension View {
    func onFirstAppear(_ action: @escaping () -> ()) -> some View {
        modifier(FirstAppear(action: action))
    }
}

private struct FirstAppear: ViewModifier {
    let action: () -> ()

    @State private var hasAppeared = false

    func body(content: Content) -> some View {
        // And then, track it here
        content.onAppear {
            guard !hasAppeared else { return }
            hasAppeared = true
            action()
        }
    }
}
```

```
extension View {
   func roundedBorder(cornderRadius: CGFloat, color: Color, lineWidth: CGFloat) -> some View {
      cornerRadius(cornderRadius)
         .overlay(
            RoundedRectangle(cornerRadius: cornderRadius)
               .strokeBorder(color, lineWidth: lineWidth)
         )
   }
}
import Foundation
import Tagged
import SwiftUI

extension EnvironmentValues {
   @Entry var appConfig: MLCAppConfig = MLCAppConfig(modelList: [])
}

struct MLCAppConfig: Decodable {
   struct Model: Decodable, Identifiable, Hashable {

      static let mock: Self = .init(
         modelId: "Llama 3.2 1B",
         modelLib: "Library",
         modelUrl: "htttps://google.com",
         estimatedVramBytes: 30000000,
         name: "Llama 3.2 1B",
         bytes: 3000000,
         group: "Llama 3.2"
      )

      typealias ModelId = Tagged<(Model, modelId: ()), String>
      typealias Lib = Tagged<(Model, lib: ()), String>
      typealias ModelUrl = Tagged<(Model, modelUrl: ()), URL>

      var id: ModelId { modelId }
      let modelId: ModelId
      let modelLib: Lib
      /// The repository URL
      let modelUrl: ModelUrl
      let estimatedVramBytes: Int
      let name: String
      let bytes: Int64
      let group: String

      /// Local directory where all the model details are stored.
      var localDirectory: URL { Constants.modelsDirectory.appending(path: modelId.rawValue)}
      var localDirectoryMlcChatConfig: URL { localDirectory.appending(path:
Constants.fileNameMlcChatConfig) }
      var localDirectoryNdarrayCache: URL { localDirectory.appending(path:
Constants.fileNameNdArrayCache)}
      /// Retrieved from: https://huggingface.co/mlc-ai/Llama-3.2-1B-Instruct-q4f16_1-
MLC/blob/d09d47e97caae5c79a01fc4889ec3b46ea551fd4/mlc-chat-config.json#L41
      /// Example: https://huggingface.co/mlc-ai/Llama-3.2-1B-Instruct-q4f16_1-
MLC/blob/main/tokenizer.json
      func localDirectory(tokenizerFile: MlcChatConfig.TokenizerFile) -> URL {
localDirectory.appending(path: tokenizerFile.rawValue) }
```

```
        /// Retrieved from: https://huggingface.co/mlc-ai/Llama-3.2-1B-Instruct-q4f16_1-
MLC/blob/d09d47e97caae5c79a01fc4889ec3b46ea551fd4/ndarray-cache.json#L7
        /// Example: https://huggingface.co/mlc-ai/Llama-3.2-1B-Instruct-q4f16_1-
MLC/blob/main/params_shard_0.bin
        func localDirectory(ndarrayCacheRecord: NdarrayCache.Record) -> URL {
localDirectory.appending(path: ndarrayCacheRecord.dataPath) }
        var localDirectoryTemporaryDownloadLocation: URL {
            Constants.temporaryDirectory
                .appending(path: modelId.rawValue)
        }
        /// Temporary download location for an `NdarrayCache.Record`
        func localDirectoryTemporaryDownloadLocation(nsarrayCacheRecord: NdarrayCache.Record) -> URL
{
            localDirectoryTemporaryDownloadLocation
                .appending(path: nsarrayCacheRecord.dataPath)
        }
        var localDirectoryFinishedFile: URL { localDirectory.appending(path: "finished.txt") }

        /// Contains config and other files
        var repositoryFilesUrl: URL { modelUrl.rawValue.appending(path: "resolve").appending(path:
"main")}
        var repositoryFileMlcChatConfigUrl: URL { repositoryFilesUrl.appending(path:
Constants.fileNameMlcChatConfig) }
        var repositoryFileNdarrayCacheUrl: URL { repositoryFilesUrl.appending(path:
Constants.fileNameNdArrayCache) }
        /// Retrieved from: https://huggingface.co/mlc-ai/Llama-3.2-1B-Instruct-q4f16_1-
MLC/blob/d09d47e97caae5c79a01fc4889ec3b46ea551fd4/mlc-chat-config.json#L41
        /// Example: https://huggingface.co/mlc-ai/Llama-3.2-1B-Instruct-q4f16_1-
MLC/blob/main/tokenizer.json
        func repositoryFile(tokenizerFile: MlcChatConfig.TokenizerFile) -> URL {
repositoryFilesUrl.appending(path: tokenizerFile.rawValue) }
        /// Retrieved from: https://huggingface.co/mlc-ai/Llama-3.2-1B-Instruct-q4f16_1-
MLC/blob/d09d47e97caae5c79a01fc4889ec3b46ea551fd4/ndarray-cache.json#L7
        /// Example: https://huggingface.co/mlc-ai/Llama-3.2-1B-Instruct-q4f16_1-
MLC/blob/main/params_shard_0.bin
        func repositoryFile(ndarrayCacheRecord: NdarrayCache.Record) -> URL {
repositoryFilesUrl.appending(path: ndarrayCacheRecord.dataPath) }
    }

