ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ
ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ
ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «On-Device Learning та його застосування у системи "розумного будинку" та промисловії автоматизації»

на здобуття освітнього ступеня бакалавра
зі спеціальності 126 Інформаційні системи та технології
(код, найменування спеціальності)
освітньо-професійної програми Інформаційні системи та технології
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____Артем НЕГОДА
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав: здобувач вищої освіти гр. ІСД-43
Артем НЕГОДА
Ім'я, ПРІЗВИЩЕ

Керівник: *К.т.н., доцент* Андрій АРОНОВ
науковий ступінь, вчене звання
Ім'я, ПРІЗВИЩЕ

Рецензент: _____
науковий ступінь, вчене звання
Ім'я, ПРІЗВИЩЕ

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Інформаційних систем та технологій
Ступінь вищої освіти бакалавр
Спеціальність Інформаційні системи та технології
Освітньо-професійна програма Інформаційні системи та технології

ЗАТВЕРДЖУЮ
Завідувач кафедрою Інф. Систем
та технологій
_____Каміла СТОРЧАК
«____» _____2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ
Негода Артем Олександрович

(прізвище, ім'я, по батькові здобувача)

1.Тема кваліфікаційної роботи: On-Device Learning та його застосування у системи "розумного будинку" та промисловії автоматизації
керівник кваліфікаційної роботи Андрій АРОНОВ к.т.н, доцент
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» березня 2025 р. № 56

2. Строк подання кваліфікаційної роботи «30» травня 2025 р.
3. Вихідні дані до кваліфікаційної роботи: наукові праці з тематики машинного навчання на пристроях (On-Device Learning) та TinyML, технічна документація до фреймворків для TinyML (наприклад, TensorFlow Lite Micro, Edge Impulse, PyTorch Mobile), офіційні матеріали та документація до мікроконтролерів та платформ для розробки (наприклад, Arduino Nano 33 BLE Sense, ESP32, SparkFun Edge), аналітичні огляди сучасних систем "розумного будинку" та промислової автоматизації, а також приклади реалізації рішень з використанням TinyML для автономного управління та оптимізації процесів.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
- Огляд теоретичних основ On-Device Learning та TinyML, їхніх алгоритмів та принципів роботи.
- Аналіз можливостей та викликів інтеграції TinyML у системи розумного будинку та промислової автоматизації.
- Вибір компонентів (мікроконтролерів, сенсорів) та фреймворків для розробки прототипу системи з On-Device Learning на основі TinyML.
- Розробка та навчання моделі TinyML для конкретного завдання (наприклад, передбачення значень функції, розпізнавання аномалій).
- Реалізація та тестування прототипу системи розумного будинку або промислової автоматизації з використанням розробленої моделі TinyML.

5. Ілюстративний матеріал: *презентація*

6. Дата видачі завдання: «24» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Вступ	11.03.2025	
2	Інтеграція tinyml у системи розумного будинку та промислової автоматизації	24.03.2025	
3	Технічна реалізація та моделювання on-device learning та tinyml	06.04.2025	
4	Створення та навчання моделі tinyml	17.04.2025	
5	Роль TinyML у розумних будинках	26.04.2025	
7	Розробка прототипу системи розумного будинку або промислової автоматизації з On-Device Learning на основі TinyML	07.05.2025	
8	Оформлення дипломної роботи	16.05.2025	

Здобувач(ка) вищої освіти _____ Артем НЕГОДА
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Андрій АРОНОВ
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

ПОДАННЯ
ГОЛОВІ ЕКЗАМЕНАЦІЙНОЇ КОМІСІЇ
ЩОДО ЗАХИСТУ КВАЛІФІКАЦІЙНОЇ РОБОТИ
на здобуття освітнього ступеня бакалавра (магістра)

Направляється здобувач(ка) Негода А.О. до захисту кваліфікаційної роботи
(прізвище та ініціали)
за спеціальністю 126, інформаційні системи та технології
(код, найменування спеціальності)
освітньо-професійної програми інформаційні системи та технології
(назва)
на тему: «On-Device Learning та його застосування у системи "розумного будинку" та промисловії автоматизації».

Кваліфікаційна робота і рецензія додаються.
Директор ННІ _____ Катерина НЕСТЕРЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Висновок керівника кваліфікаційної роботи

Здобувач Негода Артем Олександрович у своїй кваліфікаційній роботі дослідив актуальну та важливу тему інтеграції On-Device Learning та TinyML у сучасні системи. Він продемонстрував глибоке опрацювання теоретичного матеріалу, здатність до аналізу, порівняння технологій, практичні навички створення та навчання моделей машинного навчання. Поставлені завдання дослідження виконані повною мірою, а результати роботи мають наукову та практичну значущість.

Все це дозволяє оцінити виконану кваліфікаційну роботу здобувача Негоди Артема Олександровича на оцінку «відманно» та присвоїти йому кваліфікацію *бакалавр з інформаційних систем та мереж*.

Керівник кваліфікаційної роботи _____ Андрій АРОНОВ
(підпис) *(Ім'я, ПРІЗВИЩЕ)*
«___» _____2025 року

Висновок кафедри про кваліфікаційну роботу

Кваліфікаційна робота розглянута. Здобувач *Негода А.О.* допускається до захисту даної роботи в Екзаменаційній комісії.
Завідувач кафедрою Інф. систем та технологій _____ Каміла СТОРЧАК
(назва) *(підпис)* *(Ім'я, ПРІЗВИЩЕ)*

ВІДГУК РЕЦЕНЗЕНТА
на кваліфікаційну роботу
на здобуття освітнього ступеня бакалавра (магістра)

здобувача(ки) вищої освіти: Негоди Артема Олександровича

(прізвище, ім'я, по батькові)

на тему «**On-Device Learning та його застосування у системи "розумного будинку" та промисловії автоматизації**»

Актуальність.

Через зростаючі вимоги до автономних систем управління, які повинні ефективно обробляти дані в реальному часі, зберігаючи при цьому низьке енергоспоживання та високу надійність. Впровадження TinyML у такі системи дозволяє вирішити ці завдання, забезпечуючи швидке та точне прийняття рішень безпосередньо на пристроях. Робота спрямована на дослідження та практичну реалізацію TinyML, що є важливим для розвитку інтелектуальних автономних систем.

Позитивні сторони:

1. Робота глибоко розглядає основи On-Device Learning, TinyML та існуючі фреймворки.
2. Чітко аналізується застосування технологій у розумних будинках та промисловості.
3. Продемонстровано процес створення та навчання базової TinyML моделі.

Недоліки:

1. Реалізована модель (передбачення синусоїди) є радше демонстрацією принципів, ніж повноцінним рішенням для заявлених складних систем.
2. Недостатньо висвітлено практичне вирішення специфічних для промисловості чи розумного будинку інтеграційних проблем під час розробки більш комплексного прототипу.

Відзначені зауваження не впливають на загальну позитивну оцінку кваліфікаційної роботи бакалаврської.

Висновок: *кваліфікаційна робота на здобуття ступеня бакалавра заслуговує оцінку "відмінно", а здобувач(ка) заслуговує присвоєння кваліфікації: бакалавр з інформаційних систем та мереж.*

Рецензент:

науковий ступінь, вчене звання	*підпис*	*Ім'я, ПРІЗВИЩЕ*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра (магістра): 53 стор., 26 рис., 4 табл., 32 джерел.
Мета роботи – дослідження та аналіз можливостей застосування TinyML і On-Device Learning в системах розумного будинку та промислової автоматизації, а також розробка прототипу системи, що використовує ці технології для автономного управління і оптимізації процесів у реальному часі.

Об’єкт дослідження – системи розумного будинку та промислової автоматизації, процеси обробки даних та навчання моделей на пристроях з обмеженими ресурсами.

Предмет дослідження – технології On-Device Learning та TinyML, методи оптимізації моделей машинного навчання для вбудованих систем, фреймворки для розробки TinyML-рішень (наприклад, TensorFlow Lite Micro) та їх інтеграція в автономні системи.

Короткий зміст роботи:
У цій роботі було досліджено теоретичні основи On-Device Learning та TinyML, що дозволяють виконувати машинне навчання безпосередньо на пристроях з обмеженими ресурсами. Розглянуто переваги такого підходу, зокрема зниження затримок, підвищення конфіденційності та енергоефективності, а також наявні обмеження, пов'язані з обчислювальними потужностями та пам'яттю пристроїв. Проаналізовано можливості інтеграції цих технологій у сфери розумного будинку для інтелектуальної автоматизації, розпізнавання голосу, моніторингу здоров'я та безпеки, і в промислову автоматизацію для прогнозування обслуговування обладнання, управління якістю та моніторингу безпеки. Описано популярні фреймворки для TinyML, такі як TensorFlow Lite Micro, Edge Impulse та PyTorch Mobile. Розроблено та описано процес створення й навчання моделі TinyML для передбачення значень математичної функції, її конвертації для роботи на мікроконтролері та розгортання на прикладі візуалізації світлодіодами.

КЛЮЧОВІ СЛОВА: ON-DEVICE LEARNING, TINYML, РОЗУМНИЙ БУДИНОК, ПРОМИСЛОВА АВТОМАТИЗАЦІЯ, TENSORFLOW LITE, МІКРОКОНТРОЛЕРИ, МАШИННЕ НАВЧАННЯ, ВБУДОВАНІ СИСТЕМИ, ЕНЕРГОЕФЕКТИВНІСТЬ, АВТОНОМНІ СИСТЕМИ.

ЗМІСТ

ЗМІСТ

ВСТУП

Сучасні технології машинного навчання відкривають нові можливості для автоматизації та оптимізації процесів у різних сферах життя. Однією з найбільш перспективних напрямків є **On-Device Learning**, що дозволяє здійснювати обробку та навчання моделей машинного навчання безпосередньо на пристроях, зокрема на мобільних телефонах, мікроконтролерах, датчиках та інших вбудованих системах. Цей підхід знижує залежність від хмарних обчислювальних потужностей, дозволяючи обробляти дані в реальному часі з мінімальними затримками, що є критичним для багатьох індустріальних та побутових застосувань.

Однією з найбільш перспективних технологій для реалізації On-Device Learning є TinyML — область машинного навчання, що спеціалізується на виконанні моделей машинного навчання на пристроях з обмеженими ресурсами, таких як мікроконтролери та сенсори. Використання TinyML дозволяє створювати ефективні, низькоенергетичні та автономні системи для різних сфер застосування, включаючи розумний будинок та промислову автоматизацію.

Розумний будинок, як концепція, прагне створити інтегровану систему, що забезпечує автоматизацію та оптимізацію всіх аспектів життєдіяльності, таких як енергоспоживання, безпека, комфорт мешканців. У свою чергу, промислова автоматизація має на меті вдосконалення виробничих процесів, забезпечення ефективного контролю та мінімізацію людського втручання через застосування автоматичних систем управління. В обох випадках інтеграція TinyML з On-Device Learning дозволяє значно покращити ефективність, знизити затримки у прийнятті рішень та зробити системи більш автономними.

Актуальність теми дослідження обумовлена зростаючими вимогами до автономних систем управління, які повинні ефективно обробляти дані в реальному часі, при цьому зберігаючи низьке споживання енергії та високу надійність. Впровадження TinyML в такі системи дозволяє вирішити ці завдання, забезпечуючи швидке та точне прийняття рішень на місці, без необхідності передавати дані для обробки у віддалене серверне середовище.

Метою цієї дипломної роботи є дослідження та аналіз можливостей застосування TinyML і On-Device Learning в системах розумного будинку та промислової автоматизації, а також розробка прототипу системи, що використовує ці технології для автономного управління і оптимізації процесів у реальному часі.

Завдання дослідження включають:

• Ознайомлення з основними принципами On-Device Learning та TinyML;

• Аналіз можливостей інтеграції цих технологій у системи розумного будинку та промислової автоматизації;

• Оцінка переваг та обмежень застосування TinyML на вбудованих пристроях;

• Розробка та тестування прототипу системи на основі TinyML для однієї з зазначених сфер.

Таким чином, робота має на меті не лише теоретичний аналіз, але й практичну реалізацію та оцінку ефективності TinyML у реальних умовах, що є важливим кроком у напрямку розвитку інтелектуальних автономних систем.

1 ТЕОРЕТИЧНІ ОСНОВИ ON-DEVICE LEARNING ТА TINYML

1.1 Що таке On-Device Learning і як це працює

On-Device Learning — це підхід до машинного навчання, при якому всі етапи обробки даних, навчання та адаптації моделей здійснюються безпосередньо на пристрої, без потреби відправляти дані на віддалені сервери або в хмару. Це означає, що модель машинного навчання працює локально на самому пристрої, наприклад, на мобільному телефоні, розумному годиннику чи іншому IoT-пристрої, а не в хмарі чи на сервері.

Як це працює? Коли пристрій починає збирати дані (наприклад, з сенсорів або камер), ці дані обробляються і використовуються для тренування моделей машинного навчання прямо на самому пристрої. Наприклад, мобільний телефон може вчитися розпізнавати ваш голос, або розумний термостат може адаптуватися до ваших звичок і змінювати температуру в приміщенні, залежно від того, як ви використовуєте його.

Щоб це стало можливим, зазвичай модель спочатку навчається на потужних серверних системах, а потім її оптимізують для роботи на більш слабких пристроях, з урахуванням обмежених обчислювальних ресурсів. Для цього використовуються різні техніки оптимізації, такі як стиснення моделей або квантоване навчання, які дозволяють зменшити розмір і складність моделей без великої втрати точності.

Основні переваги On-Device Learning — це покращення конфіденційності (дані не залишають пристрій), зменшення часу відгуку (дані не передаються на сервери для обробки) та енергоефективність. Однак важливо враховувати, що можливості пристроїв обмежені, і складні завдання машинного навчання можуть бути виконані не так ефективно, як на потужних серверних системах.

1.2 Що таке TinyML

TinyML — це напрямок машинного навчання, який спеціалізується на розробці та оптимізації моделей машинного навчання для використання на малопотужних пристроях з обмеженими ресурсами, таких як мікроконтролери,

датчики, розумні годинники, фітнес-браслети та інші пристрої Інтернету речей (IoT).

Завдяки TinyML, можливе впровадження алгоритмів машинного навчання на пристроях з низьким споживанням енергії та обмеженими обчислювальними ресурсами, що дозволяє виконувати розпізнавання образів, голосу, а також інші завдання прямо на пристрої без необхідності підключення до потужних серверів або хмарних обчислень. Це відкриває нові можливості для створення розумних пристроїв, які можуть працювати автономно, обробляючи дані локально.

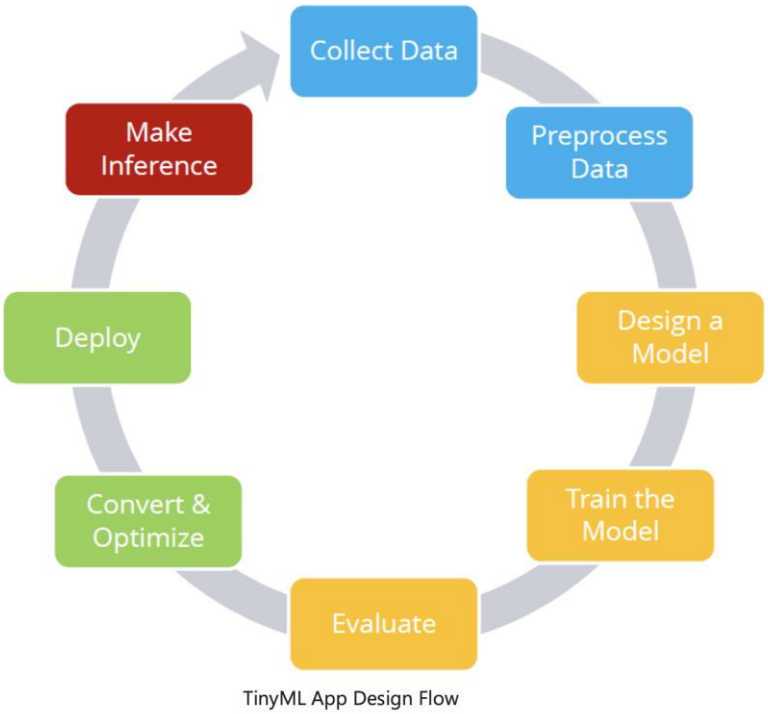


Рисунок 1.1 цикл розробки TinyML

Однією з головних переваг TinyML є те, що всі операції з обробки даних відбуваються на самому пристрої, що забезпечує низьку затримку, високу швидкість реакції та знижує потребу у відправленні даних на сервери, що, в свою чергу, підвищує рівень конфіденційності та безпеки. Крім того, таке підходить для випадків, коли з’єднання з інтернетом є нестабільним або недоступним.

TinyML використовує спеціалізовані техніки оптимізації моделей, такі як стиснення моделей, квантоване навчання і обчислення з низькою точністю, щоб моделі могли працювати навіть на пристроях з обмеженою пам’яттю та низькою обчислювальною потужністю.

Це дозволяє використовувати машинне навчання в найбільш компактних і енергоефективних рішеннях, таких як сенсори на фермах, пристрої для моніторингу здоров’я, розумні домашні пристрої та багато інших IoT-продуктів.

1.3 Як працюють системи на базі On-Device Learning?

Системи на базі On-Device Learning дозволяють здійснювати машинне навчання безпосередньо на пристрої, що має значну кількість переваг. У таких системах дані, зібрані з сенсорів або інших джерел, обробляються на місці, без потреби відправляти їх на сервери чи в хмару. Наприклад, в мобільних телефонах, носимих пристроях або розумних гаджетах дані збираються локально, і на основі цих даних здійснюється обробка і прийняття рішень. Це дозволяє пристроям швидко реагувати на зміни в реальному часі, що особливо корисно в умовах обмеженого інтернет-з’єднання або коли потрібно забезпечити високу конфіденційність.

