Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет __Комп'ютерної інженерії та управління_____
(повна назва)

Кафедра Автоматизації проектування обчислювальної техніки
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти____другий (магістерський)_____
(рівень вищої освіти)

Метод організації машинного навчання для оптимізації витрат пам'яті та часу в хмарних обчисленнях
(тема)

Виконав: студент 2 курсу, групи СКСм-22-2
Кучков Б. О.
(прізвище, ініціали)
Спеціальність *123 _Комп'ютерна інженерія*
Тип програми ____освітньо-професійна_____
(освітньо-професійна або освітньо-наукова)
Освітня програма_ *Спеціалізовані* _____
*_комп'ютерні _системи*_____
(повна назва освітньої прграми)

Керівник роботи _доцент Хаханова Г. В._
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ _____ _Чумаченко С.В
(підпис) (прізвище, ініціали)

2023 р.

Харківський національний університет радіоелектроніки

Факультет_____Комп’ютерної інженерії та управління _____

Кафедра_____Автоматизації проектування обчислювальної техніки_____

Рівень вищої освіти____другий (магістерський)_____

Спеціальність_____123_ Комп’ютерна інженерія _____
(шифр і назва)

Тип програми _____Освітньо-професійна_____
(освітньо-професійна або освітньо-наукова)

Освітня програма _____Спеціалізовані комп’ютерні системи_____
(повна назва

ЗАТВЕРДЖУЮ:

Зав.кафедри_____ __
(підпис)

“____” _____ 2023 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові ___Кучкову Бориславу Олександровичу_____
(прізвище, ім’я, по батькові)

1. Тема роботи (проекту) Метод організації машинного навчання для оптимізації витрат пам'яті та часу в хмарних обчисленнях

затверджена наказом по університету від "03"____11_____2023 р. № _1282 Ст _____

2. Термін подання студентом роботи до екзаменаційної комісії ____15.12.2023_____
3. Вихідні дані до роботи (проекту)_____

Хмарні технології_____

Машинне навчання_____

Машинне навчання у хмарі_____

Мова програмування Python_____

Середовище розробки VSCode_____

4. Перелік питань, що потрібно опрацювати у роботі _____

Особливості навчання нейронних мереж

Робота коду у хмарі

Оптимізація затрат часу та пам’яті при навчанні моделі

Маніпулювання часом та пам’яттю у хмарі

_____

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) 24 слайдів

6. Консультанти розділів роботи (проекту)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

7. Дата видачі завдання 02.09.2023

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи (проекту)	Термін виконання етапів проекту (роботи)	Примітка
1	Видача теми проекту, узгодження і затвердження	02.09.2023 - 03.09.2023	
2	Аналіз існуючих методів розпізнавання жестів рук	02.09.2023 - 10.09.2023	
3	Дослідження алгоритмів розпізнавання жестів рук	02.09.2023 - 17.09.2023	
4	Адаптація та розробка методів розпізнавання жестів руки з використанням нейронних мереж	02.09.2023 - 25.09.2023	
5	Програмна реалізація системи нейронної мережі для розпізнавання жестів рук	02.09.2023 - 05.10.2023	
6	Тестування системи	02.09.2023 - 10.10.2023	
7	Оформлення пояснювальної записки	02.09.2023 - 20.10.2023	
8	Перевірка виконаного проекту керівником	02.09.2023 - 20.10.2023	
9	Захист проекту	02.09.2023 - 12.01.2024	

Студент _____ _____
(підпис)

Керівник роботи _____ _____ доц. Хаханова Г.В.

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 60 с., 14 рис., 6 таблиць., 36 джерел.

МАШИННЕ НАВЧАННЯ, НЕЙРОННІ МЕРЕЖІ, ХМАРНІ ТЕХНОЛОГІЇ, ХМАРНІ ОБЧИСЛЮВАННЯ, AWS, PYTHON, EFS, LAMBDA, ГРАДІЄНТНИЙ СПУСК, MINI BGD, TENSORFLOW

Метою кваліфікаційної роботи є розробка методології організації машинного навчання для оптимізації витрат пам'яті та часу в хмарних обчисленнях, а саме на платформі AWS.

У ході виконання кваліфікаційної роботи було розроблено методологію машинного навчання на алгоритмі – мініпакетний градієнтний спуск, написаного мовою програмування Python, на бібліотеках TensorFlow, Keras та інших. Розгорталась модель на платформі AWS, за допомогою Lambda та EFS.

ABSTRACT

Explanatory note of the qualification work: 60 pages, 14 figures, 6 table, 36 sources.

MACHINE LEARNING, NEURAL NETWORKS, CLOUD TECHNOLOGIES, CLOUD COMPUTING, AWS, PYTHON, EFS, LAMBDA, GRADIENT DESCENT, MINI BGD, TENSORFLOW

The aim of the qualification work is to develop a methodology for organizing machine learning to optimize memory and time consumption in cloud computing, namely on the AWS platform.

In the course of the qualification work, a machine learning methodology was developed based on the mini-batch gradient descent algorithm, written in the Python programming language, using TensorFlow, Keras, and other libraries. The model was deployed on the AWS platform with the help of Lambda and EFS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

EFS – Elastic Flip Storage, гнучке зберігання;

AWS – Веб-сервіси Amazon;

ШІ – штучний інтелект;

CPU – центральний процесор, головна частина апаратного забезпечення комп'ютера чи програмованого логічного контролера;

GPU – графічний процесор, що виконує графічний рендеринг;

TPU – тензорний процесор, що відноситься до класу нейронних процесорів;

ASIC – спеціалізована інтегральна схема;

CNN – згорткові нейронні мережі;

SIFT – масштабно-інваріантне перетворення ознак;

ECR – Elastic Container Registry, реєстр еластичних контейнерів;

CIFAR10 – набір даних складається з 60000 кольорових зображень 32x32 у 10 класах, по 6000 зображень на клас.

ВСТУП

Машинне навчання та штучний інтелект деякий час є останнім трендом у сучасному світі. Машинне навчання дозволяє комп’ютерам вивчати тенденції та закономірності у даних, які надаються на етапі навчання. Комп’ютери навчені приймати рішення щодо нових вхідних даних на основі минулих даних. Ці рішення приймаються без необхідності явного кодування процесу прийняття рішень і ґрунтуються лише на знаннях, отриманих комп’ютером під час навчання. Процес навчання трудомісткий і ресурсномісткий в порівнянні зі звичайними серверами які обробляють скрипти. Ресурси, такі як графічний процесор, оперативна пам’ять і центральний процесор, є основними компонентами, які необхідні для ресурсноємного навчання.

Хмарні обчислення стали ефективним рішенням для вирішення цієї проблеми. Хмарні обчислення дозволяють користувачам використовувати ресурси за відносно невелику вартість, здаючи їх в оренду на необхідний термін. У міру впровадження хмарних обчислень все більше і більше проектів обслуговуються в хмарі. У результаті дослідники та розробники використовують все більше хмарних інстанцій, які потребують постійного моніторингу та обслуговування. Також важливо зазначити, що постачальник хмарних послуг відповідає лише за апаратне забезпечення, що лежить в основі таких послуг. Відповідальність за керування операційною системою, оновлення, виправлення та інше програмне забезпечення належить клієнту. Це також включає керування бібліотеками та їхніми версіями. Без серверні обчислення – це тип послуг, який дозволяє користувачеві зосередити свої зусилля на розробці рішень. Постачальники хмарних послуг абстрагують деталі апаратної та програмної інфраструктури, що лежить в основі, і надають користувачеві декілька опцій, які відносно легко налаштувати відповідно до потреб. Це стає корисним для розробника, оскільки він може витрачати більше

часу на зосередження на проблемі, а не на вирішення зайвих проблем, пов’язаних із сервером. Крім того, без серверні обчислення можна виконувати і в ізольованих середовищах. Це можна використовувати для одночасного тестування кількох гіпотез, що стає перевагою для розробника-дослідника.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз завдання

Представляються сучасні рішення для використання машинне навчання та нейроні мережі у хмарі. Визначаються переваги і недоліки існуючих технологій роботи у хмарі та використання у ній машинне навчання, а також проводиться огляд використовуваних інструментів. Формуються цілі і завдання дослідження, орієнтовані на усунення проблемних місць з машинним навчанням з використанням хмарних технологій. Виконано огляд і аналіз публікацій, що охоплюють:

– нейроні мережі та машинне навчання;
– типи та методи машинного навчання;
– інструменти для розробки та навчання нейронних мереж;
– хмарні обчислювання та використання храми у комп’ютерних системах;
– інструменти для розробки у хмарі;
– використання машинного навчання у хмарі;
– документація Amason Web Service;

1.1.1 Нейронні мережі та машинне навчання

Машинне навчання – це тип штучного інтелекту, який вивчає методи, що здатні автоматично навчатися [1]. Головна мета цієї наукової галузі – дозволити комп’ютерам навчатися без будь-якої допомоги або взаємодії з людиною. Зазвичай для вирішення проблеми задають вхідні дані та алгоритм. Машинне навчання працює з проблемою інакше: маючи вхідні дані та бажаний результат, воно дозволяє отримати «алгоритм». Аналізуючи приклади того, що має бути на вході та виході, метод знаходить оптимальну нелінійну функцію для вирішення задачі.

Існує два типи навчання:

– навчання за прецедентами, або індуктивне навчання, засноване на виявленні емпіричних закономірностей даних;
– дедуктивне навчання передбачає формалізацію знань експертів та його перенесення на комп'ютер як бази знань.

Дедуктивне навчання прийнято відносити до галузі експертних систем, тому терміни машинне навчання та навчання за прецедентами можна вважати синонімами.

Багато методів індуктивного навчання розроблялися як альтернатива класичним статистичним підходам. Багато методів тісно пов'язані із вилученням інформації (information extraction, information retrieval), інтелектуальним аналізом даних (data mining).

Розділ машинного навчання, з одного боку, утворився в результаті поділу науки про нейромережі на методи навчання мереж та види топологій їхньої архітектури, з іншого боку - увібрав у себе методи математичної статистики [5]. Популярні способи машинного навчання засновані на застосуванні нейромереж, хоча існують і інші методи, засновані на навчальній вибірці – наприклад, дискримінантний аналіз, що оперує узагальненою дисперсією та підступністю спостережуваної статистики, або байєсовські класифікатори.

Базові види нейромереж, такі як перцептрон та багатошаровий перцептрон [5, 6] (а також їх модифікації), можуть навчатися як з учителем, так і без вчителя, з підкріпленням та самоорганізацією. Це фундаментальний блок, з якого складаються шари та нейронна мережа загалом. Після того як було отримано значення нейронів на вихідному рівні, відбувається порівняння їх із бажаним результатом і розраховується помилка. Але деякі нейромережі та більшість статистичних методів можна віднести лише до одного із способів навчання. Тому, якщо потрібно класифікувати методи машинного навчання в залежності від способу навчання, то буде некоректним відносити нейромережі

до певного виду, правильніше було б типізувати алгоритми навчання нейронних мереж.

1.1.2 Алгоритми та методи

Існує безліч методів та підходів до машинного навчання, приведемо декілька прикладів:

- лінійна регресія - це алгоритм, який намагається знайти лінійну залежність між вхідними змінними та вихідними змінними. Використовується для задач регресії;
- логістична регресія - це алгоритм, який використовується для моделювання ймовірностей принадлежності до певного класу, використовуючи логістичну функцію. Використовується для задач бінарної класифікації;
- дерева рішень - це алгоритми, які будують дерево рішень на основі навчальних даних та використовують його для класифікації або регресії. Дерева рішень можуть бути простими або складними та використовуються в багатьох галузях;
- випадковий ліс - це алгоритм, який будує кілька дерев рішень та комбінує їх результати для зменшення перенавчання та покращення точності. Використовується для задач класифікації та регресії;
- наївний Баєс - це алгоритм, який використовує теорему Баєса для моделювання вірогідності принадлежності до класу на основі вхідних змінних. Використовується для задач класифікації;
- k-найближчих сусідів - це алгоритм, який використовує відстань між вхідними змінними для знаходження k найближчих сусідів та використовує їх для прийняття рішення про класифікацію. Використовується для задач класифікації;
- метод опорних векторів - це алгоритм, який намагається знайти оптимальну гіперп лощину між двома класами шляхом знаходження границі розділення

з максимальною шириною та використовує опорні вектори для класифікації нових прикладів. Використовується для задач бінарної класифікації;

– нейронні мережі - це алгоритми, які імітують роботу людського мозку та складаються з багатьох зв'язаних між собою штучних нейронів. Використовуються для різних задач, таких як класифікація, регресія, обробка природної мови та багато іншого;
– градієнтний спуск - це алгоритм, який використовується для оптимізації функції втрат, що використовується в багатьох алгоритмах машинного навчання. Використовується для знаходження найкращих параметрів моделі;
– метод головних компонент - це алгоритм, який зменшує кількість вхідних змінних, зберігаючи при цьому якомога більше інформації. Використовується для зменшення розмірності даних та поліпшення швидкості та точності алгоритмів машинного навчання;
– алгоритм кластеризації K-середніх - це алгоритм, який використовується для розділення вхідних даних на k кластерів на основі схожості між ними. Використовується для задач кластеризації;
– алгоритм допоміжних векторів - це алгоритм, який використовується для розв'язання задач класифікації з більш ніж двома класами. Він перетворює задачу багато класової класифікації на послідовні задачі бінарної класифікації.