    let modelList: [Model]
}

struct MLCAppConfigManager {

    func retrieve() throws -> MLCAppConfig {
        guard let file = Bundle.main.url(forResource: "bundle/mlc-app-config", withExtension: "json") else {
            throw AppError("Could not find mlc-app-config.json")
        }
        let data = try Data(contentsOf: file)
        let decoder = JSONDecoder()
        decoder.keyDecodingStrategy = .convertFromSnakeCase
        let config = try decoder.decode(MLCAppConfig.self, from: data)
        return config
    }
}

import Foundation
```

```
struct ExtendedAttributeError: Error {
   let code: Int32
   let name: String
   let description: String

   init(errno: Int32) {
      code = errno
      switch errno {
      case EEXIST:
         name = "EEXIST"
         description = "Options contains XATTR_CREATE and the named attribute already exists."
      case ENOATTR:
         name = "ENOATTR"
         description = "GET: The extended attribute does not exist. SET: Options is set to
XATTR_REPLACE and the named attribute does not exist."
      case ENOTSUP:
         name = "ENOTSUP"
         description = "The file system does not support extended attributes or has them disabled."
      case EROFS:
         name = "EROFS"
         description = "The file system is mounted read-only."
      case ERANGE:
         name = "ERANGE"
         description = "GET: Value (as indicated by size) is too small to hold the extended attribute data. SET:
The data size of the attribute is out of range."
      case EPERM:
         name = "EPERM"
         description = "Attributes cannot be associated with this type of object."
      case EINVAL:
         name = "EINVAL"
         description = "Name or options is invalid."
      case EISDIR:
         name = "EISDIR"
         description = "Path or fd do not refer to a regular file and the attribute in question is only applicable
to files."
      case ENOTDIR:
         name = "ENOTDIR"
         description = "A component of path is not a directory."
      case ENAMETOOLONG:
         name = "ENAMETOOLONG"
         description = "Name exceeded XATTR_MAXNAMELEN UTF-8 bytes, or a component of path
exceeded NAME_MAX characters, or the entire path exceeded PATH_MAX characters."
      case EACCES:
         name = "EACCES"
         description = "Search permission is denied for a component of path or permission to set the attribute
is denied."
      case ELOOP:
         name = "ELOOP"
         description = "Too many symbolic links were encountered resolving path."
      case EFAULT:
         name = "EFAULT"
         description = "Path or name points to an invalid address."
      case EIO:
         name = "EIO"
         description = "An I/O error occurred while reading from or writing to the file system."
      case E2BIG:
```

```
            name = "E2BIG"
            description = "The data size of the extended attribute is too large."
        case ENOSPC:
            name = "ENOSPC"
            description = "Not enough space left on the file system."
        default:
            name = "Unknown"
            description = "Unknown error."
        }
    }
}

extension FileManager {
    func extendedAttribute(_ name: String, on url: URL) throws -> Data {
        // Get size of attribute data
        var size = getxattr(url.path, name, nil, 0, 0, 0)
        if size == -1 {
            throw ExtendedAttributeError(errno: errno)
        }

        // Prepare buffer
        let alignment = MemoryLayout<Int8>.alignment
        let ptr = UnsafeMutableRawPointer.allocate(byteCount: size, alignment: alignment)
        defer {
            ptr.deallocate()
        }
        size = getxattr(url.path, name, ptr, size, 0, 0)

        return Data(bytes: ptr, count: size)
    }

    func setExtendedAttribute(_ name: String, on url: URL, data: Data) throws {
        try data.withUnsafeBytes { (bytes: UnsafeRawBufferPointer) -> Void in
            guard let ptr = bytes.baseAddress else {
                return
            }
            let result = setxattr(url.path, name, ptr, data.count, 0, 0)
            if result == -1 {
                throw ExtendedAttributeError(errno: errno)
            }
        }
    }

    func removeExtendedAttribute(_ name: String, from url: URL) throws {
        let result = removexattr(url.path, name, 0)
        if result == -1 {
            // If the attribute was already gone, do nothing
            if errno != ENOATTR {
                throw ExtendedAttributeError(errno: errno)
            }
        }
    }
}
//
//  AppDelegate.swift
//  AssistAI
//
```

```
//  Created by Konstantin Gonikman on 23.05.23.
//

import Cocoa
import ServiceManagement
import os.log
import Combine
import Settings

final class AppDelegate: NSObject, NSApplicationDelegate {
    private let statusItem = NSStatusBar.system.statusItem(withLength:NSStatusItem.variableLength)
    private let mStatus = NSMenuItem(title: "Index is up-to-date", action: nil, keyEquivalent: "")
    var window: NSWindow!
    private let ingester = Ingester()
    private let paletteColors1: [NSColor] = [.lightGray, .white]
    private let paletteColors2: [NSColor] = [.white, .lightGray]
    private var colorAnimationSwitch = false
    private var timer: Timer?
    private let isRunningSubject: CurrentValueSubject<Bool, Never> = .init(false)
    private var cancellables: Set<AnyCancellable> = Set<AnyCancellable>()
    private let networkService = NetworkService()

    override init() {
        super.init()

        self.setupUserSettingsManager()
        self.ingester.isRunningSubject = isRunningSubject
    }

    private var isWindowEffectivelyVisible: Bool {
        return NSApplication.shared.isActive && window.isVisible
    }

    private lazy var settingsWindowController = SettingsWindowController(
        panes: [
            FoldersSettingsViewController(),
            GeneralSettingsViewController()
        ]
    )