Однією з головних переваг є швидкість. Оскільки дані не передаються на віддалені сервери для обробки, пристрій може миттєво реагувати на нову інформацію, наприклад, в реальному часі розпізнавати обличчя або голос користувача, чи змінювати параметри температури в будинку залежно від зовнішніх умов. Важливим є й аспект конфіденційності. Оскільки обробка відбувається локально, дані не покидають пристрій, що знижує ризик витоку інформації. Це особливо важливо для чутливих даних, наприклад, медичних показників або особистих звичок.

Системи On-Device Learning також зазвичай мають низьке енергоспоживання, що дозволяє їм працювати на малопотужних пристроях, таких як смартфони, фітнес-браслети чи розумні годинники. Завдяки оптимізації моделей машинного навчання для малопотужних пристроїв, вони здатні працювати довгий час від батареї, не вимагаючи частого підзаряджання. Наприклад, розумний термостат або розумний годинник може зберігати дані про активність користувача або температуру, а потім адаптувати поведінку пристрою до його потреб без необхідності підключення до інтернету.

Однак для того, щоб система працювала ефективно на таких пристроях, необхідна спеціальна оптимізація моделей. Зазвичай, навчання машинного інтелекту проводиться на потужних серверах з великими обчислювальними ресурсами, але на пристрої, де обмеження на пам’ять і обчислювальні потужності є значними, потрібно використовувати компактніші та ефективніші моделі. Це може включати стиснення моделей або використання технік квантованих значень, щоб зменшити вимоги до пам’яті. Моделі часто потрібно адаптувати так, щоб вони могли працювати при мінімальних витратах на енергію.

Проте існують і певні обмеження. На малопотужних пристроях не завжди можна застосувати складні моделі, і це може вплинути на точність результатів. Наприклад, якщо модель повинна здійснювати складне розпізнавання образів або аналіз великих даних, її точність може бути нижчою порівняно з великими серверними рішеннями. Крім того, хоча локальне навчання дозволяє пристроям адаптуватися до нових умов або змін в поведінці користувача, цей процес часто обмежений в порівнянні з тим, що можна зробити на сервері.

В цілому, системи On-Device Learning дозволяють створювати більш персоналізовані та ефективні пристрої, здатні працювати автономно, зберігаючи конфіденційність і енергоефективність. Вони відкривають нові можливості для створення розумних гаджетів, які можуть реагувати на змінені умови в реальному часі, не залежачи від зовнішніх серверів. Однак для того, щоб ці системи працювали ефективно, важливо враховувати обмеження пристроїв і адаптувати моделі машинного навчання до конкретних умов використання.

1.4 Переваги та обмеження On-Device Learning та TinyML

On-Device Learning і TinyML революціонізують спосіб функціонування сучасних пристроїв, пропонуючи суттєві переваги порівняно з традиційними хмарними підходами до машинного навчання. Водночас ці технології мають певні обмеження, які варто враховувати.

Основною перевагою On-Device Learning є швидкість і низька затримка, оскільки всі обчислення відбуваються безпосередньо на пристрої без необхідності

передачі даних на віддалені сервери. Це забезпечує миттєву реакцію на користувацькі запити в смартфонах, розумних годинниках та інших пристроях.

Конфіденційність і безпека значно підвищуються завдяки тому, що чутливі дані, як-от медичні записи чи персональна інформація, не покидають пристрій. Відсутність передачі даних на віддалені сервери суттєво зменшує ризик витоку інформації або несанкціонованого доступу.

TinyML забезпечує високу енергоефективність, дозволяючи створювати моделі з мінімальним енергоспоживанням. Це критично важливо для пристроїв, що працюють від батарей — сенсорів, носимих гаджетів, розумних термостатів — які завдяки цьому можуть функціонувати тривалий час без підзарядки.

Доступність і автономність On-Device Learning уможливлюють використання технологій машинного навчання навіть на пристроях без постійного підключення до інтернету. Це особливо важливо для віддалених територій, автомобілів та інших середовищ з обмеженим доступом до мережі.

Масштабованість і доступність технології дозволяють впроваджувати інтелектуальні функції навіть у недорогі пристрої, відкриваючи нові можливості для Інтернету речей та інших технологій масового застосування.

Проте On-Device Learning і TinyML мають свої обмеження. Головне з них — обмежена обчислювальна потужність пристроїв. Більшість мобільних телефонів, носимих пристроїв і сенсорів не здатні виконувати складні обчислення, які зазвичай проводяться на потужних серверах. Навчання великих моделей вимагає значних ресурсів, і хоча ця проблема з часом вирішується, багато моделей залишаються надто великими для ефективного виконання на малопотужних пристроях.

Обмеження пам'яті та сховища даних також становлять виклик, оскільки більшість пристроїв мають обмежені обсяги для зберігання даних і моделей. Стиснення чи оптимізація великих моделей може призвести до втрати точності.

Розробка й оптимізація моделей для On-Device Learning потребує спеціальних навичок, оскільки розробникам необхідно враховувати обмеження пристроїв щодо потужності процесора, обсягу пам'яті та енергоспоживання. Традиційні підходи до навчання моделей часто не підходять для таких умов.

Можливості для додаткового навчання безпосередньо на пристрої також обмежені, оскільки On-Device Learning зазвичай працює з моделями, попередньо навченими на потужніших серверах. Це обмежує здатність моделі адаптуватися до нових умов.

Нарешті, існує постійна залежність від точності моделей. Оптимізація для малих пристроїв часто вимагає компромісів, які можуть негативно вплинути на точність результатів — зменшення розміру моделі може призвести до гіршої якості класифікації чи прогнозування.

1.5 Фреймворки для TinyML

Фреймворк для TinyML — це спеціалізоване програмне забезпечення або інструмент, який дозволяє розробникам та інженерам створювати моделі машинного навчання, оптимізовані для розгортання на пристроях з обмеженими ресурсами та вбудованих системах. Ці платформи надають необхідну інфраструктуру, алгоритми та ресурси для навчання моделей Tiny Machine Learning (TinyML), які оптимізовані для роботи на пристроях з обмеженими ресурсами та низьким енергоспоживанням. Фреймворки для TinyML зазвичай підтримують завдання, такі як збір даних, навчання моделей, оптимізація та розгортання на пристроях на кромці мережі, дозволяючи розробляти ефективні та потужні моделі машинного навчання, які відповідають вимогам середовищ edge computing.

TensorFlow — це відкритий фреймворк для машинного навчання від компанії Google, який допомагає швидко розробляти моделі машинного навчання. Для TinyML існує спеціалізована версія TensorFlow — TensorFlow Lite Micro, яка призначена для мікроконтролерів та інших пристроїв з дуже обмеженою пам’яттю, зазвичай кілька кілобайт. Основний рушій цієї версії займає лише 16 КБ на процесорі Arm Cortex M3 і здатний виконувати базові моделі. Для її роботи не потрібна підтримка операційної системи, стандартні бібліотеки C або C++ чи динамічне виділення пам’яті. TensorFlow Lite Micro переважно сумісний з процесорами серії Arm Cortex-M. Доступна також версія для ESP32.

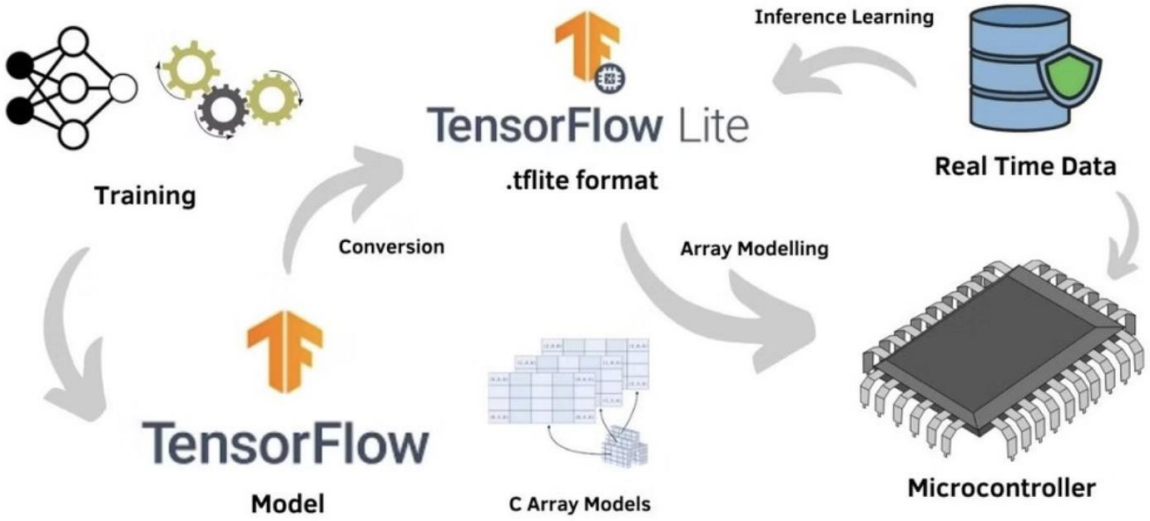


Рисунок 1.2 опис TensorFlow Lite

TensorFlow Lite пропонує значні переваги для розробників, які працюють з мобільними та вбудованими пристроями. Він забезпечує швидке виведення на мобільних пристроях завдяки використанню апаратних прискорювачів, таких як графічні процесори (GPU) та цифрові сигнальні процесори (DSP). Це дозволяє ефективно реалізувати застосунки реального часу, включаючи розпізнавання об'єктів та жестикуляцію.

Гнучкість є ще однією важливою перевагою TensorFlow Lite, оскільки він підтримує різноманітні платформи, включаючи Android, iOS, Linux і мікроконтролери. Така універсальність робить його придатним для широкого спектру пристроїв, що підтверджується даними в таблиці апаратних платформ для TinyML-фреймворків.

TensorFlow Lite також вирізняється легкістю інтеграції, оскільки він безперешкодно вписується в існуючі робочі процеси TensorFlow. Розробники можуть тренувати моделі, використовуючи повний набір інструментів TensorFlow, а потім перетворювати їх у формат TensorFlow Lite для мобільного та вбудованого розгортання. Доступність API для Python, Java та C++ значно спрощує включення можливостей машинного навчання в різноманітні застосунки.

Проте TensorFlow Lite має певні обмеження. Він підтримує лише обмежений набір операцій TensorFlow, що може ускладнити реалізацію складніших моделей на

мікроконтролерах. Також варто враховувати, що не всі пристрої підтримують TensorFlow Lite через апаратні обмеження. Робота з мікроконтролерами вимагає ґрунтовних знань C++ та навичок ручного керування пам'яттю. Крім того, TensorFlow Lite для мікроконтролерів не дозволяє навчати моделі безпосередньо на пристрої, що обмежує їхню адаптивність.

Edge Impulse, натомість, є інноваційною платформою, що дозволяє підприємствам створювати інтелектуальні продукти для edge-пристроїв. Цей фреймворк надає простий спосіб збору даних, тренування моделей та їх розгортання на мікроконтролерах.

Основною перевагою Edge Impulse є підтримка повного життєвого циклу Edge AI: від збору даних та екстракції ознак до розробки, тренування, тестування та розгортання моделей машинного навчання на кінцевих пристроях. Така комплексна підтримка дозволяє повністю автоматизувати процеси створення моделей для edge-обчислювального середовища.

Edge Impulse також легко інтегрується з іншими фреймворками машинного навчання, що значно підвищує гнучкість і адаптивність рішення, дозволяючи масштабувати та налаштовувати моделі відповідно до конкретних потреб. Завдяки цим характеристикам Edge Impulse стає ідеальним вибором для розробки ефективних і масштабованих систем машинного навчання, які працюють на пристроях з обмеженими ресурсами, таких як вбудовані системи та мікроконтролери.

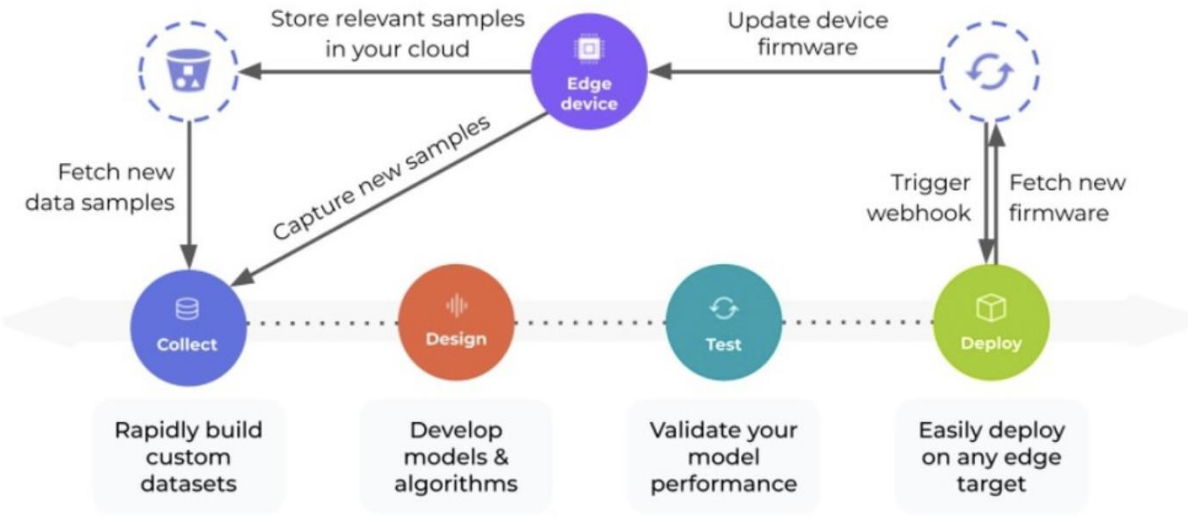


Рисунок 1.2 Розробка моделі TinyML

Edge Optimized Neural (EON™) Compiler

EON™ Compiler — це оптимізований компілятор для нейронних мереж, який дозволяє зменшити використання пам’яті та флеш-пам’яті на 25-55% порівняно з TensorFlow Lite для мікроконтролерів, зберігаючи при цьому ту саму точність моделей.

Наприклад, розглянемо різницю, яку дає EON на типовій моделі в Edge Impulse. Ось як виглядають показники часу на виведення, використання RAM і ROM для моделі розпізнавання ключових слів з 2D згортковою нейронною мережею, що працює на Cortex-M4F. Зверху: модель, оптимізована за допомогою EON, знизу: та ж модель, що працює за допомогою TensorFlow Lite для мікроконтролерів.

Використання EON дозволяє зменшити споживання ресурсів без шкоди для ефективності, що робить його чудовим вибором для пристроїв з обмеженими ресурсами.

Рисунок 1.4 використання ресурсів EON компілятора

Обмеження Edge Impulse

Проблеми сумісності

У Edge Impulse є певні обмеження, пов’язані з налаштуваннями для більш складних і спеціалізованих моделей, сумісністю з певним апаратним забезпеченням та кривою навчання для тих, хто новий у машинному навчанні та IoT технологіях.

Обмежена кастомізація

Платформа може бути обмежена у можливостях створення дуже складних або спеціалізованих моделей. Користувачі з більш просунутими потребами у машинному навчанні можуть захотіти більше можливостей для налаштування або підтримки більш складних архітектур моделей.

PyTorch Mobile є частиною екосистеми PyTorch, яка підтримує всі етапи, починаючи від тренування до розгортання моделей машинного навчання на смартфонах (Android, iOS). Платформа пропонує декілька API для попередньої обробки даних для мобільних застосунків. PyTorch Mobile підтримує скрипти та трасування TorchScript IR, а також забезпечує підтримку XNNPACK 8-бітних ядер для ARM процесорів, GPU, DSP та нейронних обробних одиниць. Оптимізація для мобільних пристроїв здійснюється через мобільний інтерпретатор. На даний момент PyTorch Mobile підтримує задачі сегментації зображень, розпізнавання об'єктів, обробки відео, розпізнавання мови та відповіді на питання.

Ключові особливості PyTorch Mobile включають підтримку багатьох платформ, оскільки він може працювати на iOS, Android та Linux, що надає розробникам широкий спектр варіантів для розгортання. Платформа також пропонує API для виконання загальних завдань попередньої обробки та інтеграції, що спрощує впровадження машинного навчання в мобільні застосунки. Важливою перевагою є підтримка TorchScript, що включає як трасування, так і скриптування через TorchScript IR, відповідаючи різним вимогам для розгортання.

Порівнюючи PyTorch з TensorFlow, можна відзначити, що обидві платформи досягають подібної точності, але TensorFlow потребує значно більше часу для тренування, хоча використовує менше пам'яті. PyTorch дозволяє швидше прототипувати моделі порівняно з TensorFlow, однак для кастомних функцій у нейронних мережах TensorFlow може бути кращим вибором.

TensorFlow трактує нейронні мережі як статичні об'єкти, тому будь-які зміни в моделі потребують початку з нуля. У PyTorch можна динамічно налаштовувати нейронну мережу під час виконання, що спрощує оптимізацію моделі. Для ефективного налагодження TensorFlow потрібен спеціальний інструмент налагодження для перевірки обчислення вузлів мережі на кожному кроці, тоді як у PyTorch можна використовувати стандартні інструменти налагодження Python.