1.1.3 Апаратне забезпечення для машинного навчання

Програму навчання моделі можна запусти практично на будь-якому апаратному забезпечені, від потужностей буде залежати швидкість навчання та об’єм даних які вона зможе обробити. Але для машинного навчання існують спеціалізовані апаратні схемі та процесори які ліпше справляються ніж звичайний CPU.

GPU. Найпопулярніший і широко використовуваний процесор який використовують для навчання ШІ у мобільних пристроях будь-то телефони чи ноутбуки. Враховуючи, що більшість цих обчислень включають матричні та векторні операції, інженери та вчені все частіше вивчають використання графічних процесорів для неграфічних обчислень; вони особливо підходять для вирішення інших бентежно паралельних завдань. Такі як навчання нейронних мереж. У дослідженні, проведеному Indigo, було виявлено, що під час навчання нейронних мереж глибокого навчання GPU може бути у 250 разів швидше, ніж CPU.

Tensor Processing Unit (TPU) – спеціалізована інтегральна схема (ASIC) прискорювача штучного інтелекту, розроблена Google для машинного навчання нейронних мереж з використанням власного програмного забезпечення Google TensorFlow. Яскравим прикладом є процесор від Google – Tensor. Google Tensor SoC продемонстрував «надзвичайно великі переваги у продуктивності порівняно з конкурентами» у тестах, орієнтованих на машинне навчання; хоча миттєве енергоспоживання також було відносно високим, покращена продуктивність означала, що споживалося менше енергії через коротші періоди, що вимагають пікової продуктивності.

У порівнянні з графічним процесором, TPU призначені для великих обсягів обчислень з низькою точністю (наприклад, з точністю до 8 біт) з великою кількістю операцій введення/виведення на джоуль, без обладнання для розтеризації/накладання текстур. За словами Нормана Джуппі, мікросхеми TPU ASIC монтуються в блоці радіатора, який може поміститися в слот для жорсткого диска в стійці центру обробки даних. Різні типи процесорів підходять до різних типів моделей машинного навчання. TPU добре підходять для CNN, у той час як GPU мають переваги для деяких повнозв'язних нейронних мереж, а CPU можуть мати переваги для RNN.

Блок обробки зображень (VPU) - це клас мікропроцесорів, що розвивається; це особливий тип прискорювача ШІ, призначений для прискорення задач машинного зору.

Блоки обробки зображень відрізняються від блоків обробки відео (які спеціалізуються на кодуванні та декодуванні відео) своєю придатністю для запуску алгоритмів машинного зору, таких як CNN (згорткові нейронні мережі), SIFT (масштабно-інваріантне перетворення ознак) та їм подібних. Вони можуть включати прямі інтерфейси для отримання даних з камер (в обхід будь-яких позачіпових буферів) і приділяти більше уваги внутрішньо-чіповому потоку даних між безліччю паралельних виконавчих блоків з блокнотною пам'яттю, наприклад багатоядерний DSP. Але, як і блоки обробки відео, вони можуть бути орієнтовані на арифметичні низько-точні операції з фіксованою точкою для обробки зображень . Найвідоміші приклади – Movidius Myriad розробки Intel або Pixel Visual Core від Google.

Вони відрізняються від графічних процесорів, які містять спеціалізоване обладнання для розтеризації та відображення текстур (для 3D-графіки) і чия архітектура пам'яті оптимізована для маніпулювання растровими зображеннями зовнішньої пам'яті (читання текстур і зміна буферів кадрів з довільним доступом). VPU оптимізовано для продуктивності на ват, а GPU в основному орієнтовані на абсолютну продуктивність. Цільовими ринками є робототехніка, Інтернет речей, нові класи цифрових камер для віртуальної та доповненої реальності, інтелектуальні камери та інтеграція прискорення машинного зору до смартфонів та інших мобільних пристроїв.

1.1.4 Інструменти для розробки та навчання нейронних мереж

Для цієї роботи було обрано мову програмування Python. Python є загально призначеною мовою програмування, яка використовується в різних сферах. Зазвичай називається "мовою скриптів", після чого вона підтримує підтримку об'єктно-орієнтованого програмування з метою на скрипті. Коли

говорять про код Python, часто вживають слово "скрипт" замість "програма". Python вибирають для написання складних програм, зокрема в області аналізу даних і машинного навчання, через його ключові переваги.

Високо-рівневість: Python орієнтований на зручність читання, послідовність і загальну якість коду, що відрізняє його від інших мов програмування для скриптів. Однорідність коду Python робить його більш зрозумілим, навіть якщо ви не писали його самого. Крім того, Python підтримує об'єктно-орієнтоване програмування.

Підвищена продуктивність розробника: Порівняно з компільованими мовами, такими як C, C++ та Java, Python дозволяє розробникам працювати більш продуктивно. Код Python зараз коротший, ніж аналогічний код на C++ або Java, що спрощує розробку та підтримку. Також програми Python запускаються миттєво, що сприяє прискоренню роботи розробників.

Портативність: Більшість програм, написаних на Python, без проблем працюють на різних комп'ютерних платформах. Наприклад, код Python можна легко переносити між системами Linux і Windows.

Багатий вибір бібліотек: Python має велику стандартну бібліотеку, яка включає багато корисних функцій і можливостей, від обробки текстових даних до роботи з мережами. Ви також можете розширювати Python за допомогою власних бібліотек або використовувати сторонні бібліотеки для вирішення конкретних задач, таких як аналіз даних (наприклад, бібліотеки NumPy і Keras).

Інтеграція: Python легко інтегрується з іншими компонентами програми, може взаємодіяти з кодом на C, C++, Java та іншими мовами. Це робить відмінним інструментом для налаштування та його розширення програмного забезпечення.

Для реалізації моделей нейронних мереж та машинного навчання використовується популярна бібліотека Tensorflow з фреймворком Keras, котрий є найшвидшим фреймворком для даних задач та має велику кількість

документації, що прискорює написання коду (також призначеним для завантаження набору даних і попередньо побудованої моделі VGG16), або PyTorch створений на базі Torch навколо цього фреймворку вибудовано екосистему, що складається з різних бібліотек, що розробляються сторонніми командами: PyTorch Lightning і Fast.ai [10], які спрощують процес навчання моделей.

TensorFlow - це програмна бібліотека з відкритим вихідним кодом для чисельних розрахунків з використанням графів потоків даних [29]. TensorFlow був створений і до сих пір підтримується командою Google Brain в дослідницькій організації Google Machine Intelligence для машинного і глибокого навчання. В даний час він випущений під ліцензією Apache 2.0 з відкритим вихідним кодом.

PyTorch – бібліотека глибокого навчання, що розвивається під крилом Facebook. Вона не схожа на інші популярні бібліотеки, такі як Caffe, Theano та TensorFlow. PyTorch є аналогом фреймворку Torch7 для мови Python [4].

Keras - це бібліотека-оболонка Python, яка надає доступ до інших інструментів глибокого навчання, таким як TensorFlow, CNTK, Theano і т.д. Вона була розроблена з метою забезпечення швидкого маніпулювання інструментами глибокого навчання і випущена під ліцензією MIT. Keras поставляється як пакет для Python версій 2.7 - 3.9 і може безперешкодно працювати на GPU і CPU з урахуванням базових структур.

Keras є одним із найкращих API для нейронних мереж високого рівня. Кілька внутрішніх механізмів обчислень нейронних мереж підтримуються в коді, написаному на Python. Керас працює з Python, є модульним і простим у використанні. «Розроблений для людей, а не для машин» і «дотримувався найкращих практик зменшення когнітивного навантаження», API був створений.

Для створення нових моделей ви можете об’єднати різні модулі, такі як нейронні рівні, функції витрат, оптимізатори, схеми ініціалізації, функції

активації та схеми регуляризації. Просто додайте нові модулі, включаючи нові класи та функції. Використовуючи код Python, а не окремі файли конфігурації моделі, моделі створюються. Рисунок 1.1 показує структуру такої конфігурації.

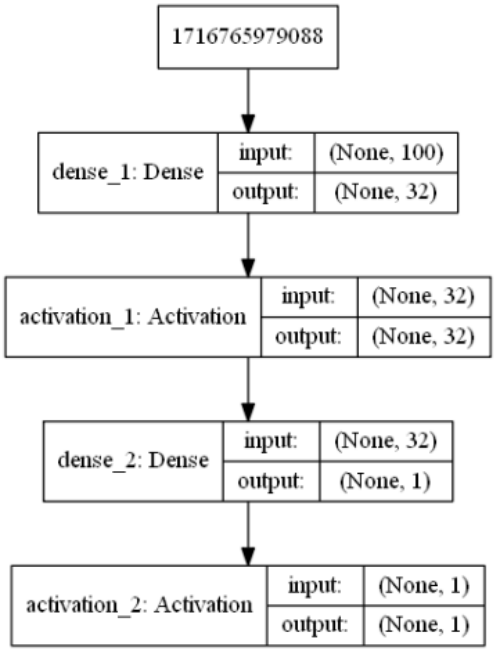

Рисунок 1.1 – Модель Keras

Для наявного набору даних з метою зниження навантаження на графічний процесор і скорочення часу навчання було взято всього 1024 випадкових примірника з навчальної вибірки і 256 примірника для тестового набору даних.

Архітектура однопрохідного знаходжувача (англ. – Single shot detector, SSD) дозволяє знаходити об’єкти будь-якого розміру в реальному часі зі швидкістю 59 фреймів за секунду (59 FPS), що робить модель ідеальною для вбудованих систем та мобільних пристроїв.

SSD використовує карти особливостей різного розміру, щоб знаходити об’єкти різного розміру. Завдяки застосуванню фільтрів невеликої розмірності до обмежувальних прямокутників за замовчуванням, модель генерує оцінку для кожного класу і редагує параметри обмежувальних прямокутників, щоб точніше окреслити об’єкт. Ці принципи дозволяють моделі досягати великої точності на зображеннях малої розмірності. Збільшення розмірності вхідного зображення, кількості ЗНМ рівнів та кількості обмежувальних прямокутників за замовчуванням покращить якість моделі, але водночас збільшить кількість ресурсів, необхідних для виведення результату.

1.1.5 Хмарні обчислювання

Хмарні сервіси [8] забезпечують легкий масштабований доступ до додатків, ресурсів і послуг, які повністю керуються провайдером хмарних сервісів. Властивість динамічного масштабування займає ключове значення для задоволення потреб користувачів і замовників [11, 12], яке так само дозволяє істотно скоротити витрати на розгортання власних ресурсів і виділення персоналу для управління службою.

Хмарні сервіси поділяються на три основні категорії:

– програмне забезпечення як послуга (SaaS) По моделі SaaS постачається апаратна інфраструктура і ПЗ, також розробник забезпечує взаємодію з користувачем через інтерфейсний портал. SaaS на даний момент є досить широким ринком. За SaaS можуть надаватись самі різноманітні послуги, від веб-пошти, до управління запасами, обробки БД. Перевагою такої моделі є те, що кінцевий користувач може вільно користуватись послугою з будь-якої точки світу;

– платформа-як-сервіс (PaaS) PaaS в хмарі визначається як набір програмних продуктів та засобів розробки, що розміщені на інфраструктурі провайдера. Розробники можуть створювати програми на платформі провайдера через Інтернет. PaaS провайдери можуть використовувати API,

сайт-портали, шлюзи, або програмне забезпечення установлене на комп'ютері клієнта;

– інфраструктура як послуга (IaaS) IaaS являє собою віртуальний сервер instanceAPI для запуску, зупинки, доступу, налаштування своїх віртуальних серверів та систем збереження. IaaS дозволяє компанії платити саме за стільки потужності, скільки їй необхідно. Дану модель іноді називають "комунальні обчислення";

А також на три основні моделі розгортання:

– приватна хмара (англ. private cloud) - це хмарна інфраструктура, яка призначена для використання виключно однією організацією, що включає декілька користувачів (наприклад, підрозділів). Приватна хмара може перебувати у власності, керуванні та експлуатації як самої організації, так і третьої сторони (чи деякої їх комбінації);

– публічна хмара (англ. public cloud) - це хмарна інфраструктура, яка призначена для вільного використання широким загалом. Публічна хмара може перебувати у власності, керуванні та експлуатації комерційних, академічних (освітніх та наукових) або державних організацій (чи будь-якої їх комбінації);

– гібридна хмара (англ. hybrid cloud) - це хмарна інфраструктура, що складається з двох або більше різних хмарних інфраструктур (приватних, громадських або публічних), які залишаються унікальними сутностями, але з’єднанні між собою стандартизованими або приватними технологіями, що уможливлюють переносимість даних та прикладних програм.