    @MainActor
    private func updateStatusMenu(isIngestingRunning: Bool) {
        if isIngestingRunning {
            mStatus.title = "Lumira is indexing your files..."
            let statusConfig = NSImage.SymbolConfiguration(hierarchicalColor: NSColor.systemYellow)
            mStatus.image = NSImage(systemSymbolName: "circle.fill", accessibilityDescription: nil)?
                .withSymbolConfiguration(statusConfig)
        } else {
            mStatus.title = "Index is up-to-date"
            let statusConfig = NSImage.SymbolConfiguration(hierarchicalColor: NSColor.systemGreen)
            mStatus.image = NSImage(systemSymbolName: "circle.fill", accessibilityDescription: nil)?
                .withSymbolConfiguration(statusConfig)
        }
    }

    private func constructMenu() {
        if let button = statusItem.button {
```

```
            button.image = NSImage(systemSymbolName: "mountain.2.fill", accessibilityDescription: nil)
        }

        let menu = NSMenu()

        let statusConfig = NSImage.SymbolConfiguration(hierarchicalColor: NSColor.systemGreen)
        mStatus.image = NSImage(systemSymbolName: "circle.fill", accessibilityDescription: nil)?
            .withSymbolConfiguration(statusConfig)
        menu.addItem(mStatus)

        menu.addItem(NSMenuItem.separator())

        let mAsk = NSMenuItem(title: "Ask Lumira...", action:
#selector(AppDelegate.didTapOpenMainWindow(_:)), keyEquivalent: "a")
        mAsk.keyEquivalentModifierMask = .command
        mAsk.image = NSImage(systemSymbolName: "questionmark.bubble", accessibilityDescription: nil)
        menu.addItem(mAsk)

        menu.addItem(NSMenuItem.separator())

        menu.addItem(NSMenuItem(title: "View Session Logs...", action:
#selector(AppDelegate.didTapViewSessionLogs(_:)), keyEquivalent: ""))
        let miSettings = NSMenuItem(title: "Settings...", action:
#selector(AppDelegate.didTapOpenSettings(_:)), keyEquivalent: ",")
        miSettings.keyEquivalentModifierMask = .command
        menu.addItem(miSettings)

        menu.addItem(NSMenuItem.separator())

        let mQuit = NSMenuItem(title: "Quit", action: #selector(AppDelegate.didTapQuit(_:)), keyEquivalent:
"q")
        mQuit.keyEquivalentModifierMask = [.command]
        mQuit.image = NSImage(systemSymbolName: "power", accessibilityDescription: nil)
        menu.addItem(mQuit)

        statusItem.menu = menu
    }

    @objc func didTapOpenMainWindow(_ sender: Any?) {
        if window != nil {
            if !self.isWindowEffectivelyVisible {
                NSApp.activate(ignoringOtherApps: true)
                return
            }
            return
        }

        let storyboard = NSStoryboard(name: "Main", bundle: nil)
        guard let windowController = storyboard.instantiateInitialController() as?
                NSWindowController else { return }

        guard let window = windowController.window else {
            return
        }

        self.window = window
```

```
        window.makeKeyAndOrderFront(self)
        NSApp.activate(ignoringOtherApps: true)
    }

    @objc func didTapViewSessionLogs(_ sender: Any?) {
        OSLog.general.debug("View logs...")
    }

    @objc func didTapOpenSettings(_ sender: Any?) {
        settingsWindowController.show()
    }

    @objc func didTapQuit(_ sender: Any?) {
        let alert = NSAlert()
        alert.messageText = "Stop indexing?"
        alert.informativeText = "Lumira is the background process that indexes your files. Quitting it means all
indexing activity will stop."
        alert.addButton(withTitle: "Don't Quit")
        alert.addButton(withTitle: "Quit")

        let modalResult = alert.runModal()

        switch modalResult {
        case .alertFirstButtonReturn:
            OSLog.general.debug("Cancel button clicked")
        case .alertSecondButtonReturn:
            NSApplication.shared.terminate(self)
        default:
            break
        }
    }

    var lastIsIngesterRunningState = false
    func applicationDidFinishLaunching(_ aNotification: Notification) {
        constructMenu()

        isRunningSubject
            .receive(on: DispatchQueue.main)
            .sink { [weak self] isLocalIngesterRunning in
                guard let self else { return }
                if lastIsIngesterRunningState != isLocalIngesterRunning {
                    OSLog.general.debug("==> Is local ingester running: \(isLocalIngesterRunning)")
                    lastIsIngesterRunningState = isLocalIngesterRunning

                    if isLocalIngesterRunning {
                        startCheckingForRemoteIngester()
                    }
                }
            }
            .store(in: &cancellables)

        self.startIngester(removedFolders: nil)
    }

    private func setupUserSettingsManager() {
        UserSettingsManager.shared.foldersChangePublisher
            .dropFirst()
```

```
            .debounce(for: .seconds(3), scheduler: DispatchQueue.main)
            .sink { [weak self] (_, removedFolders) in
                self?.startIngester(removedFolders: removedFolders)
            }
            .store(in: &cancellables)
    }

    private func startIngester(removedFolders: [URL]?) {
        Task(priority: .utility) {
            await ingester.resetQueue()
            await ingester.start()
            ingester.setupFileWatcher()

            if let removedFolders {
                removedFolders.forEach { ingester.enqueueRemoveFolderCall(folder: $0) }
            }
        }
    }

    private func startCheckingForRemoteIngester() {
        Task {
            await startTimer()
            await updateStatusMenu(isIngestingRunning: true)

            while true {
                try? await Task.sleep(for: .seconds(5)) // TODO: initial 5 secs might be too little!

                let isRemoteIngesterRunning = await networkService.isRemoteIngesterRunning()
                switch isRemoteIngesterRunning {
                case .success(let value):
                    OSLog.general.debug("==> Is remote ingester running: \(value)")
                    if !value {
                        await self.stopTimer()
                        await self.updateStatusMenu(isIngestingRunning: false)
                        return
                    }
                case .failure:
                    await self.stopTimer()
                    await self.updateStatusMenu(isIngestingRunning: false)
                    return
                }
            }
        }
    }