Обидві платформи пропонують методи прискорення розробки моделей і зменшення кількості шаблонного коду, але основна різниця між PyTorch і TensorFlow полягає в тому, що PyTorch є більш "python-орієнтованим" і базується на об'єктно-орієнтованому підході, тоді як TensorFlow пропонує більшу гнучкість.

uTensor є легким фреймворком для інференсу машинного навчання, оптимізованим для платформ Arm і заснованим на TensorFlow. Він використовує Keras для тренування нейронних мереж і конвертує їх в C++ для подальшого розгортання на пристроях з обмеженими ресурсами. uTensor є дуже компактним, займаючи всього 2 КБ на диску. Платформа використовує Python SDK для налаштування і може працювати з такими інструментами, як uTensor-CLI, Jupyter, Mbed-CLI та ST-link (для плат ST). Після створення моделі та визначення ефекту квантизації, наступним етапом є генерація коду для пристроїв на кромці.

uTensor ідеально підходить для вбудованих пристроїв, що вимагають ефективних і компактних рішень для машинного навчання, забезпечуючи гнучкість та високу продуктивність при мінімальних витратах ресурсів.

Таблиця 1

модулі TensorFlow

Module	**.text**	**.data**	**.bss**
uTensor/src/uTensor/core	1275(+1275)	4(+4)	28(+28)
uTensor/src/uTensor/tensors	791(+791)	0(+0)	0(+0)

У TensorFlow модель будується та тренується у вигляді послідовності етапів. В свою чергу, uTensor приймає вже натреновану модель і генерує файли .cpp та .hpp, які містять згенерований код на C++11 для виконання інференсу (процесу прогнозування на основі моделі).

Завдяки цьому процесу використання uTensor на вбудованих пристроях стає дуже простим — достатньо просто скопіювати згенеровані файли і вставити їх у ваш проект. Це дозволяє ефективно використовувати машинне навчання на пристроях з обмеженими ресурсами, не потребуючи складної інфраструктури або великої кількості пам’яті, що є важливим аспектом для розгортання моделей на кромці (edge devices).

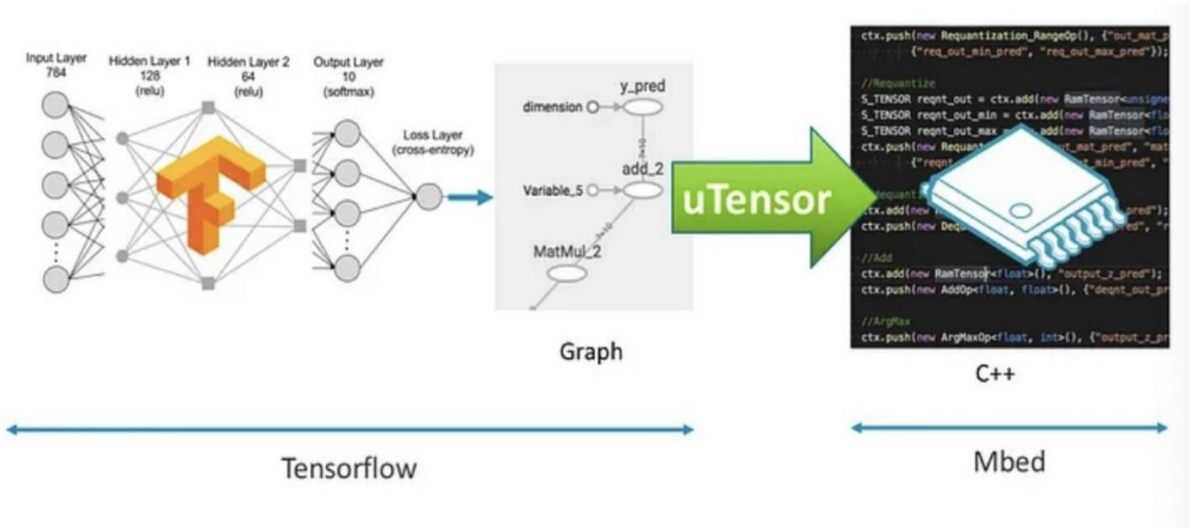


Рисунок 1.5 Приклад роботи TensorFlow

Безпечний та надійний: За рахунок управління метаданими та реальними даними у виділених регіонах пам'яті, uTensor забезпечує, щоб використання пам'яті залишалося в межах заданих обмежень, а також надає перевірку помилок під час компіляції.

Зручний API: Завдяки інтерфейсу, який нагадує стиль високорівневої мови, uTensor спрощує процес розробки та підтримує оптимізацію продуктивності безпосередньо на рівні C++.

Гнучка розширюваність: Від основної бібліотеки до стандартних реалізацій, uTensor дозволяє налаштування і оптимізацію, зокрема, у реалізаціях тензорів, операторах, аллокаторах пам'яті та інших компонентах.

STM32Cube.AI — це програмне забезпечення для генерації коду та оптимізації, яке спрощує завдання, пов'язані з машинним навчанням та штучним інтелектом для плат на базі ARM Cortex-M, таких як STM32. За допомогою STM32Cube.AI можна безпосередньо реалізувати нейронні мережі на платах STM32, конвертуючи їх у оптимізований код для найбільш підходящих мікроконтролерів (MCU). Програмне забезпечення також дозволяє оптимізувати використання пам'яті під час виконання та підтримує використання моделей, натренованих за допомогою традиційних інструментів, таких як TensorFlow, ONNX, Matlab і PyTorch. Це інструмент, який є розширенням оригінальної платформи STM32CubeMX, що

дозволяє STM32Cube.AI виконувати генерацію коду для цільових пристроїв STM32 та оцінку параметрів програмного забезпечення.

Генерація бібліотек з попередньо натренованих моделей: Дає змогу генерувати бібліотеки з нейронних мереж та типових моделей машинного навчання, оптимізованих для STM32.

Підтримка популярних фреймворків: Підтримує найпопулярніші фреймворки, такі як TensorFlow Lite, Keras, qKeras, PyTorch, ONNX та інші.

Легка переносність: Забезпечує легку переносимість між різними серіями мікроконтролерів STM32 завдяки інтеграції з STM32Cube.

Зручність інтеграції: Дає можливість легше інтегрувати перетворені бібліотеки нейронних мереж завдяки прикладам коду, орієнтованим на конкретні застосунки (функціональні пакети).

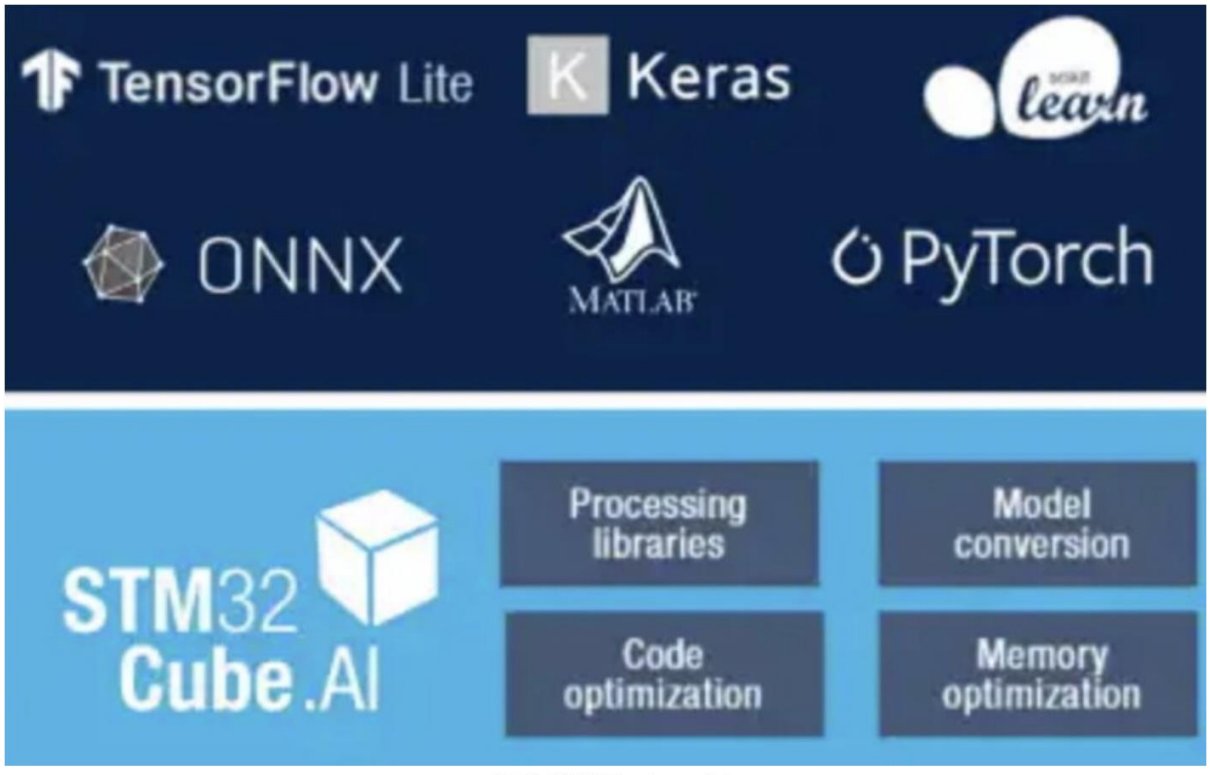


Рисунок 1.6 CubeAI

NanoEdgeAIStudio

NanoEdgeAIStudio — це інструмент автоматизованого машинного навчання, розроблений для програмістів, що працюють з платами STM32. Цей інструмент не

вимагає від користувачів спеціалізованих знань з обробки даних або штучного інтелекту (ШІ), оскільки пропонує зручне для користувача середовище та підтримує всі продукти STM32.

Однією з ключових особливостей NanoEdgeAIStudio є можливість реєстрації даних, що дозволяє збирати і керувати високошвидкісними даними з промислових датчиків без необхідності написання коду для їх обробки. Ця функція робить його особливо зручним для промислових застосунків, де швидкість і точність обробки даних є критично важливими.

NanoEdgeAIStudio також пропонує інші потужні можливості, такі як автоматичний пошук моделей, виявлення аномалій, а також алгоритми для класифікації та регресії, що робить машинне навчання на пристроях з обмеженими ресурсами більш доступним для розробників.

Завдяки таким функціям, NanoEdgeAIStudio спрощує процес використання машинного навчання на пристроях кромки (edge devices), дозволяючи навіть тим, хто не має глибоких знань у галузі ШІ, створювати ефективні моделі для різноманітних завдань.

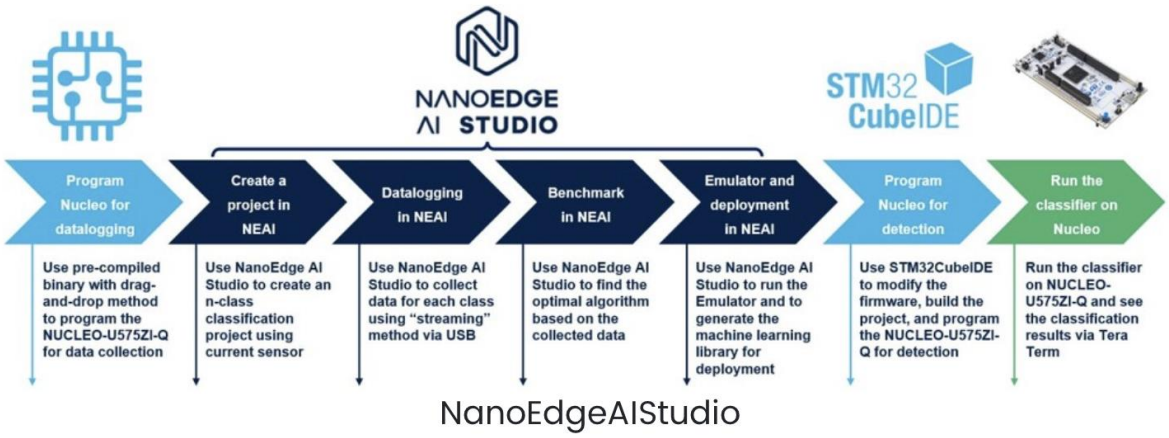


Рисунок 1.7 NanoEdgeAIStudio

NXP eIQ® Machine Learning Software Development Environment — це комплекс бібліотек та інструментів для розробки, призначений для використання з мікропроцесорами та мікроконтролерами від NXP Semiconductors. Це програмне забезпечення надає всі необхідні інструменти для впровадження алгоритмів машинного навчання на вбудованих системах. Включає в себе DeepViewRT™ —

власний інтерпретатор для інференсії, який дає змогу робити прогнози на основі моделей нейронних мереж (NN) і штучного інтелекту (AI) на пристроях з обмеженими ресурсами.

Програмне забезпечення eIQ (що означає “edge intelligence”) надає основні інструменти для розгортання різноманітних алгоритмів машинного навчання на пристроях кромки (edge devices). Це включає в себе інтерпретатори для інференсії, компілятори нейронних мереж, рішення для обробки зображень та сенсорів, а також шари абстракції апаратного забезпечення, що дозволяють працювати з різноманітними платформами та пристроями.

Основні інтерпретатори та бібліотеки, які підтримуються в **eIQ**, включають:

OpenCV — для обробки зображень.

Arm® NN — для роботи з нейронними мережами.

Arm CMSIS-NN — оптимізація роботи з нейронними мережами на архітектурі Arm.

TensorFlow Lite — відомий фреймворк для роботи з машинним навчанням на мобільних і вбудованих пристроях.

DeepViewRT — власний інтерпретатор NXP для інференсії на пристроях.

Цей набір інструментів дозволяє ефективно впроваджувати алгоритми машинного навчання та інтелекту на пристрої, що працюють на основі технологій NXP, забезпечуючи високу ефективність та масштабованість для різноманітних завдань в реальному часі.

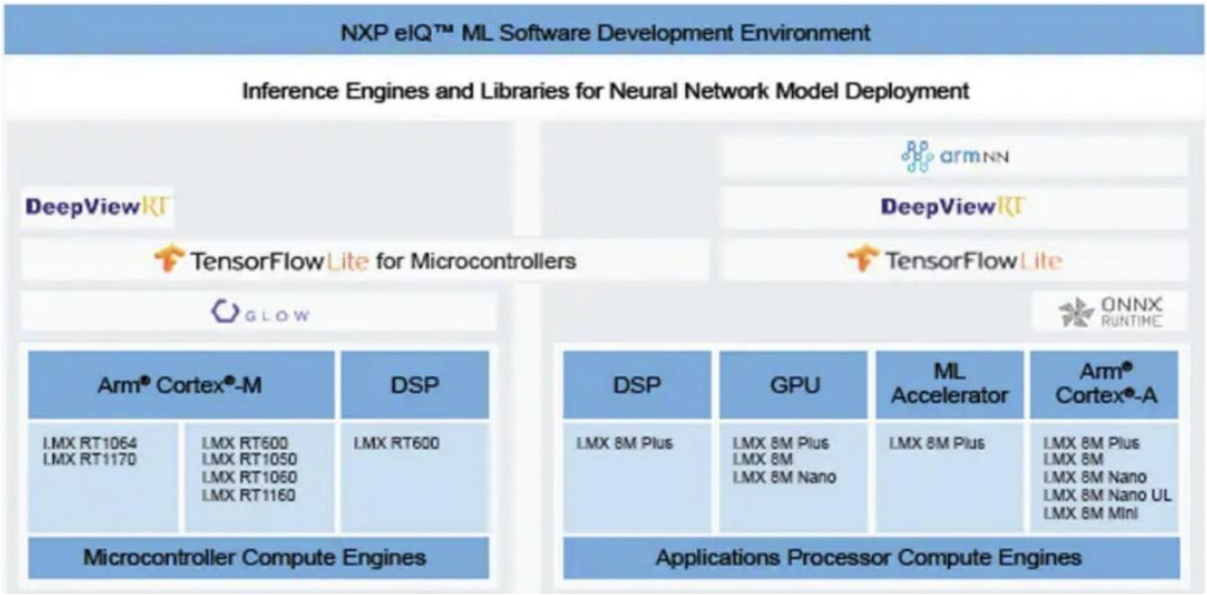


Рисунок 1.7 Програмне забезпечення NXP eIQ

Embedded Learning Library (ELL)

Embedded Learning Library (ELL) — це інструмент, розроблений компанією Microsoft, який підтримує екосистему TinyML для вбудованого навчання. Ця бібліотека надає можливості для створення та впровадження моделей машинного навчання на різноманітних вбудованих платформах, таких як Raspberry Pi, Arduino та micro:bit. Моделі, які використовуються на цих пристроях, не потребують підключення до хмари, оскільки вони є незалежними від інтернет-з’єднання.

Однією з основних переваг ELL є можливість розгортання моделей для класифікації зображень та аудіо. Це дозволяє реалізувати функції, як-от розпізнавання об’єктів чи аналіз звуків на пристроях з обмеженими ресурсами. Бібліотека також надає набір програмних інструментів та опціональний інтерфейс на Python, що дозволяє зручно інтегрувати TinyML рішення у проекти.

Ключові Особливості:

Широкий набір інструментів для оптимізації та Model Zoo: ELL надає інструменти для оптимізації моделей машинного навчання та бібліотеку готових моделей (Model Zoo), що значно спрощує процес створення ефективних рішень для вбудованих систем.

Великий вибір інтерпретаторів для інференсії: бібліотека підтримує різноманітні варіанти інтерпретаторів з автоматичним налаштуванням якості, що

дозволяє ефективно вибирати найбільш підходящий інтерпретатор для конкретної задачі.

Можливість запуску ML задач на кількох ядрах процесора: підтримка паралельної обробки дозволяє значно збільшити ефективність виконання машинних завдань на вбудованих платформах з кількома процесорними ядрами.

ELL забезпечує зручне рішення для інтеграції машинного навчання в низькоресурсні вбудовані системи, роблячи TinyML доступним і практичним для різноманітних IoT-застосунків.

Серед цих платформ TensorFlow вирізняється своєю видатною гнучкістю, підтримуючи понад 20 апаратних платформ, таких як Arduino Nano, Sparkfun Edge і STM32F746 Discovery Kit. Це робить його підходящим для різноманітних цільових застосувань, зокрема для класифікації зображень та аудіо, обнаруження об’єктів, оцінки пози, розпізнавання мови та жестів.