У нашій роботі розглядаються хмарні обчислення або функція як послуга (FaaS), так як вона була розроблена для короткочасних завдань [8]. Ці завдання були б періодичними, або керується подією, щоб мати можливість ініціювати код. Застосування хмарних обчислень є серверна частина веб-додатку [8]. Веб-запити керуються подіями та нетривалі і ідеально підходить для такого випадку використання. Завдяки хмарним обчисленням один може

інтегрувати інший хмарні служби та забезпечують надійну програму для користувачів. Однак перевага марного обчислення полягає в тому, що воно приховує керування сервером від користувача та дозволяє щоб зосередитися на поставленому завданні, як PaaS. Ця перевага корисна для розробників, які можуть зосередитися на програмі, не турбуючись про сервер управління [20]. Перенесення цієї керованої подіями властивості хмарних обчислень до навчання моделі машинного навчання є складним завданням. Але хмарні обчислення викликає проблеми з часом і пам'яттю, які потрібно вирішити [7].

Зараз на ринку хмарних технологій існують такі основні провайдери:

- Amazon Web Services (AWS) - це провідний постачальник хмарних послуг, який надає дуже широкий спектр хмарних сервісів, включаючи IaaS, PaaS та SaaS. AWS є найбільшим постачальником хмарної інфраструктури та контролює понад 30% ринку хмарних технологій;

- Microsoft Azure – популярна платформа хмарних технологій, яка надає послуги IaaS, PaaS та SaaS. Azure є провідним постачальником хмарних послуг для підприємств та великих корпорацій в основному для продуктів від Microsoft, з більш ніж 20% долею ринку;

- Google Cloud Platform (GCP) – це хмарна платформа, яка надає послуги IaaS, PaaS та SaaS. GCP пропонує багато інноваційних технологій, таких як машинне навчання та штучний інтелект, які забезпечують конкурентні переваги у галузі нейронних мереж;

- IBM Cloud – це платформа хмарних технологій, яка надає послуги IaaS, PaaS та SaaS, а також додаткові сервіси для розробки та керування додатками у хмарі. IBM Cloud в основному орієнтована на галузь блокчейну та інтернету мов (IoT);

- Oracle Cloud – це платформа хмарних технологій, яка надає послуги IaaS, PaaS та SaaS для корпоративних клієнтів. Особливості Oracle Cloud включають засоби для розробки та управління базами даних, аналітики даних та інтеграції з іншими технологіями.

1.1.6 Огляд схожих рішень

Якщо просто розгорнути модель для навчання на облачному сховищу, то таке виконання скрипта зіткнеться з деякими труднощами. По-перше, оскільки базове обладнання та програмне забезпечення є відповідальністю постачальника хмарних послуг, час виконання коду обмежено кількома хвилинами. Це додає складності для навчання, що перевищує цей ліміт часу. По-друге, обсяг пам’яті, який надає ця послуга, обмежений мегабайтами (МБ). Це ускладнює використання таких бібліотек, як PyTorch і TensorFlow, оскільки вони займають простір, який можна використовувати для інших цілей. Ще одна проблема, яку створює обмежене сховище, полягає в тому, що розмір навченої моделі може перевищувати ліміт пам’яті. Як вирішення – загружати на сервер уже навчену модель.

Але на стан 2023 року практично усі крупні провайдери хмарних технологій мають свої продукти по розгортанню моделей машинного навчання на своїх платформах. Наприклад на AWS таких більше десяти різноманітних продуктів. Розглянемо недоліки декількох з них:

Google Cloud. Високі витрати: Google Cloud пропонує різні тарифні плани, вартість використання сервісів машинного навчання може бути дуже високою, що робить його менш доступним для маленьких компаній та індивідуальних користувачів.

Наявність додаткових інструментів: Google Cloud має базовий набір інструментів для машинного навчання. Для побудови складних моделей, що потребують більшої кількості спеціалізованих інструментів, можна знадобитися додаткове програмне забезпечення.

Amazon SageMaker: обмежена підтримка: Amazon SageMaker підтримує обмежену кількість алгоритмів та бібліотек машинного навчання. Це може обмежити можливості для розгортання складних проектів.

Висока вартість: підписка на Amazon SageMaker може бути високою, залежно від того, як часто ви використовуєте сервіс. Для маленьких компаній та індивідуальних користувачів це може бути недосяжним.

AWS Deep Learning Containers. Налаштування складності: для того, щоб користувачі могли використовувати AWS Deep Learning Containers, вони повинні мати досвід роботи з Docker та налаштуваннями серверів. Це може бути складним для початківців або для користувачів без технічної освіти.

Витрати на ресурси: AWS Deep Learning Containers можуть вимагати значних витрат на обробку даних та пам’ять. Якщо користувачі працюють з великими об'ємами даних, вони можуть стикнутися з проблемами виконання завдань у розумний час.

Підтримка: хоча AWS надає документацію та інструкції для користувачів, можна отримати постійну підтримку у разі виникнення проблеми. Користувачі можуть звернутися до форумів або спільнот для отримання допомоги, але це може бути неефективним у випадку серйозних проблем.

Обмежені можливості: AWS Deep Learning Containers не надає повних можливостей для машинного навчання, порівняно з іншими сервісами AWS, такими як Amazon SageMaker. Наприклад, немає можливості використовувати готові моделі або побудовані бібліотеки для машинного навчання.

Отже головні проблеми у популярних провайдерів це – висока вартість послуги, та обмежений функціонал, яких дозволяє користуватися лише тими алгоритмами та моделями які надають самі провайдери. Це стосується також і інших сервісів AWS, Google Cloud та Azure.

1.2 Постановка задачі

Отже задача цієї роботи є розробка оптимального рішення для навчання моделі машинного навчання за допомогою хмарних обчислень. Для створення

моделі, використовувати AWS Lambda – безсерверна обчислювальна служба, яку пропонує Amazon Web Service. Обчислювальна потужність AWS Lambda збільшується зі збільшенням її конфігурація пам'яті. Отже, хмарна функція більшої пам’яті може отримати вигоду від вищої обчислювальної потужності в його розпорядженні. Також обрати систему зберігання екземплярів у AWS, роздивитися такі як EFS, S3 та Lambda.

2 РОЗРОБКА МЕТОДОЛОГІЇ

Успішне навчання моделі машинного навчання за допомогою хмарних обчислень включатиме подолання проблем пам'яті та часових обмежень. Це основні проблеми, з якими стикаються безсерверні обчислення, крім обчислювальної потужності. Для цього ми зосередимося на розв’язанні цього завдання на AWS

2.1 Зберігання

Проблема зберігання виникає через те, що AWS Lambda має лише 512 МБ непостійної пам’яті під час виконання в каталозі «/tmp». Будь-які дані, збережені під час виконання, будуть недоступні під час наступних виконання. У результаті стає неможливим динамічне завантаження таких бібліотек, як PyTorch або TensorFlow під час виконання.

2.1.1 Налаштування AWS Lambda

Одне з рішень – це створення образу Docker і завантаження його в AWS Lambda. Цей метод дозволяє нам завантажити необхідну бібліотеку та, якщо потрібно, набір даних у зображення. Це зменшує час і затримку в отриманні даних, оскільки дані доступні локально. Однак із таким підходом ми також маємо додати пакет розробки програмного забезпечення AWS (SDK), а також бібліотеку, яку будемо використовувати для навчання. Це також вимагало використання репозиторію, що належить AWS, для зображень контейнерів під назвою Elastic Container Registry (ECR) [32]. AWS ECR схожий на Docker Hub у тому сенсі, що він розміщує зображення контейнерів, створені користувачем.

Інший підхід, який ми можемо використати, це додавання шарів до AWS Lambda. Шари це zip-файли, які додаються до лямбда-функції. Шари можуть

містити бібліотеки в шарі. Потім вміст шарів доступний у каталозі «/opt», звідки ми можемо використовувати бібліотеки. Завдяки такому підходу ми можемо швидко розробити код і внести незначні зміни, не завантажуючи великі обсяги даних для невеликих змін.

2.1.2 AWS Simple Storage Service

Іншим підходом буде доступ до AWS Simple Storage Service, більш відомого як S3[18]. Це дозволяє зберігати об’єкти, до яких можна легко отримати доступ за допомогою AWS SDK. Пакети SDK AWS доступні в усіх середовищах виконання AWS Lambda. Ми можемо зберігати набір даних у сегменті S3, а потім отримувати його за потреби. Це дозволяє звільнити місце для набору даних. S3 також підтримує потокове передавання байтів, яке можна використовувати для зберігання моделі. Однак нам потрібно буде створити файл або об’єкт для кожного робочого елемента, а потім попросити додаткових робочих об’єднати ці файли. Це додає додаткові витрати на процес. S3 не дозволяє блокувати об’єкт, якщо один процес може оновити модель. Тому стає важко використовувати один і той же об’єкт для всіх працівників.

2.1.3 AWS Elastic File Storage

Замість зовнішнього сховища ми можемо використовувати послуги, які пропонує AWS, щоб вирішити проблему зберігання. Як зазначають Sindi.[33], ми можемо розширити сховище за допомогою Elastic File Storage (EFS) [16], яка є мережевою файловою системою, яку можна монтувати в AWS Lambda. EFS повністю керується AWS і є масштабованим, що означає, що він не працює без сервера та може збільшуватися та зменшуватися відповідно до потреб користувача. AWS Lambda використовує точку монтування в EFS, щоб дозволити монтувати файлову систему. Це дає Lambda необхідну додаткову пам’ять для навчання моделі. EFS також підтримує блокування файлів,

включаючи спільні та ексклюзивні блокування. Це дозволяє декільком працівникам оновлювати файл під час блокування, не турбуючись про узгодженість файлу. У результаті ми можемо оновлювати вміст моделі без накладних витрат на об’єднання кількох файлів в один.

2.2 Час

Іншим важливим фактором, що впливає на впровадження хмарних обчислень, є часові обмеження на виконання коду. Оскільки користувач не керує базовим сервером, він не має доступу до налаштування часу, протягом якого має виконуватися код. Це робиться з метою безпеки, щоб будь-який шкідливий код не міг довго працювати на серверах, якими керує постачальник хмарних послуг. Та з економічних причин, щоб не витрачати ресурси на підтримання завислих скриптів. Постачальник хмарних послуг, у цьому випадку AWS, дозволяє користувачеві налаштувати максимальний час виконання коду. Однак це обмежено максимум 15 хвилинами.

2.2.1 Послідовне виконання

Ми можемо використовувати хмарні екземпляри, щоб викликати один одного та передати поточний стан навчання. Викликаючи екземпляри передають параметри моделі як подію, а викликаний екземпляр отримує їх і продовжує навчання з цього моменту. Коли абонент успішно викликав іншу функцію, її можна завершити. Викликаний екземпляр тепер відповідає за продовження навчання. Так можна продовжувати до досягнення результату. Це означає, що ми будемо тренуватися серійно, тобто один екземпляр за іншим. У результаті нам знадобиться стільки ж часу, скільки більше, залежно від накладних витрат на послідовний виклик екземплярів. На рисунку 2.1 показано послідовне виконання лямбда-виразів для машинного навчання.

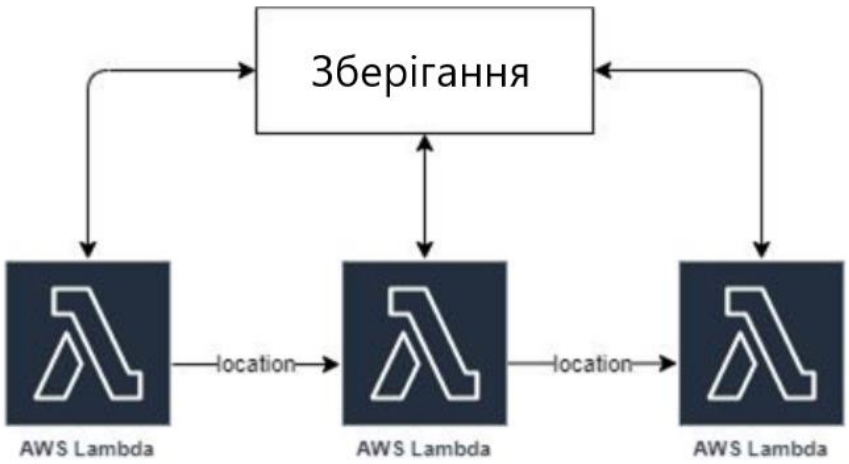

Рисунок 2.1 - Демонстрація методу послідовного виконання

2.2.2 Організований робочий процес

Системи оркестровки будують організовані робочі процеси на основі бізнес-логіки. Ці робочі процеси контролюють виконання послуг FaaS, які надає постачальник хмарних послуг. AWS має систему оркестровки під назвою AWS Step Functions.

