Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Федеративне машинне навчання для рішень автопілоту

(тема)

Виконав:
студент 2 курсу, групи СШІм-20-2
Ларка І.В.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

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

Освітня програма Системи штучного інтелекту
(повна назва спеціалізації)

Керівник проф. Терзіян В.Я.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ В.О. Філатов
(підпис) (прізвище, ініціали)

2022 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

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

Освітня програма Системи штучного інтелекту (СШІ)
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____» _____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Ларка Ілля Вікторович
(прізвище, ім'я, по батькові)

1. Тема роботи «Федеративне машинне навчання для рішень автопілоту»

затверджена наказом університету від 24 березня 20 22 р. № 414Ст

2. Термін подання студентом роботи до екзаменаційної комісії 13 травня 20 22 р.

3. Вихідні дані до роботи Книги з розробки нейронних мереж, статі на тему федеративне машинне навчання та машинний зір. Документація хмарного провайдера, документація сервісів хмарного провайдера

4. Перелік питань, що потрібно опрацювати в роботі Аналіз предметної області, та постановка задачі, Аналіз федеративного навчання та методів забезпечення конфіденційності, Огляд методів та алгоритмів федеративного навчання, Огляд моделей федеративного навчання, Огляд Tesla Inc, Огляд Comma AI, Проектування системи, Архітектурне рішення, Secure Aggregation протоколу, Алгоритм федеративного агрегування, Окрестратор мікросервісних рішень, Проектування інтелектуальної складової, Проектування інтелектуальної складової, Розробка додатка, Розробка інтелектуальної частини додатка, Розробка інфраструктури додатка, Розробка алгоритму керування автомобіля.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Рисунок 2.1 – Концептуальна схема дерева рішень; Рисунок 3.1 – Загальна схема роботи федеративного навчання; Рисунок 3.2 – Концептуальна архітектурна діаграма інтелектуальної складової в середовищі хмарного провайдера Azure; Рисунок 3.3 – Опис протоколу на високому рівні; Рисунок 3.4 – Втрати на навчальному наборі MNIST; Рисунок 4.5 – Загальний потік роботи на високому рівні абстракції; Рисунок 4.6 – OpenCV engine потік; Рисунок 4.1 – Схема міграції підходів; Рисунок 4.2 – Приклад симуляції автоводіння;

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

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

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1.	Отримання завдання	28.03.2022	Виконано
2.	Аналіз завдання та пошук літератури	26.03.2022-30.03.2022	Виконано
3.	Аналіз предметної області	30.03.2022-05.04.2022	Виконано
4.	Огляд існуючих рішень	05.04.2022-06.04.2022	Виконано
5.	Проектування архітектури додатку	06.04.2022-10.04.2022	Виконано
6.	Розробка інфраструктурної частини додатка	10.04.2022-20.04.2022	Виконано
7.	Розробка інтелектуальної складової	20.04.2022-29.04.2022	Виконано
8.	Оформлення пояснювальної записки	29.04.2022-09.05.2022	Виконано
9.	Захист роботи	17.05.2022	Виконано

Дата видачі завдання 28 березня 20 22 р.

Студент _____
(підпис)

Керівник роботи _____ _____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 104 с., 9 рис., 3 дод., 8 джерел.

АВТОПІЛОТ АВТОМОБІЛЯ, МАШИННЕ НАВЧАННЯ, ФЕДЕРАТИВНЕ МАШИННЕ НАВЧНАННЯ, ХМАРНІ ТЕХНОЛОГІЇ МАШИННОГО НАВЧАННЯ

Об'єкт дослідження – розробка системи федеративного навчання для навчання самостійного водіння автомобіля.

Мета роботи – проектування та реалізація системи федеративного навчання на великій кількості агентів-автомобілів з використанням хмарних технологій та поширення моделей навчання між агентами.

Області застосування – використання будь-якою людиною в повсякденному житті для переміщення, використання логістичними компаніями в роботі.

Методи дослідження – аналіз теоретичного матеріалу, технічної літератури, ринку існуючих рішень та практична реалізація самостійно розробленого додатка.

Визначено мету розробки та цільову аудиторію програми. Здійснено моделювання процесу інтелектуальної обробки відео фрагментів водіння з подальшою автоматичною генерацією моделі навчання.

ABSTRACT

The explanatory note: 104 p., 9 fig., 3 app., 8 sources.

AUTOPILOT OF THE CAR, MACHINE LEARNING, FEDERAL MACHINE LEARNING, CLOUD TECHNOLOGIES OF MACHINE LEARNING

The object of research – is the development of a system of federal training for learning to drive a car.

The purpose of the work is to design and implement a system of federal training for many car agents using cloud technologies and the dissemination of training models among agents.

Scopes – is used by any person in everyday life to move, used by logistics companies at work.

Research methods – analysis of theoretical material, technical literature, the market of existing solutions, and practical implementation of the self-developed application.

The purpose of development and target audience of the program is defined. The modelling of the process of intelligent processing of video fragments of driving with the subsequent automatic generation of the learning model is carried out.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ ТА ТЕРМІНІВ

AAD – Azure Active Direcotry – Ажур Активна Директорія;

Angular – написаний на TypeScript front-end фреймворк з відкритим кодом, який розробляється під керівництвом Angular Team у компанії Google.

ArpaNet – перші версії мережі інтернет;

ASP.NET Core – Active Server Pages Network Enabled Technologies – платформа розробки web-додатка;

Cosmos DB – Cosmos Database – сервіс хмарного провайдера призначений для надійного збереження даних не реляційної структури.

DB – Database – база даних;

Desktop − робочий стіл;

FL – Federated Learning – федеративне навчання;

ML – Machine Learning – машинне навчання;

OAuth – стандарт авторизації в Web-середовищі;

PKCE – Proof Key for Code Exchange – ключ для підтвердження обміну повідомленнями;

TCP – Transmission Control Protocol – транспортний протокол передачі даних;

TCP/IP – Transmission Control Protocol/Internet Protocol – набір протоколів мережі інтернет;

UDP – User Datagram Protocol – найпростіший протокол транспортного рівня моделі OSI;

WWW – World Wide Web – всесвітня мережа інтернет.

ВСТУП

Наразі в світі ведеться декілька процесів розробки автопілотів для управління наземним транспортом, від трамваїв до автомобілів на дорогах загального користування. Найбільш відомий розробника автопілота – Tesla Motors. Сфери використання машин з автоматичним управлінням рухомих засобів – величезна.

Для логістичних компаній використання пристроїв і програм автоматичного керування транспортними засобами дозволить скоротити бюджет. Адже утримання робітників та водіїв – це щомісячні витрати на заробітну плату, медичне страхування, проживання водіїв в містах локаціях, тощо.

Зосередивши увагу, на звичайних автомобілях які використовуються людьми для подорожей, роботи, можна сказати, що кожен автомобіль являється актором в системі і має свої характеристики. Очевидно, що для управління автомобілем треба мати гарну модель комп’ютерного бачення та алгоритми використання цього бачення. Так як кожний автомобіль і кожна відеозапис – це клієнтські дані які містять приватну інформацію, то доцільно використовувати федеративне навчання.

Федеративне навчання (FL) допомагає в навчанні алгоритму машинного навчання і зберігає дані на рівні пристрою. Це означає, що FL дозволяє кожному пристрою зберігати свої особисті та місцеві дані. Ця технологія дозволить широко використовувати рішення машинного навчання, а також гнучкі та керовані дані в режимі реального часу. Методика може бути використана для вирішення численних завдань і в різних контекстах. Вона включає процедури навчання алгоритмів в режимі «оф лайн» і «онлайн». Залежно від умов експлуатації і типу даних алгоритм підбере відповідну методологію. Традиційний метод, такий як централізоване машинне навчання, не включає ці переваги і пов'язаний з високим ризиком для захисту даних і передачі великих файлів.

Коли ми розглядаємо заходи конфіденційності в комплексних навчальних додатках, традиційні методи забезпечують більшу безпеку. Основним недоліком уніфікованого навчання є те, що воно включає градієнтну інформацію, а не необроблені дані. Повідомляючи про оновлення під час навчального процесу, можна зрозуміти, чи використовує центральний сервер і сторонні сервери конфіденційну інформацію. За допомогою нового підходу ви можете використовувати такі інструменти, як диференціальна конфіденційність або багатопартійні обчислення, як безпечні параметри. Використовуючи ці інструменти, ви можете підвищити конфіденційність, знизивши ефективність системи та продуктивність моделі.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ТЕОРЕТИЧНІ ОСНОВИ ФЕДЕРАТИВНОГО МАШИНОГО НАВЧАННЯ

1.1 Аналіз предметної області

У міру того, як експерти заглядають у майбутнє, яке може принести користь світу, вони наголошують на конфіденційності даних. Оскільки штучний інтелект розвиває здатність наслідувати моделі поведінки, ми скоро зможемо передавати такі дані, як медична ультразвукова візуалізація, по всьому світу. Це допоможе алгоритмам машинного навчання розширити досвід людей, а також навчити їх новим завданням та методам за допомогою наборів даних. Штучний інтелект генерує найкращі результати з великою кількістю даних.

Через проблеми з конфіденційністю ми досі не можемо обмінюватися медичними ультразвуковими зображеннями, такими як магнітно-резонансна томографія головного мозку. Ми зберігаємо всі документи пацієнта на території лікарні, але не ділимося жодними даними з міркувань конфіденційності. Федеративне навчання – це штучний інтелект нового покоління з найкращими ідеями конфіденційності даних. Ми будуємо модель, якій можна довіряти, щоб ховати дані.

1.2 Федеративне машинне навчання

1.2.1 Технологія федеративного машинного навчання

Федеративне навчання спрямоване на навчання алгоритму машинного навчання, наприклад, глибокого нейронні мережі, на декількох локальних наборах даних, що містяться в локальних вузлах, без явного обміну зразками даних. Загальний принцип полягає у навчанні місцевих моделей на місцевих зразках даних та обміні ними параметри (наприклад, ваги та упередження

глибокої нейронної мережі) між цими локальними вузлами на деякій частоті, щоб сформувати глобальну модель, спільну для всіх вузлів.

Федеративне навчання допомагає у навчанні алгоритму машинного навчання та зберігає дані на рівні пристроїв. Це означає, що FL дозволяє кожному пристрою зберігати свої особисті та локальні дані. Ця технологія забезпечить широке поширення рішень машинного навчання, а також гнучкі та керовані дані в режимі реального часу.

Методику можна використовувати для вирішення численних завдань та у різних контекстах. Вона включає процедури навчання алгоритмів в режимі «офлайн» і «онлайн». В залежності від умов експлуатації та типу даних алгоритм підбере відповідну методику. Традиційний метод, наприклад, централізоване машинне навчання, не включає ці переваги і пов'язаний з високим ризиком для захисту даних і передачі великих файлів.

Горизонтальне федеративне навчання та однорідне федеративне навчання можуть вирішувати технічні та практичні завдання шляхом поділу даних на різні підрозділи. Процес працює за рахунок введення аналогічних наборів даних у порівнянний простір. Алгоритм порівнює характеристики та пов'язує їх відповідним чином.

У об'єднаному вертикальному навчанні різні набори даних мають однакові ідентифікатори вибірки, але різні функціональні простори.

1.2.2 Порівняння федеративного та класичного розподіленого машинного навчання

Основна різниця між федеративним навчанням та розподіленим навчанням полягає у припущеннях щодо властивостей локальних наборів даних, як розподілене навчання спочатку націлений на паралелізація обчислювальної потужності де федеративне навчання спочатку спрямоване на навчання з неоднорідними наборами даних.

Можливості пристроїв можуть змінюватись в залежності від мережі. зв'язок, обладнання та живлення. Крім того, системні обмеження та розмір мережі призведуть лише до невеликої кількості пристроїв. Кожен пристрій ненадійно і зазвичай падає за певної ітерації.

Оскільки у федеративних мережах підключається велика кількість пристроїв, мережа може працювати повільніше. Це може спричинити зв'язок. Крім того, зв'язок може бути дорожчим, ніж при використанні традиційних методів. Для раціоналізації процесу федеративного навчання потрібно розробити ефективну структуру зв'язку. Для навчання моделі необхідно надсилати невеликі повідомлення замість ділитися повним набором даних через мережу.

Коли ми розглядаємо заходи щодо забезпечення конфіденційності у федеративних навчальних програмах, традиційні методи забезпечують велику безпеку. Основним недоліком об'єднаного навчання є те, що воно включає градієнтну інформацію, а не необроблені дані. Повідомляючи про оновлення в процесі навчання, ви можете зрозуміти, чи не використовують центральний сервер і сторонні сервери конфіденційну інформацію.

За допомогою нового підходу можливо використовувати такі інструменти, як диференціальна конфіденційність або багатопартійні обчислення як безпечні опції. Використовуючи ці інструменти, можливо підвищити конфіденційність, знизивши ефективність системи та продуктивність моделі.

1.3 Механізм забезпечення конфіденційності

Конфіденційність є важливим аспектом в зборі та аналізі даних. Наприклад збереження конфіденційності місця положення користувачів були вивчені багатьма дослідниками. Для того щоб формально зберегти конфіденціальність загальних типів даних, декілька механізмів таких як k-анонімність та протокол конфіденціального обчислення були опубліковані

в різних роботах. Останнім часом, диференціальна приватність вказує на декілька вразливостей k-анонімності і стає все популярнішою через можливість використання кількісної міри ризику приватності. Диференціальна приватність була використана в платформі аналізу даних для забезпечення конфіденційності, для шумування даних параметрів навченої моделі, а також для збереження конфіденційності при вилученні даних з розподілених частин часових рядів.

1.3.1 Диференційна конфіденціальність як засіб забезпечення приватності даних

Конфіденційність є важливою проблемою в розподіленому машинному навчанні. У той самий час основна перевага використання федеративних підходів до машинного навчання – забезпечення конфіденційності чи секретності даних за використанням диференційної конфіденціальність. Ніякі локальні дані не вивантажуються ззовні, не поєднуються або обмінюються. Оскільки вся база даних сегментована на локальні біти, це ускладнює її зламування.

При федеративному навчанні здійснюється обмін лише параметрами машинного навчання. Крім того, такі параметри можуть бути зашифровані перед спільним використанням між циклами навчання для розширення конфіденційності, а схеми гомоморфного шифрування можуть використовуватися для безпосереднього виконання обчислень із зашифрованими даними без попереднього дешифрування.