    @MainActor
    private func startTimer() {
        self.timer = Timer.scheduledTimer(timeInterval: 0.37, target: self, selector:
#selector(updateStatusItemConfig), userInfo: nil, repeats: true)

        if let timer = self.timer {
            RunLoop.current.add(timer, forMode: .common)
        }
    }

    @MainActor
    private func stopTimer() {
```

```
        self.timer?.invalidate()
        self.timer = nil

        if let button = statusItem.button {
            button.image = NSImage(systemSymbolName: "mountain.2.fill", accessibilityDescription: nil)
        }
    }

    @objc func updateStatusItemConfig() {
        let colors: [NSColor] = colorAnimationSwitch ? paletteColors1 : paletteColors2
        let statusConfig = NSImage.SymbolConfiguration(paletteColors: colors)

        if let button = statusItem.button {
            button.image = NSImage(systemSymbolName: "mountain.2.fill", accessibilityDescription: nil)?
                .withSymbolConfiguration(statusConfig)
        }

        self.colorAnimationSwitch.toggle()
    }

    func applicationWillTerminate(_ aNotification: Notification) {
        // Insert code here to tear down your application
    }

    func applicationSupportsSecureRestorableState(_ app: NSApplication) -> Bool {
        return true
    }
}
//
//  Ingester.swift
//  Ingester
//
//  Created by Konstantin Gonikman on 22.05.23.
//

import Foundation
import os.log
import FileWatcher
import CryptoKit
import Combine

final class Ingester {
    private var filewatcher: FileWatcher?
    private let validExtensions = [
        "pdf"
    ]

    private let uploadCallQueue = APICallQueueActor()
    private let networkService = NetworkService()
    private var cancellables: Set<AnyCancellable> = Set<AnyCancellable>()

    var isRunningSubject: CurrentValueSubject<Bool, Never>?

    func start() async {
        OSLog.ingester.log("Start Ingester...")

        guard let isRunningSubject = self.isRunningSubject else {
```

```
        OSLog.ingester.fault("isRunningSubject not set.")
        fatalError("isRunningSubject not set.")
    }

    // It has to be a separate task, otherwise this call blocks
    Task {
        await uploadCallQueue.run(isRunningSubject: isRunningSubject)
    }

    let allFiles = filesInAllDirectories()
    let filesToIndex = filesToBeIndexed(at: allFiles)

    OSLog.ingester.log("Files to be indexed:")
    filesToIndex.forEach { OSLog.ingester.log("=> \($0.path(percentEncoded: false))") }

    filesToIndex.forEach { enqueueCall(filePath: $0) }
}

func setupFileWatcher() {
    OSLog.ingester.log("Setup FileWatcher...")

    if filewatcher != nil {
        filewatcher?.stop()
    }

    let pathDirectories = UserSettingsManager.shared.getFolders().map { $0.path }
    guard pathDirectories.count > 0 else {
        OSLog.ingester.warning("No folders selected, FileWatcher not started.")
        return
    }

    filewatcher = FileWatcher(pathDirectories)
    filewatcher?.queue = DispatchQueue.global(qos: .utility)

    filewatcher?.callback = { [weak self] event in
        guard let self else { return }

        OSLog.ingester.log("=> \(event.path); \(event.description)")

        guard event.fileCreated || event.fileRemoved || event.fileRenamed || event.fileModified else {
            OSLog.ingester.log("==> Insignificant event at `\(event.path)`, ignoring.")
            return
        }

        let url = URL(filePath: event.path)
        let shouldBeIndexed = fileShouldBeIndexed(at: url)
        if shouldBeIndexed {
            OSLog.ingester.log("==> File at `\(event.path)` should be indexed.")
            self.enqueueCall(filePath: url)
        } else {
            OSLog.ingester.log("==> File at `\(event.path)` should NOT be indexed.")
        }
    }