AWS Step Functions дозволяє виконувати послідовне виконання, а також паралельне виконання AWS Lambda на основі певних умов. Lopez [19] порівнює різні системи оркестровки для FaaS від відповідних постачальників хмарних послуг. Їхнє дослідження також виявило, що стан можна передавати між екземплярами розміром до 32 Кб. Моделі не можуть бути передані в таких коротких обмеженнях пам’яті. Однак ми можемо зберегти модель у S3 або EFS, як описано в розділах 2.1.2 і 2.1.3 відповідно. Потім ми можемо надіслати розташування моделі під час передачі стану. Цього можна досягти з обмеженням у 32 КБ. Тут слід зауважити, що плата за

використання функцій AWS Step може бути високою під час навчання великих моделей.

2.2.3 Методики навчання

Вибір відповідної методики навчання є запорукою швидшого отримання результату. Хмарні обчислення створюють проблеми для традиційних методів навчання. Традиційно навчання моделі включає всі дані та необхідні атрибути моделі, доступні для процесу навчання. У хмарних обчисленнях, заснованих на методі обчислень, коли обчислення переходять від одного екземпляра до іншого, ці атрибути потрібно відтворювати для продовження навчання. Це створює проблему для послідовного режиму виконання, описаного в розділі 4.2.1. Крім того, хмарні обчислювальні екземпляри не мають стану, тобто жодна інформація про поточне виконання не буде збережена під час наступного виконання. Використовуючи цю властивість, ми можемо зосередитися на розподіленому навчанні моделі.

2.2.4 Розподілене навчання

Навчання моделі передбачає ітерацію по набору даних і оновлення вагових коефіцієнтів моделі. Частота оновлення ваг залежить від алгоритму, який ми використовуємо для оновлення ваг. Градієнтний спуск є популярним вибором для таких проблем. Градієнтний спуск має три варіанти, а саме пакетний Градієнтний спуск (BGD), Стохастичний градієнтний спуск (SGD) і Мініпакетний градієнтний спуск (mini BGD).

BGD повторює весь набір даних, щоб оновити ваги моделі. Він розглядає всі точки даних, доступні в даному наборі даних, перш ніж коригувати ваги. Для великих наборів даних, що містять мільйони точок даних, одна ітерація займає багато часу. Цей процес потрібно повторювати знову і знову кожного разу для всіх точок даних, щоб досягти бажаного результату. Як наслідок, цей процес займає багато часу. Враховуючи тимчасові проблеми хмарних

обчислень, процес пакетного градієнтного спуску може вийти за межі допустимого часу виконання.

SGD забезпечує швидший спосіб оновлення ваг. На відміну від BGD, він розглядає кожну точку даних як цілий набір даних і оновлює вагу після кожної точки даних. Це забезпечує миттєвий зворотний зв'язок із розробником. Може здатися, що цей підхід найкращий, оскільки він забезпечує миттєвий зворотний зв'язок і налаштовує ваги на основі окремих точок даних, але це не так. Оскільки він розглядає всі точки даних однаково, він також враховує викиди. Викиди – це точки даних, які не відповідають загальній тенденції всього набору даних. Це дозволяє викидам спотворювати ваги моделі та може зашкодити процесу навчання. Хоча SGD дає миттєвий зворотний зв'язок, результат BGD кращий, ніж SGD. SGD також виявляється важко реалізувати в розподіленому середовищі. Кожне оновлення потребує блокування для оновлення параметрів, що призводить до накладних витрат. Виявлено, що процес оновлення параметрів за допомогою блокування уповільнює процес, оскільки процеси в кінцевому підсумку чекають блокування, та не виконують фактичні обчислення. Якщо екземпляр без сервера чекає на блокування, це може перевищити максимально допустимий час, і нам, можливо, доведеться повторити процес. Це додає додаткову роботу для обчислень і витрачає ресурси.

Mini BGD бере найкраще з обох алгоритмів і поєднує їх в один. Він випадковим чином розподіляє набір даних на менші групи. Можна зазначити, що кожна менша група буде представляти весь набір даних, якщо розділити випадковим чином. Кожна партія тепер оброблятиметься як цілий набір даних, а ваги оновлюватимуться після однієї партії, а не всього набору даних. Однак цей процес передбачає ітерацію по всьому набору даних. Цей процес допомагає зменшити вимоги до пам'яті AWS Lambda. Весь набір даних не обов'язково повинен бути в пам'яті. Лише менша партія, яка зараз використовується, може бути в оперативній пам'яті, тоді як інші можуть бути

в постійному сховищі. Це дозволяє хмарним екземплярам працювати лише з частиною даних і може бути виконано в межах допустимого часу. Однак вибір правильного розміру для міні-партій стає незначною проблемою.

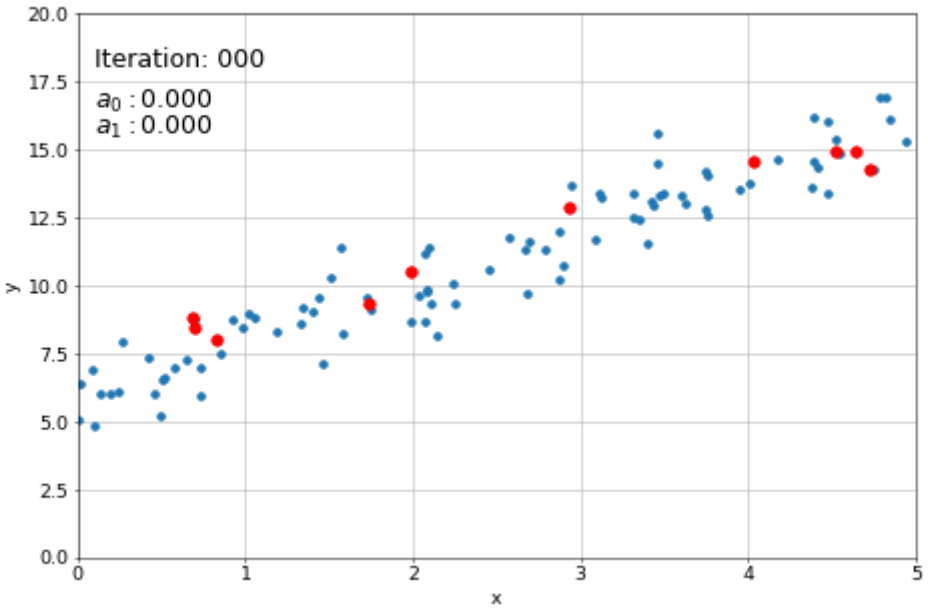

Рисунок 2.2 - Процес конвергенції міні-пакетного градієнтного спуску

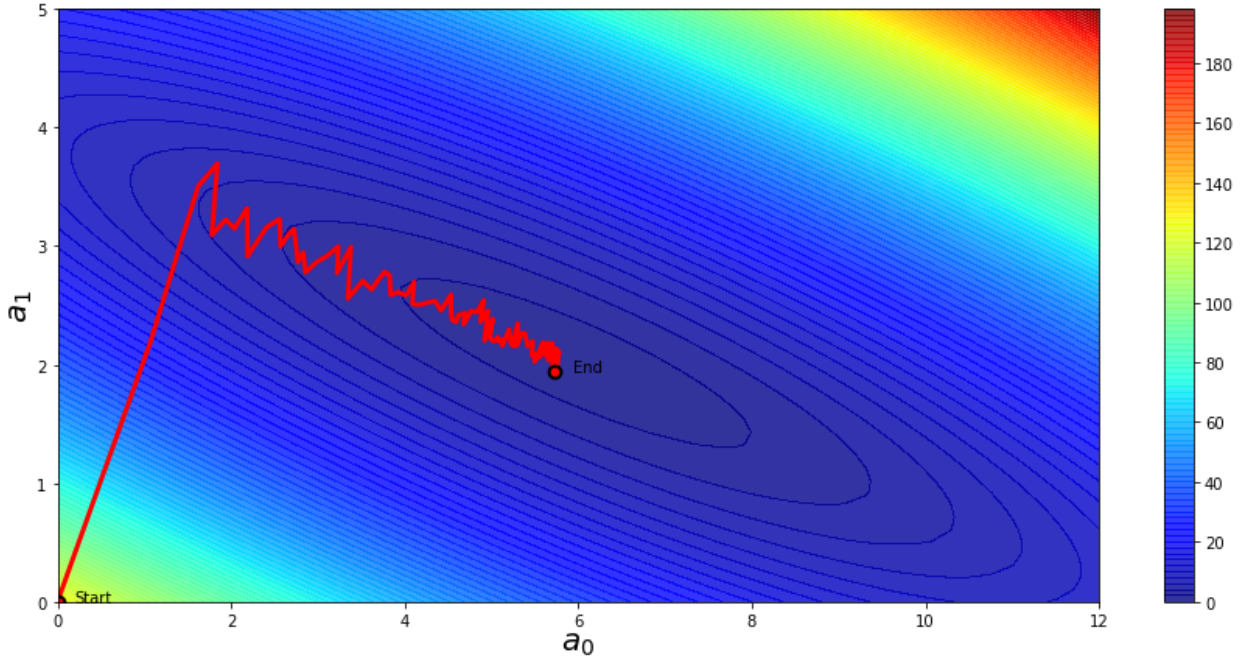

Рисунок 2.3 - Траєкторія міні-пакетного градієнтного спуску

Коефіцієнт збіжності цього алгоритму лежить десь між BGD і mBGD і становить

$$O(\frac{1}{\sqrt{bk}} + \frac{1}{k}),$$

де b – розмір партії.

2.2.5 Параметри сервера

Сервер відповідає за оновлення параметрів, коли нові параметри надходять від робочих вузлів, і розповсюдження параметрів, коли робочі вузли запитують їх. Кожен сервер підтримує головну копію параметрів, за які він відповідає, а також дублікати параметрів з інших серверів для відмовостійкості. У контексті хмарних обчислень ми можемо мати екземпляр хмарного обчислення, який виконує роль сервера параметрів і оновлює та розподіляє значення оновлених ваг. Однак такий підхід означає, що ми завжди матимемо хмарний екземпляр, запущений для цієї мети. Натомість ми можемо розподілити відповідальність за оновлення параметрів на окремі екземпляри без сервера. Екземплярам не потрібно поширювати вагові коефіцієнти, оскільки все оновлення відбуватимуться на центральній копії вагових коефіцієнтів, що можна виконати за допомогою блокування. Екземпляри зчитують параметри, а потім працюватимуть над оновленням параметра та оновлять параметри після завершення обчислення. Поки він виконує обчислення, зніме блокування та дозволить іншим програмам зчитати або оновити параметри.

Всупереч загальній інтуїції заборони оновлень під час роботи над поточними даними, які зараз обробляються, безпечно дозволити оновлення вагових коефіцієнтів моделі в машинному навчанні. Niu пропонують нову ідею розпаралелювання SGD без блокування під час оновлення ваг. Їхні дослідження показують, що більшість оновлень ваги моделі є рідкісними.

Оновлення змінює лише набір параметрів. У результаті їхній алгоритм досягає таких результатів, як серійна версія виконання.

2.3 Архітектура

Переглядаючи різні підходи, згадані в розділі 4, ми можемо побачити, що різні варіанти добре підходять для певних завдань. Поєднуючи необхідні властивості кожного підходу, які відповідають нашим завданням, ми можемо досягти системи, яка добре підходить для досягнення бажаного результату. Щоб вирішити проблему зберігання, ми можемо створити систему з комбінацією EFS і Lambda. Ми виключаємо використання S3, оскільки це додає додаткові накладні витрати на керування кількома працівниками. Крім того, S3 використовує механізм багаторазового запису (WORM), що означає, що дані можна записати лише один раз. Для внесення змін до файлу необхідно перезаписати весь файл. Як наслідок, для вирішення проблеми сховища ми будемо використовувати комбінацію налаштування лямбда та додавання сховища за допомогою EFS. Ми зберігатимемо набір даних, бібліотеки та модель у EFS.

Щоб подолати часові обмеження, ми можемо використовувати комбінацію серійних і розподілених робітників. Спочатку ми починаємо з набору робітників, які працюють паралельно розподіленим способом. Кожен з них працюватиме над власним набором даних. Після завершення процесу навчання працівники зберігатимуть градієнти в каталозі в EFS. Потім ці градієнти будуть підібрані лямбда-функцією, яка об’єднує градієнти та оновлює вагу. Після оновлення ваги об’єднана лямбда знову викликає робочих, і процес продовжується. Кількість робітників, які працюють паралельно, визначатиметься значеннями параметрів, які ми передаємо на початку навчання. Для методології навчання ми будемо використовувати підхід розподіленого навчання з використанням мініпакетного градієнтного спуску (mini BGD). Ми розподілимо набір даних серед робочих вузлів, і кожен

вузол оброблятиме та оновлюватиме вагу моделі. Ми плануємо включити частину сервера параметрів у кожен робочий вузол, таким чином усуваючи потребу в автономному сервері. Це зменшує мережеві накладні витрати на передачу та отримання параметрів і може бути безпосередньо отримано з підключеного сховища.