Таким чином, можливість запитів вузлів є основним питанням, яке можна вирішити за допомогою диференціальної конфіденційності та безпечного агрегування.

1.4 Постановка задач дослідження

1.4.1 Класифікація обраної задачі

Вирішувана задача відносяться до рівня критичного для сучасного світу, а саме полягає в розробці алгоритмів федеративного машинного навчання на великій кількості агентів-авто. А саме необхідно реалізувати:

- модель комп’ютерного бачення;
- управління автомобілем в залежності від результатів обробки кожного кадру комп’ютерним баченням;
- розробка алгоритму та підходу до збереження моделі;
- розробити алгоритм для поширення збереженої та навченої моделі на агенті через мережу інтернет;
- розробити алгоритм для шифрування моделі;
- розробити алгоритм для злиття моделей всіх підмножин агентів в єдину цілу для поширення з іншими агентами;

1.4.2 Визначення вимог до функціоналу

Ця робота визначає мету як: отримання працездатної системи розподіленого та федеративного навчання на великій кількості агентів підключених до неї. Ідеальним результатом було би розробити пристрій який з’єднується з телефоном в кого є доступ до мережі інтернет. Розроблений пристрій повинен бути не великого розміру, та легко монтуватися близько дзеркалу заднього виду.

Пристрій повинен через телефон проводити оновлення моделі навченої та відправлення того, чому локальна модель була навчена.

2 МЕТОДИ ТА АЛГОРИТМИ ФЕДЕРАТИВНОГО МАШИННОГО НАВЧАННЯ

2.1 Аналіз алгоритмів федеративного машинного навчання

Щоб забезпечити хороше виконання завдань остаточної центральної моделі машинного навчання, федеративне навчання ґрунтується на ітеративному процесі, розбитому на атомарний набір взаємодій клієнт-сервер, відомий як цикл федеративного навчання. Кожен раунд цього процесу полягає у передачі поточного стану глобальної моделі вузлам, що навчають, навчанні локальних моделей на цих локальних вузлах для створення набору потенційних оновлень моделі на кожному вузлі, а потім агрегації та обробки цих локальних оновлень в одне глобальне оновлення і застосовуючи його до глобальної моделі. У методології нижче центральний сервер використовується для агрегування, а локальні вузли виконують локальне навчання, залежно від замовлень центрального сервера. Однак інші стратегії призводять до тих самих результатів без центральних серверів, в одноранговому підході, з використанням методології пліток або консенсусу.

Припускаючи, що федеративний раунд складається з однієї ітерації процесу навчання, процедуру навчання можна резюмувати таким наступним чином.

Ініціалізація: відповідно до вхідних даних сервера вибирається модель машинного навчання (наприклад, лінійна регресія, нейронна мережа, посилення) для навчання на локальних вузлах та ініціалізації. Потім вузли активуються і чекають, доки центральний сервер видає обчислювальні завдання.

Вибір клієнта: вибирається частина локальних вузлів початку навчання на локальних даних. Вибрані вузли отримують поточну статистичну модель, тоді як інші чекають на наступний раунд об'єднання.

Конфігурація: центральний сервер наказує вибраним вузлам пройти навчання моделі на їх локальних даних заздалегідь заданим способом (наприклад, для деяких міні-пакетних оновлень градієнтного спуску).

Звітність: кожен вибраний вузол надсилає свою локальну модель на сервер для агрегування. Центральний сервер агрегує отримані моделі та відправляє назад оновлені моделі на вузли. Він також обробляє збої для вимкнених вузлів або втрачені оновлення моделі. Починається наступний раунд федерації, який повертається до фази вибору клієнтів.

Припинення: при дотриманні попередньо визначеного критерію завершення (наприклад, досягається максимальна кількість ітерацій або точність моделі перевищує граничне значення) центральний сервер об'єднує оновлення та завершує глобальну модель.

Розглянута раніше процедура передбачає синхронізоване оновлення моделі. Нещодавні розробки в галузі федеративного навчання представили нові методи вирішення проблеми асинхронності під час тренувального процесу або навчання з використанням моделей, що динамічно змінюються. У порівнянні з синхронними підходами, коли обмін локальними моделями відбувається після того, як обчислення були виконані для всіх рівнів нейронної мережі, асинхронні використовують властивості нейронних мереж для обміну оновленнями моделей, як тільки стають доступними обчислення певного рівня. Ці методи також називають роздільним навчанням, і вони можуть застосовуватися як під час навчання, так і під час виведення, незалежно від централізованих або децентралізованих налаштувань федеративного навчання.

2.1.1 Неідентифікаційні дані

Найчастіше припущення про незалежні та ідентично розподілені вибірки по локальним вузлам не виконується для установок федеративного навчання. При цьому налаштуванні характеристики процесу навчання

можуть різнитися залежно від незбалансованості локальних вибірок даних, а також конкретного розподілу ймовірностей навчальних прикладів (тобто функцій і міток), що зберігаються в локальних вузлах. Опис даних, ґрунтується на аналізі спільної ймовірності між функціями та мітками для кожного вузла. Це дозволяє відокремити кожен вклад відповідно до конкретного розподілу, доступного на локальних вузлах. Основні категорії даних без ідентифікаторів можна резюмувати наступним чином:

Коваріативний зсув: локальні вузли можуть зберігати приклади, які мають різні статистичні розподіли в порівнянні з іншими вузлами. Приклад зустрічається в наборах даних обробки природної мови, де люди зазвичай пишуть ті самі цифри/літери з різною шириною штриха або нахилом.

Апріорний зсув ймовірності: локальні вузли можуть зберігати мітки, статистичні розподіли яких відрізняються від інших вузлів. Це може статися, якщо набори даних розділені по регіонах та/або демографічно. Наприклад, набори даних, що містять зображення тварин, значно відрізняються від країни до країни.

Зміщення концепцій (одна й та сама мітка, різні функції): локальні вузли можуть мати одні й ті самі мітки, але деякі з них відповідають різним функціям у різних локальних вузлах. Наприклад, зображення, на яких зображений конкретний об'єкт, можуть відрізнятися залежно від погодних умов, за яких вони були зняті.

Зсув концепції (однакові функції, різні мітки): локальні вузли може мати одні й самі функції, але з них відповідають різним міткам у різних локальних вузлах. Наприклад, при обробці природної мови аналіз тональності може давати різні настрої, навіть якщо проглядається той самий текст.

Незбалансованість: обсяг даних, доступних на локальних вузлах, може значно відрізнятися за розміром. Втрата точності через дані, відмінні від iid, може бути обмежена за допомогою складніших засобів нормалізації даних, а не пакетної нормалізації.

2.2 Моделі федеративного машинного навчання

2.2.1 Нейронні мережі

Виконання машинного навчання передбачає створення моделі, яка навчається за деякими навчальними даними, а потім може обробляти додаткові дані для прогнозування. Для систем машинного навчання були використані та досліджені різні типи моделей.

Штучні нейронні мережі (ANN) або коннекціоністські системи – це обчислювальні системи, невиразно натхненні біологічними нейронними мережами, які складають мозок тварин. Такі системи "навчаються" виконувати завдання, розглядаючи приклади, як правило, без програмування для яких-небудь конкретних правил. ANN – це модель, заснована на сукупності з'єднаних одиниць або вузлів, званих "штучними нейронами", які вільно моделюють нейрони в біологічному мозку. Кожне з'єднання, як і синапси в біологічному мозку, може передавати інформацію, "сигнал", від одного штучного нейрона до іншого.

Штучний нейрон, який отримує сигнал, може його обробити, а потім сигналізувати про додаткові штучні нейрони, підключені до нього. У поширених реалізаціях ANN сигнал при з'єднанні між штучними нейронами є дійсним числом, і вихід кожного штучного нейрона обчислюється деякою нелінійною функцією від суми його входів. Зв'язки між штучними нейронами називаються "краями". Штучні нейрони та краї зазвичай мають вагу, яка регулюється під час навчання. Вага збільшує або зменшує силу сигналу при з'єднанні. Штучні нейрони можуть мати такий поріг, що сигнал надсилається лише в тому випадку, якщо сукупний сигнал перетинає цей поріг. Як правило, штучні нейрони агрегуються в шари. Різні шари можуть виконувати різні види трансформацій на своїх вхідних даних. Сигнали проходять від першого шару (вхідного шару) до останнього шару (вихідного шару), можливо, після кількох обходів шарів. Початковою метою підходу

ANN було вирішення проблем так само, як це робив би людський мозок. Однак з часом увага перейшла до виконання конкретних завдань, що призвело до відхилень від біології.

Штучні нейронні мережі використовувалися для вирішення різноманітних завдань, включаючи комп'ютерний зір, розпізнавання мови, машинний переклад, фільтрацію соціальних мереж, гра в настільні та відеоігри та медичну діагностику. Глибоке навчання складається з декількох прихованих шарів у штучній нейронній мережі. Цей підхід намагається моделювати, як людський мозок перетворює світло і звук на зір та слух. Деякі успішні застосування глибокого навчання – це комп'ютерний зір та розпізнавання мови.

2.2.2 Дерева рішень

Вивчення дерева рішень використовує дерево рішень як модель прогнозування для переходу від спостережень за предметом (представленим у гілках) до висновків щодо цільового значення елемента (представленого у листках). Це один із підходів до прогнозного моделювання, який використовується у статистиці, видобутку даних та машинному навчанні.

Моделі дерев, де цільова змінна може приймати дискретний набір значень, називаються деревами класифікації; у цих деревних структурах листя представляють мітки класів, а гілки – поєднання ознак, які ведуть до цих міток класів. Дерева рішень, де цільова змінна може приймати безперервні значення (зазвичай це дійсні числа), називаються деревами регресії. При аналізі рішень дерево рішень може бути використано для наочного та явного представлення рішень та прийняття рішень. У видобутку даних дерево рішень описує дані, але отримане дерево класифікації може бути вхідним для прийняття рішень. На рисунку 2.1 зображено приклад дерева рішень. Корінь дерева – це перше питання, та кожен наступне

питання представлено вузлом q_{ij}, де i – ідентифікаційний номер питання, а j – номер рівня в дереві рішень. Кінцевим результатом є вершина виводу.

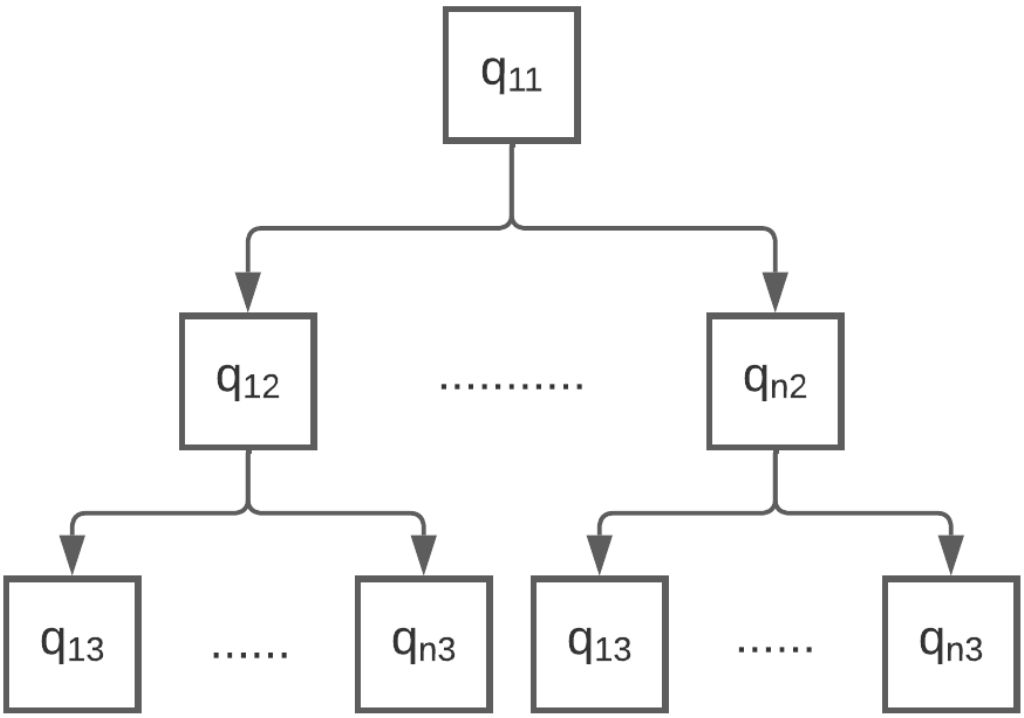


Рисунок 2.1 – Концептуальна схема дерева рішень

2.2.3 Опорно-векторні машини

Машини опорних векторів (SVM), також відомі як мережі векторних опор,-це набір пов'язаних методів навчання, що використовуються для класифікації та регресії. Враховуючи набір навчальних прикладів, кожен з яких позначений як приналежні до однієї з двох категорій, алгоритм навчання SVM будує модель, яка передбачає, чи потрапляє новий приклад до тієї чи іншої категорії. Навчальний алгоритм SVM – це неімовірнісний, двійковий, лінійний класифікатор, хоча існують такі методи, як масштабування плати для використання SVM у ймовірнісній класифікації. На додаток до виконання лінійної класифікації, SVM можуть ефективно

виконувати нелінійну класифікацію, використовуючи те, що називається трюком ядра, неявно відображаючи свої вхідні дані у високо вимірні простори ознак.

2.2.4 Регресійний аналіз

Регресійний аналіз охоплює велику різноманітність статистичних методів для оцінки зв'язку між вхідними змінними та їх пов'язаними ознаками. Його найпоширенішою формою є лінійна регресія, де окрема лінія проводиться так, щоб найкраще відповідати даним відповідно до такого математичного критерію, як звичайні найменші квадрати. Останнє часто поширюється методами регуляризації (математики) для пом'якшення надмірності та упередженості, як у регресії хребта. При вирішенні нелінійних проблем перехідні моделі включають поліноміальну регресію (наприклад, використовувану для підгонки лінії тренду в Microsoft Excel), логістичну регресію (часто використовується в статистичній класифікації) або навіть регресію ядра, яка вводить нелінійність, користуючись перевагами трюка ядра для неявного відображення вхідних змінних у простір вищого виміру.