    filewatcher?.start()
}
```

ДОДАТОК Г
Ілюстративна частина

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА МЕТОДІВ ТА ПРОГРАМНОГО ЗАСОБУ ДЛЯ ОПТИМІЗАЦІЇ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ ДЛЯ IOS

ВІННИЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ
КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА НА ТЕМУ:
«РОЗРОБКА МЕТОДІВ ТА ПРОГРАМНОГО ЗАСОБУ ДЛЯ ОПТИМІЗАЦІЇ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ ДЛЯ IOS»

АВТОР: СТ. ГРУПИ 2ПІ-23М ЛУЦЕНКО Р.С.
НАУКОВИЙ КЕРІВНИК: К.Т.Н., ДОЦ. КАФ. ПЗ РОМАНЮК О.В.

ВІННИЦЯ – 2024

Рисунок Г.1 – Назва роботи

АКТУАЛЬНІСТЬ ТЕМИ

До оптимізації:
❖Параметри моделі:
- ❖ Висока кількість ваг.
- ❖ Багато нейронів у шарах.
- ❖ Використання 32-бітних форматів (FP32).
- ❖ Велика кількість параметрів та значне споживання пам'яті.
- ❖ Велике навантаження на процесор.

Після оптимізації:
❖Зміни в параметрах:
- ❖ Скорочення кількості ваг
- ❖ Видалення зайвих шарів і нейронів.
- ❖ Перехід до 8-бітних форматів (INT8) через квантування.
- ❖ Зменшення розміру моделі.
- ❖ Менше навантаження на процесор.

Рисунок Г.2 – Актуальність теми

РОЗРОБКА МЕТОДІВ ТА ПРОГРАМНОГО ЗАСОБУ ДЛЯ ОПТИМІЗАЦІЇ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ ДЛЯ IOS.

- **Мета дослідження.** Метою магістерської кваліфікаційної роботи є підвищення ефективності роботи великих мовних моделей на мобільних платформах шляхом розробки спеціалізованих методів оптимізації, які дозволять адаптувати такі моделі для використання на пристроях з обмеженими обчислювальними ресурсами.
- **Об'єкт дослідження.** Процес розробки та оптимізації великих мовних моделей для мобільних пристроїв на платформі iOS
- **Предмет дослідження.** Методи та засоби оптимізації великих мовних моделей для мобільних пристроїв на платформі iOS.

Рисунок Г.3 – Мета, об'єкт та предмет дослідження

РОЗРОБКА МЕТОДІВ ТА ПРОГРАМНОГО ЗАСОБУ ДЛЯ ОПТИМІЗАЦІЇ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ ДЛЯ IOS.

ЗАВДАННЯ

- проаналізувати існуючі методи оптимізації мовних моделей для мобільних пристроїв;
- розробити метод гібридного адаптивного скорочення рангу;
- розробити метод динамічного налаштування ваг на основі апаратних характеристик пристрою;
- розробити метод квантової компресії мовних моделей для мобільних пристроїв;
- розробити програмні модулі компонентів мобільного застосунку «OptiLang» для оптимізації великих мовних моделей;
- розробити зручний та зрозумілий графічний інтерфейс;
- повести експериментальні дослідження розроблених методів;
- провести тестування розробленої iOS-системи;
- розробити інструкцію користувача та описати системні вимоги розробленої програми.

Рисунок Г.4 – Завдання дослідження