Edge Impulse презентує Edge Optimized Neural (EON™) Compiler, який дозволяє знизити використання оперативної пам’яті нейронними мережами на 25-55% та зменшити обсяг флеш-пам’яті до 35%, зберігаючи при цьому ту ж точність.

PyTorchпропонує швидшу розробку прототипів порівняно з TensorFlow, тоді як uTensor є компактним модулем, що потребує лише 2 КБ місця на диску.

Крім того, провідні індустріальні лідери, такі як ST, NXP і Microsoft, також представили свої власні платформи для впровадження TinyML, що сприяє подальшому розвитку технології TinyML.

2 ІНТЕГРАЦІЯ TINYML У СИСТЕМИ РОЗУМНОГО БУДИНКУ ТА ПРОМИСЛОВОЇ АВТОМАТИЗАЦІЇ

2.1 Що таке розумний будинок та промислова автоматизація

Перш ніж говорити про TinyML у розумних будинках та промислових системах, варто зрозуміти, що взагалі ці системи собою представляють.

Розумний будинок — це будинок, оснащений різноманітними технологіями для автоматизації повсякденних процесів. Наприклад, розумні термостати, які самі регулюють температуру, системи освітлення, що підлаштовуються під ваш розклад, або навіть охоронні системи, які можуть виявляти рух і реагувати на непередбачувані ситуації. Розумний будинок значно полегшує життя, роблячи його комфортнішим і безпечнішим.

Промислова автоматизація — це використання технологій для автоматизації виробничих процесів, управління підприємствами та контролю за різними процесами. Тут можуть використовуватися роботи, датчики, системи моніторингу та аналізу для поліпшення ефективності, зниження витрат і забезпечення безпеки на підприємствах.

Важливо, що обидві ці сфери — розумний будинок і промислова автоматизація — все більше спираються на інтернет речей (IoT), де різні пристрої об’єднуються в мережу і можуть взаємодіяти між собою.

2.2 Як TinyML допомагає у розумних будинках

TinyML грає ключову роль у розвитку **розумних будинків**, надаючи можливість використовувати машинне навчання безпосередньо на пристроях з обмеженими ресурсами. Це дозволяє здійснювати швидку та ефективну обробку даних, не потребуючи передачі їх на віддалені сервери чи хмару. Завдяки цьому розумні будинки можуть стати більш автономними, зручними та енергоефективними.

Інтелектуальна автоматизація

TinyML дає змогу пристроям в будинку адаптуватися до звичок і поведінки користувачів. Наприклад, розумні термостати можуть навчатися з часом і самостійно регулювати температуру в приміщенні в залежності від того, коли ви зазвичай вдома або коли ви спите. Це дозволяє зекономити енергію та створити комфортну атмосферу без необхідності вручну налаштовувати температурні режими.

Розпізнавання голосу та звуків

Використовуючи TinyML, розумні колонки або інші аудіопристрої можуть обробляти звукові сигнали безпосередньо на місці, розпізнаючи голоси або звуки. Наприклад, розумний помічник може визначати, коли ви говорите йому команду, чи навіть реагувати на певні звуки, наприклад, дощу або шум в коридорі, що може викликати запуск певних сценаріїв (закрити вікна або увімкнути освітлення).

Моніторинг стану здоров’я

TinyML дозволяє використовувати носимі пристрої для моніторингу здоров’я, такі як фітнес-браслети або розумні годинники, для збору даних про пульс, кроки, рівень кисню в крові та інші показники. Обробка цих даних може здійснюватися на самому пристрої, даючи змогу своєчасно виявляти аномалії, наприклад, підвищення пульсу або надмірну активність, і вживати відповідних заходів (наприклад, сповіщення користувача чи лікаря).

Розпізнавання обличчя та безпека

Пристрої відеоспостереження з TinyML можуть безпосередньо на місці здійснювати розпізнавання обличчя для ідентифікації осіб. Це дозволяє створювати більш безпечне середовище, де лише авторизовані особи мають доступ до певних зон будинку, або де можуть автоматично включатися певні сценарії (наприклад, відкриття дверей для знайомої людини).

Інтелектуальне освітлення

Розумне освітлення може використовувати TinyML для автоматичного налаштування яскравості світла в залежності від часу доби або рівня природного освітлення в приміщенні. Такі пристрої можуть також розпізнавати присутність

людини в кімнаті та автоматично включати або вимикати світло залежно від того, чи знаходиться хтось у приміщенні.

Енергоефективність

TinyML дає змогу пристроям розумного будинку моніторити споживання енергії та адаптувати свої налаштування для зниження витрат. Наприклад, розумні кондиціонери або обігрівачі можуть оптимізувати свою роботу, використовуючи дані про температуру, вологості та час доби, без необхідності постійного з’єднання з хмарними сервісами, що дозволяє знижувати енергоспоживання.

Автономні системи безпеки

Системи безпеки в розумному будинку, такі як датчики руху або розпізнавання звуків (ударів, відкриття дверей), можуть працювати на основі TinyML, обробляючи дані локально та автоматично реагуючи на загрози, наприклад, сповіщаючи власника або автоматично активуючи сигналізацію.

2.3 TinyML у промисловій автоматизації

TinyML в промисловій автоматизації відкриває нові можливості для зниження витрат, підвищення ефективності і безпеки виробничих процесів. Завдяки своїй здатності працювати на малопотужних пристроях, TinyML дозволяє інтегрувати машинне навчання в різноманітні промислові системи без необхідності великих інвестицій у потужні сервери або хмарні інфраструктури.

Однією з головних переваг TinyML у промисловості є можливість здійснювати аналіз даних безпосередньо на місці, що дозволяє швидше реагувати на зміни і покращувати прийняття рішень. Наприклад, у системах моніторингу обладнання можна використовувати TinyML для визначення аномалій у роботі машин, таких як зміни температури, вібрації або рівня шуму, що може сигналізувати про потенційні поломки. Це дозволяє вчасно провести технічне обслуговування або заміну деталей, запобігаючи серйознішим збоям і зупинкам виробництва.

TinyML в прогнозуванні і обслуговуванні обладнання дозволяє автоматично визначати майбутні проблеми, використовуючи дані сенсорів, що моніторять стан машин. Наприклад, у виробничих лініях, де є великий обсяг

обладнання, кожен пристрій може мати свій локальний “інтелект”, який аналізує дані про його роботу і вчасно попереджає про необхідність технічного обслуговування. Це зменшує ймовірність непередбачених поломок і зупинок виробничих процесів, що, в свою чергу, покращує ефективність і знижує витрати.

Інтелектуальні датчики і пристрої на основі TinyML можуть також допомогти в **управлінні якістю** продукції. Наприклад, сенсори, які оцінюють якість виробів на конвеєрних лініях, можуть використовувати алгоритми машинного навчання для виявлення дефектів або відхилень у продукції, таких як неправильні розміри, поверхневі дефекти чи порушення кольору. Це дозволяє значно зменшити кількість відходів і покращити якість кінцевого продукту.

Важливим є й аспект **енергоефективності**. На малопотужних пристроях, які працюють на базі TinyML, можна здійснювати моніторинг споживання енергії в реальному часі та коригувати налаштування для досягнення оптимального використання ресурсів. Наприклад, в автоматизованих системах управління енергоспоживанням на виробництвах можна за допомогою TinyML адаптувати освітлення, опалення, вентиляцію та інші системи для зниження витрат енергії в залежності від навантаження або зовнішніх умов.

Моніторинг безпеки та охорона праці також отримує значну користь від TinyML. Датчики, що виявляють небезпечні гази, високі температури або навіть людські помилки (наприклад, якщо працівник неправильно налаштував обладнання), можуть працювати на основі локальних моделей машинного навчання. Це дозволяє швидко виявляти потенційні загрози без необхідності передавати дані на центральний сервер, що дозволяє миттєво реагувати на ситуацію.

Ще однією цікавою можливістю є **використання TinyML для розпізнавання об’єктів і робототехніки**. Роботи та автоматизовані системи на базі TinyML можуть “бачити” навколишнє середовище завдяки камерам і сенсорам, і на основі цих даних приймати рішення, наприклад, для упаковки товарів, переміщення предметів або для точного керування виробничими лініями.

Завдяки своїй здатності працювати в реальному часі, на малопотужних пристроях і з мінімальним енергоспоживанням, **TinyML** відкриває величезний

потенціал для впровадження розумних, автономних систем в промислову автоматизацію. Ці технології дозволяють значно знизити витрати, покращити ефективність, якість і безпеку виробничих процесів, зберігаючи при цьому контроль над витратами енергії та мінімізуючи ризики для здоров’я та безпеки працівників.

2.4 Переваги інтеграції TinyML у ці системи

Інтеграція **TinyML** у промислові автоматизовані системи має низку значних переваг, які можуть суттєво підвищити ефективність, безпеку та економічність виробничих процесів. Ось деякі з основних переваг:

Енергоефективність

Оскільки TinyML працює на пристроях з обмеженими ресурсами, його впровадження дозволяє знизити споживання енергії, що особливо важливо для промислових систем. Пристрої з TinyML можуть працювати в режимі низького енергоспоживання, що дозволяє зберігати ресурси та скорочувати витрати на енергоспоживання, навіть коли працюють у реальному часі.

Автономність

Однією з найбільших переваг TinyML є здатність працювати автономно без постійного з’єднання з хмарними сервісами або потужними центральними серверами. Це дозволяє системам здійснювати обробку даних і прийняття рішень без необхідності передавати великі обсяги інформації в хмару. Це дуже корисно для промислових середовищ, де є обмежений доступ до Інтернету або з’єднання з хмарою є нестабільним.

Швидка обробка даних у реальному часі

TinyML дозволяє здійснювати обробку даних локально, без затримок, які можуть виникати при передачі інформації до віддалених серверів. Це дає можливість миттєво реагувати на зміни в системах або виробничих процесах, що важливо для забезпечення високої точності і швидкості виявлення аномалій або нештатних ситуацій.

Зниження витрат на інфраструктуру

Завдяки тому, що TinyML дозволяє здійснювати обробку даних на малопотужних пристроях без необхідності в потужних серверах або хмарних інфраструктурах, зменшуються витрати на розгортання і обслуговування IT-інфраструктури. Це дозволяє підприємствам значно зекономити кошти на апаратному забезпеченні та його обслуговуванні.

Покращена безпека і конфіденційність даних

Обробка даних безпосередньо на пристрої зменшує потребу в передачі чутливої інформації на сервери чи в хмару. Це значно покращує рівень конфіденційності і безпеки, оскільки менш ймовірно, що дані будуть виведені з-під контролю підприємства або перехоплені при передачі.

Підвищення продуктивності

TinyML дозволяє автоматизувати численні завдання, які раніше вимагали людської участі або централізованого оброблення даних, що дозволяє знизити час реакції і зменшити людський фактор. Наприклад, системи з TinyML можуть вчасно виявляти дефекти на виробничих лініях, запускати автоматичні налаштування на основі сенсорних даних або навіть здійснювати прогнозування майбутніх поломок.

Масштабованість

TinyML дозволяє легко масштабувати рішення для великих виробничих систем. Оскільки моделі машинного навчання можуть бути оптимізовані для роботи на малопотужних пристроях, не потрібно використовувати дорогі сервери для обробки даних з кожного окремого пристрою. Це робить можливим масове впровадження інтелектуальних рішень без великих капіталовкладень.

Прогнозування та профілактика поломок

Інтеграція TinyML в системи моніторингу та управління обладнанням дозволяє вчасно виявляти несправності та попереджати про необхідність технічного обслуговування. Моделі машинного навчання, які працюють на пристроях, можуть оцінювати стан обладнання в режимі реального часу, визначати аномалії, що свідчать про знос або потенційні поломки, і автоматично реагувати на ці зміни.

Миттєве виявлення аномалій та збоїв

Завдяки вбудованим алгоритмам машинного навчання, TinyML може оперативно виявляти будь-які аномалії в процесах, збої в роботі пристроїв або відхилення від нормальних параметрів. Це дозволяє швидко виявляти проблеми і скорочувати час простоїв або поломок.

Покращення якості продукції

TinyML дозволяє виявляти дефекти на ранніх етапах виробничого процесу, тим самим знижуючи кількість неякісної продукції. Це забезпечує кращу якість кінцевого продукту і зменшує витрати на повернення або виправлення дефектів.

Завдяки всім цим перевагам, інтеграція TinyML у промислові автоматизовані системи дозволяє досягти значних результатів у підвищенні ефективності, економії ресурсів, покращенні безпеки та зниженні витрат на інфраструктуру та обслуговування. Це робить технології TinyML ідеальним рішенням для сучасної промисловості, яка прагне до інновацій та оптимізації своїх виробничих процесів.

2.5 Виклики інтеграції TinyML у ці сфери

Попри численні переваги, інтеграція **TinyML** у промислові автоматизовані системи має ряд **викликів**, які потребують ретельного вирішення для досягнення оптимальних результатів. Ось деякі з основних проблем, з якими стикаються компанії при впровадженні цієї технології:

Обмежені ресурси пристроїв

Однією з головних проблем є обмежені обчислювальні потужності пристроїв, на яких працює TinyML. Оскільки ці пристрої зазвичай мають низьку потужність та пам’ять, вони можуть не справлятися з великими або складними моделями машинного навчання. Це потребує ретельного налаштування моделей, оптимізації коду та навіть обмеження функціоналу для досягнення бажаної ефективності.

Обмеження в обробці великих даних

Важливим аспектом TinyML є обробка даних локально на пристрої, але при цьому є обмеження на кількість даних, які можуть бути оброблені. В промислових умовах, де велика кількість датчиків генерує величезні обсяги даних, локальна обробка може не бути достатньо швидкою або точною. Це вимагає розробки

складних алгоритмів для вибірки релевантних даних, що потребує додаткових витрат часу і ресурсів.

Навчання та оптимізація моделей

Моделі машинного навчання для TinyML повинні бути компактними та ефективними, але при цьому забезпечувати достатню точність. Створення таких моделей вимагає спеціалізованих знань і досвіду в області оптимізації та машинного навчання. Виробникам необхідно витратити час на тренування моделей на обмежених даних і тестування їх на реальних умовах, що може бути складним і трудомістким процесом.

Сумісність з існуючими системами

Інтеграція TinyML у вже існуючі промислові системи може бути складною через необхідність адаптувати старі платформи до нових вимог. Наприклад, багато промислових пристроїв, що працюють на основі традиційних систем, можуть не підтримувати потрібні стандарти або технології для роботи з TinyML. Це може вимагати значних змін у програмному забезпеченні або апаратному забезпеченні, що збільшує витрати і час на впровадження.

Безпека даних

Незважаючи на те, що TinyML дозволяє здійснювати обробку даних локально і зменшує необхідність передачі даних на хмарні сервіси, промислові системи все одно стикаються з проблемами безпеки. Дані, що обробляються на пристроях, можуть бути вразливими до зломів або маніпуляцій, тому вимагається впровадження ефективних механізмів шифрування, автентифікації та захисту даних.

Інтеграція в розподілені мережі

У промислових системах часто існує велика кількість розподілених датчиків і пристроїв, що мають бути інтегровані в одну систему для ефективної роботи. Впровадження TinyML в таку розподілену мережу потребує ретельного проектування для забезпечення синхронізації, взаємодії між пристроями та забезпечення безперебійної роботи системи.

Обмеження в масштабуванні

Хоча TinyML дозволяє використовувати машинне навчання на малопотужних пристроях, масштабування таких рішень на великі промислові об’єкти може бути складним. Кожен пристрій потребує своєї окремої моделі або налаштування, що

може збільшити складність управління великими мережами пристроїв, а також ускладнити їх оновлення і технічне обслуговування.

Програмне забезпечення та стандарти

Системи TinyML вимагають наявності специфічного програмного забезпечення, а також чітких стандартів і протоколів для забезпечення сумісності між пристроями різних виробників. Відсутність єдиних стандартів може створити додаткові труднощі у впровадженні TinyML у складні промислові середовища, де можуть використовуватися різні моделі і технології.

Проблеми з навчанням персоналу

Інтеграція TinyML вимагає, щоб працівники, які займаються обслуговуванням таких систем, мали достатньо високий рівень кваліфікації. Вони повинні розуміти принципи роботи машинного навчання та мати навички роботи з новими технологіями. Підготовка такого персоналу може вимагати додаткових інвестицій у навчання та тренінги.

Вартість впровадження

Впровадження TinyML, особливо на великих підприємствах, може вимагати значних початкових витрат на модернізацію інфраструктури, включаючи придбання нових датчиків, пристроїв і програмного забезпечення. Хоча в довгостроковій перспективі ці витрати можуть окупитися завдяки зниженню енергоспоживання та збільшенню ефективності, початкові витрати можуть бути високими.

Всі ці виклики вимагають комплексного підходу та ретельного планування на етапі інтеграції **TinyML** в промислові автоматизовані системи. Виробники повинні враховувати ці аспекти, щоб забезпечити ефективне впровадження та оптимальну роботу систем на основі TinyML.

3 ТЕХНІЧНА РЕАЛІЗАЦІЯ ТА МОДЕЛЮВАННЯ ON-DEVICE LEARNING ТА TINYML

Навчання на пристроях, зокрема в контексті **TinyML**, вимагає використання алгоритмів, які здатні працювати в умовах обмежених ресурсів. Це означає, що алгоритми повинні бути не тільки точними, але й ефективними з точки зору обчислювальної потужності та пам’яті. Ось кілька основних методів і алгоритмів, які можуть бути використані для **навчання на пристроях**:

1. Лінійна регресія

Лінійна регресія є одним з найпростіших і найбільш ефективних алгоритмів для навчання на пристроях, оскільки її обчислювальні вимоги мінімальні. Вона підходить для завдань, де є лінійний зв’язок між входами і виходами. Лінійна регресія намагається знайти таку модель, яка мінімізує різницю між передбаченням та реальними значеннями. Завдяки простоті, цей алгоритм може бути ефективно реалізований на пристроях з обмеженими обчислювальними можливостями.