2.2.5 Байєсові мережі

Байєсова мережа, мережа переконань або орієнтована ациклічна графічна модель – це імовірнісна графічна модель, яка представляє набір випадкових величин та їх умовну незалежність із спрямованим ациклічним графіком (DAG). Наприклад, байєсівська мережа може представляти ймовірні зв'язки між хворобами та симптомами. Враховуючи симптоми, мережу можна використовувати для обчислення ймовірності наявності різних захворювань. Існують ефективні алгоритми, які здійснюють висновки та навчання.

Байєсові мережі, які моделюють послідовності змінних, наприклад мовні сигнали або білкові послідовності, називаються динамічними байєсівськими мережами. Узагальнення байєсівських мереж, які можуть представляти та вирішувати проблеми прийняття рішень в умовах невизначеності, називаються діаграмами впливу.

2.2.6 Генетичні алгоритми

Генетичний алгоритм (GA) – це алгоритм пошуку та евристична техніка, що імітує процес природного відбору, використовуючи такі методи, як мутація та кросовер, для створення нових генотипів у надії знайти хороші рішення даної проблеми. У машинному навчанні генетичні алгоритми використовувалися у 1980 -х та 1990 -х роках. І навпаки, методи машинного навчання були використані для покращення продуктивності генетичних та еволюційних алгоритмів.

2.3 Огляд існуючих рішень

2.3.1 Tesla Inc

Tesla, Inc. – американська автомобільна та екологічна компанія, що базується в Остіні, штат Техас. Tesla розробляє та виробляє електромобілі (електромобілі та вантажівки), накопичувачі енергії від дому до мережі, сонячні панелі та сонячну черепицю, а також супутні продукти та послуги.

Tesla є однією з найдорожчих компаній у світі та залишається найдорожчим виробником автомобілів у світі з ринковою капіталізацією понад 900 мільярдів доларів США. Компанія мала найбільше в світі продажів акумуляторних електромобілів і електромобілів із підзарядкою, захопивши 23% ринку акумуляторних електричних (чисто електричних) і 16% ринку підключаються (включаючи гібриди, що підключаються) у 2020

році. Через свою дочірню компанію Tesla Energy, компанія розробляє та є основним установником фотоелектричних систем у Сполучених Штатах. Tesla Energy також є одним із найбільших світових постачальників систем накопичення енергії акумуляторів, у 2021 році встановлені 3,99 гігават-годин (ГВт-год).

Рік тому компанія випустила бета тестування Full-self driving автопілоту.

Можливості автопілоту:

– навігація на автопілоті (бета-версія): активно керує вашим автомобілем від рампи на автомагістралі до з'їзду, включаючи пропозицію щодо зміни смуги руху, навігаційну розв'язку, автоматичне включення покажчика повороту та правильний з'їзд;

– автоматична зміна смуги: допомагає рухатися на сусідню смугу на шосе, коли ввімкнено автоматичне керування;

– автопарк: допомагає автоматично припаркувати автомобіль паралельно або перпендикулярно одним дотиком;

– summon: переміщує ваш автомобіль в тісний простір і з нього за допомогою мобільного додатка або ключа;

– розумний виклик: ваш автомобіль буде переміщатися в складніших середовищах і паркувальних місцях, маневруючи навколо об'єктів, якщо це необхідно, щоб знайти вас на стоянці;

– контроль дорожнього руху та знаків зупинки (бета-версія): визначає знаки зупинки та світлофори та автоматично сповільнює ваш автомобіль до зупинки під час наближення під вашим активним наглядом.

2.3.2 Comma AI

Comma Openpilot – це система допомоги водієві з відкритим вихідним кодом. Наразі openpilot виконує функції адаптивного круїз-контролю (ACC) і автоматизованого центрування смуги руху (ALC) для сумісних

транспортних засобів. Він працює аналогічно автопілоту Tesla і GM Super Cruise. Openpilot може автоматично керувати, прискорюватися та гальмувати для інших транспортних засобів у своїй смузі.

Багато факторів можуть вплинути на продуктивність openpilot ALC і openpilot LDW, через що вони не зможуть функціонувати належним чином. Вони включають, але не обмежуються:

– погана видимість (сильний дощ, сніг, туман тощо) або погодні умови, які можуть заважати роботі датчика;

– камера на дорозі заблокована, покрита або пошкоджена брудом, льодом, снігом тощо;

– заборона, спричинена нанесенням надмірної кількості фарби або клейких продуктів на автомобіль;

– пристрій змонтовано неправильно;

– на різких поворотах, як-от пандуси, перехрестя тощо; openpilot розроблений таким чином, щоб обмежувати крутний момент керма, який він може створити;

– за наявності заборонених смуг або зон забудови;

– під час руху по дорогах з великим нахилом або при сильному боковому вітру;

– надзвичайно високі або низькі температури;

– яскраве світло;

– їзда по пагорбах, вузьких або звивистих дорогах.

2.3.3 Круїз контроль реалізація від інших розробників автомобілів

Кожний автовиробник в преміальних функціях автомобілів має опцію підключення круїз-контолю. В більшості випадків, це не складні алгоритми, в яких автомобіль має досить обмежені можливості, як слідкування автомобіля спереду, або слідкування за дорожніми лініями.

3 ПРОЕКТУВАННЯ СИСТЕМИ

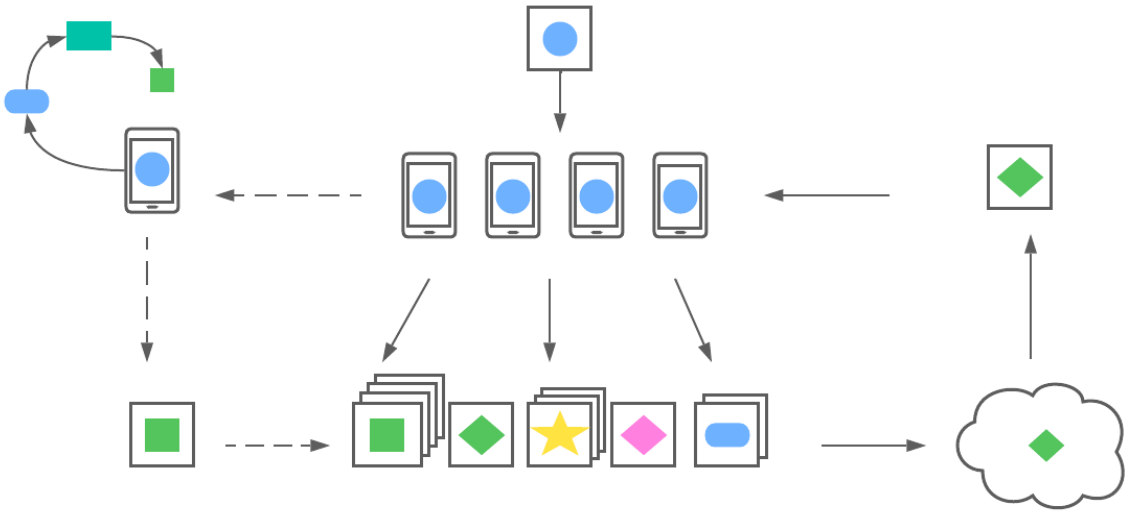
3.1 Стандартна архітектура

Стандартні підходи машинного навчання вимагають централізації навчальних даних на одній машині або в центрі обробки даних. Розроблено одну з найбільш безпечних і надійних хмарних інфраструктур для обробки цих даних, щоб покращити наші послуги. Тепер для моделей, які навчаються за допомогою взаємодії користувачів із мобільними пристроями або автономними агентами, вводиться додатковий підхід: федеративне навчання.

Об’єднане навчання дає змогу мобільним пристроями спільно вивчати спільну модель прогнозування, зберігаючи при цьому всі навчальні дані на пристрої, відокремлюючи можливість машинного навчання від необхідності зберігати дані в хмарі. Це виходить за рамки використання локальних моделей, які роблять прогнози на мобільних пристроях (наприклад, Mobile Vision API і On-Device Smart Reply), також забезпечуючи навчання моделі на пристрої. Це працює так: пристрій завантажує поточну модель, покращує це, вивчаючи дані на вашому пристрої, а потім підсумовує зміни як невелике цілеспрямоване оновлення. Лише це оновлення моделі надсилається до хмари за допомогою зашифрованого зв’язку, де воно негайно усереднюється з іншими оновленнями користувача для покращення спільної моделі. Усі дані про тренування залишаються на вашому пристрої, а окремі оновлення не зберігаються в хмарі. На рисунку 3.1 представлена загальна модель роботи федеративного навчання.

Federated Learning дає змогу створювати розумніші моделі, меншу затримку та менше енергоспоживання, забезпечуючи при цьому конфіденційність. І цей підхід має ще одну безпосередню перевагу: на додаток до оновлення спільної моделі, покращену модель на вашому

телефоні також можна використовувати негайно, забезпечуючи персоналізований досвід використання телефону.



Рисунку 3.1 – Загальна схема роботи федеративного навчання

Щоб зробити федеративне навчання можливим, довелося подолати багато алгоритмічних і технічних проблем. У типовій системі машинного навчання алгоритм оптимізації, як-от Stochastic Gradient Descent (SGD), працює на великому наборі даних, однорідно-розділеному на сервери в хмарі. Такі високоітераційні алгоритми вимагають з'єднання з навчальними даними з низькою затримкою та високою пропускною здатністю. Але в налаштуваннях федеративного навчання дані розподіляються між мільйонами пристроїв дуже нерівномірно. Крім того, ці пристрої мають значно більшу затримку та меншу пропускну здатність і доступні лише періодично для навчання. Ці обмеження пропускної здатності та затримки мотивують алгоритм федеративного усереднення, який може навчати глибокі мережі, використовуючи в 10-100 разів менше зв'язку в порівнянні з наївно об'єднаними версія SGD. Ключова ідея полягає в тому, щоб використовувати потужні процесори в сучасних пристроях для обчислення оновлень більш високої якості, ніж прості кроки градієнта. Оскільки для

створення хорошої моделі потрібно менше ітерацій високоякісних оновлень, навчання може використовувати набагато менше комунікації. Оскільки швидкість завантаження зазвичай набагато нижча, було також розроблено новий спосіб зменшити витрати на передачу даних ще в 100 разів шляхом стиснення оновлень за допомогою випадкових обертань і квантування. Хоча ці підходи зосереджені на навчанні глибоких мереж, потрібно також розроблено алгоритми для високорозмірних розріджених опуклих моделей, які відмінно справляються з такими проблемами, як прогнозування шляху. Розгортання цієї технології на мільйонах різнорідних пристроях, які працюють під керуванням системи контролю і керування автомобіля, вимагає складних рішень. На навчанні пристрою використовується мініатюрна версія TensorFlow. Ретельне планування гарантує, що навчання відбуватиметься лише тоді, коли пристрій неактивний, підключений до мережі та має безкоштовне бездротове з'єднання, тому це не впливає на продуктивність пристрою.

Потім системі необхідно обмінюватися інформацією та об'єднувати оновлення моделі безпечним, ефективним, масштабованим і відмовостійким способом. Лише поєднання досліджень із цією інфраструктурою робить можливими переваги федеративного навчання. Федеральне навчання працює без необхідності зберігати дані користувача в хмарі, але це ще не всі переваги. Було застосовано протокол Secure Aggregation, який використовує криптографічні методи, тому координуючий сервер може розшифрувати середнє оновлення лише за умови участі 100 або 1000 пристроїв – жодне оновлення окремого пристрою не може бути перевірено перед усередненням. Це перший протокол у своєму роді, який практичний для вирішення проблем, пов'язаних із великими мережами, і реальних обмежень підключення. Було розроблено федеративне усереднення, щоб сервер координації потребував лише середнього оновлення, що дозволяє використовувати безпечну агрегацію; однак протокол є загальним і може застосовуватися і до інших проблем.

Federated Learning не може вирішити всі проблеми машинного навчання (наприклад, навчитися розпізнавати різні породи собак шляхом навчання на ретельно позначених прикладах), а для багатьох інших моделей необхідні навчальні дані вже зберігаються в хмарі (наприклад, навчальні фільтри спаму для Gmail). Тому вдосконалення найсучасніших технології для хмарного машинного навчання продовжується. Крім того, коло проблем, які ми можемо вирішити за допомогою Federated Learning, розширюється. Застосування Federated Learning вимагає від практиків машинного навчання прийняти нові інструменти та новий спосіб мислення: розробку моделей, навчання й оцінку без прямого доступу до необроблених даних чи маркування.

3.2 Архітектурне рішення

На рисунку 3.2 зображено архітектура інтелектуальної складової. Маючи N-ну кількість клієнтів (користувачів використовуючих автопілот) які використовуючи DNS Zone Server для знаходження кінцевих IP адресів Application Gateway. Application Gateway захищений за допомогою Azure Firewall. Якщо говорячи про Application Gateway – який дає можливості форсувати HTTPS/TSL протоколи та коригування заголовків відповідно до рекомендацій та кращих практик, то Azure Firewall – дає глибокий аналіз трафіку та налаштування пропуску чи відмови деяким запитам.

Вся інфраструктура інтелектуальної складової додатка знаходиться в Private Virtual Network, що в свою чергу дає додатковий захист, так як всі запити і вся комунікація між сервісами відбувається в середині мережі та не виходить за її межи.

VM Scale Sets – це набори віртуальних машин для масштабування навантаження для агрегування та обновлення моделей клієнтів. Та на самому кінці знаходиться сховище моделі, яке було розроблено як сховище бінарних даних.