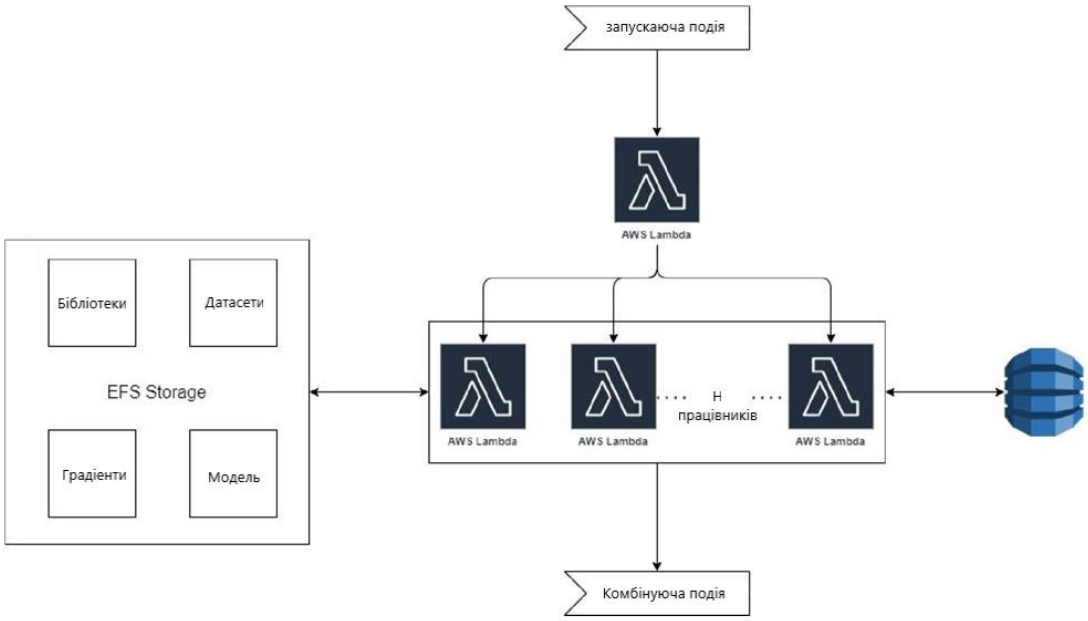

Рисунок 2.4 – Архітектура запуску навчального процесу

На рисунку 2.4 і 2.5 показана архітектура запропонованої системи. На рисунку 2.4 зосереджено увагу на архітектурі для ініціації процесу навчання. На рисунку 2.5 зосереджено увагу на архітектурі для поєднання фази та продовження процесу навчання. Архітектура складається з AWS Elastic File System, Lambda та DynamoDB. EFS зберігає бібліотеки, які використовуються для машинного навчання, набір даних, необхідний для навчання, градієнти та модель під час навчання. Весь код для лямбда записується в одній лямбда функції. Та сама лямбда-функція повторно використовується як сервер параметрів для обчислення градієнта та об’єднання градієнтів після

завершення кроку обчислення. Кожна лямбда-функція оновлює таблицю DynamoDB, щоб надати оновлення користувачеві. Користувачі можуть відстежувати поточний стан тренувального процесу за допомогою даних, вставлених у DynamoDB.

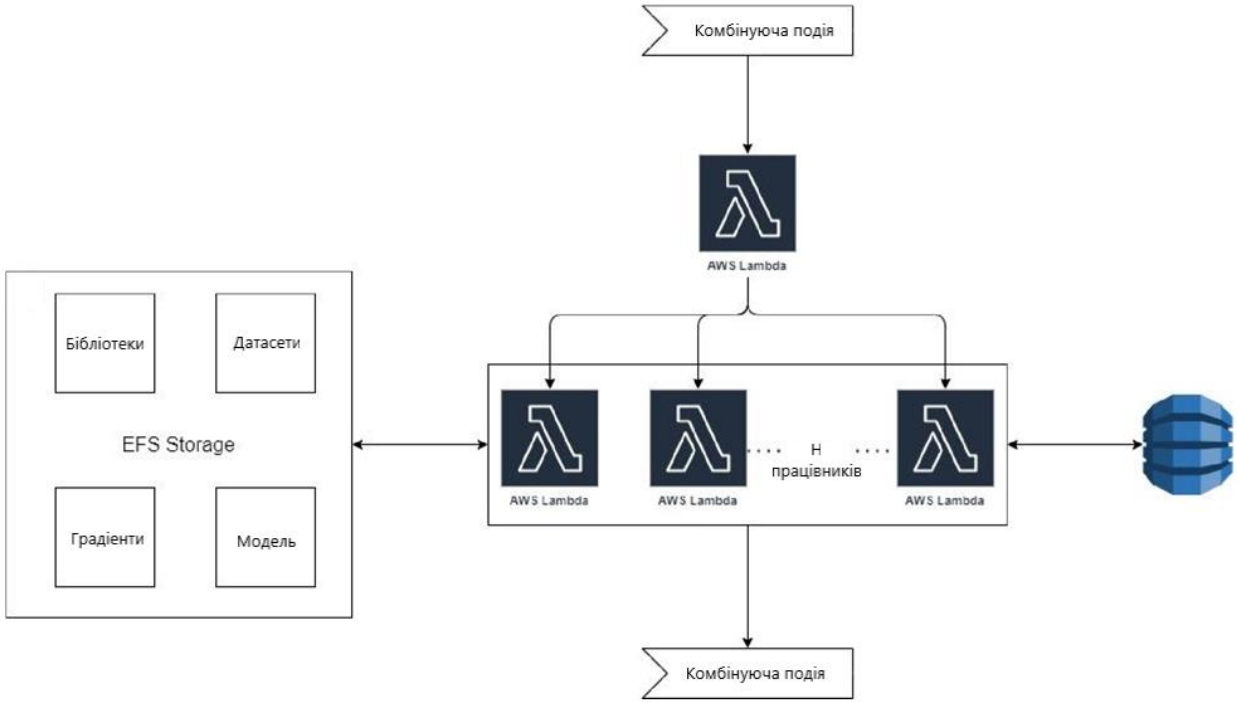

Рисунок 2.5 – Архітектура для комбінування градієнтів

2.3.1 Навчання

Щоб почати процес навчання, нам потрібно вперше викликати AWS Lambda. Функція приймає подію для запуску, і ми можемо надати цю подію вручну через консоль. Ця подія є об’єктом JSON і має містити ключ «start», щоб почати процес навчання. Крім того, він також повинен містити кількість працівників, яких необхідно залучити для навчання.

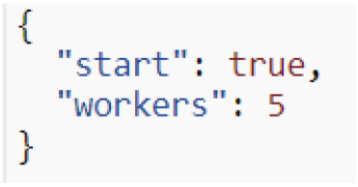

Рисунок 2.6 - Початкова подія виклику

На рисунку 2.6 показано подію виклику для процесу навчання. Як тільки лямбда отримує подію, вона викликає потрібну кількість робітників і призначає кожному з них ідентифікатор робітника. Ідентифікатори працівників призначаються від 0 до N-1, де N - кількість запитаних працівників. Щоб запустити працівника, ми створюємо подію в самій початковій лямбді. Лямбда-випромінювачі робітників вимагають ідентифікації працівників разом із загальною кількістю працівників у процесі навчання. Ми також передаємо інформацію про кількість виконаних епох.

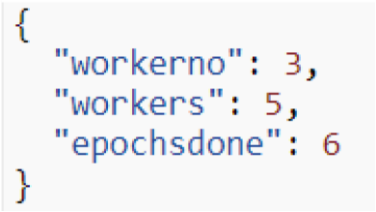

Рисунок 2.7 - Подія виклику працівника

На рисунку 2.7 показано подію виклику для робочого. Працююча лямбда отримує ці подію та починається із завантаження даних, збережених у EFS. Спочатку вона завантажує вагові коефіцієнти попередньо навченої моделі, а потім починає виконувати навчання на наборі даних. Він зберігає градієнти в

EFS для кожного кроку. Перед завершенням виконання, лямбда-функція об’єднує всі свої градієнти, щоб зменшити робоче навантаження на об’єднану лямбда. Після завершення навчання лише одна лямбда-функція починає фазу об’єднання. Кожен робочий файл перевіряє кількість файлів градієнта в EFS. Якщо вони дорівнюють кількості працівників, то це створює подію для фази об’єднання.

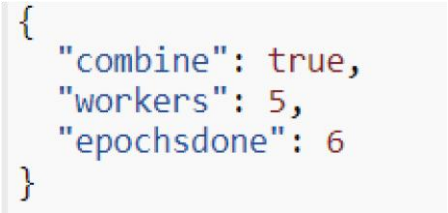

Рисунок 2.8 - Подія виклику фази комбінування

На рисунку 2.8 показано подію для початку фази комбінування. Фаза події містить ключ «комбінувати», щоб дати лямбда-функції знати, що вона знаходиться у фазі комбінування. На цьому етапі вона підсумовує всі градієнти, створені робочими лямбда-сигналами, а потім продовжує оновлювати вагу на основі формули. Фаза комбінування також визначає, чи потрібно продовжувати навчання чи його можна зупинити.

2.4 Налаштування

Щоб налаштувати всю архітектуру, ми використовуємо CloudFormation, який дозволяє нам створювати ресурси в AWS за допомогою шаблонів. Користувачі можуть використовувати шаблони для розгортання тієї самої інфраструктури в кількох облікових записах, гарантуючи, що архітектура

залишається незмінною. За допомогою шаблону ми створюємо ресурси, встановлюємо бібліотеки та попередньо обробляємо набір даних.

2.4.1 Ресурси

Нам потрібно створити ресурси до AWS Lambda та AWS EFS, щоб мати можливість проводити експерименти. Ці ресурси надає AWS. У нас також є ролі Identity and Access Management (IAM), які дозволяють лише вибраним об’єктам отримувати доступ до файлової системи. Ресурси, необхідні для цього експерименту:

a) віртуальна приватна хмара (VPC) VPC - це логічна межа для ізоляції ресурсів в обліковому записі AWS. VPC діє як приватна мережа для взаємодії ресурсів. Кожен користувач може мати свій VPC та ресурси всередині себе, щоб дозволити йому працювати незалежно або працювати в тому самому VPC;
b) підмережа. Підмережі - це віртуальні підмережі в межах VPC. Це менші логічні розділи VPC, які використовуються ресурсами для взаємодії. Нам знадобиться 2 типи підмереж:
    1) публічна підмережа. Ресурси в цих підмережах мають доступ до Інтернету;
    2) приватна підмережа. Ці підмережі не мають доступу до Інтернету. Доступ до Інтернету можна надати шляхом маршрутизації запитів через публічну підмережу. В першу чергу EFS і Lambda будуть розгорнуті в цій підмережі з метою безпеки;
c) шлюз NAT. Шлюз трансляції мережевих адрес (NAT) використовується для надання доступу до Інтернету до приватної підмережі. Він пересилає запит до місця призначення, маскуючи або перекладаючи IP-адресу ресурсу, який його запитує;

d) інтернет-шлюз. Internet Gateway забезпечує доступ до всього VPC. Це забезпечує доступ до загальнодоступних підмереж і шлюзу NAT для підключення до Інтернету;
e) EFS. Файлова система, яка використовуватиметься для зберігання набору даних, бібліотек і моделі. Він також зберігатиме градієнти під час навчання;
f) цільові групи та групи безпеки. Цільові групи— це логічні точки модифікування EFS, які дозволяють іншим ресурсам модифікувати файлову систему. Групи безпеки діють як брандмауери та дозволяють джерелам з білого списку, одночасно забороняючи будь-який інший трафік;
g) таблиця DynamoDB. Таблиця DynamoDB використовується для моніторингу ходу поточного навчального процесу. Ми можемо зберігати інформацію про епоху, а також значення втрат;
h) Lambda та IAM ролі. Лямбда використовується для навчання моделі. Будь яка лямбда, яка намагається отримати доступ до файлової системи, потребує дозволу на це. Роль IAM має дозволи, які дозволяють лямбда отримати доступ до файлової системи. Щоб забезпечити безпеку архітектури, ми дозволяємо лише Lambda взяти на себе роль, і жодна інша служба не може використовувати її;

2.4.2 Бібліотеки

Під час налаштування ресурсів ми також встановлюємо бібліотеки в EFS для доступу лямбда-виробників. Ми встановлюємо бібліотеки в EFS, що дозволяє нам вийти за межі 512 МБ Lambda. Ми можемо встановити такі популярні бібліотеки машинного навчання, як PyTorch і Tensorflow. Ці бібліотеки можуть використовуватися кількома працівниками через ту саму EFS, що зменшує конфлікти залежностей. Усі працівники використовують

однакову версію та єдине середовище розробки. Бібліотеки, встановлені для цього проекту, згадані нижче:

– Keras (v 2.4.3);
– Matplotlib (v 3.4.0);
– NumPy (v 1.19.3);
– Open CV (v 4.5.1);
– Pandas (v 1.2.3);
– Pickle (v 0.0.11);
– SciPy (v 1.6.2);
– TensorFlow (v 2.4.1);

2.4.3 Набір даних

Для цього експерименту ми використовуємо набір даних CIFAR10 [34]. Набір даних містить 60 000 зображень, розділених на 10 різних класів. Кожен клас має 6000 зображень. Набір даних розділено на 50 000 зображень для навчання та 10 000 зображень для тестування.