2. Дерева рішень

Дерева рішень — це алгоритм, який за допомогою структури дерева намагається класифікувати або регресувати вхідні дані. Дерево рішень працює шляхом прийняття послідовних рішень (розгалужень) на основі значень вхідних параметрів, що дозволяє ефективно поділяти простір для класифікації або прогнозування. Для **TinyML** можна використовувати спрощені варіанти дерев рішень, такі як **пошук по бінарних деревах** або використання обмеженої глибини дерева для економії пам’яті і обчислювальних ресурсів.

3. Логістична регресія

Логістична регресія є покращеною формою лінійної регресії, яку часто використовують для класифікаційних задач. Вона дає змогу оцінити ймовірність того, що вхідні дані належать до певного класу (наприклад, ймовірність того, що зображення містить об’єкт певного типу). Логістична регресія є дуже ефективною для простих класифікаційних задач і також може бути використана в обмежених умовах, таких як пристрої з малим обсягом пам’яті.

4. Нейронні мережі (включаючи компактні моделі)

Нейронні мережі, зокрема **глибокі нейронні мережі (DNN)**, є потужними алгоритмами для розв’язання складних задач, але вони також можуть бути досить вимогливими до обчислювальних потужностей. Однак для **TinyML**існують оптимізовані нейронні мережі, які можуть працювати навіть на малопотужних пристроях. Для цього використовуються такі стратегії:

Просторова та часові згортки (Convolutional Neural Networks, CNN): використовується для обробки зображень або сигналів, але з обмеженнями на розмір мережі та кількість параметрів.

Мережі з малою кількістю параметрів (наприклад, MobileNet): ці моделі мають зменшену кількість параметрів і забезпечують баланс між швидкістю роботи та точністю.

Мережі з прямих зв’язків (Feed-forward Networks): менш складні, мають менше параметрів і можуть бути оптимізовані для простих задач.

У **TinyML** часто використовуються зменшені нейронні мережі, такі як **SqueezeNet** або **TinyYolo**, що дозволяє використовувати переваги нейронних мереж навіть на пристроях з низькими вимогами до пам’яті та потужності.

5. Метод опорних векторів (SVM)

Алгоритм **методу опорних векторів** є ефективним для задач класифікації, особливо при наявності малих обсягів даних. Він знаходить гіперплощину, яка максимально відокремлює дані різних класів. Оскільки цей метод потребує значно менше обчислювальних ресурсів, ніж глибокі нейронні мережі, він може бути корисним для завдань, де обмежена потужність пристроїв.

6. K-найближчих сусідів (K-NN)

Алгоритм **K-NN** є простим і ефективним для задач класифікації та регресії. Він полягає в тому, щоб для нового зразка знаходити найближчі до нього точки з тренувальних даних і визначати його клас на основі їхніх класів. Цей метод добре підходить для невеликих, обмежених за ресурсами пристроїв, оскільки він не потребує тренування моделі, але має високі вимоги до пам’яті для зберігання даних.

7. Кластеризація (наприклад, K-середніх)

Алгоритм **кластеризації K-середніх** використовується для поділу даних на групи або кластери, що мають подібні характеристики. Він часто використовується в задачах сегментації даних і може бути корисним для аналізу даних у реальному часі на пристроях.

8. Підсилення моделей (Boosting та Bagging)

Алгоритми **підсилення** (Boosting), такі як **AdaBoost** або **Gradient Boosting**, можуть використовуватися для поліпшення результатів класифікації чи регресії, комбінуючи кілька слабких моделей для створення сильної моделі. Для **TinyML** можуть бути використані спрощені варіанти таких методів, що дозволяють отримувати ефективні результати з мінімальними витратами пам’яті.

9. Періодичні та часові моделі (RNN, LSTM)

Для задач, пов’язаних з аналізом часових рядів або послідовних даних (наприклад, сенсорних даних або даних з камер спостереження), можна використовувати рекурентні нейронні мережі (RNN) або довгої короткочасної пам’яті (LSTM). Хоча вони зазвичай є досить обчислювально складними, для TinyML можуть бути адаптовані легші варіанти, що забезпечують можливість обробки послідовних даних в реальному часі.

Кожен з цих методів має свої сильні та слабкі сторони залежно від конкретної задачі та умов використання. Вибір оптимального алгоритму залежить від обмежень пристроїв, складності задачі та доступних ресурсів для обробки даних.

3.1 Розробка прототипу системи розумного будинку або промислової автоматизації з On-Device Learning на основі TinyML

Розробка прототипу системи розумного будинку або промислової автоматизації з On-Device Learning на основі TinyML є цікавою та перспективною задачею, яка включає в себе вибір відповідних компонентів, оптимізацію моделей машинного навчання для роботи на малопотужних пристроях і забезпечення ефективної роботи всієї системи в реальному часі. Ось як можна розробити прототип такої системи.

1. Вибір пристроїв та мікроконтролерів

Для реалізації прототипу системи з **TinyML** важливо обрати пристрої з достатньою обчислювальною потужністю, але з обмеженими вимогами до пам’яті та енергоспоживання. У цьому контексті використовуються мікроконтролери з можливістю локального виконання моделей машинного навчання.

Ось кілька варіантів мікроконтролерів та платформ, які можна використовувати:

Arduino Nano 33 BLE Sense — це мікроконтролер на основі **ARM Cortex-M4**, який оснащений сенсорами для вимірювання температури, вологості, освітленості, а також акселерометром і гіроскопом. Його можна використовувати для простих задач з машинним навчанням, таких як класифікація чи виявлення аномалій на основі сенсорних даних.

Raspberry Pi Pico з мікроконтролером **RP2040** — ще один вибір для низькобюджетних рішень. Він має достатньо обчислювальних ресурсів для виконання моделей машинного навчання, включаючи прості нейронні мережі.

ESP32 — потужний мікроконтролер з двоядерним процесором **Xtensa LX6**, який забезпечує достатню потужність для роботи з більш складними моделями **TinyML**. Платформа ESP32 підтримує Wi-Fi і Bluetooth, що дозволяє інтегрувати пристрій у мережу **IoT**. **NVIDIA Jetson Nano** — хоча це вже більше платформа для **edge computing**, вона може використовуватися для більш складних задач у промисловій автоматизації, де обчислювальні потужності необхідні для виконання великих моделей машинного навчання.

2. Вибір сенсорів

Сенсори є основними компонентами, які збирають дані для подальшої обробки. В залежності від задачі (розумний будинок чи промислова автоматизація) використовуються різні типи сенсорів:

Температурні та вологісні сенсори (наприклад, **DHT22**, **BME280** або **SHT30**) — вони можуть використовуватись для моніторингу умов в будинку або на виробничих підприємствах.

Датчики руху та PIR сенсори (наприклад, **HC-SR501**) — для систем безпеки в розумному будинку або для виявлення аномалій у промислових процесах.

Датчики освітленості (наприклад, **BH1750**) — для автоматизації освітлення в будинку.

Вібраційні сенсори (наприклад, **SW-420**) — використовуються для виявлення вібрацій у промислових установках, що може свідчити про несправність обладнання.

Газові датчики (наприклад, **MQ series**) — застосовуються для моніторингу якості повітря в будинках чи на підприємствах.

Мікрофони та звукові сенсори (наприклад, **MAX9814**) — використовуються для збору аудіоданих, що можуть бути аналізовані за допомогою **TinyML** для виявлення певних звуків чи аномалій.

3.3 Реалізація On-Device Learning

Після того як вибрані необхідні пристрої та сенсори, наступним кроком є **розробка моделей машинного навчання** та їх навчання безпосередньо на пристроях. Ось як це можна реалізувати:

Збір даних: Першим кроком є налаштування сенсорів для збору даних. Наприклад, для системи розумного будинку можна використовувати температурні та вологісні датчики для моніторингу умов. У промисловості можна використовувати вібраційні сенсори для виявлення несправностей на виробничих лініях.

Попереднє навчання на потужному пристрої: На початковому етапі модель машинного навчання можна тренувати на потужних комп’ютерах, використовуючи зібрані дані. Наприклад, модель може бути нейронною мережею або деревом рішень, залежно від типу задачі.

Оптимізація моделі для TinyML: Після тренування, модель оптимізується для роботи на малопотужних пристроях. Це включає в себе:

Квантування — зменшення точності даних, щоб зменшити розмір моделі.

Прайонізація — видалення неважливих ваг нейронної мережі, що знижує кількість операцій.

Згортання — використання методів згортки для зменшення кількості параметрів, що зберігаються.

Розгортання на пристрої: Оптимізована модель завантажується на мікроконтролер. Пристрій використовує цю модель для **локальної обробки даних**, що дозволяє здійснювати розпізнавання чи прогнозування в реальному часі. Наприклад, розумний будинок може автоматично регулювати температуру чи освітлення, в залежності від отриманих даних.

3.2 Прототип системи розумного будинку

Метою цього прототипу є створення системи розумного будинку, яка використовує технології TinyML для автоматизації управління температурою та освітленням. Система буде приймати рішення на основі даних, зібраних сенсорами, і передбачень, зроблених за допомогою моделей машинного навчання, щоб забезпечити комфорт і енергоефективність у будинку. Вона має здатність адаптуватися до змін в середовищі, забезпечуючи оптимальне керування будинковими пристроями.

Для реалізації цієї системи вибрано кілька ключових компонентів. Одним з них є мікроконтролер, наприклад, Arduino Nano 33 BLE Sense або ESP32. Arduino Nano 33 BLE Sense оснащений вбудованими сенсорами для збору даних, таких як температура, вологість та освітленість, що робить його зручним для простих завдань. Якщо ж потрібне підключення до Wi-Fi та обмін даними між кількома пристроями, використовують ESP32. Для збирання даних використовуються такі сенсори, як температурний датчик BME280, датчик освітленості BH1750 та PIR сенсори для виявлення руху. Ці сенсори дозволяють автоматично керувати освітленням та температурою залежно від змін у середовищі. Активаційні пристрої, такі як реле для управління освітленням і терморегулятори для контролю опалення та кондиціонування, забезпечують ефективне використання енергії.

Система працює за принципом локального оброблення даних. Кожен пристрій у будинку (сенсори, реле, терморегулятори) інтегрується у єдину систему через Wi-Fi або Bluetooth, залежно від вибору мікроконтролера. Мікроконтролер збирає дані від сенсорів і передає їх до моделі машинного навчання, що працює безпосередньо на пристрої. За допомогою цих моделей, система може прогнозувати оптимальні налаштування для управління освітленням і температурою, з урахуванням таких факторів, як зміна освітленості протягом дня чи зниження температури вночі.

Перший етап реалізації прототипу включає в себе збір даних з сенсорів. Для цього можна записувати показники протягом кількох днів чи тижнів для подальшого аналізу. Наступним кроком є навчання моделі, де використовуються зібрані дані для створення алгоритмів, таких як лінійна регресія або дерева рішень, які допоможуть передбачити оптимальні налаштування для керування пристроями. Після цього модель оптимізується для роботи на малопотужних пристроях шляхом мініатюризації та стиснення.

Коли модель буде навчена та оптимізована, її завантажують на мікроконтролер. Цей мікроконтролер використовуватиме вхідні дані для управління пристроями, такими як освітлення чи терморегуляція. Для цього також необхідно програмувати реле та терморегулятори, щоб вони працювали на основі передбачень, отриманих від моделі. Останнім етапом є тестування системи в реальних умовах будинку і коригування моделі на основі зворотного зв’язку.

Для інтеграції системи в більш складну мережу розумного будинку, можна використовувати платформи типу Home Assistant або OpenHAB. Ці системи дозволяють інтегрувати різноманітні пристрої та взаємодіяти з ними через MQTT або HTTP. Для моніторингу й оптимізації роботи можна створити мобільний додаток або веб-інтерфейс, де користувачі можуть вручну налаштовувати параметри, а також отримувати зворотний зв’язок від системи.

Завдяки TinyML, система буде здатна обробляти дані безпосередньо на пристрої, що дозволяє зменшити навантаження на зовнішні сервери і забезпечує швидке реагування системи в реальному часі. Це дасть можливість значно знизити

енергоспоживання, забезпечуючи при цьому високу ефективність та комфорт в управлінні температурою і освітленням у будинку.

3.3 Прототип для промислової автоматизації

У промисловій автоматизації **TinyML** може бути використано для моніторингу обладнання та виявлення аномалій. Наприклад:

• Вибираємо **ESP32** з вбудованими Wi-Fi і Bluetooth для підключення до мережі.

• Використовуємо **вібраційні датчики** для моніторингу робочих механізмів.

• Навчаємо модель на великих даних про нормальну роботу обладнання, щоб вона могла виявляти відхилення (наприклад, надмірні вібрації).

• Оптимізована модель завантажується на пристрій, який локально аналізує вібрації і, у разі виявлення аномалії, сповіщає оператора про необхідність перевірки або ремонту.

Розробка прототипу системи на основі **TinyML** для розумного будинку або промислової автоматизації вимагає ретельного вибору відповідних компонентів — пристроїв, сенсорів та мікроконтролерів, а також оптимізації моделей машинного навчання для ефективної роботи на малопотужних пристроях. В результаті, такі системи можуть забезпечити автономність, низьке енергоспоживання та швидку реакцію в реальному часі, що робить їх дуже перспективними для широкого застосування.

Алгоритм обробки даних та застосування моделей машинного навчання на пристрої (TinyML)

Розглянемо загальний алгоритм обробки даних та застосування моделей машинного навчання на пристрої. Спочатку відбувається збір даних. Пристрій збирає інформацію з різних джерел: сенсорів температури, вологості, руху, камери, мікрофону та інших. Ці дані можуть бути як аналоговими, так і цифровими.

Наступний етап - попередня обробка даних. Тут важливо очистити дані від шуму та артефактів, нормалізувати або масштабувати їх до єдиного діапазону, а

також вилучити ознаки, тобто перетворити необроблені дані в набір релевантних характеристик, які будуть використовуватись моделлю.

Далі відбувається вибір моделі машинного навчання. Важливо обрати модель, яка найкраще підходить для конкретної задачі та обмежень пристрою, таких як обчислювальна потужність, пам'ять та енергоспоживання. Прикладами таких моделей можуть бути лінійна регресія, дерева рішень, SVM, нейронні мережі, зокрема згорткові та рекурентні.

Навчання моделі може відбуватися як безпосередньо на пристрої, якщо він має достатньо ресурсів, так і попередньо на потужних серверах з подальшою оптимізацією та завантаженням на пристрій

Після навчання моделі настає етап її застосування, або інференсу. Модель отримує оброблені дані на вхід, виконує обчислення та видає результат: прогноз, класифікацію, виявлення аномалій тощо.

Отримані результати потребують постобробки. На цьому етапі відбувається інтерпретація результатів, їх візуалізація у вигляді графіків або діаграм, а також прийняття рішень на основі цих результатів, наприклад, активація сигналізації або зміна режиму роботи пристрою.

За необхідності модель може бути оновлена, якщо пристрій збирає нові дані. Це дозволяє покращити точність та адаптувати модель до змін.

Технічна реалізація такого алгоритму включає вибір апаратної платформи, такої як мікроконтролери ARM Cortex-M або одноплатні комп'ютери Raspberry Pi, а також програмного забезпечення, наприклад, фреймворків TinyML (TensorFlow Lite Micro, Edge Impulse, uTensor) та мов програмування C/C++ або Python. Важливим аспектом є оптимізація моделі, яка включає квантизацію, прунінг та дистиляцію знань для зменшення розміру та обчислювальної складності моделі.

Як приклад, можна розглянути задачу виявлення аномалій у роботі промислового обладнання. В цьому випадку дані збираються з датчиків вібрації, температури та тиску. Після попередньої обробки даних, яка включає фільтрацію шуму та нормалізацію, застосовується навчена модель One-Class SVM. Якщо модель класифікує дані як аномальні, активується сигнал тривоги.

Цей алгоритм є загальним, і ви можете адаптувати його до конкретної задачі та пристрою у вашій дипломній роботі. Не забудьте також описати особливості реалізації алгоритму на обраній вами апаратній платформі та програмному забезпеченні.

4 СТВОРЕННЯ ТА НАВЧАННЯ МОДЕЛІ TINYML

1 Вибір обладнання та середовища розробки

Для тестування коду можливо використовувати персональний комп’ютер із операційною системою Mac, Linux або Windows. Однак, для повноцінного розгортання моделі необхідний один із наступних вбудованих пристроїв:

Arduino Nano 33 BLE Sense

SparkFun Edge

ST Microelectronics STM32F746G Discovery kit

Для створення моделі машинного навчання ми будемо використовувати такі програмні засоби:

Python

TensorFlow

Google Colaboratory – хмарне інтерактивне середовище для експериментів із кодом Python

Всі ці інструменти є загальнодоступними та безкоштовними для використання.

2 Процес створення моделі

Протягом цього розділу буде виконано наступні етапи:

1. Отримання простого набору даних.
2. Навчання моделі глибокого навчання.
3. Оцінка продуктивності моделі.
4. Конвертація моделі для роботи на пристрої.
5. Написання коду для виконання висновків на мікроконтролері.
6. Компіляція коду у двійковий файл.
7. Розгортання двійкового файлу на мікроконтролері.

Усі програмні компоненти, що використовуються, доступні у відкритому репозиторії GitHub TensorFlow, що забезпечує прозорість та повторюваність експерименту.