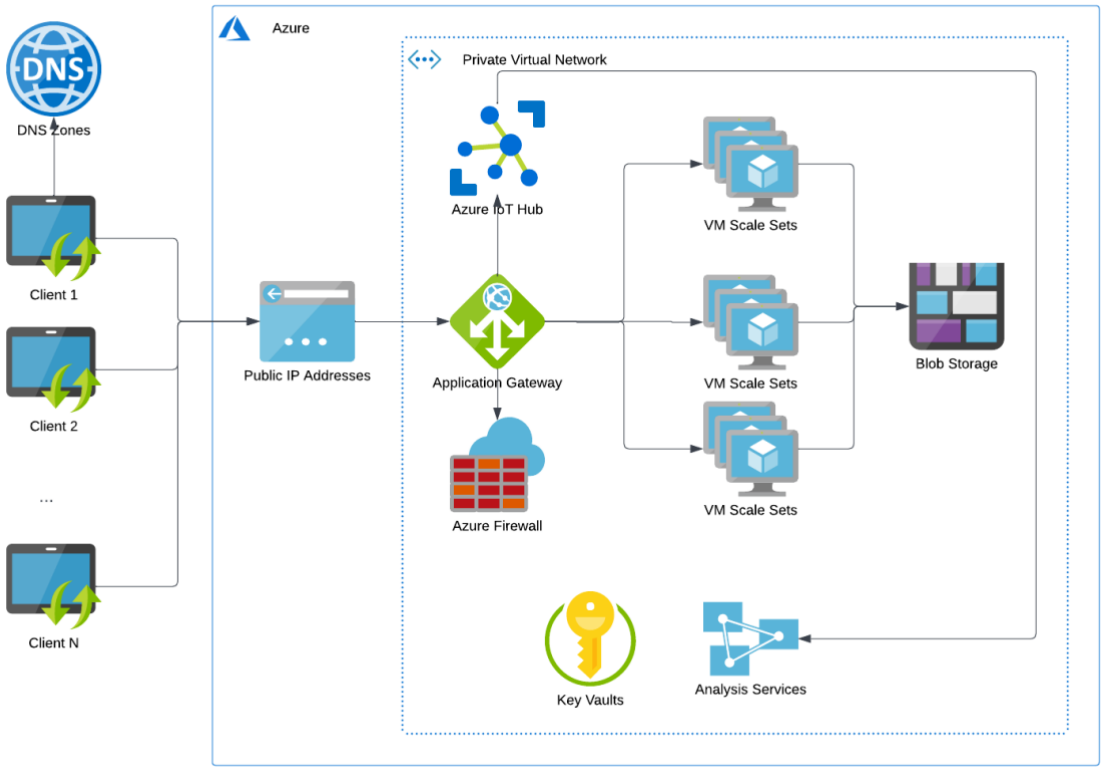


Рисунок 3.2 – Концептуальна архітектурна діаграма інтелектуальної складової в середовищі хмарного провайдера Azure

3.3 Secure Aggregation протокол

Моделі машинного навчання, навчені на чутливих реальних даних, обіцяють покращення у всьому, від медичного скринінгу до виявлення спалаху захворювання. А широке використання мобільних пристроїв означає, що стають доступні ще багатші – і більш чутливі – дані.

Щоб навчити глибоку інтелектуальну складову керувати автомобілем, який користувач введе під час подорожі. Розробник моделювання може захотіти навчити таку модель для всієї місцевості на великій групі користувачів. Однак дані про переміщення, а тим більше відео з камери, часто містять конфіденційну інформацію; користувачі можуть неохоче завантажувати їх копії на сервери розробника. Замість цього ми розглядаємо навчання такої моделі в умовах федеративного навчання, де кожен користувач безпечно підтримує приватну базу даних своїх подорожей на

своєму автомобілі і пристрої, а спільна глобальна модель навчається під координацією центрального сервера на основі високо оброблених, мінімально обмежених, ефемерні оновлення від користувачів.

Ці оновлення є великовимірними векторами, заснованими на інформації з приватної бази даних користувача. Навчання нейронної мережі зазвичай здійснюється шляхом багаторазового повторення цих оновлень з використанням варіанта міні-пакетного правила стохастичного градієнтного спуску.

Хоча кожне оновлення є ефемерним і містить не більше (і, як правило, значно менше) інформації, ніж приватна база даних користувача, користувач все ще може хвилюватися, яка інформація залишається. За деяких обставин можна дізнатися окремі місця, де був користувач, перевіривши останні оновлення цього користувача. Однак у федеративному навчанні, серверу не потрібен доступ до будь-яких оновлень окремого користувача, щоб виконати стохастичний градієнтний спуск; він вимагає лише поелементно зважених середніх векторів оновлення, взятих для випадкової підмножини користувачів. Використання протоколу Secure Aggregation для обчислення цих середньозважених значень гарантує, що сервер зможе дізнатися лише про те, що один або кілька користувачів у цій випадково вибраній підмножині відвідав те чи інше місце, але не про те, які користувачі.

Системи федеративного навчання стикаються з кількома практичними проблемами. Мобільні пристрої мають лише спорадичний доступ до живлення та підключення до мережі, тому набір пристроїв, які беруть участь у кожному етапі оновлення, непередбачуваний, і система має бути надійною для користувачів, які вибули. Оскільки нейронна мережа може бути параметризована мільйонами значень, оновлення можуть бути великими, що представляє прямі витрати для користувачів на тарифних планах мережі. Мобільні пристрої також, як правило, не можуть встановити прямі канали зв’язку з іншими мобільними пристроями (покладаючись на

сервер або постачальника послуг такого зв'язку), а також вони не можуть самостійно аутентифікувати інші пристрої.

Таким чином, федеративне навчання мотивує потребу в протоколі безпечної агрегації.

Аутентифіковане шифрування поєднує гарантії конфіденційності та цілісності повідомлень, якими обмінюються дві сторони. Він складається з алгоритму генерації ключа, який виводить приватний ключ, алгоритму шифрування $AE.enc$, який приймає на вхід ключ і повідомлення та виводить зашифрований текст, і алгоритму дешифрування $AE.dec$, який приймає як вхідний зашифрований текст та виводить оригінальний відкритий текст, або спеціальний символ помилки $\perp$. Для коректності ми вимагаємо, щоб для всіх ключів $c \in \{0,1\}^k$ і всіх повідомлень x, $AE.dec(c, AE.enc(c,x)) = x$. Для безпеки ми вимагаємо нерозрізненна під обраною атакою відкритого тексту (IND-CPA) і цілісності шифрованого тексту (IND-CTXT). Неофіційно гарантія полягає в тому, що для будь-якого противника M, якому надано шифрування повідомлень за своїм вибором за випадковим вибіркою ключа c (де c невідомий M), M не може розрізнити нові шифрування під c двох різних повідомлень, а також M створювати нові дійсні зашифровані тексти (відмінні від тих, які він отримав) щодо c з перевагою краще, ніж незначну.

Протокол спирається на стандартну схему безпечного підпису $UF-CMA$ $(SIG.gen, SIG.sign, SIG.ver)$.Алгоритм генерації ключа $SIG.gen(k) \rightarrow (d^{PK}, d^{SK})$ приймає як вхідний параметр безпеки та виводить секретний ключ d^{SK} та відкритий ключ d^{PK} ; алгоритм підпису $\mathrm{SIG.sign}(\mathrm{d^{SK}}, \mathrm{m}) \rightarrow \sigma$ приймає на вхід секретний ключ і повідомлення і виводить підпис; алгоритм перевірки $SIG.ver(d^{PK}, m, \sigma) \rightarrow \{0,1\}$ приймає на вхід відкритий ключ, повідомлення та підпис і повертає біт, що вказує, чи слід вважати підпис дійсним. Для правильності нам потрібно, щоб

$\forall_m, Pr[(d^{PK}, d^{SK})] \leftarrow SIG.gen(k), \sigma \leftarrow SIG.sing(d^{SK}, m) :$
$SIG.ver(\ d^{PK}, m,\ \sigma)\ =\ 1] = 1$

Згенерований відкритий ключ і доступ до Oracle, який створює підписи для довільних повідомлень, повинні мати можливість створити дійсний підпис для повідомлення, щодо якого Oracle був запитаний з більш ніж незначною ймовірністю.

Потрібно відмітити, що можливо зменшити комунікацію, якщо сторони домовляться про загальні початкові дані для генератора псевдовипадкових дій (PRG), а не про всю маску $s_{u,v}$. Ці спільні початкові дані будуть обчислюватися, якщо сторони передають відкриті ключі Діффі-Хеллмана та укладають ключову угоду.

Одним із підходів до роботи з користувачами, що вибули, було б сповістити тих користувачів, які вибули, і щоб кожен із них відповів загальним початковим числом, яке вони обчислили з вилученим користувачем. Цей підхід все ще має проблему: додаткові користувачі можуть вийти на етапі відновлення, перш ніж відповісти з початковими матеріалами, що, таким чином, вимагатиме додаткової фази відновлення для повідомлення про вихідні дані користувачів, які нещодавно викинули, і так далі, керуючи кількістю раундів. максимум до кількості користувачів.

Ця проблема вирішується використанням порогову схему спільного доступу до секрету та запропонувавши кожному користувачеві надсилати частки свого секрету Діффі-Хеллмана всім іншим користувачам. Це дозволяє відновити попарні початкові дані, навіть якщо під час відновлення випадуть додаткові сторони, доки деяка мінімальна кількість сторін (дорівнює порогу) залишається в живих і відповідає частками втрачених ключів користувачів.

Цей підхід вирішує проблему необмежених раундів відновлення, але все ще має проблему: існує ймовірність того, що дані користувача можуть випадково потрапити на сервер. Розглянемо сценарій, коли користувач u занадто повільно надсилає своє y_u на сервер. Сервер припускає, що

користувач відмовився, і просить всіх інших користувачів розкрити свої частки секретного ключа u, щоб видалити нескасовані маски u з z. Однак одразу після отримання цих спільних ресурсів та обчислення кожного із значень $s_{u,v}$. сервер може отримати від u. Тепер сервер може видаляти всі маски з y_u та вивчати x_u в чистому вигляді, порушуючи безпеку для u. Більше того, змагальний сервер в активній моделі може аналогічним чином дізнатися x_u , просто збрехавши про те, чи вибув користувач u.

Щоб вирішити подвійне маскування для захисту безпеки, було введено структуру подвійного маскування, яка захищає x_u , навіть якщо сервер може відновити маски u. На рисунку 3.3 переставлений опис протоколу на високу рівні.

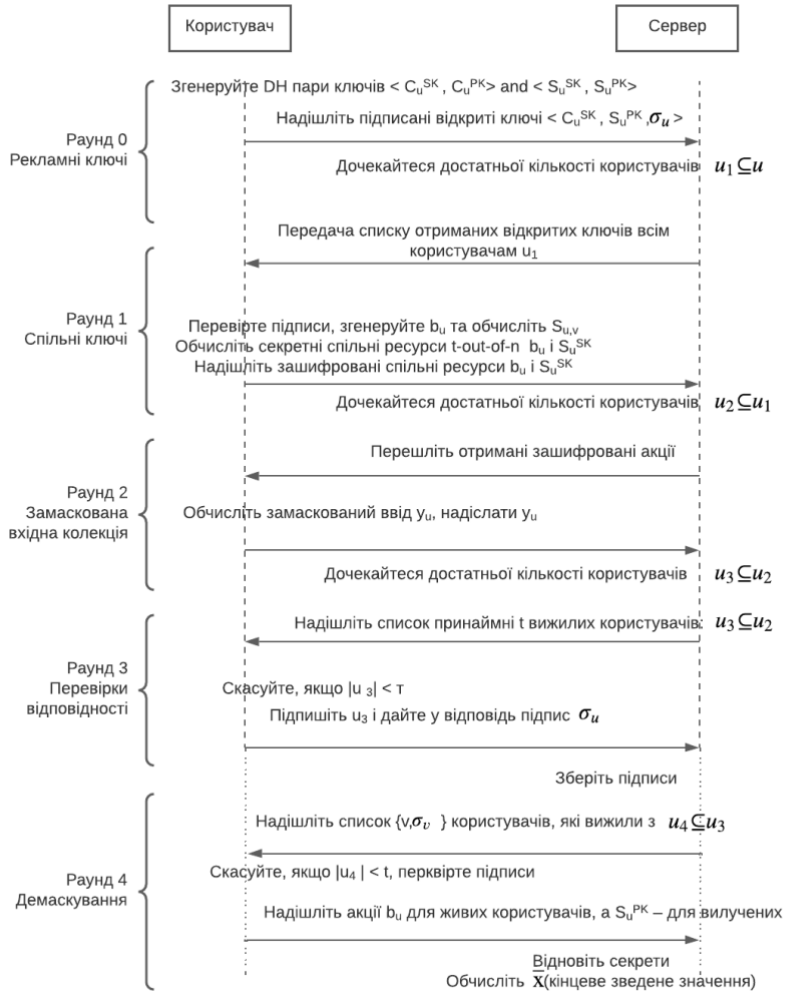


Рисунок 3.3 – Опис протоколу на високому рівні

По-перше, кожен користувач u відбирає додаткове випадкове початкове b_u протягом того самого раунду, коли генеруються значення $s_{u,v}$. Під час раунду секретного обміну користувач також створює та розподіляє частки b_u кожному з інших користувачів. Під час створення y_u користувачі також додають вторинну маску:

$$y_u = x_u + PRG(b_u) + \sum_{v \in U: u<v} PRG(s_{u,v}) - \sum_{v \in U: u>v} PRG(s_{u,v})$$

Під час раунду відновлення сервер повинен зробити явний вибір щодо кожного користувача u: від кожного члена, що вижив, v сервер може запитати або частку загального секрету $s_{u,v}$, пов'язаного з u, або частку b_u. for u; чесний користувач v ніколи не розкриє обидва види спільного доступу для одного користувача. Зібравши принаймні t часток $s_{u,v}$ для всіх вилучених користувачів і t часток b_u для всіх користувачів, що вижили, сервер може відняти маски, що залишилися, щоб виявити суму.

Протокол виконується (в синхронній мережі) між сервером і набором з n користувачів і складається з чотирьох раундів. Кожен користувач u має як вхідний вектор x_u (рівної довжини m), що складається з елементів з Z_R для деякого R. Сервер не має вхідних даних, але може спілкуватися з користувачами через безпечні (приватні та аутентифіковані) канали. У будь-який момент користувачі можуть вийти з протоколу (у цьому випадку вони повністю припиняють надсилання повідомлень), і сервер зможе виробляти правильний результат, доки t з них доживе до останнього раунду. Щоб спростити позначення, ми припускаємо, що кожному користувачеві u присвоюється унікальна «логічна ідентичність» (також позначена u у вигляді цілого числа від 1 до n, так що жодне з двох чесних користувачів не має однакового індексу.

Щоб підтвердити безпеку в моделі активного супротивника, також припускається існування інфраструктури відкритих ключів, яку для

простоти абстраговано, припускаючи, що всі клієнти отримують як вхід (від довіреної третьої сторони) відкриті ключі підпису для всіх інших клієнтів.