РОЗРОБКА МЕТОДІВ ТА ПРОГРАМНОГО ЗАСОБУ ДЛЯ ОПТИМІЗАЦІЇ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ ДЛЯ IOS.

НАУКОВА НОВИЗНА ОТРИМАНИХ РЕЗУЛЬТАТІВ

- Подальшого розвитку отримав метод гібридного адаптивного скорочення рангу (HARR), який, на відміну від відомих, поєднує техніки LoRA та прунингу з урахуванням індивідуальних технічних характеристик мобільного пристрою, що дозволило зменшити розмір моделі на 25.7% при збереженні високої точності та знизити максимальне використання процесора до 93.5%.
- Подальшого розвитку отримав метод динамічного налаштування ваг на основі апаратних характеристик пристрою (DHWA), який, на відміну від існуючих методів, автоматично адаптує параметри моделі залежно від поточних обчислювальних можливостей пристрою, що дозволило зменшити енергоспоживання та час генерації відповіді на 39%, підвищуючи продуктивність у реальних умовах.
- Подальшого розвитку набув метод квантової компресії мовних моделей для мобільних пристроїв, який, на відміну від існуючих методів, використовує адаптивну стратегію вибору рівня квантування для кожного шару моделі залежно від його важливості, що забезпечило зменшення кількості ваг моделі на 30%, зниження використання пам'яті на 37% при збереженні високої точності.

Рисунок Г.5 – Наукова новизна

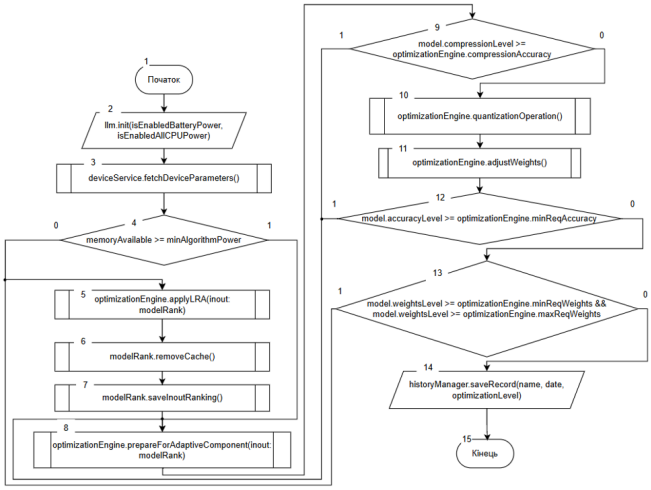
РОЗРОБКА МЕТОДІВ ТА ПРОГРАМНОГО ЗАСОБУ ДЛЯ ОПТИМІЗАЦІЇ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ ДЛЯ IOS.

ПРАКТИЧНА ЦІННІСТЬ ОТРИМАНИХ РЕЗУЛЬТАТІВ

Практична цінність полягає у тому, що на основі отриманих у магістерській кваліфікаційній роботі теоретичних положень запропоновано алгоритми та програмні засоби для оптимізації великих мовних моделей та їх адаптації для мобільних пристроїв.

Рисунок Г.6 – Практична цінність отриманих результатів

МЕТОД ГІБРИДНОГО АДАПТИВНОГО СКОРОЧЕННЯ РАНГУ



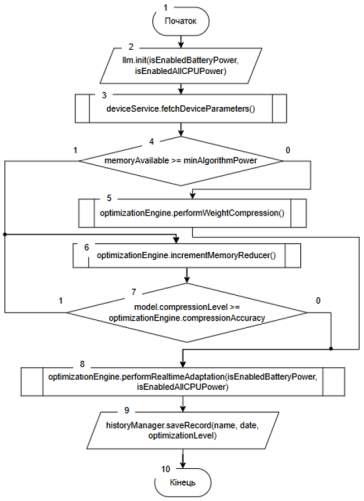
Метод гібридного адаптивного скорочення рангу (HARR) поєднує техніки Low-Rank Adaptation (LoRA) та адаптивного прунінгу для ефективної оптимізації мовних моделей на мобільних пристроях, враховуючи індивідуальні технічні характеристики. Основна ідея полягає у поетапному зменшенні розміру моделі, збереженні її точності та адаптації до ресурсів пристрою.

Основні етапи методу:

- **Ініціалізація:** Завантаження моделі та збирання апаратних характеристик пристрою.
- **Аналіз ресурсів:** Оцінка доступної пам'яті та потужності процесора.
- **Скорочення рангу:** Застосування LRA для початкової оптимізації структури моделі.
- **Адаптивна компресія:** Використання прунінгу для додаткового зменшення розмірності.
- **Квантування:** Зниження розрядності параметрів моделі для економії пам'яті.