3 Завдання моделі та її застосування

Основне завдання моделі – передбачати значення математичної функції синуса. Незважаючи на те, що синус можна легко обчислити математично, створення нейронної мережі для цього завдання дозволяє продемонструвати ключові принципи машинного навчання.

Наступним етапом є запуск моделі на апаратному пристрої. Оскільки синусоїда є плавною кривою з періодичними коливаннями, вона добре підходить для створення візуальних ефектів. У нашому випадку вихідні дані моделі будуть використані для керування світлодіодами або графічною анімацією, що дозволить на практиці продемонструвати взаємодію машинного навчання та апаратного забезпечення.

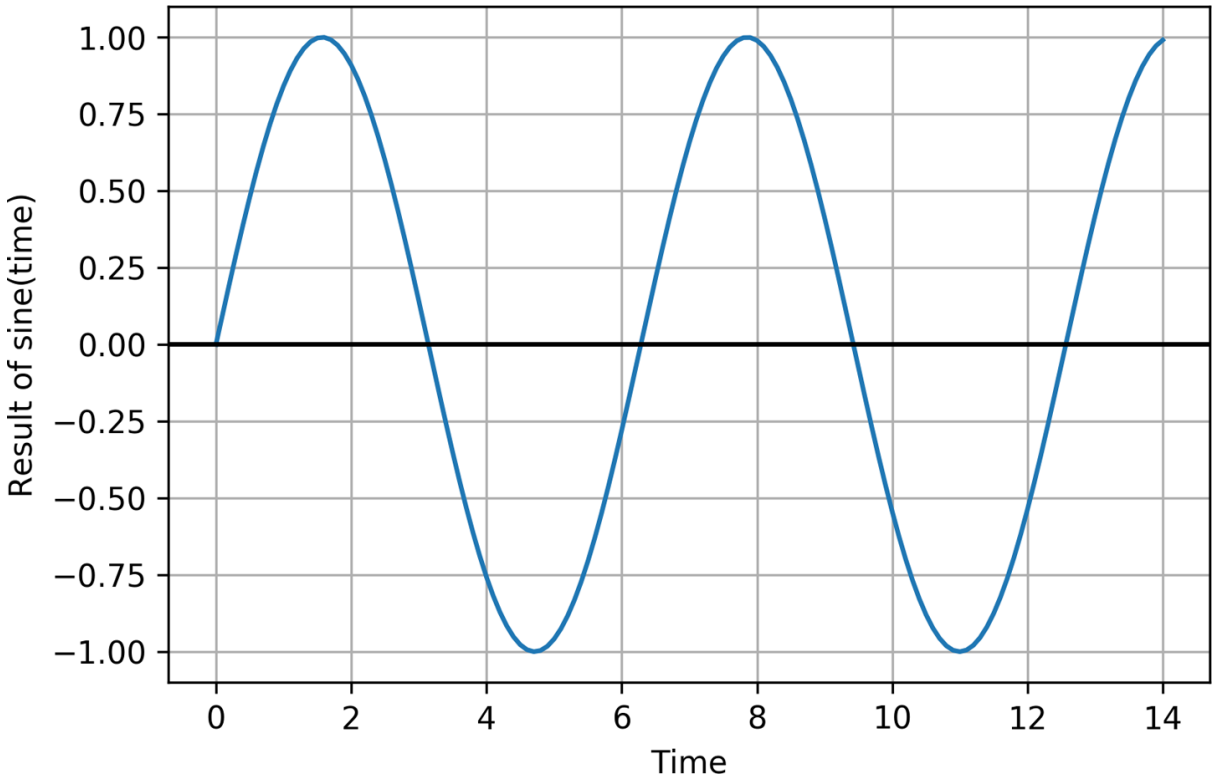


Рисунок 4-1. Синусоїда.

4 Візуалізація та розгортання на різних пристроях

Онлайн доступна анімована GIF-зображення, що демонструє, як цей код керує світлодіодами SparkFun Edge. На рисунку 4-2 зображено статичний кадр із цієї анімації, де видно декілька активних світлодіодів пристрою. Хоча це не є безпосередньо корисним застосуванням машинного навчання, у рамках концепції "Hello World" цей приклад є простим і наочним для розуміння основних принципів.

Після перевірки працездатності базового коду ми розгорнемо його на трьох різних пристроях:

- SparkFun Edge
- Arduino Nano 33 BLE Sense
- ST Microelectronics STM32F746G Discovery kit

Примітка: Оскільки TensorFlow є відкритим проєктом, що активно розвивається, можливі незначні відмінності між кодом, представленим у цій роботі, та версією, розміщеною онлайн. Однак, навіть якщо окремі рядки коду зміняться, основні принципи залишаться незмінними.

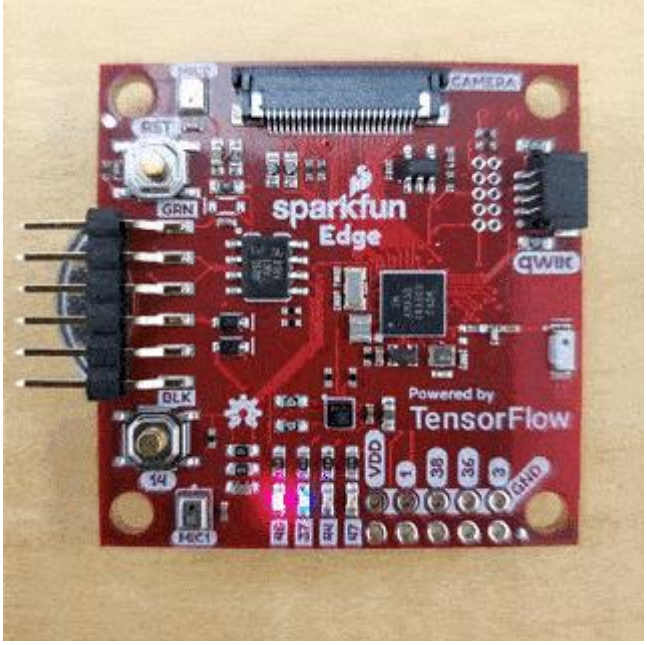


Рисунок 4-2. Код, що виконується на SparkFun Edge.

Інструменти для машинного навчання

Для розробки компонентів машинного навчання в цьому проєкті ми використовуватимемо ті ж самі інструменти, що й професійні розробники. У цьому підрозділі розглянемо їх детальніше.

Python та Jupyter Notebooks

Python є однією з найпопулярніших мов програмування серед дослідників та інженерів у сфері машинного навчання. Він простий у вивченні, має велику кількість бібліотек для роботи з даними та математичними обчисленнями. Значна

частина сучасних досліджень у сфері глибокого навчання проводиться саме за допомогою Python.

Однією з особливостей Python є інтеграція з Jupyter Notebooks — форматом документів, який дозволяє поєднувати текст, графіку та виконуваний код. Jupyter широко використовується для опису, аналізу та дослідження алгоритмів машинного навчання.

У рамках цього проєкту ми створимо модель у середовищі Jupyter Notebook, що дозволить нам наочно візуалізувати процес навчання та аналізувати точність моделі.

Google Colaboratory

Для запуску нашого блокнота ми скористаємося сервісом Google Colaboratory (Colab). Це безкоштовне онлайн-середовище для роботи з Jupyter Notebook, яке забезпечує зручний доступ до потужного апаратного забезпечення Google.

Зазвичай для роботи з Jupyter Notebook потрібно встановлювати на локальному комп’ютері безліч залежностей, що може ускладнювати налаштування середовища. Крім того, запуск складних моделей машинного навчання вимагає значних обчислювальних ресурсів. Використання Colab дозволяє уникнути цих труднощів, оскільки обчислення виконуються на серверах Google. Крім того, Colab підтримує прискорене виконання моделей за допомогою графічних процесорів (GPU) та тензорних процесорів (TPU).

TensorFlow та Keras

TensorFlow — це набір інструментів для створення, навчання та розгортання моделей машинного навчання. Спочатку розроблений у Google, TensorFlow зараз є відкритим проєктом із тисячами розробників по всьому світу. Це одна з найпопулярніших платформ у сфері машинного навчання.

У цьому розділі ми використовуватимемо Keras — високорівневий API TensorFlow, який спрощує процес створення та навчання глибоких нейронних мереж. Для розгортання моделі на вбудованих пристроях ми застосуємо TensorFlow

Lite — набір інструментів, що дозволяє оптимізувати моделі для роботи на малопотужному обладнанні.

4.3 Процес створення моделі

Протягом цього розділу буде виконано наступні етапи:

1. Отримання простого набору даних.
2. Навчання моделі глибокого навчання.
3. Оцінка продуктивності моделі.
4. Конвертація моделі для роботи на пристрої.
5. Написання коду для виконання висновків на мікроконтролері.
6. Компіляція коду у двійковий файл.
7. Розгортання двійкового файлу на мікроконтролері.

Завантаження блокнота

Для виконання коду необхідно завантажити відповідний Jupyter Notebook. Після відкриття сторінки слід натиснути кнопку "Run in Google Colab" (див. рисунок 4-3). Це дозволить скопіювати блокнот із GitHub у середовище Colab, після чого ви зможете його виконати та редагувати відповідно до потреб проєкту.

Усі програмні компоненти, що використовуються, доступні у відкритому репозиторії GitHub TensorFlow, що забезпечує прозорість та повторюваність експерименту.

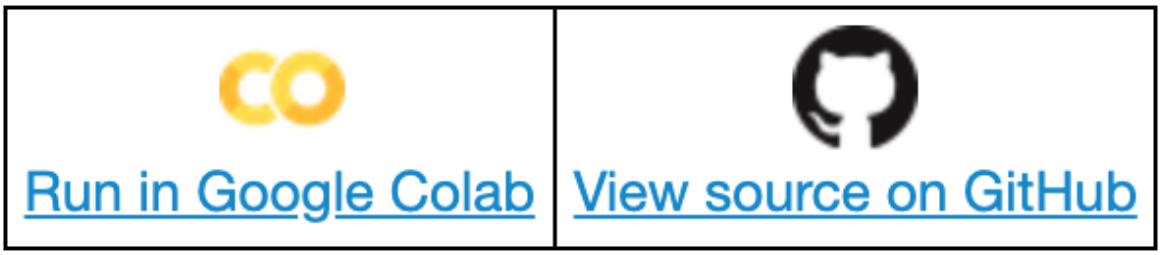


Рисунок 4-3. Кнопка "Запустити в Google Colab".

Вирішення проблем із завантаженням блокнота

Іноді при відкритті Jupyter Notebook на GitHub може з’явитися повідомлення "Sorry, something went wrong. Reload?". Це відома проблема, пов’язана з роботою

GitHub. Якщо ви зіткнулися з нею, можна відкрити блокнот безпосередньо в Colab, використовуючи наступний спосіб:

1. Скопіюйте частину URL-адреси блокнота після https://github.com. Наприклад: tensorflow/tensorflow/blob/master/tensorflow/lite/micro/examples/hello_world/create_sine_model.ipynb
2. Додайте перед нею https://colab.research.google.com/github, щоб отримати повний URL: https://colab.research.google.com/github/tensorflow/tensorflow/blob/master/tensorflow/lite/micro/examples/hello_world/train/train_hello_world_model.ipynb
3. Вставте цей URL у веб-браузер та відкрийте блокнот безпосередньо в Colab.

Усі програмні компоненти, що використовуються, доступні у відкритому репозиторії GitHub TensorFlow, що забезпечує прозорість та повторюваність експерименту.

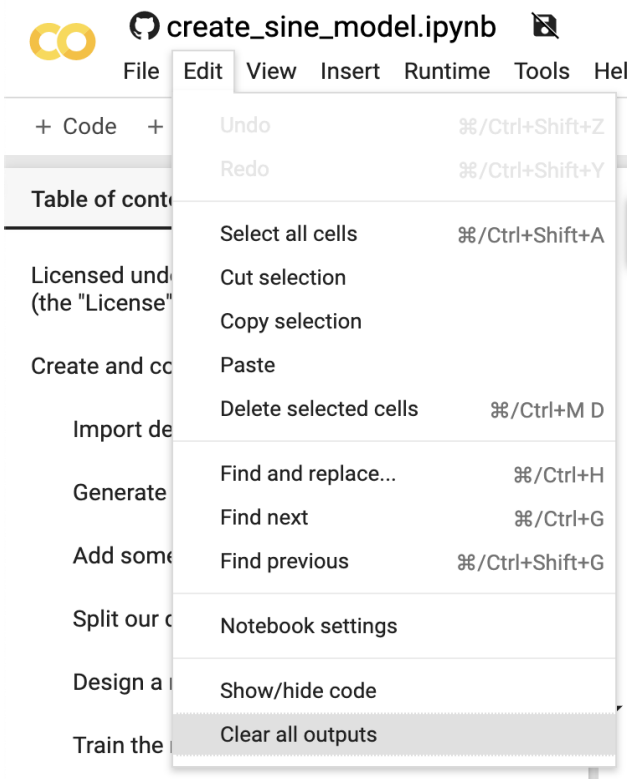


Рисунок 4.4 Опція "Очистити всі вихідні дані"

Імпорт необхідних бібліотек

Першим кроком у реалізації моделі є імпорт необхідних залежностей. У середовищі Jupyter Notebook код та текст організовані у вигляді окремих комірок. Виділяють два основні типи комірок:

• Кодові комірки, які містять виконуваний код мовою Python.

• Текстові комірки, які використовуються для форматованого тексту, пояснень та документації.

Перший кодовий блок, розташований у розділі «Імпорт необхідних бібліотек», містить перелік бібліотек, що будуть використані для навчання та конвертації моделі. Наведений нижче код здійснює їх підключення:

```
# TensorFlow is an open source machine learning library
!pip install tensorflow==2.0
import tensorflow as tf
# NumPy is a math library
import numpy as np
# Matplotlib is a graphing library
import matplotlib.pyplot as plt
# math is Python's math library
import math
```

Імпорт бібліотек у Python

Оператор import у мові Python дозволяє завантажувати необхідні бібліотеки для подальшого використання у програмному коді. Розглянутий вище код виконує такі дії:

• Встановлює бібліотеку TensorFlow 2.0 за допомогою pip – стандартного менеджера пакетів для Python.

• Імпортує такі бібліотеки:

• TensorFlow – фреймворк для машинного навчання.

• NumPy – бібліотеку для виконання числових обчислень.

• Matplotlib – інструмент для візуалізації даних.

• math – стандартну бібліотеку Python для математичних операцій.

При імпорті бібліотек їм можна призначати псевдоніми для спрощення доступу до їх функцій. Наприклад:

• import numpy as np означає, що всі функції бібліотеки NumPy будуть доступні через скорочену назву np.

• import matplotlib.pyplot as plt дозволяє звертатися до функцій побудови графіків через plt.

Виконання коду у Jupyter Notebook здійснюється шляхом натискання кнопки запуску у верхньому лівому куті активної комірки. Для вибору комірки необхідно клікнути на неї мишею. На рисунку 4-5 зображено приклад вибраної комірки.

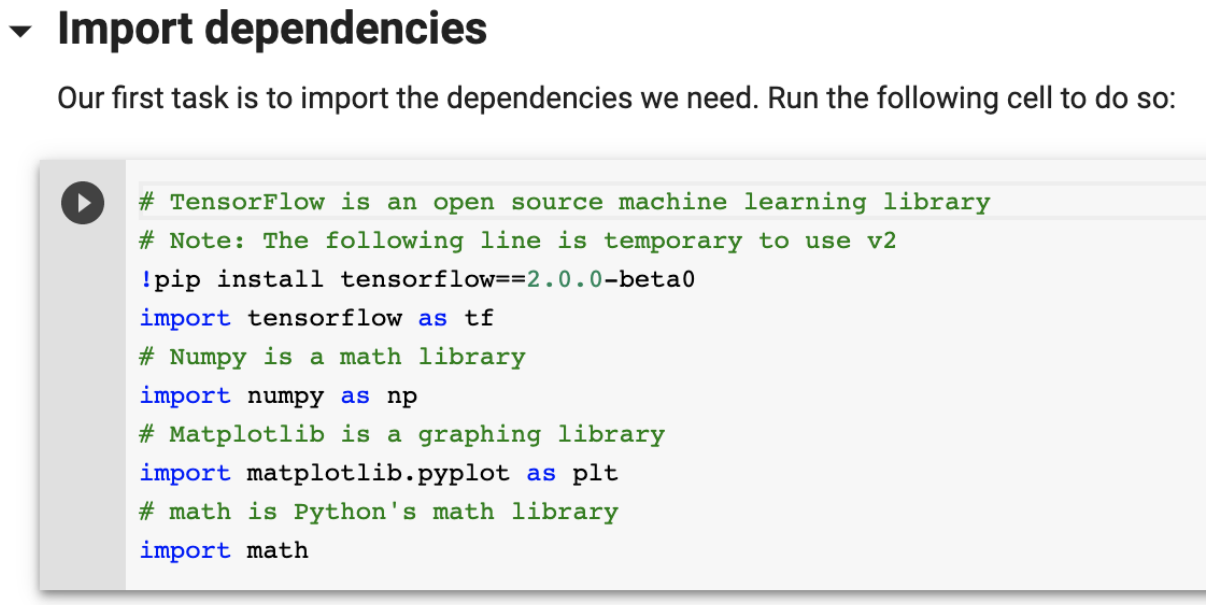


Рисунок 4-5. Комірка "Імпорт залежностей" у вибраному стані.

Виконання коду та встановлення залежностей

Для запуску коду в середовищі Jupyter Notebook необхідно натиснути кнопку виконання, яка розташована у верхньому лівому куті комірки з кодом. Під час виконання кнопка буде анімовано відображати процес, що позначається у вигляді обертового кола (див. рисунок 4-6).