Загалом, протокол параметризовано за параметром безпеки k, який можна налаштувати, щоб обмежити ймовірність успіху будь-якого зловмисника. У всіх теоремах припущено, що кількість клієнтів n поліноміально обмежена в параметрі безпеки. Більше того, деякі з примітивів також вимагають додаткових глобальних параметрів.

Отже, було представлено практичний протокол для безпечного агрегування даних, гарантуючи, що введені дані клієнтів вивчаються сервером лише в сукупності. Накладні витрати такого протоколу дуже низькі, і він може витримувати велику кількість несправних пристроїв, що робить його ідеальним для мобільних додатків. Потрібен лише один постачальник послуг, що спрощує розгортання. Цей протокол має негайне застосування до реального федеративного навчання.

3.4 Алгоритм федеративного агрегування

Сучасні мобільні пристрої мають доступ до великої кількості даних, придатних для вивчення моделей, що, у свою чергу, може значно покращити роботу користувача на пристрої. Наприклад, мовні моделі можуть покращити розпізнавання мовлення та введення тексту, а моделі зображень можуть автоматично вибирати гарні фотографії. Однак ці багаті дані часто чутливі до конфіденційності, великі за кількістю або і те, і інше, що може перешкоджати реєстрації в центрі обробки даних і навчанню там із застосуванням звичайних підходів. Існує альтернатива, яка залишає навчальні дані розподіленими на мобільних пристроях і вивчає спільну модель шляхом агрегування локально обчислених оновлень. Цей децентралізований підхід називається федеративним навчанням.

Проблема оптимізації, яка присутня в федеративному навчанні відома як федеративна оптимізація, яка встановлює зв'язок (і контраст) з розподіленою оптимізацією.

Федеративна оптимізація має кілька ключових властивостей, які відрізняють її від типової задачі розподіленої оптимізації:

– тренінгові дані, надані клієнтам, які не належать до IID, як правило, засновані на використанні мобільного пристрою конкретним користувачем, і, отже, локальний набір даних будь-якого конкретного користувача не буде репрезентативним для розподілу населення;

– незбалансований. Так само деякі користувачі набагато важче використовують сервіс або додаток, ніж інші, що призводить до різної кількості локальних навчальних даних;

– широко розповсюджений. Очікується, що кількість клієнтів, які беруть участь в оптимізації, буде набагато більшою, ніж середня кількість прикладів на одного клієнта;

– обмежений зв'язок. Мобільні пристрої часто перебувають у автономному режимі або на повільних або дорогих з'єднаннях.

Останні численні успішні застосування глибокого навчання майже виключно покладалися на варіанти стохастичного градієнтного спуску (SGD) для оптимізації; насправді, багато досягнень можна розуміти як адаптацію структури моделі (і, отже, функції втрат), щоб вона була більш піддатною для оптимізації за допомогою простих методів, заснованих на градієнті. Таким чином, природно, що ми будуємо алгоритми для федеративної оптимізація, починаючи з SGD.

SGD можна застосувати до проблеми федеративної оптимізації, де за раунд зв’язку виконується один пакетний розрахунок градієнта (скажімо, для випадково вибраного клієнта). Цей підхід є обчислювально ефективним, але вимагає дуже великої кількості раундів навчання для створення хороших моделей.

У федеративних умовах залучення більшої кількості клієнтів не вимагає великих витрат, тому для базового сценарію буде використанно синхронну SGD великої партії; експерименти Chen et al. показують, що цей підхід є найсучаснішим у центрі обробки даних, де він перевершує асинхронні підходи. Щоб застосувати цей підхід у об'єднаних налаштуваннях, обирається C-частку клієнтів у кожному раунді та обчислюється градієнт втрати для всіх даних, які зберігаються у цих клієнтів. Таким чином, C керує глобальним розміром пакету, при цьому C = 1 відповідає повному (нестохастичному) градієнтному спуску. Цей базовий алгоритм має назву FederatedSGD (або FedSGD).

Типова реалізація FedSGD з C = 1 і фіксована швидкість навчання η має кожен клієнт k обчислити $g_k = \nabla F_k(w_t)$ середній градієнт за його локальними даними у поточній моделі w_t , і центральний сервер об'єднує ці градієнти і застосовує оновлення $w_{t+1} \leftarrow w_t - \eta \sum_{k=1}^{K} \frac{n_k}{n} g_k$, оскільки $\eta \sum_{k=1}^{K} \frac{n_k}{n} g_k = \nabla F_k(w_t)$. Еквівалентне оновлення дає $\forall k,\ w_{t+1}^k \leftarrow w_t - \eta g_k$ і потім $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$.

Тобто кожен клієнт локально виконує один крок градієнтного спуску на поточній моделі, використовуючи свої локальні дані, а потім сервер приймає середнє зважене значення результуючих моделей. Як тільки алгоритм буде написаний таким чином, можна додати більше обчислень до кожного клієнта, повторивши локальне оновлення $w^k \leftarrow w^k - \eta \nabla F_k(w^k)$ кілька разів перед кроком усереднення. Цей підхід називається FederatedAveraging (або FedAvg). Обсяг обчислень контролюється трьома ключовими параметрами: C – частка клієнтів, які виконують обчислення в кожному раунді; E, потім кількість проходів навчання, які кожен клієнт проводить у своєму локальному наборі даних у кожному раунді; і B, локальний розмір міні-пакету, який використовується для оновлень клієнта. Пишеться $B = \infty$, щоб вказати, що повний локальний набір даних розглядається як один мініпакет. Таким чином, в одній кінцевій точці цього

сімейства алгоритмів можна взяти $B = \infty$ і $E = 1$, що точно відповідає FedSGD. Для клієнта з n_k локальними прикладами кількість локальних оновлень за раунд визначається $u_k = E\frac{n_k}{B}$.

На рисунку 3.4 представлено втрата на повному навчальному наборі MNIST для моделей, згенерованих усередненням параметрів двох моделей w та w' за допомогою $\theta w + (1 - \theta) w'$ для 50 рівномірно розташованих значень $\theta \in$ [-0,2, 1,2]. Моделі w та w' навчалися за допомогою SGD на різних невеликих наборах даних. Для лівого графіка w та w'були ініціалізовані за допомогою різних випадкових початкових величин; для правильної ділянки використовували спільне насіння. Зверніть увагу на різні масштаби осі Y. Горизонтальна лінія дає найкращі втрати, досягнуті за допомогою w та w'. При спільній ініціалізації усереднення моделей призводить до значного зменшення втрат у загальному навчальному набору (набагато краще, ніж втрата будь-якої батьківської моделі).

Дотримуючись підходу Goodfellow, можна побачити саме таку погану поведінку, коли усереднюються дві моделі розпізнавання цифр MNIST3, навчені з різних початкових умов. Для цієї фігури кожна з батьківських моделей w та w' була навчена на неперекриваючих вибірках IID із 600 прикладів із навчального набору MNIST. Навчання проводилося через SGD з фіксованою швидкістю навчання 0,1 для 240 оновлень на міні-пакетах розміром 50 (або E = 20 проходів над міні-наборами даних розміром 600). Це приблизно той обсяг навчання, коли моделі починають переповнювати свої локальні набори даних. Останні роботи показують, що на практиці втрата поверхонь достатньо параметризовані NN добре поводяться і, зокрема, менш схильні до поганих локальних мінімумів ніж вважалося раніше. І справді, якщо запустити дві моделі з однієї випадкової ініціалізації та потім знову тренувати кожен окремо на іншій підмножині даних (як описано вище), можна побачити, що цей наївний параметр усереднення працює напрочуд добре (рисунок 3.4, праворуч).

Алгоритм федеративного усереднення представлено в лістингу 3.1, де K клієнтів індексуються за k; B – локальний розмір міні-пакету, E – кількість локальних епох, η– швидкість навчання. Сервер виконує:

Лістинг 3.1 – Алгоритм федеративного усереднення

Ініціалізує w_0

for кожного раунду t = 1,2 … **do**

$$m \leftarrow max\,(C * K, 1)$$

$S_t \leftarrow$ (випадковий набір із m клієнтів)

for кожного клієнту k $\in S_t$ паралельно **do**

$$w_{t+1}^k \leftarrow ClientUpdate\,(k, w_t)$$

$$w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$$

ClientUpdate (k, w) : // Виконати на клієнті k

$\beta \leftarrow$ (розділити P_k на партії розміру B)

for кожної локальної епохи i від 1 до E **do**

for партії b $\in \beta$ **do**

$$w \leftarrow w - \eta \nabla l(w;\, b)$$

return w до серверу

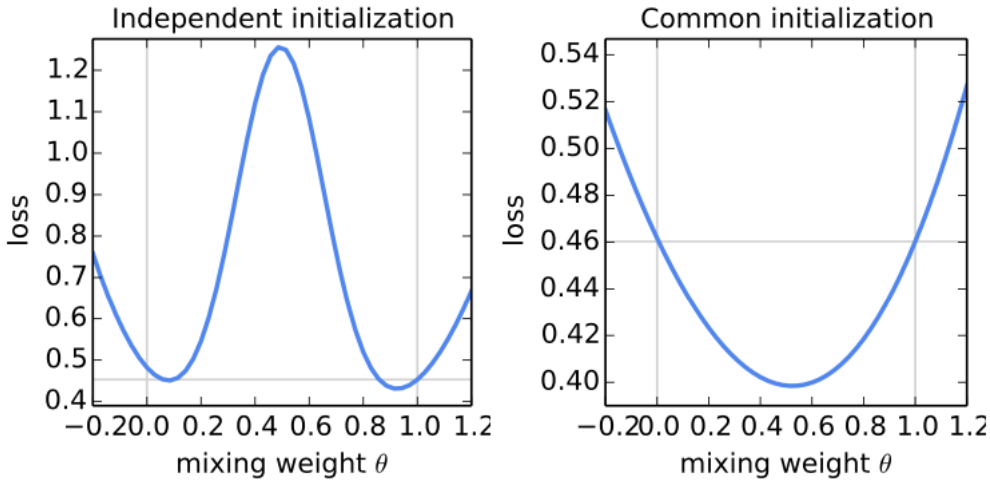


Рисунок 3.4 – Втрати на навчальному наборі MNIST

3.5 Оркестратор мікросервісних рішень

Мікросервісний стиль розробки додатка – це нова концепція розробки та розділу монолітнього додатка на менші сервіси для простоти управління змінами.

Мікросервісний підхід при якому єдиний додаток будується з набору менших сервісів, кожен з яких працює в особистому процесі та комунікує з іншими сервісами використовуючи легкі механізми, як правило HTTP або AMQ. Ці сервіси побудовані навколо потреб бізнесу і розгортаються незалежно з використанням повністю автоматизованого середовища. Самі по собі ці сервіси можуть бути написані на різних мовах програмування і використовувати різні технології. Перевагою мікросервісів над монолітними рішеннями:

– будь-які зміни, навіть, невеликі потребують перебірки та розгортання тільки одного сервісу;

– легке горизонтальне масштабування шляхом, спочатку, збільшенням кількості процесів одного і того же сервісу як одинці модулю, а потім збільшення обчислювальних можливостей машини на які виконується додаток.

Розбиття на сервіси відповідає бізнес сутностям, та сутностям архітектури.

Децентралізоване управління – це тенденція до стандартизації використовуваних платформ. Замість того, щоб використовувати набір визначених стандартів, написаних кимось, вони надають перевагу ідеї побудови корисних інструментів, які інші розробники можуть використовувати для вирішення подібних проблем. Ці інструменти як правило виокремлені з коду одного з проектів і розподілені між різними командами, іноді використовуючи при цьому модель внутрішнього відкритого коду. Тепер, коли git і github стали де-факто стандартною системою контролю версій, практики відкритого коду стають все більш і

більш популярними у внутрішніх проектах компаній. Децентралізоване управління даними постає в різному вигляді. У найбільш абстрактному сенсі це означає, що концептуальна модель побудови у різних систем буде відрізнятися. Це звичайна проблема, що виникає при інтеграції різних частин великих enterprise-додатків. Поняття «Клієнт сервісу» буде відрізнятися в різних командах. Одна команда може розробляти допоміжний сервіс, який буде спілкуватися тільки з іншими сервісами, а інша команда може розробляти відкритий для зовнішнього користування сервіс. Деякі атрибути «Клієнта» можуть бути присутніми в контексті одної команда і бути відсутнім в контекст іншої команди. Більш того, атрибути з однаковою назвою можуть мати різне значення. Ця проблема зустрічається не тільки у різних додатків, але також і в рамках єдиного додатка, особливо в тих випадках коли цей додаток розділений на окремі компоненти або сервіси. Цю проблему добре вирішує поняття Bounded Context з Domain-Driven Design (DDD). DDD пропонує ділити складну предметну область на кілька контекстів і відносини між ними. Цей процес корисний як для монолітної, так і для мікросервісній архітектури, але між сервісами і контекстами існує природний зв'язок, який допомагає прояснювати і підтримувати межі контекстів. Розподілені системи складні самі по собі, а управління сервісами на розподілених системах – одна з найскладніших проблем, з якими стикаються команди управління.

Kubernetes – програмне забезпечення для управління та контролю виконання та розгортування мікросервісних додатків. Ця відкрита система автоматичного розгортання та масштабування застосунків у контейнерах. Розроблена компанією Google. Kubernetes дотримується архітектури master/slave.

Azure Kubernetes Service – це платформа, яка керує контейнерними програмами та пов'язаними з ними мережевими та складськими компонентами. Основна увага приділяється розгортуванню сервісів, а не базовим компонентам інфраструктури. Kubernetes забезпечує

декларативний підхід до опису мікросервісів, підкріплений надійним набором API для здійснення операцій управління. Дозволяє створювати та запускати сучасні, портативні програми на основі мікросервісів, які користуються Kubernetes оркеструванням та керуванням наявністю цих компонентів-додатків. Як відкрита платформа, Kubernetes дозволяє створювати додатки за допомогою бажаної мови програмування, ОС, бібліотек або шини обміну повідомленнями. Існуючі інструменти безперервної інтеграції та безперервної постачання (CI / CD) можуть інтегруватися з Kubernetes для планування та розгортання релізів. Служба Azure Kubernetes (AKS) забезпечує керовану послугу Kubernetes, яка зменшує складність розгортання та основних завдань управління, включаючи координацію оновлень. Площиною управління AKS керує платформа Azure, оплата має місце лише за вузли AKS, які запускають програми. AKS побудований на вершині сервісного двигуна Azure Kubernetes з відкритим кодом (aks engine).