Набір даних поділено на п'ять навчальних пакетів та один тестовий пакет, кожен з яких містить 10 000 зображень. Тестова партія містить рівно 1000 випадково вибраних зображень кожного класу. Навчальні пакети містять зображення у випадковому порядку, але деякі навчальні пакети можуть містити більше зображень з одного класу, ніж з іншого. Загалом навчальні пакети містять 5000 зображень з кожного класу. Кожен із цих файлів є об’єктом Python (лістинг 2.1).

Лістинг 2.1 – Налаштування набору у Python3

```
def unpickle(file):
import pickle
with open(file, 'rb') as fo:
dict = pickle.load(fo, encoding='bytes')
return dict
```

Завантажений таким чином кожен пакетний файл містить словник із такими елементами:

- дані – 10000x3072 масив numpy uint8s. У кожному рядку масиву зберігається кольорове зображення розміром 32x32. Перші 1024 записи містять червоні значення каналу, наступні 1024 зелені, а останні 1024 сині;

- мітки – список із 10000 чисел у діапазоні 0-9. Число в індексі і вказує на мітку і-го зображення в масиві даних.

Набір даних містить інший файл під назвою batches.meta. Він також містить об’єкт словника Python. У ньому є такі записи: label_names – 10-елементний список, який дає значущі назви числовим міткам у описаному вище масиві міток.

Рисунок 2.9 – Попередня обробка набору даних

На рисунку 2.9 показано попередню обробку набору даних. Набір даних завантажується та розпаковується у форматі tar безпосередньо до EFS.

Витягнуті дані розділені на 6 файлів: 5 файлів для тестових даних і 1 файл для перевірки. Ці файли зберігаються у форматі .pickle і потребують бібліотеки .pickle для читання даних. Щоб заощадити час під час кожного лямбда-виконання, ми зчитуємо всі дані під час процесу налаштування та попередньо обробляємо дані. Попередня обробка включає зміну розміру зображень відповідно до вхідних даних моделі, а також швидке кодування вихідних міток. Зображення для навчання і тестові зображення потім перетворюються на масиви NumPy для більш ефективного зберігання. Потім ми зберігаємо дані в чотирьох окремих файлах. По одному файлу для навчальних даних, навчальних міток, тестових даних і тестових міток. Ці файли можна читати швидше та економити час, оскільки лямбда має щоразу читати менше файлів.

Подібний підхід використовує Sony Playstation, вона використовує різні методи для архівації та розархівації файлів-присетів у ігорах під час їх виконання на консолі. Перед тим, як ігрова графіка, звуки, тексти та інші ресурси будуть завантажені в пам'ять консолі, вони можуть бути затиснуті. Це зменшує обсяг даних, які потрібно завантажити з носія, таким чином зменшуючи час завантаження і ресурси, витрачені на зберігання. Якщо ігра потребує доступу до певного ресурсу, такого як текстура чи звук, консоль може розпаковувати цей ресурс з архіву в реальному часі. Це означає, що дані розпаковуються на «лету», коли їх достатньо для відтворення гри. Консоль також може використовувати кешування для збереження розпакованих ресурсів у пам'яті на деякий час. Це дозволяє швидше отримати доступ до ресурсів, які були розпаковані раніше, замість розпакування їх знову в кожному разі.

Таблиця 2.1 показує час, необхідний для читання файлів із даних без архівування та файлів NumPy.

Таблиця 2.1 - Час затримки читання файлів

	Не заархівовані дані(у секундах)	NUMPY дані (у секундах)
1	4.07	2.04
2	3.43	1.76
3	4.80	2.17
4	4.23	2.07
5	3.23	1.61
6	4.18	2.01
7	5.09	2.29
8	4.64	2.16
9	3.97	2.03

3 ЕКСПЕРИМЕНТ

3.1 Вимоги

3.1.1 Вимоги до обладнання:

- машина 1: AWS Lambda (RAM: 1 ГБ – 10 ГБ; з кроком 1 ГБ);
- машина 2: AWS EC2: t2.xlarge (4 vCPU, 16 GB RAM);
- машина 3: AWS EC2: p4d.24xlarge (96 vCPU, 8 GPU, 320 GB RAM).

Інстанси Amazon EC2 T2 – це інстанси з продуктивністю, що підвищується, які забезпечують базовий рівень продуктивності ЦПУ з можливістю його підвищення.

Безлімітні інстанси T2 можуть підтримувати високу продуктивність ЦПУ до того часу, поки цього вимагає робоче навантаження. Більшість робочих навантажень загального призначення безлімітні інстанси T2 забезпечують достатню продуктивність без додаткової плати.

Базовий рівень продуктивності та можливість його перевищення визначаються кредитами ЦПУ. Інстанси T2 регулярно одержують певну кількість кредитів, яка залежить від розміру інстансу. Вони накопичують кредити ЦПУ під час простою та споживають їх в активному стані. Інстанси T2 добре підходять для різних робочих навантажень загального призначення, включаючи мікросервіси, інтерактивні програми з низькою затримкою, малі та середні бази даних, віртуальні робочі столи, середовища розробки, збирання та тестування, репозиторії коду та прототипи продуктів.

Інстанси Amazon EC2 P4 забезпечують високу продуктивність для навчання машинного навчання та високопродуктивних обчислень у хмарі:

- процесори другого покоління Intel Xeon Scalable (Cascade Lake P-8275CL) із тактовою частотою 3,0 ГГц;

- до 8 графічних процесорів NVIDIA A100 Tensor Core;
- пропускна здатність інстансу 400 Гбіт/с за допомогою Elastic Fabric Adapter (EFA) та NVIDIA GPUDirect RDMA (віддалений прямий доступ до пам'яті);
- зв'язок між графічними процесорами зі швидкістю 600 Гбіт/с із використанням NVIDIA NVSwitch;
- розгортається в кластерах Amazon EC2 UltraCluster, кожен з яких містить понад 4000 графічних процесорів NVIDIA A100 Tensor Core з пропускною здатністю, яка виражається в петабітах, і має сховище з низькою затримкою, що масштабується, з Amazon FSx for Lustre.

3.1.2 Вимоги до програмного забезпечення

Використовуємо python3.7 як мову програмування для експерименту. Бібліотеки, про які йшлося раніше, встановлюються за допомогою інсталятора пакетів Python – pip. Бібліотеки встановлюються на EFS і спільно використовуються між усіма машинами, щоб мати ідентичні середовища навчання та тестування, щоб запобігти будь-які не рівні умови.

3.2 Модель

Використовуємо VGG19 [34] для експериментів із запропонованою системою. VGG19 має розмір навченої моделі 549 МБ [35]. Цей розмір перевищує обмеження в 512 МБ AWS Lambda. Тому виконуємо трансферне навчання моделі VGG19. Навчання передачі займає більше 15 хвилин, що є максимально допустимим часом роботи AWS Lambda.

На рисунку 3.1 показано шари моделі VGG19 з модифікованим SoftMax шар. Вхідний розмір – це зображення або масив розмірів (224, 224, 3). Перші два значення – це ширина та висота зображення, а третє значення – кількість

кольорових каналів. «3» означає, що на піксель припадає три значення кольору.

Рисунок 3.1 –VGG19 [34] з новим шаром SoftMax

Для цього експерименту вхідні дані CIFAR10 потрібно змінити з 32 x 32 до 224 x 224. Зображення вже є в трьох каналах, тому подальші зміни – не потрібні. Зображення зі зміненим розміром зберігаються як масиви NumPy у EFS. Після вхідного етапу є два шари згортки 3x3 із 64 фільтрами. Вихід шарів згортки переходить у шар максимального пулу. Рівень MaxPool має фільтр розміром 2x2, і він генерує вихідні дані розміром 112x112x64 із вхідного розміру 224x224x64. Цей результат переходить у два шари згортки 3x3 із 128 фільтрами та MaxPool шар, щоб отримати вихід 56x 56x128. Це повторюється для чотирьох шарів згортки 3x3 із 256 фільтрами з шаром максимального пулу

та двічі для чотирьох шарів згортки 3x3 із 512 фільтрами та шаром максимального пулу, щоб отримати кінцевий результат із частини нейронної мережі згортки, щоб отримати вихід 7x7x512. Цей вхід зрівняно подається на повністю зв’язаний рівень із 4096 виходами, що мають активацію ReLU. Це знову проходить через повністю підключений рівень перед тим, як пройти через остаточний класифікатор SoftMax, який класифікує вхідне зображення в категорію.

3.2.1 Трансфер навчання

Трансферне навчання – це метод, за якого модель, попередньо навчена на наборі даних, налаштовується для роботи з набором даних. Щоб цей підхід працював, набори даних мають бути схожими за функціями. Попередньо навчена модель, яку ми використовуємо для нашого експерименту, є моделлю, навченою на наборі даних ImageNet [36]. ImageNet складається з понад 21 000 класів. Для набору даних CIFAR10 [34] нам потрібна частина попередньо підготовленої моделі для виділення ознак. У результаті ми замінюємо остаточний рівень класифікації SoftMax із 1000 виходами та представляємо новий класифікаційний рівень SoftMax із 10 виходами, який потрібно навчити. Ми заморожуємо ваги попередніх шарів, щоб вони не постраждали в процесі навчання.

3.2.2 Шар SoftMax

Замінюємо шар SoftMax попередньо навченої моделі на рівень SoftMax, який виводить 10 класів. Класифікатор SoftMax перетворює вхідні дані в ймовірності, а потім нормалізує їх. Нормований вихід знаходиться в діапазоні від 0,0 до 1,0, а сума всіх вихідних даних дорівнює 1,0. Вихідний індекс із найвищим значенням є прогнозованим класом. Нам потрібно навчити цей рівень під час нашого навчання передачі, щоб ми могли передбачити результат на основі 10 класів, які ми маємо в нашому наборі даних.

4 РЕЗУЛЬТАТИ

4.1 Лямбда-конфігурація – час

Перша конфігурація, яку можна налаштувати, це тривалість виконання лямбда-функції. Мінімальний час роботи становить 1 секунду, а максимальний – 15 хвилин або 900 с. Наступна конфігурація – пам’ять. Ми зберігаємо оперативну пам’ять на рівні 10240 МБ, тобто 10 ГБ, щоб уникнути будь-яких конфліктів через пам’ять, для навчання моделі ми ставимо максимальне значення. Мова програмування моделі – Python3. Ми експериментуємо з налаштуванням з інтервалом у 1 хвилину, щоб знайти оптимальну конфігурацію часу виконання. Для цього експерименту ми встановили 20 кроків на епоху. У таблиці 4.1 наведено параметри для експерименту.

Таблиця 4.1– Параметри для пошуку оптимального часу роботи

Параметр	**Значення**
Мова програмування	Python3.7
Оперативна пам’ять	10 GB
Кроки	20 на епоху
Епох	5
Розмір партії	8
Тривалість виконання	1 мін інкремент (Мін: 1с, Макс: 15хв)
Кількість працівників	1

Таблиця 4.2 –Результати для оптимального часу роботи для AWS Lambda

Тривалість виконання (у хвилинах)	**Епохи**	**Кроки**	**Загальна кількість кроків**
1	0	1	1
2	0	4	4
3	0	9	9
4	0	15	15
5	1	0	20
6	1	5	25
7	1	11	31
8	1	17	37
9	2	2	42
10	2	7	47
11	2	11	51
12	2	15	55
13	2	19	59
14	3	4	64
15	3	8	68

Таблиця 4.2 показує нам, що ми можемо досягти 3 епох і 8 кроків або 68 кроків навчання за виконання. Це досягається за допомогою значень параметрів у таблиці 4.2. Для кожної додаткової хвилини, доданої до виконання, ми отримуємо додаткові чотири-п’ять кроків. Щоб підтримувати послідовний підрахунок епох, ми встановимо 3 епохи для кожного працівника та залишимо додатковий час, якщо партія займає більше часу, ніж очікувалося.