Після запуску коду розпочнеться встановлення необхідних бібліотек. У процесі інсталяції виводитимуться повідомлення про хід виконання, а після успішного завершення ви побачите наступний рядок:

Successfully installed tensorboard-2.0.0 tensorflow-2.0.0 tensorflow-estimator-2.0.0

```
# TensorFlow is an open source machine learning library
# Note: The following line is temporary to use v2
!pip install tensorflow==2.0.0-beta0
import tensorflow as tf
# Numpy is a math library
import numpy as np
# Matplotlib is a graphing library
import matplotlib.pyplot as plt
# math is Python's math library
import math
```

```
...  Collecting tensorflow==2.0.0-beta0
```

Рисунок 4-6. Комірка "Імпорт залежностей" в стані виконання.

Виконання комірок у Google Colaboratory

Після запуску комірки у середовищі **Google Colaboratory** у її верхньому лівому куті з’явиться номер виконання. Якщо комірка більше не є активною, відображатиметься число **1**, як показано на рисунку 4-7.

Це число є лічильником запусків і збільшується щоразу, коли комірку виконують повторно. Таким чином, користувач може відстежувати кількість виконань кожної комірки та їхню послідовність у межах блокнота.

```
[1]  # TensorFlow is an open source machine learning library
     # Note: The following line is temporary to use v2
     !pip install tensorflow==2.0.0-beta0
     import tensorflow as tf
     # Numpy is a math library
     import numpy as np
     # Matplotlib is a graphing library
     import matplotlib.pyplot as plt
     # math is Python's math library
     import math
```

Рисунок 4-7. Лічильник запусків осередку в верхньому лівому куті.

Генерація даних

Глибокі навчальні мережі вчаться моделювати патерни в основних даних. Як ми згадували раніше, ми будемо тренувати мережу для моделювання даних, що генеруються синусоїдальною функцією. Це призведе до моделі, яка зможе приймати значення x і передбачати його синус, y.

Перш ніж рухатися далі, нам потрібні деякі дані. У реальному світі ми могли б збирати дані з датчиків і виробничих журналів. Однак для цього прикладу ми використовуємо простий код для генерації набору даних.

Наступний осередок — це те, де це станеться. Наш план полягає в тому, щоб згенерувати 1 000 значень, які представляють випадкові точки вздовж синусоїдальної хвилі. Давайте подивимося на рисунок 4-8, щоб згадати, як виглядає синусоїдальна хвиля.

Кожен повний цикл хвилі називається її періодом. З графіка ми можемо побачити, що повний цикл завершується приблизно кожні шість одиниць на осі x. Насправді період синусоїдальної хвилі — це $2 \times \pi$, або 2π.

Щоб ми мали повний набір даних для тренування на основі синусоїди, наш код згенерує випадкові значення x від 0 до 2π. Потім він обчислить синус для кожного з цих значень.

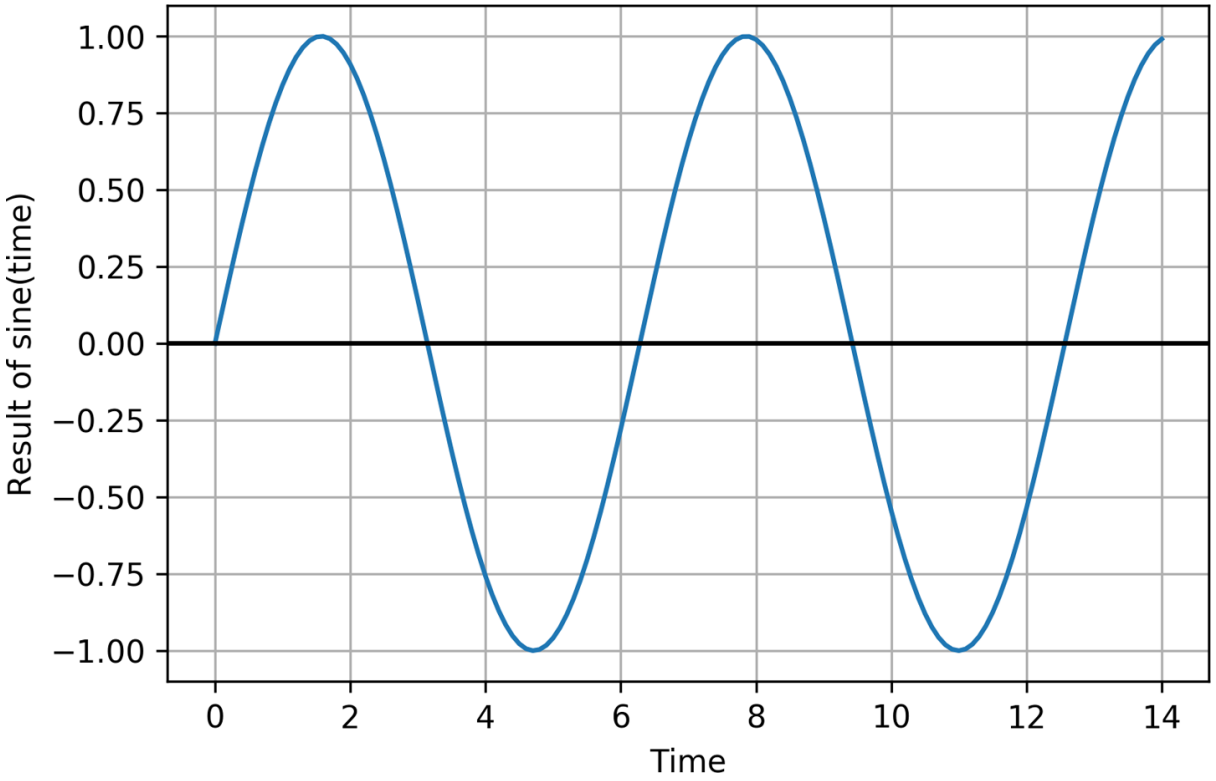


Рисунок 4-8. Синусоїда

Ось повний код для цього осередку, який використовує NumPy (np, який ми імпортували раніше) для генерування випадкових чисел і обчислення їхнього синуса:

```
# We'll generate this many sample datapoints
SAMPLES = 1000
# Set a "seed" value, so we get the same random numbers each time we run this
# notebook. Any number can be used here.
SEED = 1337
np.random.seed(SEED)
tf.random.set_seed(SEED)
# Generate a uniformly distributed set of random numbers in the range from
# 0 to 2π, which covers a complete sine wave oscillation
x_values = np.random.uniform(low=0, high=2*math.pi, size=SAMPLES)
# Shuffle the values to guarantee they're not in order
np.random.shuffle(x_values)
# Calculate the corresponding sine values
```

```
y_values = np.sin(x_values)

# Plot our data. The 'b.' argument tells the library to print blue dots.
plt.plot(x_values, y_values, 'b.')
plt.show()
```

Є кілька речей, на які варто звернути увагу в цьому коді. По-перше, що ми використовуємо np.random.uniform() для генерації значень x. Цей метод повертає масив випадкових чисел у заданому діапазоні. NumPy містить багато корисних методів, які працюють з цілими масивами значень, що дуже зручно при роботі з даними.

По-друге, після генерації даних ми їх перемішуємо. Це важливо, оскільки процес тренування, який використовується в глибокому навчанні, залежить від того, щоб дані подавалися в дійсно випадковому порядку. Якщо дані будуть впорядковані, отримана модель буде менш точною.

Далі зверніть увагу, що ми використовуємо метод sin() з NumPy для обчислення синусів наших значень x. NumPy може обчислити це для всіх наших значень x одночасно, повертаючи масив. NumPy справді чудовий!

Нарешті, ви побачите деякий загадковий код, який викликає plt, що є нашим псевдонімом для Matplotlib:

```
# Plot our data. The 'b.' argument tells the library to print blue dots.
plt.plot(x_values, y_values, 'b.')
plt.show()
```

Цей код створює графік наших даних. Однією з найкращих можливостей Jupyter-ноутбуків є їх здатність відображати графіки, що генеруються кодом, який ви запускаєте. Matplotlib — це відмінний інструмент для створення графіків з даних.

Оскільки візуалізація даних є важливою частиною робочого процесу машинного навчання, це буде надзвичайно корисно під час тренування нашої моделі.

Щоб згенерувати дані та відобразити їх як графік, запустіть код в осередку. Після того як осередок завершить виконання, ви повинні побачити чудовий графік, який з’явиться нижче, як на рисунку 4-9.

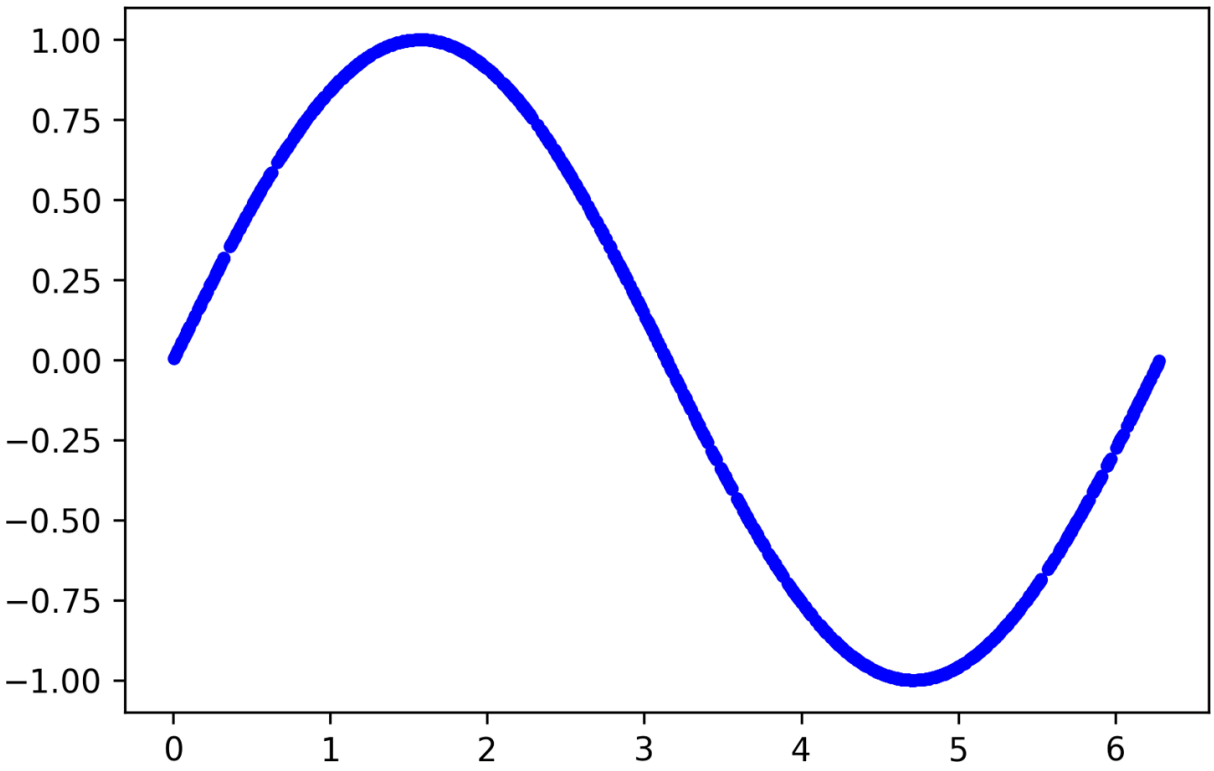


Рисунок 4-9. Графік наших згенерованих даних.

Це вибір випадкових точок на гарній, гладкій синусоїдальній кривій. Ми могли б використати їх для тренування нашої моделі. Однак це було б занадто просто. Одна з захоплюючих особливостей глибоких навчальних мереж — це їх здатність виявляти патерни серед шуму. Це дозволяє їм робити прогнози навіть при тренуванні на неохайних, реальних даних. Щоб продемонструвати це, давайте додамо випадковий шум до наших даних і побудуємо ще один графік:

```
# Add a small random number to each y value
y_values += 0.1 * np.random.randn(*y_values.shape)
# Plot our data
plt.plot(x_values, y_values, 'b.')
plt.show()
```

На даному етапі ми згенерували вибірку випадкових точок, що слідують за синусоїдальною кривою, що може бути використано для тренування моделі. Однак, для досягнення більш реалістичного підходу, ми додаємо випадковий шум до цих

даних. Це дозволяє створити набір даних, який більш точно відображає умови реального світу, де дані часто є зашумленими і неідеальними.

Після додавання шуму, наші точки розподіляються навколо синусоїдальної хвилі, а не утворюють рівномірну, гладку криву. Такий підхід набагато краще відображає реальні ситуації, де дані можуть бути непередбачуваними і містити помилки вимірювань або інші фактори, що впливають на точність.

Для візуалізації цих даних використовуємо бібліотеку Matplotlib, що дозволяє будувати графіки прямо в середовищі Jupyter. Це важливо для наочного представлення даних під час тренування моделі, оскільки візуалізація є невід'ємною частиною процесу аналізу даних у машинному навчанні.

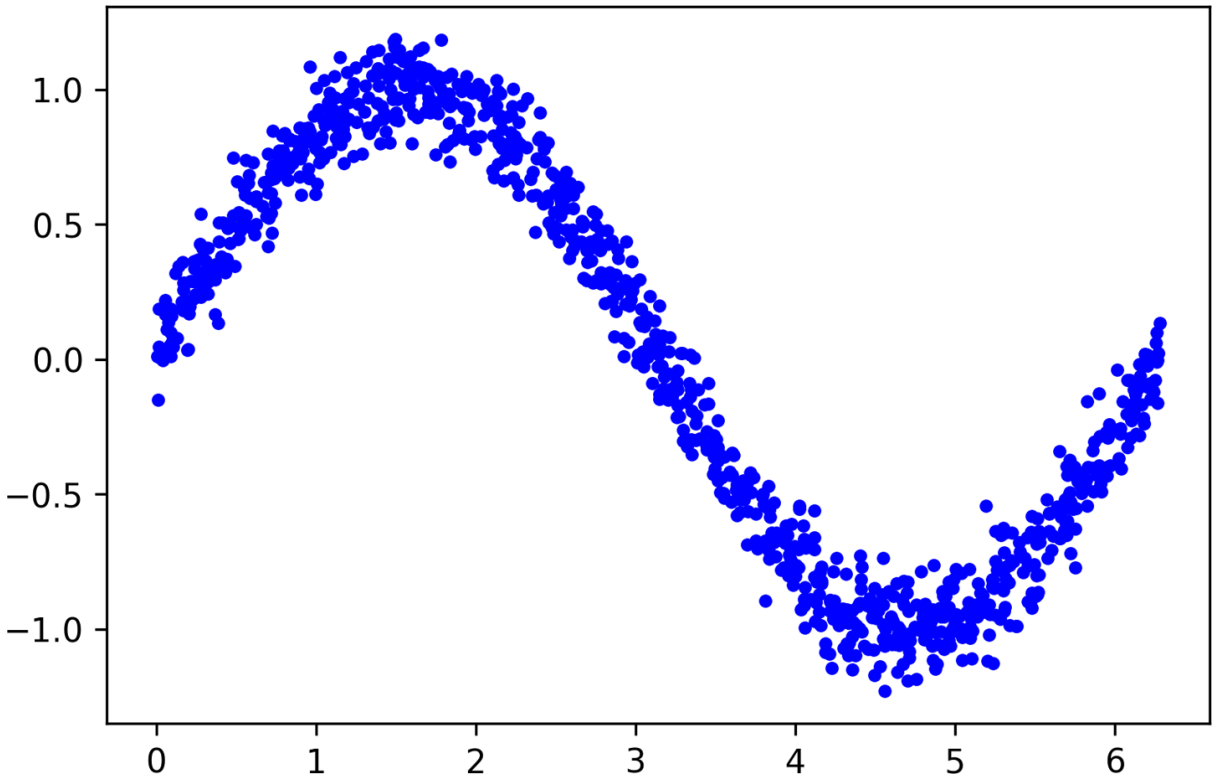


Рисунок 4-10. Графік наших даних з доданим шумом.

Розподіл даних

Як ми обговорювали в попередньому розділі, набір даних часто поділяється на три частини: для тренування, валідації та тестування. Для оцінки точності моделі, яку ми тренуємо, необхідно порівняти її передбачення з реальними даними та перевірити, наскільки вони відповідають один одному.

Ця оцінка відбувається під час тренування (де вона називається валідацією) і після тренування (де вона називається тестуванням). Важливо, щоб в кожному випадку використовувалися нові дані, які не були використані для тренування моделі.

Щоб забезпечити наявність даних для оцінки, ми відкладемо частину даних до початку тренування. Ми зарезервуємо 20% даних для валідації та ще 20% для тестування. Решту 60% ми використовуватимемо для тренування моделі. Це типова схема розподілу даних при тренуванні моделей.

Наступний код здійснює розподіл наших даних, після чого будує графік для кожного набору даних різним кольором:

```
# We'll use 60% of our data for training and 20% for testing. The remaining 20%
# will be used for validation. Calculate the indices of each section.
TRAIN_SPLIT =  int(0.6 * SAMPLES)
TEST_SPLIT = int(0.2 * SAMPLES + TRAIN_SPLIT)

# Use np.split to chop our data into three parts.
# The second argument to np.split is an array of indices where the data will be
# split. We provide two indices, so the data will be divided into three chunks.
x_train, x_validate, x_test = np.split(x_values, [TRAIN_SPLIT, TEST_SPLIT])
y_train, y_validate, y_test = np.split(y_values, [TRAIN_SPLIT, TEST_SPLIT])

# Double check that our splits add up correctly
assert (x_train.size + x_validate.size + x_test.size) ==  SAMPLES
```

```
# Plot the data in each partition in different colors:
plt.plot(x_train, y_train, 'b.', label="Train")
plt.plot(x_validate, y_validate, 'y.', label="Validate")
plt.plot(x_test, y_test, 'r.', label="Test")
plt.legend()
plt.show()
```

Для розподілу наших даних ми використовуємо ще один корисний метод NumPy — split(). Цей метод приймає масив даних і масив індексів, після чого розрізає дані на частини за вказаними індексами.