Таким чином, описаний сервіс хмарного провайдера, дозволяє перекласти відповідальність за управління мікросервісною архітектурою на інший інструмент, який на базі визначених правил, керує сервісами та дає гарантію, що сервіс буде стійким до збоїв, шляхом реплікації одиниць сервісу.

3.6 Проектування інтелектуальної складової

3.6.1 Комп’ютерне бачення

OpenCV (Open Source Computer Vision Library) – бібліотека комп’ютерного бачення та машинного навчання з відкритим кодом. OpenCV створено для забезпечення загальної інфраструктури для додатків комп’ютерного зору та для прискорення використання машинного

сприйняття в комерційних продуктах. Будучи ліцензованим продуктом BSD, OpenCV дозволяє компаніям легко використовувати та змінювати код.

OpenCV надає АПІ методи для швидкої реалізації алгоритму комп'ютерного зору. Оскільки завдання не стоїть у розробці ідеального автопілота, він буде досить стандартним. Загалом, кожен вид з камери можна розділити на кадри в середньому з частотою 24 кадри в секунду. Таким чином кожен кадр з камери повинен потрапити в алгоритм розпізнання та прийняття рішень. На рисунку 3.5 зображено типовий потік роботи на високому рівні абстракції. Карди подаються в двигун обробки OpenCV обробляючи і приймаючи рішення що перед камерою зображено і які рішення в контролі над автомобілем потрібно прийняти, двигун відсилає події в обробник подій який на підставі кожної події приймає рішення повернути вправо або вліво, збільшити швидкість і зупинятися.

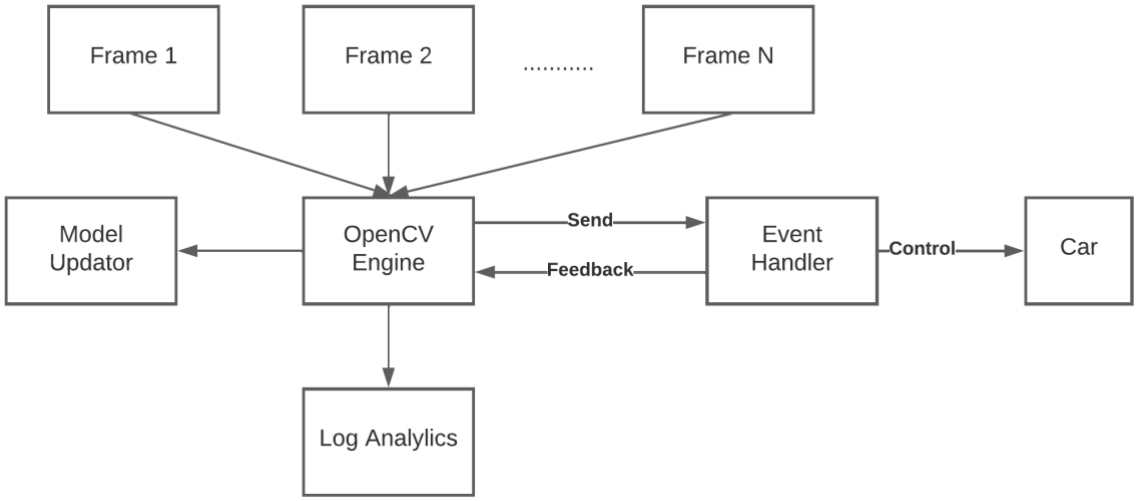


Рисунок 3.5 – Загальний потік роботи на високому рівні абстракції

Варто відзначити що для аналізів хронологічних записів і того, що відбувається, зокрема, чому було прийнято те або інше рішення на малюнку, додано агрегатор хронологічних записів для відстеження поведінки автомобіля.

Розглянувши двигун та алгоритм прийняття рішень можна зобразити діаграму (рисунок 3.6).

Обробник кадрів відповідає за первинне перетворення кадру, таке як додавання чи зменшення яскравості, кластеризація, побудова бітової мапи. Згенерований набір кадрів передається на маркування. Головне завдання цього кроку помітити всі об'єкти, які можуть бути виявлені на фотографії, включаючи машини та пішоходи.

З маркування будуватися тривимірний світ прийняття рішень навігації в тривимірному просторі. Побудови такого 3 мірного простору необхідно для утримання в пам'яті реального становища і змін його після обробки кожного кадру.

Система збору логів простий лог агрегатор з можливість сортування та вибірки логів за період часу у хронологічному порядку.

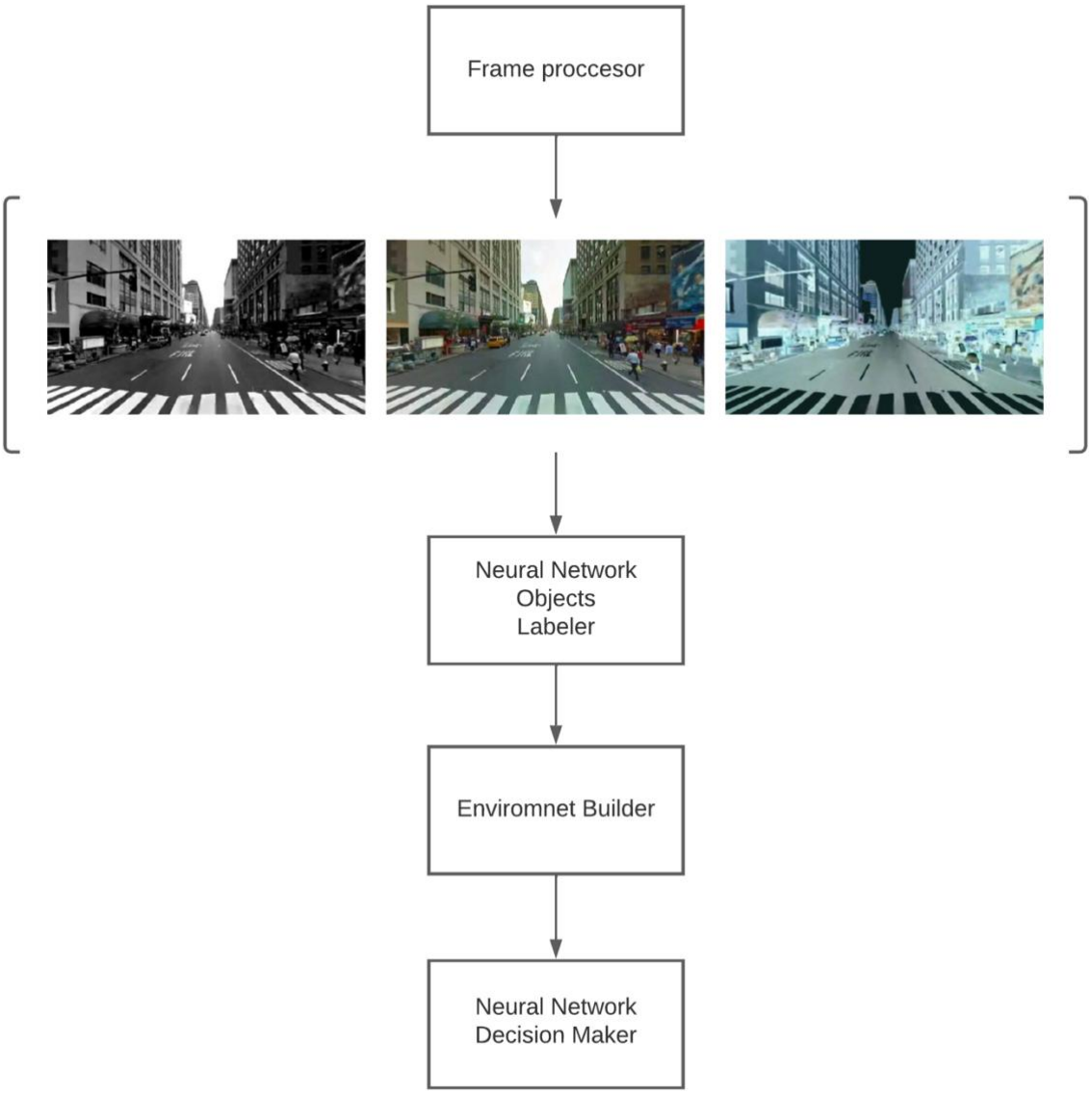


Рисунок 3.6 – OpenCV engine потік

4 РОЗРОБКА ДОДАТКА

4.1 Розробка інфраструктури додатка

Для зручного доступу до функціональності додатка і його інтелектуальної частини, необхідно реалізувати Web-додаток. Для розробки серверної частини було вирішено використовувати ASP.NET Core, та мову програмування C#. Даний фреймворк було вибрано, по ряду причин. Ефективність запуску програм реалізованих мовою програмування під .NET знаходиться на високому рівні. Можна стверджувати, що це фреймворк має найкращу ефективність виконання програмного забезпечення написаних на керованих мовах програмування. Варто зазначити, що часи, коли розробляти програми використовуючи .NET технології можливо було тільки на платформі Windows, пройшли. Зараз .NET Core – це кросс платформне рішення, для побудови прогресивного програмного забезпечення. Додатковою привабливою стороною вибраного фреймворку – є його спеціалізованість в розробці додатків для хмарних технологій Microsoft Azure. Обидва продукти розроблені в стінах Microsoft.

Для реалізації та управління інфраструктурою мікросервісів було вибрано Kebernetes. Kubernetes надає декларативний підхід для опису сервісів. Такий підхід дає змогу описувати сервіси та ресурси у вигляді yaml файлів. Конфігурація мережевої комунікаціях між сервісами, збереження секретів спрощує розробку сервісів. Можливість інтеграції з сервісами збереження образів в сховищі, для легкого розповсюдження між розробниками команди, та введення нових змін в існуючий кластер.

Балансир вхідних запитів відповідає за рівномірне розподілення запитів між віртуальними машинам, що виконують програмне забезпечення на першому рівні. Replica-Set – це набір віртуальних машин які виконують одну версію Web додатка. Збільшення кількості віртуальних машин – називається горизонтальним масштабуванням. Кожен сервіс реалізує

бізнес-логіку додатка. Далі буде говоритися про Web.BusinessLogic сервіс, як про єдиний. Цей сервіс має змогу комунікувати з сервісами другого рівня. Виконуюче середовище Kubernetes Service контролю життєвий цикл кожного сервісу. Якщо виникають помилки під час виконання, сервіс буде намагатися створити нові репліки існуючого сервісу, який потерпів провалу.

Вузол в розумінні Kubernetes – це контрольований набір віртуальних машин, які виконують програмне забезпечення написане для реалізації бізнес логіки.

Третій рівень хмарного додатка, це рівень сховищ даних. Сховище даних необхідно надавати надійні сервіси для збереження і швидкого доступу до інформації. Так як, було вибрано сервіс хмарного провайдера Cosmos Db, то комунікація з сервісами відбувається за допомогою HTTPS. Вимоги до сховища даних є типовим, швидкість доступу. Так як, важливість ефективності запису даних для нас не грає ролі, тому що, збереження словників буде відбуватися до запуску додатка в готовому режимі. Проте запис та покращення повинне бути в ході роботи додатка, і нові збережені дані про нові слова в словнику не є важливою функціональність. Такі дані можливо назвати холодними (Cold), швидкість доступу до яких є важливішою ніж збереження.

Гео-реплікація даних робить доступними данні до кінцевого користувача швидше ніж інші системи сховища даних. Так як первинний ринок роботи додатка – Європа, то сервіс гео-реплікації можливо налаштувати тільки на цей регіон, що спочатку знизить витрати на оплату сервісів.

Архітектурне рішення наслідує останні тенденції розробки мікросервісних додатків. Вибраних архітектурний стиль, надає величезну кількість переваг від традиційного, багаторівневого підходу, проте це збільшує складність реалізації і робить майже неможливим ACID операції в розподілених системах. Традиційна ера розгортання на рисунку 4.1.

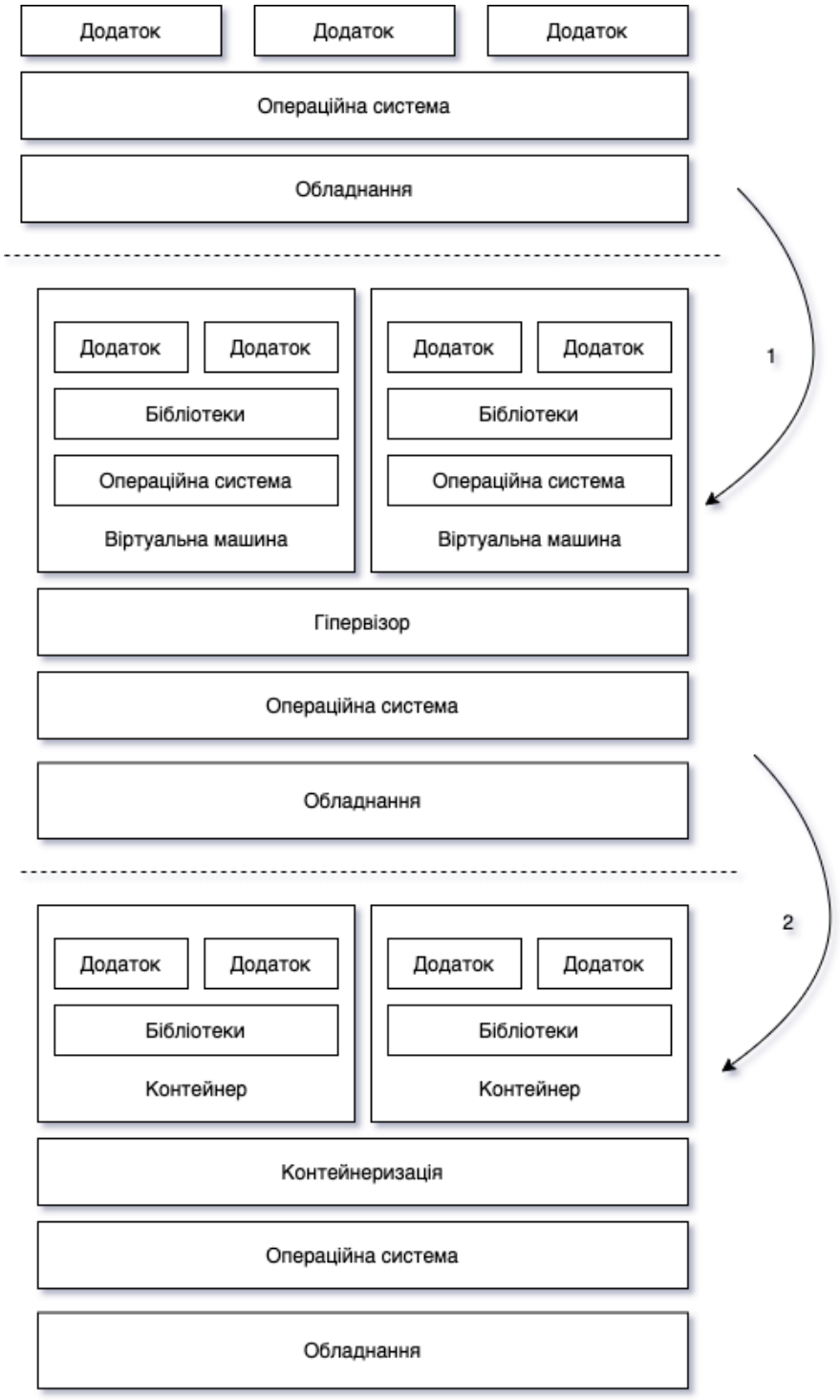


Рисунок 4.1 – Схема міграції підходів

Раніше організації запускали додатки на фізичних серверах. Не було ніякого способу визначити межі ресурсів для додатків на фізичному сервері, і це викликало проблеми з розподілом ресурсів. Наприклад, якщо кілька