4.2 Лямбда конфігурація – пам’ять

Наступна конфігурація, яку можна змінити для лямбда-функції, це конфігурація пам’яті. Мінімальний обсяг пам’яті, який може мати лямбда-функція, становить 128 МБ, а максимальний – 10240 МБ або 10 ГБ. Ми експериментуємо з налаштуванням із кроком 1 ГБ і використовуємо журнал максимально використаної пам’яті, наданий AWS для кожного виконання функції. Ми притримуємося часу виконання у 15 хвилин, щоб перевірити вимоги до пам’яті протягом усього процесу. Беремо максимальну кількість епох. У таблиці 4.3 наведено параметри для експерименту.

Таблиця 4.3 – Параметри для знаходження оптимального значення пам'яті

Параметр	**Значення**
Мова програмування	Python3.7
Тривалість	15 хв
Кроки	20 на епоху
Епох	3
Розмір партії	8
Пам’ять	1 Гб інкремент (Мін: 128Мб, Макс: 10Гб)
Кількість працівників	1

Таблиця 4.4 – Результати оптимального значення пам’яті для AWS Lambda

Пам'ять (у ГБ)	**Епохи**	**Кроки**	**Загальна кількість кроків**	**Час**
1	0	0	0	Недостатньо пам’яті

2	0	0	0	Недостатньо пам’яті
3	0	0	0	Недостатньо пам’яті
4	0	1	1	Недостатньо пам’яті
5	2	16	56	15 хвилин
6	3	0	60	14хв 26с
7	3	0	60	13хв 48с
8	3	0	60	13хв 05с
9	3	0	60	12хв 20с
10	3	0	60	11хв 42с

З таблиці 4.4 ми бачимо, що будь-яка конфігурація пам’яті понад 5 ГБ може дати нам результати навчання для 3 епох менше 15 хвилин. Для 5 ГБ пам’яті ми можемо навчити модель, але ми не можемо завершити 3 епохи. Варіант пам’яті у 8 ГБ є оптимальною конфігурацією пам’яті, оскільки ми можемо отримати 2 хвилини буферного часу у випадку, якщо ми перевищимо виконуваний час.

4.3 Кількість робітників

Ми будемо працювати з кількома працівниками паралельно, щоб досягти найшвидшого часу навчання. Щоб отримати максимальну віддачу від паралелізму, нам потрібно визначити оптимальну кількість працівників. У цьому експерименті ми будемо запускати кілька робочих процесів паралельно та дивитися на загальний час, необхідний налаштуванням для завершення процесу навчання. Ми починаємо зі запуску лише одного робочого елемента,

що є лінійним підходом, де лямбда продовжує викликати себе лінійно.

Потім ми експоненціально збільшуємо кількість працівників до 64 паралельно. Підрахунок епох тут є загальною кількістю всіх робочих разом.

Таблиця 4.5 – Час виконання та кількість працівників

Кількість працівників	**Епохи за працівника**	**Епохи**	**Час (хв)**
1	45	45	221
2	36	72	193
4	30	120	157
8	27	216	139
16	21	336	118
32	18	576	132
64	16	1024	168

Зі спостережень ми бачимо, що спочатку, коли ми збільшуємо кількість працівників, час, витрачений на навчання, зменшується. Однак, коли кількість працівників перевищує 16, час починає збільшуватися. Для 32 робітників витрачений час майже дорівнює наявності 8 робітників, а для 64 – 3 робітника. Наявність додаткових працівників не приносить нам користі з точки зору часу. У результаті оптимальна кількість робітників становить від 16 до 20 робітників.

Причина того, що навчання триває довше, навіть якщо у нас є додаткові працівники, полягає в тому, що етап об’єднання повинен збирати градієнти від додаткових працівників і обчислювати суму для всіх градієнтів. Цей процес збільшує накладні витрати на всій фазі навчання та зменшує переваги наявності додаткових працівників. Це тому, що у нас більше мільйона ваг, і кожна вага матиме відповідний градієнт. Кожен працівник генерує ці мільйони

градієнтів, і функція об’єднання повинна зібрати ці ваги та обробити їх перед оновленням ваг. На рисуноку 4.1 наведено графік результатів таблиці 4.5. Він показує час, витрачений працівниками на завершення процесу навчання.

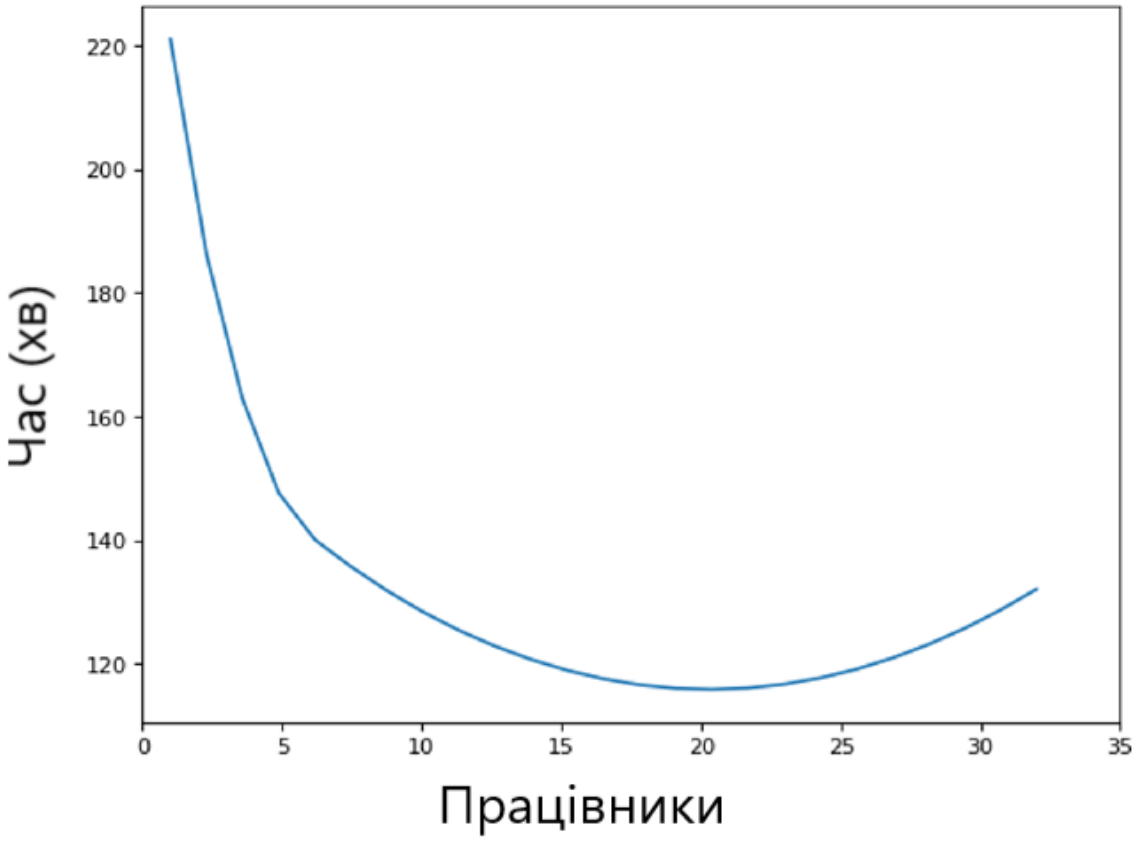

Рисунок 4.1 – Залежність часу обчислення від кількості працівників

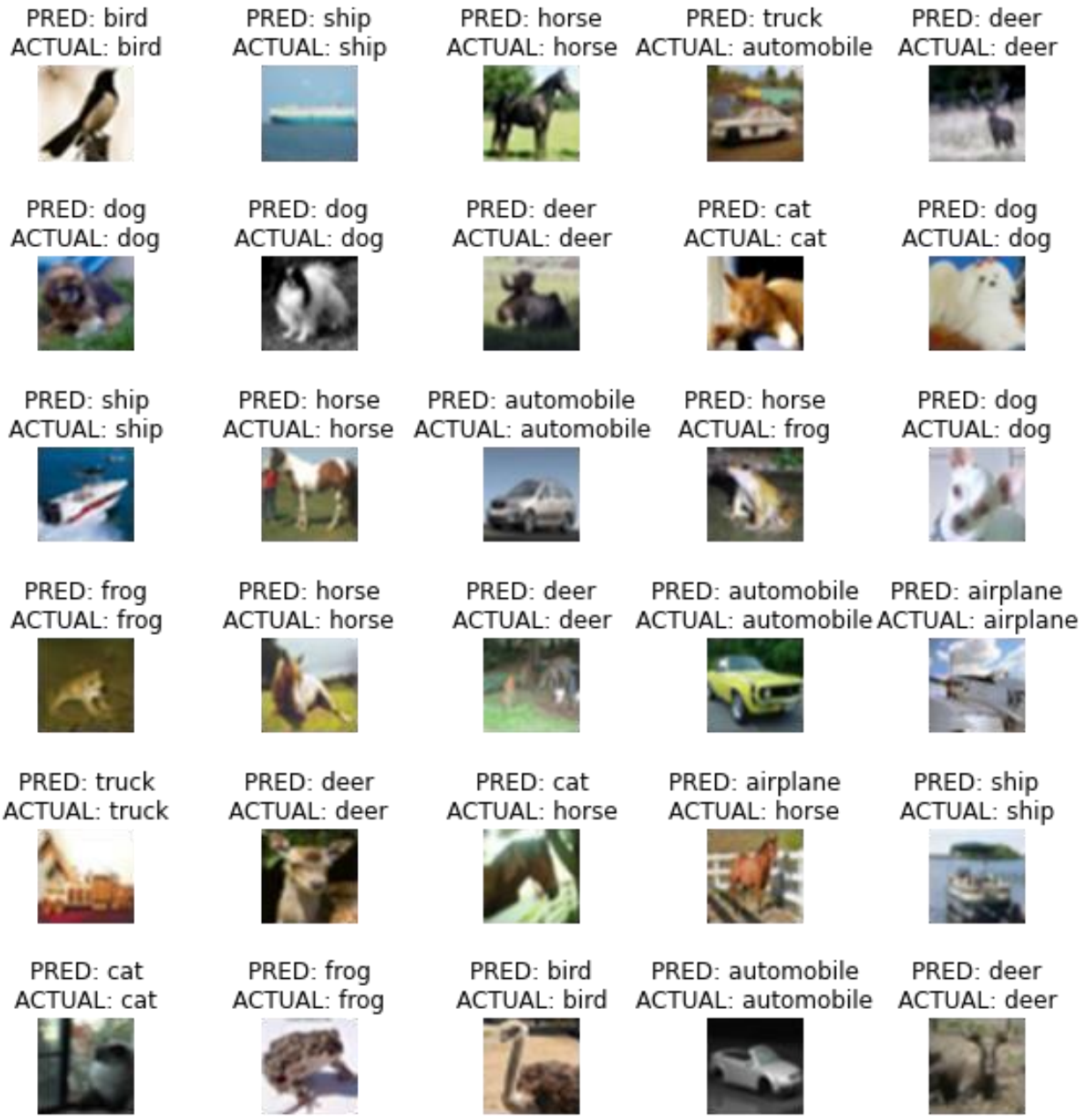

Рисунок 4.2 – Прогнози навченої моделі

На рисунку 4.2 показано прогнозування нашої моделі, навченої 16 працівниками на Lambda елементах. Ми бачимо, що модель точно прогнозує 26 із 30 тестових зображень. Що є не поганим результатом для невеликої моделі.

4.4 Порівняння з іншими екземплярами віртуальних машин

Для порівняння з традиційними екземплярами віртуальних машин ми використовуємо екземпляр EC2 у AWS, який надає віртуальні машини як у попередньо визначених, так і в індивідуальних конфігураціях. Ми використовуємо машину 2, яка має 4 vCPU і 16 ГБ оперативної пам'яті, та машину 3, яка має 96 vCPU, 8 GPU, 360 GB оперативної пам'яті. Ми навчаємо модель за допомогою бібліотек, встановлених на EFS. Це дозволяє нам мати однакове середовище для всіх машин. Ми використовуємо той самий код на Python з бібліотеками Tensor Flow та Keras, який використовується в AWS Lambda, для запуску на екземплярі з модифікаціями, щоб він запускався на екземплярах віртуальних машин.