- **Налаштування ваг:** Динамічне налаштування параметрів залежно від характеристик пристрою.
- **Валідація:** Перевірка точності моделі після кожного етапу.
- **Збереження:** Збереження оптимізованої моделі для швидкого доступу.

Рисунок Г.7 – Метод гібридного адаптивного скорочення рангу

МЕТОД ДИНАМІЧНОГО НАЛАШТУВАННЯ ВАГ



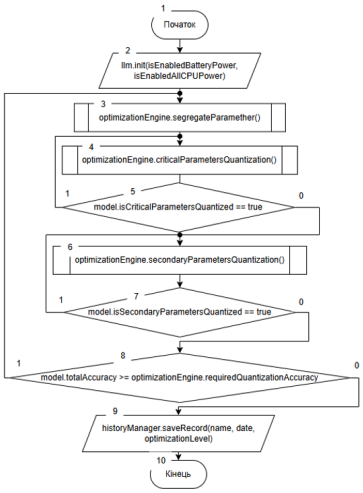
Метод динамічного налаштування ваг (DHWA) адаптує параметри моделі в реальному часі залежно від обчислювальних можливостей пристрою. Це дозволяє досягти оптимального балансу між продуктивністю, точністю та енергоефективністю. Основна мета – адаптивно зменшувати ресурсні вимоги моделі, враховуючи поточний стан пристрою.

Основні етапи методу:

- **Ініціалізація моделі:** Завантаження моделі та отримання її параметрів.
- **Аналіз характеристик пристрою:** Збір інформації про доступну пам'ять, потужність CPU та інші ресурси.
- **Адаптація ваг:** Компресія ваг із врахуванням доступних ресурсів, що включає кілька ітерацій для досягнення оптимального рівня компресії.
- **Динамічне налаштування в реальному часі:** Під час роботи моделі параметри ваг коригуються залежно від змін навантаження та умов роботи пристрою.
- **Збереження:** Результати оптимізації зберігаються в кеші для швидкого доступу.

Рисунок Г.8 – Метод динамічного налаштування ваг

МЕТОД КВАНТОВОЇ КОМПРЕСІЇ МОВНИХ МОДЕЛЕЙ ДЛЯ МОБІЛЬНИХ ПРИСТРОЇВ



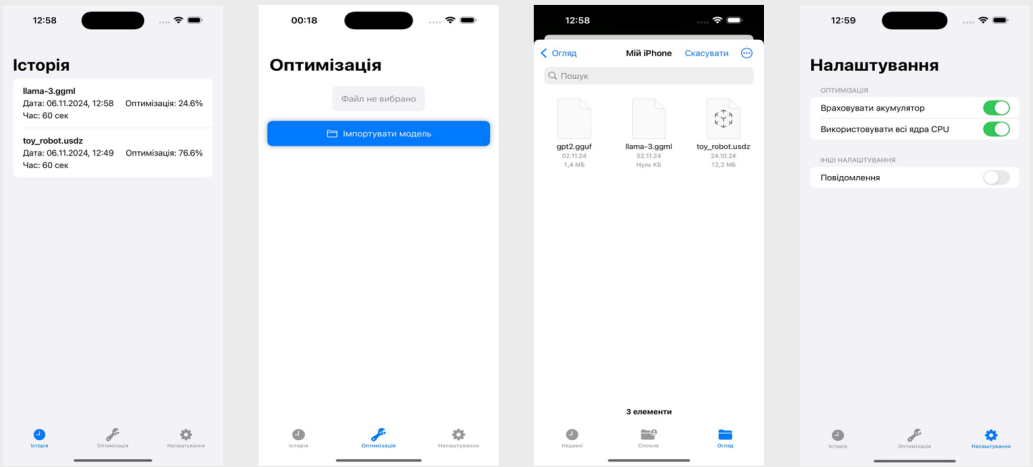
Метод квантової компресії мовних моделей оптимізує розмір моделі шляхом зменшення розрядності її параметрів. Особливістю методу є адаптивна стратегія, яка застосовує високоточне квантування для критичних параметрів і стандартне квантування для некритичних. Це дозволяє забезпечити ефективну компресію при мінімальних втратах точності.

Основні етапи методу:

- **Ініціалізація моделі:** Завантаження параметрів моделі та підготовка до компресії.
- **Сегментація параметрів:** Розподіл параметрів на критичні (що впливають на точність) і некритичні.
- **Високоточне квантування:** Квантування критичних параметрів з використанням точних алгоритмів для збереження якості.
- **Стандартне квантування:** Квантування некритичних параметрів з метою економії ресурсів.
- **Валідація:** Перевірка точності моделі після компресії та її відповідності заданим вимогам.
- **Збереження:** Збереження оптимізованих параметрів у кеш для подальшого використання.

Рисунок Г.9 – Метод квантової компресії мовних моделей для мобільних пристроїв

ТЕСТУВАННЯ IOS-СИСТЕМИ ДЛЯ ОПТИМІЗАЦІЇ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ



Головні вікна програми

Рисунок Г.10 – Головні вікна програми

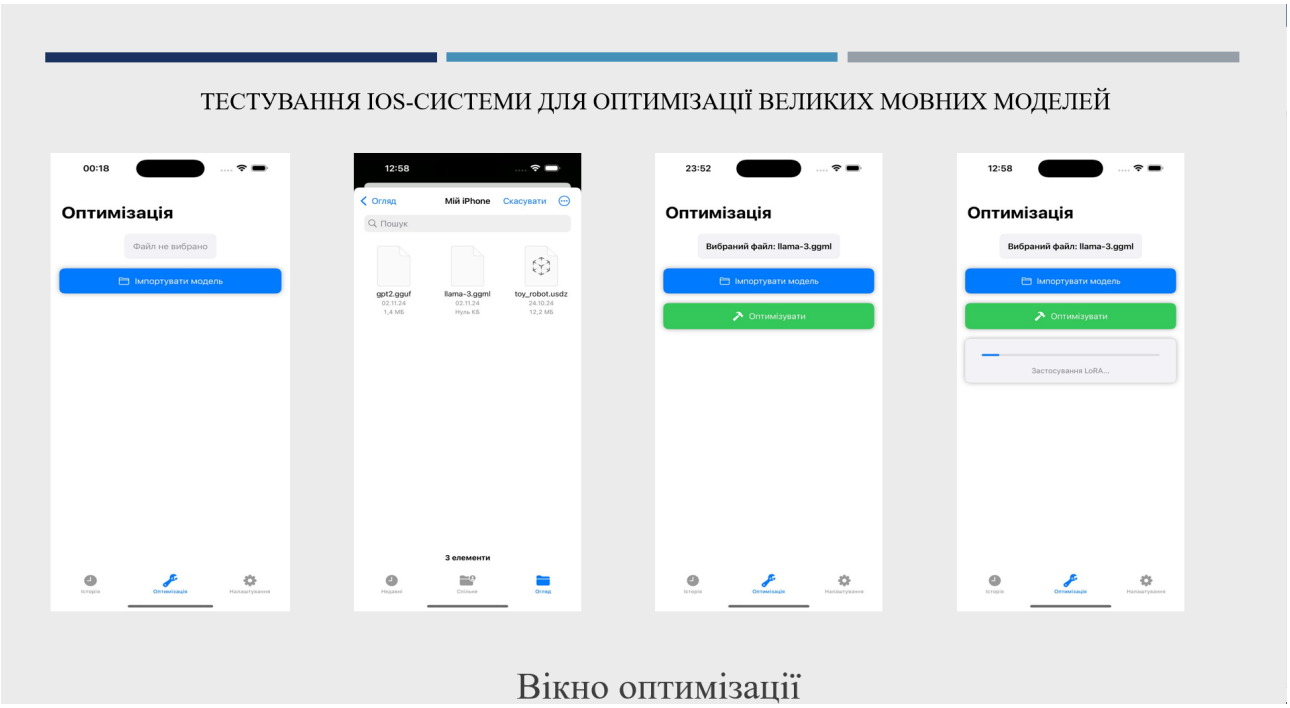


Рисунок Г.11 – Вікно оптимізації