додатків виконуються на фізичному сервері, можуть бути випадки, коли один додаток буде займати більшу частину ресурсів, і в результаті чого інші додатки будуть працювати гірше. Рішенням цього було запустити кожен додаток на іншому фізичному сервері. Але це неможливо масштабувати, оскільки ресурси використовувалися не повністю, через що організаціям було дорого підтримувати безліч фізичних серверів. Ера віртуального розгортання: Для вирішення була представлена віртуалізація. Вона дозволила запускати кілька віртуальних машин (ВМ) на одному фізичному сервері. Віртуалізація ізолює додатки між віртуальними машинами і забезпечує певний рівень безпеки, оскільки інформація одного додатка не може бути вільно доступна іншому додатка. Віртуалізація дозволяє краще використовувати ресурси на фізичному сервері і забезпечує кращу масштабованість, оскільки додаток можна легко додати або оновити, крім цього знижуються витрати на обладнання і багато іншого. За допомогою віртуалізації можна перетворити набір фізичних ресурсів в кластер одноразових віртуальних машин. Кожна віртуальна машина являє собою повноцінну машину, на якій виконуються всі компоненти, включаючи власну операційну систему, поверх віртуалізованого обладнання. Контейнери схожі на віртуальні машини, але у них є властивості ізоляції для спільного використання операційної системи (ОС) між додатками. Тому контейнери вважаються легкими. Подібно віртуальній машині, контейнер має свою власну файлову систему, процесор, пам'ять, простір процесу і багато іншого. Оскільки вони не пов'язані з базовою інфраструктурою, вони переносяться між хмарами і збірками ОС.

Декларативний підхід до опису сервісів вносить спрощення до побудови мікросервісів.

Додаток можливо розділити на декілька рівнів. Доступ з навколишнього середовища в кінцевого користувача до додатка можливо з використанням настільних, мобільних та планшетних пристроїв. Клієнтські рішення може спілкуватися тільки з сервісом першого рівня.

4.3 Розробка інтелектуальної частини додатка

4.3.1 Розробка алгоритму обробки потоку

Використовується OpenCV SDK для маніпуляції кадрами та виконання маркування об’єктів базуючись на нейронній мережі відповідно для кожного типу. Для спрощення було обрано два типи об’єктів, пішохід та інший автомобіль. Більшість монотонної логіки реалізовано в бібліотеках, тому використання цього SDK спростить та зменшить кількість коду, для обробки зображень. Бібліотеки розроблені за допомогою мови програмування C++, оскільки основні сервіси розроблюються на базі мови C# то для виклику методів SDK було використано COM API інтерфейс. Компонентна об'єктна модель (COM) – стандарт бінарного інтерфейсу для компонентів програмного забезпечення, представлений Microsoft в 1993 році. Він використовується для забезпечення можливості створення міжпроцесорних комунікаційних об'єктів у великому діапазоні мов програмування. COM є основою для декількох інших технологій та рамок Microsoft, включаючи OLE, OLE Automation, Browser Helper Object, ActiveX, COM +, DCOM, оболонку Windows, DirectX, UMDF та Windows Runtime.

OpenCV – надає високорівневий інтерфейс для реалізації логіки обробка програмного забезпечення з розпізнавання об’єктів на зображенні. Команди SAS поєднуються в програмі з командами Windows API. Якщо можливо ідентифікувати команду, структуру або повідомлення які належать до API Win, вони, ймовірно, належать до SAS або стосуються того, що потрібно для обробки зображення. Наприклад, повідомлення WM_RECOEVENT, яке визначено в додатка. Це не стосується SAS, але це повідомлення потрібне для роботи додатку. Також є кілька шаблонів одного типу, визначених у програмі (Point (), Scalar (), imshow() та drawDescription()). Всередині цих функцій, визначених програмою,

називаються методи SAS. Також у верхній частині файлу є кілька #define, які використовуються для включення спеціальних заголовків SAS 5.

Для початку будь-якого кодування необхідно імпортувати заголовні файли з SAS:

```
#include "opencv2/imgproc.hpp"
#include "opencv2/highgui.hpp"
```

imgproc.h – містить глобальні змінні та функції для реалізаці основного функціонала з роботи з розпізнавання і обробки картинки. highhgui.h – містить допоміжні інструментальні функції для відображення інформації на екрані. Він містить список функцій, які об'єднують по кілька SAS-методів в один виклик. Використовувати допоміжні функції не обов'язково, але якщо все ж використовувати, процес кодування може бути значно спрощений. У цьому прикладі використовуються допоміжні функції там, де це тільки можливо.

Для початку роботи з SDK необхідно ініціалізувати бібліотеки. Перш за все, так як використовується COM, то необхідно впевнитись, що вся COM інформація завантажена та доступна до виклику. Використовуємо COM-команди CoInitialize() і, пізніше, CoUninitialize(). Щоб гарантувати активність COM протягом всього часу роботи програми, ці команди повинні знаходитися відповідно до і після головного циклу обробки повідомлень. У лістингу 4.1 представлено код обробки циклу.

Лістинг 4.1 – Обробка циклу

```
if(SUCCEEDED(CoInitialize(NULL)))
{
while(GetMessage(&msg,NULL,0,0))
{
}
CoUninitialize();
}
```

По-друге, сам об’єкт. Він забезпечує доступ до реалізованої бібліотеки. У лістингу 4.2 представлено код ініціалізації контексту.

Лістинг 4.2 – Ініціалізація контексту

```
CComPtr g_cpEngine;
hr=g_cpEngine.CoCreateInstance(CLSID_SpSharedRecognizer);
if(FAILED(hr)) exit(1);
```

Об'єкт створений. Існують дві опції, щоб визначити цей об'єкт як розділений або InProc-об'єкт. Використовуються такі глобальні ідентифікатори класів, щоб встановити об'єкт: CLSID_SpSharedRecoContext – створюється об'єкт ресурсу, CLSID_SpInprocRecognizer – створюється неподільні об'єкти типу InProc. Об'єкт дозволяє ресурсам, таким як двигун розпізнавача, і пристрої виведення, використовувати кілька різних програм одночасно. Це кращий варіант для більшості програм. Зазвичай настільні системи мають тільки одну камеруі, використовуючи колективні об'єкти, різні додатки. Це програма, яка запускається у фоновому режимі. Вона доставляє події в власний додаток.

Об'єкт InProc, навпаки, дозволяє одному єдиному додатком контролювати ресурс. Це відноситься до мікрофонів і двигуна розпізнавання. Використання InProc-процедури дуже обмежує можливості роботи і допустимо тільки в особливих обставинах. Наприклад, ви можете поступити так, якщо Вам потрібна камера для відображення даних тільки одним додатком. Тож рішення синхронізації потоків цей підхід вирішує.

Наступний необхідним кроком – є створення контексту розпізнавання. У лістингу 4.3 представлений код використання контексту розпізнавання.

Контекст – це одиничне поняття програми, необхідне для розпізнавання картинки. У найпростішому випадку всьому додатком зіставляється тільки один контекст розпізнавання.

Лістинг 4.3 – Використання контексту розпізнавання

```
CComPtr g_cpRecoCtxt;
hr = g_cpEngine->CreateRecoContext(&g_cpRecoCtxt );
if ( FAILED( hr ) ) //Leave application
```

Не має значення, де саме користувач перебуває в додатка, всі об’єкти події і повідомлення обробляються однією і тією ж процедурою. Однак кожна частина програми може мати свій власний контекст. Окремі вікна, діалогові панелі, панелі меню або навіть окремі пункти меню (такі як «Відкрити» або «Друк») можуть мати свій власний контекст. Події та повідомлення, генеровані з таких просторів обробляються своїми власними процедурами. Це схоже на те, як окремі вікна обробляють події повідомлення в стандартних додатках Win API. Кожному вікна зіставляється віконна процедура, яка і обробляє події і повідомлення. Точно так же кожному контексту розпізнаванню зіставляється окрема процедура. В цьому випадку у більший контроль над програмою і обробкою картинок. Контексти створюються динамічно в той момент, коли вони потрібні і знищуються, коли необхідність зникає. Однак Ви можете створити один контекст і зберігати його протягом всієї роботи програми. Але наш перший приклад матиме тільки один контекст.

Середовище ініціалізоване, наступник кроком – є завантаження моделі навчання нейронної мережі. Доступ до сховища моделі нейронної мережі представлено в лістингу 4.4.

Лістинг 4.4 – Доступ до сховища моделі нейронної мережі

```
hr=g_cpRecoCtxt->CreateGrammar(NN,&g_cpCmdNN);
if(FAILED(hr)) exit(1);

hr = g_cpCmdNN->LoadCmdFromResource(
NULL,
MAKEINTRESOURCEW(IDR_CMD_CFG),
```

```
L"Nerual Netword",
```

Продовження лістингу 4.4

```
MAKELANGI
const utility::string_t
storage_connection_string(U("UseDevelopmentStorage=true;")
);D( LANG_NEUTRAL, SUBLANG_NEUTRAL), TRUE);
string.
azure::storage::cloud_storage_account storage_account =
azure::storage::cloud_storage_account::parse(storage_conne
ction_string);
azure::storage::table_query_iterator end_of_results;
if ( FAILED( hr ) ) exit(1);
```

Список, використаний в коді, попередньо створюється і вдає із себе зовнішній стосовно програми ресурс, що містить набір команд. Використовується формат JSON. Використовується бібліотека для доступу та запиту даних з постійного сховища даних. Два перші параметра NULL означають, що ніякі правила не будуть включені або активовані.

Обробка кадра відбувається у фоновому режимі. Коли у SAS є інформація, він повертає її в додаток. SAS сповістить програму, коли подія відбудеться. Подія – це умова спеціального інтересу для SAS. Прикладами подій можуть бути знаходження пішохода перед автомобілем (WA_PEDESTRIANT_WARN), коли його нема (WA_ROAD_CLEAR) або успішне прийняття рішення (WA_TRUN_RIGHT_30). SAS має кілька типів подій, перерахованих в типі SPEVENTENUM, що містить повний список.

Повідомлення показує, що подія SAS сталося і додаток може на нього реагувати. Щоб реагувати на повідомлення, додаток повинен зв'язати з ним окрему процедуру. Є кілька способів зробити це. Інтефейс ISpNotifySource має чотири методи: SetNotifyCallbackFunction, SetNotifyCallbackInterface, SetNotifyWin32Event і SetNotifyWindowMessage. Ще є ISpNotifySink :: Notify, що представляє загальний метод, який дозволяє спеціальні та

незвичайні умови. Можливо використовувати деякі або всі з цих методів в залежності від своїх потреб. Наприклад, може бути легше в додатка керувати повідомленнями прямим викликом функції (:: SetNotifyCallbackFunction), наприклад в разі виведення нового діалогового вікна або автоматичного запису діяльності в файл-журнал. Три перших метода (:: SetNotifyCallbackFunction, :: SetNotifyCallbackInterface і :: SetNotifyWindowMessage) вимагають циклу обробки повідомлень і тому можуть використовуватися тільки в Windows-додатках. Ви можете використовувати інші два (:: SetNotifyWin32Event і :: ISpNotifySink :: Notify) без циклу обробки повідомлень.

Оскільки було встановлення зображення тільки до одної події – SPEI_RECOGNITION. Тільки подія успішного розпізнання генерує повідомлення. SAS не буде сповіщати програму про будь-які інші події.

Встановлюються два значення, перший параметр містить список всіх зацікавлених обробників. Він визначає всі можливі види подій, які можуть зацікавити. Другий параметр містить список подій, який повинен бути поставлений в чергу, так щоб додаток міг їх обробити в належний момент часу. В даному випадку, цікавить кожна поява SPEI_RECOGNITION, навіть якщо вони приходять так швидко, що програма не може їх обробити за раз. Часто ці два параметри ідентичні, але, іноді це не так. SPFEI () – це допоміжна функція, яка використовується, щоб перетворити перелічуваних події в число типу ULONGLONG.

Програма може звернутися до SAS, отримати зображення з камери, спробувати розпізнати її і повернути подію в разі вдалого розпізнання об’єкту. В циклі обробки подій необхідно визначити обробник подій, для того, щоб додаток опрацював подію. Наступний фрагмент коду, що представлений в лістингу 4.5 міститься в головному циклі.

Лістинг 4.5 – Обробка події в головному циклі

```
case WM_RECOEVENT:
ProcessRecoEvent (hWnd);
```

```
break;
```

Потрібно три речі, щоб повністю описати подію. Перша – це CSpEvent. Це допоміжний клас, який містить кілька корисних функцій і структуру SPEVENT. Один метод :: GetFrom робить дві речі одночасно. Він витягує таку подію з черги і завантажує відповідну інформацію в структуру SPEVENT, роблячи її готовою для обробки. Друга необхідна річ – визначити, які SAPI-події справді мають місце. Для цієї мети достатньо всього лише визначити атрибут eEventId. Якщо це подія, яка не цікавить обробник, то він ігнорує її і продовжує перевіряти інші повідомлення. Остання важлива річ – використання події для своїх потреб. У цьому прикладі цікавить тільки SPEI_RECOGNITION. Якщо воно сталося , а оператор switch зробить все інше. У лістингу 4.6 представлено цикл обробки події.