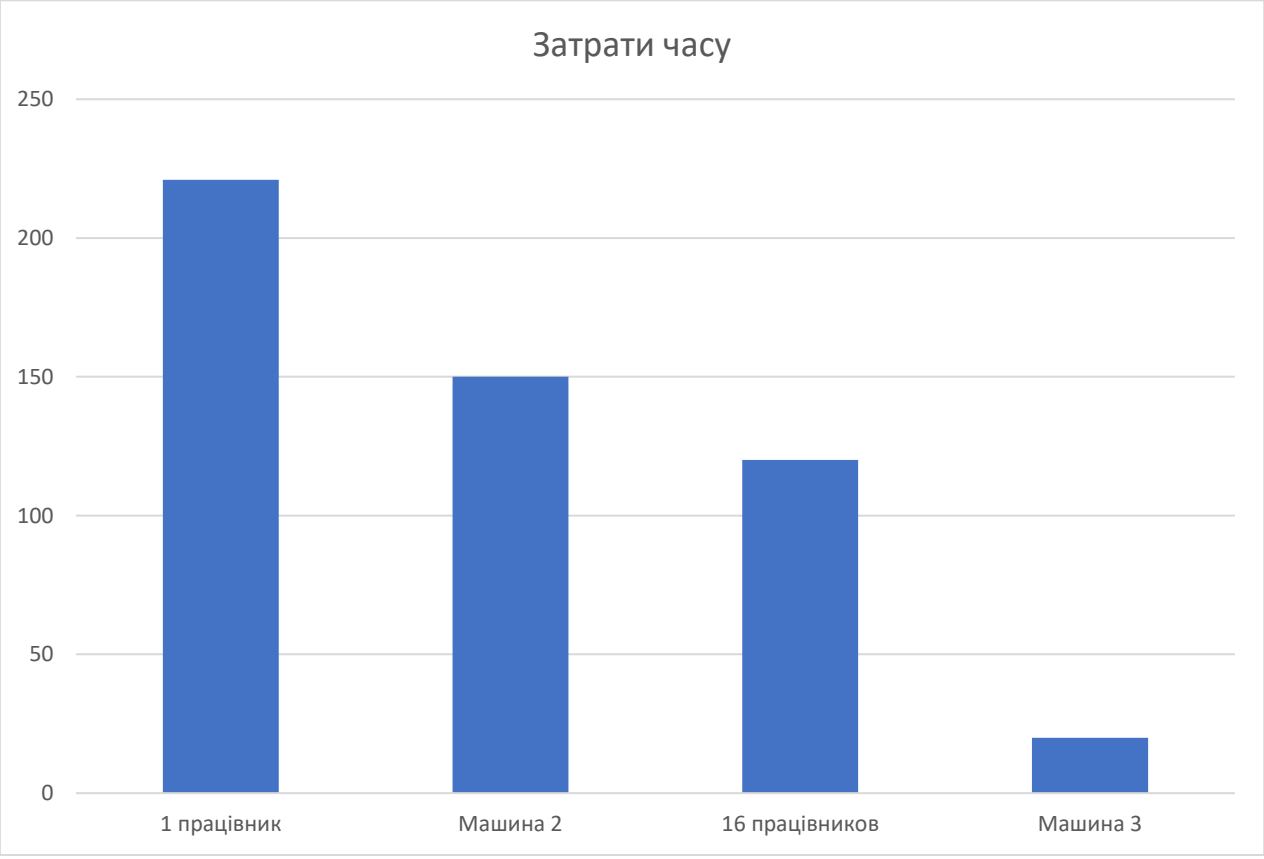

Рисунок 4.3 – Порівняння часу виконання з традиційними віртуальними машинами

З рисунка 4.3 ми бачимо, що машина 2 швидша, ніж один працівник на лямбді, але повільніша, ніж 16 робочих, що працюють паралельно. Одна робоча лямбда діє як традиційна віртуальна машина, але має накладні витрати на етап об'єднання, що робить її повільнішою, ніж традиційна віртуальна машина 2. Для лямбда-системи з 16 робочими елементами переваги паралельності переважають накладні витрати на об'єднання градієнтів. Працівники обчислюють градієнти паралельно, що сприяє прискоренню навчання. Ми бачимо, що наявність більшої кількості працівників призводить до швидшого навчання, ніж машина 2. Однак існує обмеження щодо кількості працівників, які ми можемо мати, перш ніж накладні витрати на наявність цих додаткових працівників переважать їхні переваги.

З іншого боку спеціалізована машина 3, має значні переваги над машиною 2, та зі 16 робочими елементами лямбди. Швидкість навчання перебільшує у 6 разів швидкість навчання на лямбда елементах, та у 7.5 разів ніж звичайна віртуальна машина. Це досягається великою кількістю оперативної пам'яті – 360 ГБ, та надпотужним спеціалізованим 8 ядерним графічним процесором - NVIDIA A100 Tensor Core. Але така машина потребує багато фінансових витрат. Також треба мати на увазі що такі великі потужності потрібі далеко не для всіх задач, та підходять для промислових, великих моделей.

У лямбда елементах основним вузьким місцем навчання на паралелі є фаза комбайну. Фаза об'єднання повинна чекати, поки всі робочі лямбда-вирази завершають обчислення, і лише тоді вона зможе розпочати обробку градієнтів. Крім того, фаза об'єднання повинна зчитувати всі градієнти та підсумовувати їх, що затримує процес навчання. Як наслідок, чим більше працівників ми додаємо до процесу навчання, тим довше буде час очікування початку фази комбайна. Крім того, фаза об'єднання має тривати довше, щоб обчислити всі градієнти. Отже, додавання додаткових працівників поза

точкою максимум призводить до додаткових накладних витрат, які не подолаються паралелізмом лямбда-виробників.

Ще одним обмеженням процесу є синхронізація лямбда-виразів, щоб переконатися, що ми маємо лише одну подію, що запускає фазу об’єднання. Може статися, що кілька лямбда задовольняють умову для фази комбінування. Якщо у нас є декілька лямбда-виразок, що запускають фазу об’єднання, у нас може виникнути більше ніж N робочих процесів, оскільки кожна об’єднана лямбда запускає N своїх власних робочих процесів. Як наслідок, стає необхідним контролювати початок фази об’єднання та переконатися, що запускається одна і тільки одна з цих лямбда.

ВИСНОВКИ

Застосування машинного навчання зростає з кожним днем. Нові моделі навчаються швидкими темпами, і потреба в нових і швидших методах навчання також зростає. Постачальники хмарних послуг мають різноманітні послуги для обчислень. Хмарні обчислення, незважаючи на свої обмеження, виявилися життєво важливою послугою в хмарному домені в цілому. Обмеження хмарних обчислень можна використати на нашу користь і перетворити на ресурс. Обмеження часу та пам’яті в поєднанні породжує новий підхід до економного та розподіленого навчання.

Отримані основні результати: виконана необхідна формалізація, розроблено та досліджено різноманітні методи та алгоритми. Необхідні теореми та твердження були розроблені та підтверджені. Було вивчено тему, досліджено та проаналізовано роботи вітчизняних і іноземних авторів, спроектовано методологію організації машинного навчання у хмарі та проведенні експерименти.

Серед недоліків системи можна виділити наступні речі:

– недостатня швидкодія, порівняно зі спеціалізованими системами;
– неможливість «простого» масштабування, просто додавши більшу кількість працівників.

Серед переваг розробленої системи можна виділити такі аспекти:

– оптимізовані технічні вимоги;
– навіть при короткому періоді навчання моделі нейронної мережі, система має досить високий рівень успішного розпізнавання моделей.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. A Simple Guide to Machine Learning. Warren E. Agin, 2017, ст 1 – 5.
2. Deep Convolution Neural Network sharing for the multi-label images classification // Machine Learning with Applications. Solemane Coulibaly, Bernard Kamsu-Foguem, Dantouma Kamissoko, 2022, ст 4 – 10.
3. Deep learning in computer vision: A critical review of emerging techniques and application scenarios // Machine Learning with Applications. Juniy Chai, Hao Zeng, Anming Li, Eric W.T. Ngai, 2021, ст 3 – 7.
4. Automated Deep Learning Using Neural Network Intelligence: Develop and Design Pytorch and TensorFlow Models Using Python. Ivan Gridin. 2019. ст 277 – 333.
5. Нейронні мережі: повний курс. Хайкін С. М, 2019. ст 1104 с.
6. Нейронні мережі: Короткий довідник. Р.Каллан, М.Вільямс, І.Д, 2017, ст 288.
7. Performance optimization of serverless edge computing function offloading based on deep reinforcement learning, Xuyi Yao, Ningjiang Chen, Xuemei Yuan, Pingjie Ou, 2022.
8. Serverless computing: One step forward, two steps back, Hellerstein J.M, 2018.
9. Reinforcement learning-assisted autoscaling mechanisms for serverless computing platforms. Anastasios Zafeiropoulos, Eleni Fotopoulou, Nikos Filinis, Symeon Papavassiliou, 2022.
10. Convolutional Neural Network (CNN): TensorFlow Core, TensorFlow, 2021, [Електронний ресурс] – https://www.tensorflow.org/tutorials/images/cnn.
    Cloud programming simplified. A berkeley view on serverless computing. Jonas E, 2019.
11. Transparent serverless execution of Python multiprocessing applications, Aitor Arjona, Gerard Finol, Pedro García López, 2022.

12. Amazon Web Services, Optimize CPU options, Amazon EC2 User Guide for Linux Instances, 2021, [Електронний ресурс] – https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-optimize-cpu.html].
13. Amazon Web Services. Amazon EBS volume types, 2021. [Електронний ресурс]. https:// aws.amazon.com/AWSEC2/latest/UserGuide/ebs-volume-types
14. Amazon Web Services, Developer Guide for AWS Lambda, 2021, [Електронний ресурс] – https:// aws.amazon.com/lambda/latest/dg/configuration-function-common
15. Amazon Web Services: EFS, 2021 [Електронний ресурс] – https://aws.amazon.com/efs/
16. Amazon Web Services: Step functions, 2021 [Електронний ресурс] – https://aws.amazon.com/step-functions.
17. Amazon Web Services, S3, 2021 2021 [Електронний ресурс] – https://aws.amazon.com/s3
18. Serverless end game: Disaggregation enabling transparency, López P.G., Slominski A., Shillaker S., Behrendt M., Metzler B. 2020.
19. Occupy the cloud: Distributed computing for the 99% Proceedings of the 2017 Symposium on Cloud Computing, Jonas E., Pu Q., Venkataraman S., Stoica I., Recht B, 2021. ст 445-451
20. Triggerflow Trigger-based orchestration of serverless workflows, Aitor Arjona Pedro, García López, Josep Sampé, Aleksander Slominski, Lionel Villard, 2022.
21. Triggerflow, [Електронний ресурс] – https://github.com/triggerflow/triggerflow.
22. CNCF, Serverless Workflow, [Електронний ресурс] – https://serverlessworkflow.io/.
23. Formal foundations of serverless computing. Proc. ACM Program Lang. Jangda A., Pinckney D., Brun Y., Guha A., 2022, ст 1-26

24. Convolutional Neural Network (CNN): TensorFlow Core, TensorFlow, 2021. [Електронний ресурс] – https://www.tensorflow.org/tutorials/images/cnn.
25. Docker Hub. [Електронний ресурс] – https://hub.docker.com/.
26. Energy price forecasting - problems and proposals for such predictions IEEE Power and Energy Magazine, N. Amjady and M. Hemmati, 2006, ст 20–29
27. Going serverless Communications of the ACM. N. Savage, 2018, ст 15–16
28. Convolutional Neural Network (CNN): TensorFlow Core, TensorFlow. 2021 [Електронний ресурс] – https://www.tensorflow.org/tutorials/images/cnn.
29. Exploring Serverless Computing for Neural Network Training 11th International Conference on Cloud Computing (CLOUD), L. Feng, P. Kudva, D. Da Silva, and J. Hu, 2018.
30. Comparison of FaaS Orchestration Systems IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion). P. Garcia Lopez, M. Sanchez-Artigas, G. Paris, D. Barcelona Pons, A. Ruiz Ollobarren, and D. Arroyo Pinto, 2018.
31. "ECR", 2021 [Електронний ресурс] – https://aws.amazon.com/ecr/.
32. "Using Container Migration for HPC Workloads Resilience," 2019 IEEE High Performance Extreme Computing Conference (HPEC). M. Sindi and J. R. Williams, 2019.
33. [Електронний ресурс] – https://www.cs.toronto.edu/kriz/cifar.html
34. K. Simonyan and A. Zisserman, Very deep convolutional networks for largescale image recognition, 2014.
35. A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images", 2009.
36. "Keras Applications," Team Keras, 2020. [Online]. Available: https://keras.io/api/applications/. [Accessed: 29-Mar-2021]

Відомості кваліфікаційної роботи

Метод організації машинного навчання для оптимізації витрат пам'яті та часу в хмарних обчисленнях

	Прізвище та ініціали відповідальної особи	Підпис	Дата
Роботу виконав студент групи СКСм-22-2. Структура кваліфікаційної роботи: – пояснювальна записка 62 с.; – графічний матеріал 24 арк..	Кучков Б.О.		01.12.2023
Керівник роботи	Хаханова Г.В		01.12.2023
Перевірка на плагіат здійснена. Оригінальність авторського тексту складає %	Литвинова Є.І.		06.12.2023
Нормоконтроль проведено :	Хаханова Г.В		05.12.2023