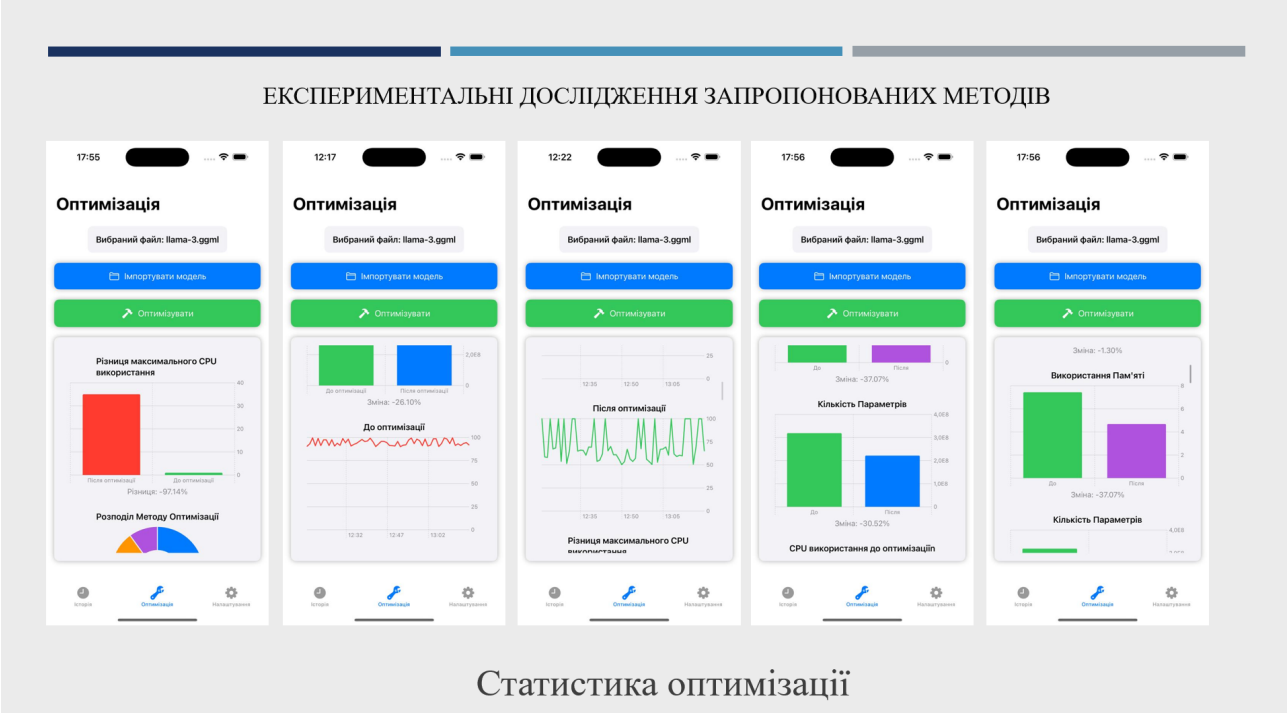


Рисунок Г.12 – Експериментальні дослідження запропонованих методів (частина перша)

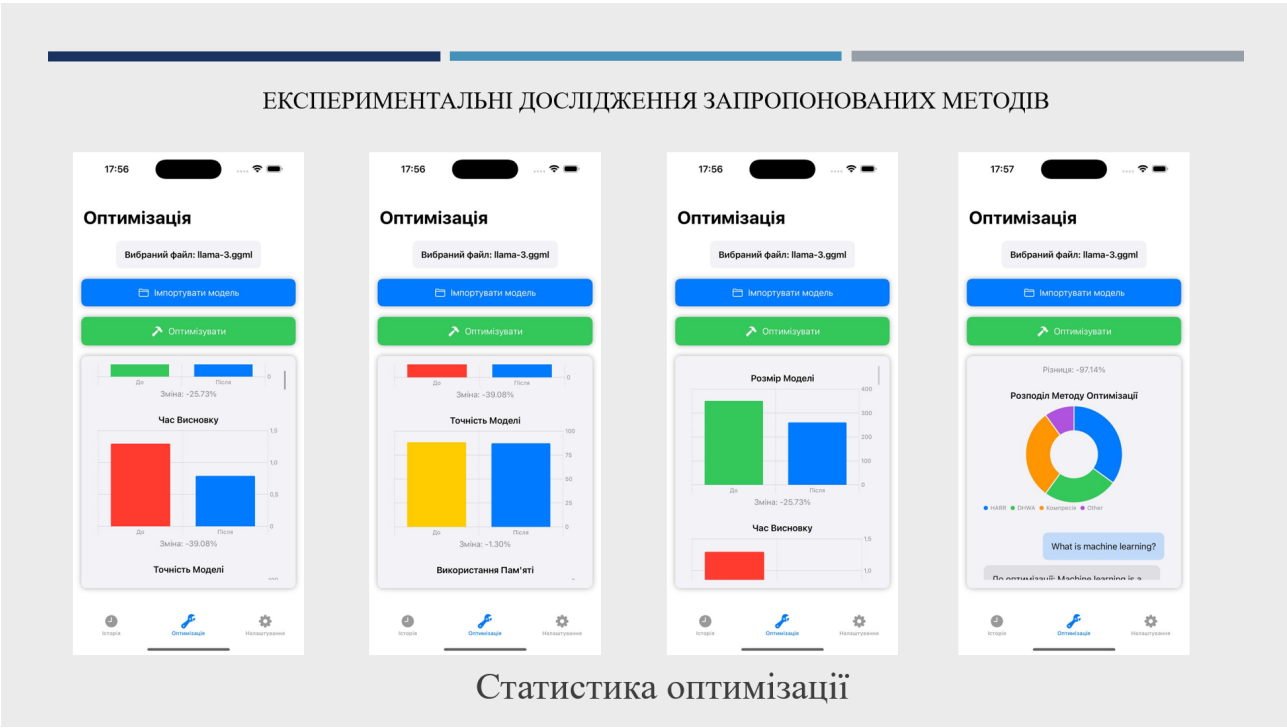


Рисунок Г.13 – Експериментальні дослідження запропонованих методів (частина друга)

Рисунок Г.14 – Демонстрація програмного засобу

АПРОБАЦІЇ ТА ПУБЛІКАЦІЇ

Апробація матеріалів магістерської кваліфікаційної роботи. Результати магістерської кваліфікаційної роботи обговорювалися на:

- IV Всеукраїнській науково-технічній конференції молодих вчених, аспірантів і студентів «Комп'ютерні ігри та мультимедіа як інноваційний підхід до комунікації» (Одеса, 2024 р.);
- Міжнародній науково-практичній Інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ» (Суми/Вінниця, 2024).

Публікації. Результати дослідження опубліковані в тезах доповіді:

- Луценко Р.С. Перспективи застосування гібридного адаптивного скорочення рангу для оптимізації великих мовних моделей на мобільних пристроях / Р.С. Луценко, О.В. Романюк // Електронні інформаційні ресурси: створення, використання, доступ. Збірник матеріалів міжнародної науково-практичної Інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ та управління – 2024» 20-21 листопада 2024 р. – Суми/Вінниця: НІКО/КЗВО «Вінницька академія безперервної освіти», 2024 р.– с. 102-103.
- Романюк О.В. Вплив оптимізацій великих мовних моделей для iOS на розвиток освітніх, медичних і розважальних додатків / О.В. Романюк, Р.С. Луценко // Електронні інформаційні ресурси: створення, використання, доступ. Збірник матеріалів IV конференції молодих вчених, аспірантів та студентів «Комп'ютерні ігри та мультимедіа як інноваційний підхід до комунікації – 2024» 26-27 вересня 2024 р. – Одеса: Видавництво ОНТУ, 2024 р.– с. 294-296.

Рисунок Г.15 – Апробації та публікації

ВИСНОВКИ

- У процесі виконання магістерської кваліфікаційної роботи було проаналізовано існуючі методи оптимізації мовних моделей для мобільних пристроїв;
- Розроблено метод гібридного адаптивного скорочення рангу;
- Розроблено метод динамічного налаштування ваг на основі апаратних характеристик пристрою;
- Розроблено метод квантової компресії мовних моделей для мобільних пристроїв;
- Створено програмні модулі компонентів мобільного застосунку «OptiLang» для оптимізації великих мовних моделей;
- Реалізовано зручний та зрозумілий графічний інтерфейс користувача;
- Проведено експериментальні дослідження розроблених методів із детальним аналізом результатів;
- Виконано тестування розробленої iOS-системи, що підтвердило її відповідність технічному завданню;
- Розроблено інструкцію користувача та описано системні вимоги для стабільної роботи програми.

Рисунок Г.16 – Висновки

Дякую за увагу!

Рисунок Г.17 – Фінальний слайд