Лістинг 4.6 – Цикл обробки події

```
void ExecuteCommand(ISpPhrase *pPhrase, HWND hWnd)
{
SPPHRASE *pElements;
if (SUCCEEDED(pDecision->GetHandler(&pElements))){
switch ( pElements->Rule.ulId ){
case VID_Navigation:
{
switch( pElements->pProperties->vValue.ulVal )
{
case VID_Counter:
PostMessage( hWnd, WM_GOTOCOUNTER, NULL, NULL );
break;
}
}
break;
}
::CoTaskMemFree(pElements);
```

```
}
}
```

Команди управління очевидно короткочасні. Вони обмежені в використанні кадрів. Це те, що роблять традиційні програми розпізнавання об’єктів. Щоб використовувати двигун розпізнавання треба замінити модель команд і управління на існуючу. Замість побудованого на звичайній моделі, база об’єктів використовує набагато більшу кількість об’єктів і визначає кожне нове розпізнання, ґрунтуючись на контексті. Розпізнані об’єкти негайно до і після обробки запам'ятовуються і база вибирає найбільш кращий варіант вихідного виразу.

4.3.2 Розробка інтелектуальної моделі в хмарному середовищі

Хмарний провайдер реалізовує сервіс Azure Machine Learning. Microsoft розбиває Azure Machine Learning Service на три великі стадії: підготовка даних, сам експеримент зі створенням моделі, розгортання.

Сервіс розгортується як Web-сервіс, доступний за протоколом HTTPs. Тож, розроблений алгоритм обробки клієнтських подій буде звертатися до сервісу машинного навчання агрегування і усереднення оновлення моделі.

Azure Machine Learning Service дозволяє робити вставки з коду мов програмування для реалізації додаткової логіки, яку він не підтримує сам. Тож, мовою програмування R було реалізовано алгоритм Secure Aggregation.

Розгорнутий сервіс в кожному пристрої для оновлення моделі звертається до сервісу машинного навчання кожен раз коли потрібно оновоити та трансформувати модель.

4.4 Розробка алгоритму керування автомобіля

Використовуючи методи бібліотеки OpenCV та спеціальні алгоритми та обчислення математичного аналізу можна будувати. І запам'ятовувати оточення автомобіля у тривимірному просторі у радіусі 10-20 метрів навколо автомобіля. Об'єкти розпізнані перетворюватися на координати в тривимірному просторі і відображаються на оточенні. Це дає змогу відстежувати динаміку зміни окремих об'єктів у просторі та пронозувати їх зміни.

Загалом не потрібно генерувати та відображати цей світ, для цього потрібна лише математична модель спостереження за ним та застосування всіх подійю. У лістингу 4.7 представлено обчислення приблизно копланарної точки.

Лістинг 4.7 – Обчислення приблизно копланарної точки

```
    Size patternSize(parser.get<int>("width"),
parser.get<int>("height"));
    float squareSize = (float)
parser.get<double>("square_size");
    poseEstimationFromCoplanarPoints(parser.get<String>("image
"),
    parser.get<String>("intrinsics"),
    patternSize, squareSize);
```

4.4.1 Комунікація пристрою з машиною

Сучасні транспортні засоби спілкуються за допомогою CAN-повідомлень, протоколу трансляції, який дозволяє багатьом комп’ютерам спілкуватися разом у спосіб, прийнятний до шумного середовища. З точки зору машинного бачення, перевага протоколу CAN полягає в тому, що він за своєю суттю довірливий, що дозволяє підробляти повідомлення. Погана річ у протоколі CAN полягає в тому, що кожен виробник створює свій власний словник ідентифікаторів повідомлень CAN і визначень даних.

Наприклад, один виробник може вказати кут повороту для повідомлення ID 0x30 у байтах 3 і 4, тоді як інший виробник описує кут повороту для повідомлення ID 0xe4 у байтах 2 і 3.

Тож навіть після того, як автопілот визначить, яким має бути прискорення та кут повороту, потрібно ще багато роботи, щоб перетворити це на повідомлення CAN, яке буде зрозуміло даній марці/моделі автомобіля.

У лістингу 4.8 наведено код для перетворення обчислень CC, у специфічні для виробника повідомлення CAN за допомогою інтерфейсу автомобіля CI, який містить інформацію про марку/модель автомобіля.

Лістинг 4.8 – Перетворення обчислень CC у повідомлення CAN

```
# selfdrive/controls/controlsd.py

# send car controls over can
can_sends = CI.apply(CC)
pm.send('sendcan', can_list_to_can_capnp(can_sends))
```

Поки було розглянуто механізми транспортування даних у системі. Далі спробуємо зрозуміти інтерфейси, необхідні для підтримки автономного керування сотнями різних автомобілів. Для цього давайте почнемо з визначення, що таке відбиток пальця.

Відбиток даних – це словник ідентифікаторів повідомлень CAN і довжини даних (у байтах). Відбиток пальця використовується для ідентифікації автомобіля на основі набору повідомлень CAN, надісланих протягом кількох секунд, ігноруючи вміст повідомлень.

Тепер припустимо, що у Всесвіті є лише два автомобілі, XTRAIL і LEAF. І припустимо, що кожен автомобіль може мати два різні можливі відбитки пальців, залежно від деяких виробничих змін (наприклад, модельного року). Дві машини з двома відбитками пальців дають нам чотири можливі відбитки пальців, приклад представлений у лістингу 4.9.

Лістинг 4.9 – Можливі відбитки пальців машин

```
 # selfdrive/car/nissan/values.py
fingerprints = {
```

Продовження лістингу 4.9

```
    CAR.XTRAIL: [
    {
      2: 5, 42: 6, 346: 6, 347: 5, 348: 8, 349: 7, 361: 8,
386: 8, 389: 8, 397: 8, 398: 8, 403: 8, 520: 2, 523: 6,
548: 8, 645: 8, 658: 8, 665: 8, 666: 8, 674: 2, 682: 8,
683: 8, 689: 8, 723: 8, 758: 3, 768: 2, 783: 3, 851: 8,
855: 8, 1041: 8, 1055: 2, 1104: 4, 1105: 6, 1107: 4, 1108:
8, 1111: 4, 1227: 8, 1228: 8, 1247: 4, 1266: 8, 1273: 7,
1342: 1, 1376: 6, 1401: 8, 1474: 2, 1497: 3, 1821: 8,
1823: 8, 1837: 8, 2015: 8, 2016: 8, 2024: 8
    },
    # 2020 Leaf SV Plus
    {
      2: 5, 42: 8, 264: 3, 361: 8, 372: 8, 384: 8, 389: 8,
403: 8, 459: 7, 460: 4, 470: 8, 520: 1, 569: 8, 581: 8,
634: 7, 640: 8, 643: 5, 644: 8, 645: 8, 646: 5, 658: 8,
682: 8, 683: 8, 689: 8, 724: 6, 758: 3, 761: 2, 772: 8,
773: 6, 774: 7, 775: 8, 776: 6, 777: 7, 778: 6, 783: 3,
852: 8, 853: 8, 856: 8, 861: 8, 943: 8, 944: 1, 976: 6,
1008: 7, 1009: 8, 1010: 8, 1011: 7, 1012: 8, 1013: 8,
1019: 8, 1020: 8, 1021: 8, 1022: 8, 1057: 3, 1227: 8,
1228: 8, 1261: 5, 1342: 1, 1354: 8, 1361: 8, 1402: 8,
1459: 8, 1477: 8, 1497: 3, 1549: 8, 1573: 6, 1821: 8,
1837: 8
    },
  ],
}
```

Значення відбитків пальців – це набір пар ключ/значення, де ключ – це ідентифікатор повідомлення, а значення – довжина даних. Кожного разу,

коли запускається система, невідомо, який у нас автомобіль. Тож починаємо прослуховувати повідомлення CAN з автомобіля.

Для прикладу припустимо, що отримується ідентифікатор повідомлення 2 довжиною 5 байт. Враховуючи наведену вище інформацію, будь-який з чотирьох відбитків пальців може бути дійсним. Тепер припустимо, що отримується повідомлення ID 264 із довжиною 3. Тепер ми можемо видалити обидва відбитки пальців із CAR.XTRAIL, оскільки жоден із відбитків пальців не містить повідомлення ID 264. Аналогічно, якщо ми отримаємо повідомлення 42 із довжиною 8, ми можемо видалити перший CAR .LEAF відбиток пальця. Якщо після прослуховування ще багатьох повідомлень другий відбиток пальця CAR.LEAF таким чином не був усунутий, можна зробити висновок, що автомобіль – Nissan Leaf.

Цей висновок був використаний, щоб завантажити правильні словники CAN (DBC), важливу інформацію про геометрію та набір функцій автомобіля, а також функції для читання та запису повідомлень CAN, що стосуються марки/моделі.

Зчитування шини CAN здійснюється через `selfdrive/car/<make>/carstate`, а запис на шину CAN здійснюється через `selfdrive/car/<make>/carcontroller`.

Кожна марка автомобіля містить спеціальний виробник, `CarInterface`, `CarState` і `CarController`, які можна знайти в каталозі `selfdrive/car/<make>`. За потреби ці класи забезпечують диференціацію між моделлю автомобіля.

4.4.2 CARLA Simulation програмне забезпечення

CARLA – це симулятор автономного водіння з відкритим кодом. Він був створений з нуля, щоб служити модульним і гнучким API для вирішення низки завдань, пов’язаних із проблемою автономного водіння. Однією з головних цілей CARLA є допомога в демократизації досліджень і розробок

автономного водіння, слугуючи інструментом, до якого користувачі можуть легко отримати доступ і налаштувати. Для цього симулятор повинен відповідати вимогам різних випадків використання в рамках загальної проблеми водіння (наприклад, вивчення правил водіння, алгоритми навчання сприйняття тощо). CARLA базується на Unreal Engine для запуску симуляції та використовує стандарт OpenDRIVE (1.4 як сьогодні) для визначення доріг та міських умов. Контроль над моделюванням надається через API, який обробляється на Python і C++, який постійно розвивається разом із проектом.

Щоб згладити процес розробки, навчання та перевірки систем водіння, CARLA перетворилася на екосистему проектів, побудовану спільнотою навколо основної платформи. У цьому контексті важливо зрозуміти деякі речі про те, як працює CARLA, щоб повністю зрозуміти її можливості.

Симулятор CARLA складається з масштабованої архітектури клієнт-сервер.

Сервер відповідає за все, що стосується самої симуляції: сенсорний рендеринг, обчислення фізики, оновлення світу-держави та його акторів та багато іншого. Оскільки він прагне до реалістичних результатів, найкращим буде запуск сервера з виділеним графічним процесором, особливо коли це стосується машинного навчання.

Клієнтська сторона складається із суми клієнтських модулів, які контролюють логіку акторів на сцені та встановлюють умови світу. Це досягається завдяки використанню API CARLA (на Python або C++), шару, який є посередником між сервером і клієнтом, який постійно розвивається, щоб надати нові функції. На рисунку 4.2 зображено приклад симуляції автоводіння.

Рисунок 4.2 – Приклад симуляції автоводіння

ВИСНОВКИ

У результаті виконання роботи було розроблено архітектуру додатка, який призначений для роботи безвідмовно під великим навантаженням. Спроектовано систему комунікації в хмарному середовищі з підходом до розробки мікросервісного додатка.

У ході роботи було розроблено програмне забезпечення для збагачення знань нейронної мережі, розпізнавання та маркування об’єктів зображених на кадрі. Крім цього, побудовано та реалізована інфраструктурна частина додатка в хмарному середовищі.

Було проаналізовано бібліотеки та фреймворки які можуть вирішувати завдання поставлені перед програмним забезпеченням додатка.

Проведений огляд сервісів хмарного провайдера та моделей оплати за користування сервісами.

Вивчено алгоритм побудови нейронної мережі. Розроблений алгоритм було інтегровано в хмарне середовище, за допомогою сервісів хмарного провайдера.

В атестаційні роботі було розроблено напівфінальний веб-додаток за допомогою технологій та фреймворків .NET Core та C++.

Встановлені правила комунікації сервісів в мікросервісному підходи та конфігурація кожного сервісу в підконтрольній Kubernetes гіпервізору, для коректного маніпулювання виконуваними сервісами.

Розроблений додаток, дозволяє виконувати задачі, які було поставлено, та головну функціональність, інтелектуальна складова розпізнання та прийняття рішень з підходом розподіленого навчання нейронної мережі зі збереженням конфіденційності користувачів.

Оскільки, розробленим програмний засіб відповідає вимогам, що були поставлені для функціональності додатка, то можна зробити висновок, що атестаційна робота виконана в повному обсязі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Collective Intelligence for Exploratory Keyword Search. URL: https://www.researchgate.net/publication/313732060_TB- Structure_Collective_ Intelligence_for_Exploratory_Keyword_Search (дата зверенення: 20.04.2022).

2. Freeman A .NET Core for professionals. Redmond: Oreilly, 2019. 1008 с.

3. Sumit M., Mund Sumit. Microsoft Azure Machine Learning: навч. посіб. New York: Packt Enterprise: 2005. 278 с.

4. М. Мак-Дональд, Фримен, М. Microsoft ASP.NET 4 с примерами на C# 2010 для профессионалов. – 4–е изд: довідник. Redmond: ООО "И.Д. Вильямс", 2011. 1424 с.

5. REST API Tutorial. URL: https://www.restapitutorial.com/ (дата звернення: 21.04.2022).

6. Jhon A, Justin D. Cloud Native DevOps with Kubernetes: Building, Deploying, and Scaling Modern Applications in the Cloud: навч. посіб.. Minnesota Oreilly, 2019, 309с.

7. A. Troelsen, P. Japikse. Pro C# 7 With .NET and .NET Core: навч. посіб. USA Oreilly, 2019, 1201 с.

8. Probabilistic Neural Networks. URL: https://www.mathworks.com/help/deeplearning/ug/probabilistic-neural-networks.html;jsessionid=db3e7bc2229763ba64c867ce7f84 (дата звернення: 01.05.2022).