Запустіть цей осередок, щоб побачити результати нашого розподілу. Кожен тип даних буде представлений різним кольором (або відтінком, якщо ви читаєте друковану версію цієї книги), як показано на рисунку 4-11.

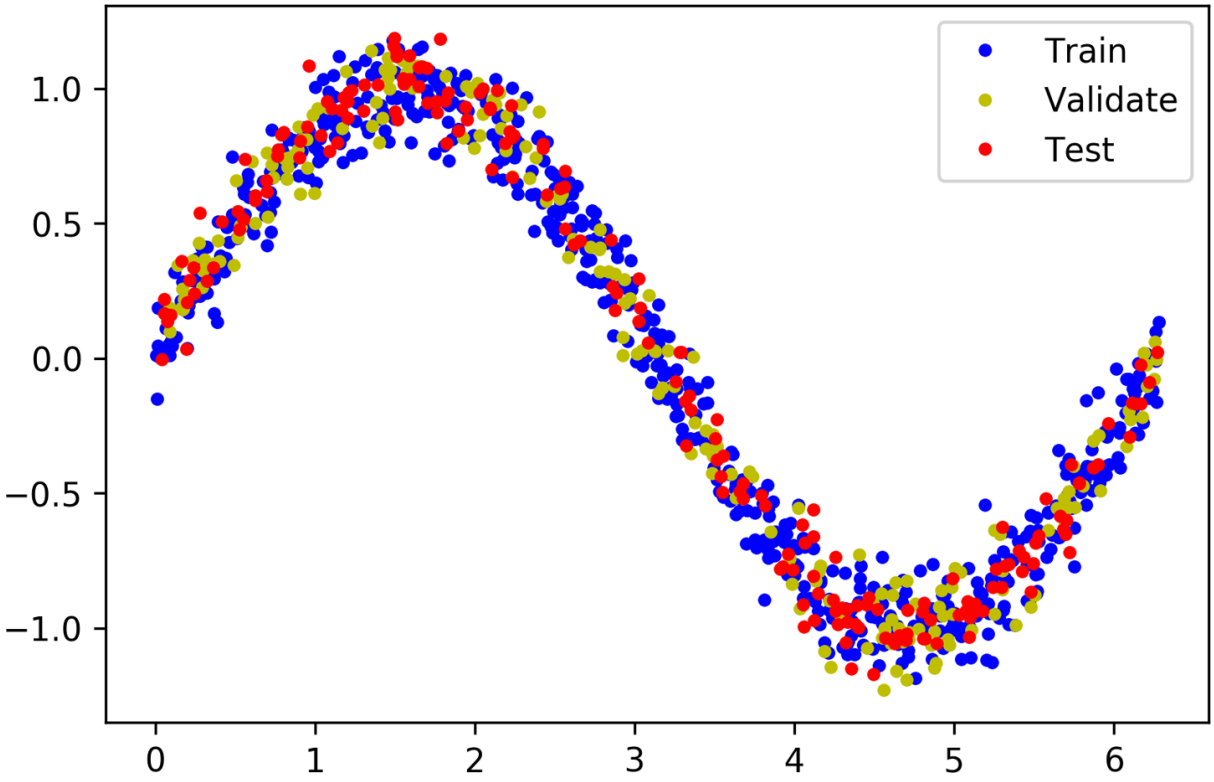


Рисунок 4-11. Графік наших даних, розділених на навчальний, перевірочний та тестовий набори.

Визначення базової моделі

Тепер, коли ми маємо наші дані, час створити модель, яку ми будемо тренувати для підгонки до цих даних.

Ми побудуємо модель, яка прийматиме вхідне значення (у цьому випадку x) і використовуватиме його для прогнозування числового вихідного значення (синус x). Цей тип задачі називається регресією. Моделі регресії можуть бути використані для різноманітних завдань, що потребують числових результатів. Наприклад, модель регресії може намагатися передбачити швидкість бігу людини в милях на годину, ґрунтуючись на даних акселерометра.

Щоб створити нашу модель, ми розробимо просту нейронну мережу. Вона використовує шари нейронів для вивчення будь-яких патернів, що лежать в основі тренувальних даних, щоб згодом робити прогнози.

Код для створення моделі досить простий і використовує Keras — високорівневий API для створення глибоких нейронних мереж в TensorFlow:

```
# We'll use Keras to create a simple model architecture
from tf.keras import layers
model_1 = tf.keras.Sequential()
# First layer takes a scalar input and feeds it through 16 "neurons." The
# neurons decide whether to activate based on the 'relu' activation function.
model_1.add(layers.Dense(16, activation='relu', input_shape=(1,)))
# Final layer is a single neuron, since we want to output a single value
model_1.add(layers.Dense(1))
# Compile the model using a standard optimizer and loss function for regression
model_1.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
# Print a summary of the model's architecture
model_1.summary()
```

По-перше, ми створюємо послідовну модель за допомогою Keras, що означає, що кожен шар нейронів розташований один за одним, як це показано на рисунку 3-1. Потім ми визначаємо два шари. Ось як визначено перший шар:

```
model_1.add(layers.Dense(16, activation='relu', input_shape=(1,)))
```

Перший шар має один вхід — наше значення x — і 16 нейронів. Це щільний шар (також відомий як повнозв’язаний шар), що означає, що вхід буде переданий в кожен з його нейронів під час інференції, коли ми робимо прогнози. Кожен нейрон

потім активується в певній мірі. Ступінь активації для кожного нейрона залежить як від його ваги та зміщення, так і від функції активації. Активація нейрона виводиться як число.

Активація обчислюється за простою формулою, яку ми показуємо нижче на Python. Нам не потрібно писати цей код самостійно, оскільки він обробляється Keras та TensorFlow, але це корисно для розуміння процесу глибокого навчання:

```
activation = activation_function((input * weight) + bias)
```

Щоб обчислити активацію нейрона, його вхід множиться на вагу, і додається зміщення. Обчислене значення передається у функцію активації. Результатом є активація нейрона.

Функція активації — це математична функція, що використовується для формування виходу нейрона. У нашій мережі ми використовуємо функцію активації, яка називається **Rectified Linear Unit (ReLU)**. Це вказано в Keras через аргумент activation='relu'.

ReLU — це проста функція, яку можна записати так:

```
def relu(input):
    return max(0.0, input)
```

ReLU повертає більше значення: або його вхід, або нуль. Якщо вхідне значення є від’ємним, ReLU повертає нуль. Якщо вхідне значення більше за нуль, ReLU залишає його незмінним.

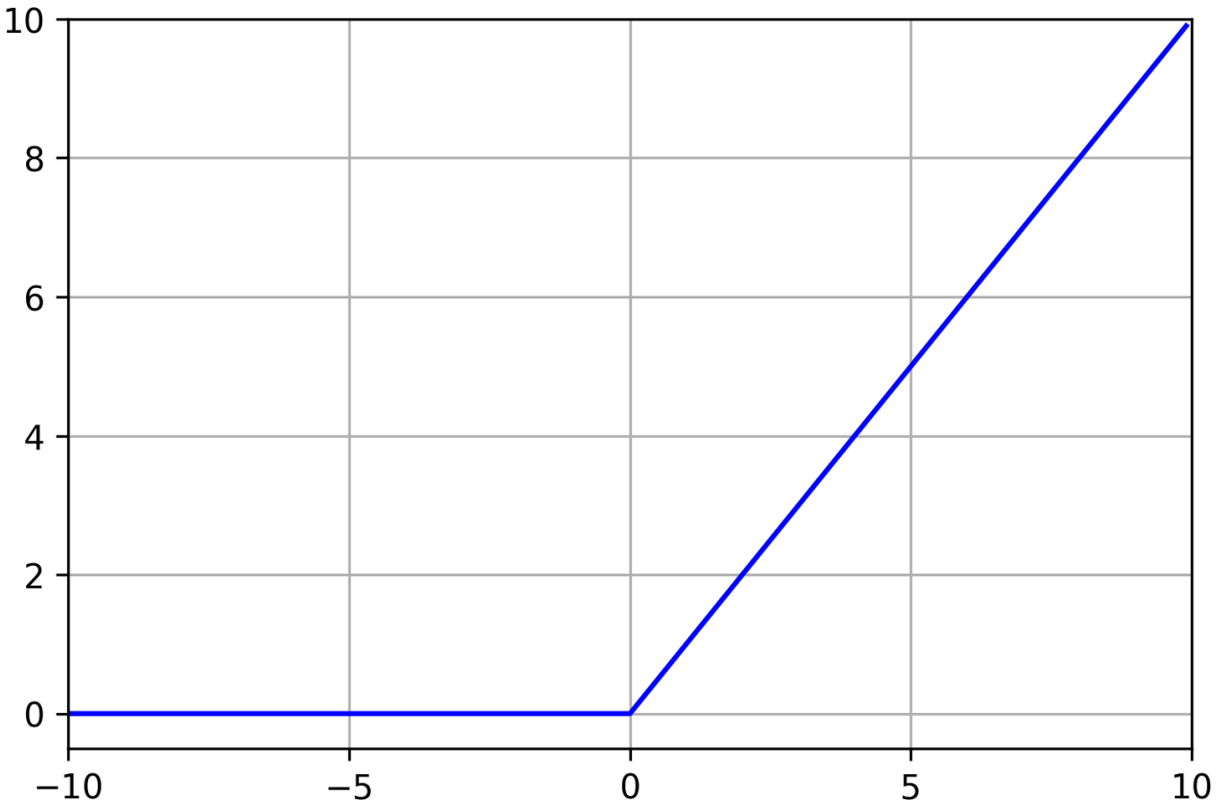


Рисунок 4.12 Графік ReLU для вхідних даних від -10 до 10

Функція активації ReLU є нелінійною, що дозволяє кільком шарам нейронів об’єднуватися і моделювати складні нелінійні взаємозв’язки, де значення у не збільшується на однакову величину з кожним кроком x.

Після цього активація з першого шару буде передана в другий шар, який визначено наступним чином:

model_1.add(layers.Dense(1))

Оскільки цей шар має один нейрон, він отримає 16 вхідних значень, по одному від кожного з нейронів попереднього шару. Його мета — об’єднати всі активації з попереднього шару в одне вихідне значення. Оскільки це вихідний шар, ми не вказуємо функцію активації — ми просто хочемо отримати сирий результат.

Цей нейрон має відповідну вагу для кожного вхідного значення. Вихід нейрона обчислюється за наступною формулою:

```
# Here, `inputs` and `weights` are both NumPy arrays with 16 elements each
output = sum((inputs * weights)) + bias
```

Вихідне значення обчислюється шляхом множення кожного вхідного значення на відповідну вагу, підсумовування результатів і додавання зміщення нейрона.

Ваги та зміщення мережі навчаться під час тренування. Компіляція моделі налаштовує важливі параметри для тренувального процесу та готує модель до навчання:

```
model_1.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
```

• **optimizer** — це алгоритм, який буде коригувати мережу для моделювання її входів під час тренування.

• **loss** — функція втрат, яка використовується для обчислення, наскільки відрізняються передбачення мережі від реальних даних. У нашому випадку це **mse** (середньоквадратична помилка).

• **metrics** — дозволяє вказати додаткові функції для оцінки продуктивності моделі. Ми використовуємо **mae**(середня абсолютна помилка), що є корисною функцією для оцінки регресійної моделі.

Після компіляції моделі ми можемо вивести її підсумок:

```
# Print a summary of the model's architecture
model_1.summary()
```

Запустивши цей код, отримуємо таблицю, що показує шари мережі, їхні форми виходу та кількість параметрів.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 16)                32
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 17
=================================================================
Total params: 49
```

Trainable params: 49

Non-trainable params: 0

_____

Навчання нашої моделі

Після того, як ми визначили нашу модель, наступний крок — її тренування та оцінка ефективності, щоб зрозуміти, наскільки добре вона працює. Після того, як ми побачимо метрики, ми зможемо вирішити, чи модель задовільна, чи потрібно внести зміни в її дизайн і провести повторне тренування.

Для тренування моделі в Keras ми просто викликаємо метод fit(), передаючи йому всі наші дані та деякі інші важливі аргументи. Код у наступній клітинці показує, як це зробити:

```
history_1 = model_1.fit(x_train, y_train, epochs=1000, batch_size=16,
                        validation_data=(x_validate, y_validate))
```

Запустивши код у комірці, щоб розпочати навчання ми побачимо, як почнуть з'являтися деякі журнали:

```
Train on 600 samples, validate on 200 samples
Epoch 1/1000
600/600 [==============================] - 1s 1ms/sample - loss: 0.7887 - mae: 0.7848 - val_loss: 0.5824 - val_mae: 0.6867
Epoch 2/1000
600/600 [==============================] - 0s 155us/sample - loss: 0.4883 - mae: 0.6194 - val_loss: 0.4742 - val_mae: 0.6056
```

Пояснення параметрів методу fit()

Коли ми тренуємо модель за допомогою методу fit(), важливо зрозуміти, як працюють його аргументи та налаштування для досягнення оптимальних результатів.

1. x_train, y_train

Перші два аргументи методу fit() — це значення вхідних (x) та вихідних (y) даних для навчання. Важливо пам’ятати, що частина наших даних зберігається для

валідації та тестування, тому тільки тренувальний набір даних використовується для навчання мережі.

2. epochs

Аргумент epochs визначає кількість разів, які весь тренувальний набір даних буде пропущений через мережу під час тренування. Чим більше епох, тим більше тренування відбудеться. Однак важливо зазначити, що занадто велика кількість епох може призвести до **перенавчання** (overfitting), коли модель стає занадто специфічною до тренувальних даних і не може добре працювати на нових, не бачених раніше даних. У нашому випадку ми почнемо з 1,000 епох і після завершення тренування оцінемо метрики, щоб зрозуміти, чи це оптимальна кількість.

3. batch_size

Аргумент batch_size визначає, скільки частин тренувальних даних буде подано до мережі перед тим, як оцінити її точність та оновити ваги і зсуви. Якщо ми встановимо batch_size рівним 1, то кожен елемент даних оброблятиметься по черзі, що призведе до великої кількості оновлень і дуже довгого часу тренування.

Якщо ми встановимо batch_size рівним 600 (розмір всього тренувального набору), то кожен етап тренування буде відбуватися тільки один раз, що значно зменшить час тренування. Однак це знижує точність моделі, оскільки мережа може почати перенавчатися.

Оптимальний варіант — це середнє значення. Ми використовуємо розмір пакету **16**. Це означає, що ми будемо вибирати 16 випадкових точок даних, обробляти їх, обчислювати сумарні втрати і оновлювати мережу після кожного пакету. Це забезпечує баланс між ефективністю тренування і точністю моделі.

4. validation_data

У цьому параметрі вказується набір даних для валідації. Дані з цього набору будуть проходити через мережу протягом всього процесу тренування, а прогнози мережі порівнюватимуться з очікуваними значеннями. Ми можемо оцінити результати валідації в логах та об’єкті history_1. Це дозволяє відстежувати, як добре модель працює на нових даних і коригувати її тренування при необхідності.

Метрики навчання

На даному етапі навчання моделі має бути завершено. Якщо ні, необхідно зачекати декілька хвилин до його завершення.

Для оцінки ефективності навчання мережі будемо використовувати ряд метрик. Розглянемо журнали, що були згенеровані під час навчання, які відображають прогрес моделі від початкового випадкового стану.

Приклад журналів для першої та останньої епох:

Epoch 1/1000

600/600 [==============================] - 1s 1ms/sample - loss: 0.7887 - mae: 0.7848 - val_loss: 0.5824 - val_mae: 0.6867

Epoch 1000/1000

600/600 [==============================] - 0s 124us/sample - loss: 0.1524 - mae: 0.3039 - val_loss: 0.1737 - val_mae: 0.3249

Розглянемо значення loss, mae, val_loss та val_mae:

1. loss (функція втрат):

Це результат функції втрат, в нашому випадку - середньоквадратична помилка (MSE), що виражається позитивним числом.

Загалом, чим менше значення loss, тим краще.

Порівнюючи першу та останню епохи, можна побачити значне покращення навчання: значення losszменшилось з ~0.7 до ~0.15.

2. mae (середня абсолютна помилка):

Це середня абсолютна помилка для навчальних даних, що показує середню різницю між прогнозами мережі та очікуваними значеннями y.

Початкова помилка є досить високою (~0.78), що очікувано для нетренованої мережі.

Після навчання mae становить ~0.30, що все ще є відносно високим значенням, враховуючи діапазон очікуваних значень від -1 до 1.

3. val_loss (функція втрат на валідаційних даних):

Це значення функції втрат на валідаційних даних.

В останній епосі значення val_loss (~0.17) трохи вище, ніж loss (~0.15). Це може свідчити про перенавчання моделі, оскільки вона гірше працює на даних, які не бачила під час навчання.

4. val_mae (середня абсолютна помилка на валідаційних даних):

Це середня абсолютна помилка для валідаційних даних.

Значення val_mae (~0.32) вище, ніж mae на навчальних даних, що також свідчить про можливе перенавчання.

Графічне відображення історії навчання

Для візуалізації процесу навчання та оцінки його ефективності, побудуємо графіки зміни значень loss та maeпротягом епох.

1. Графік функції втрат (loss):

Відображає динаміку зміни loss на навчальних та валідаційних даних.

Дозволяє оцінити швидкість та стабільність навчання.

Допомагає виявити ознаки перенавчання (розбіжність кривих loss на навчальних та валідаційних даних).

2. Графік середньої абсолютної помилки (mae):

Відображає динаміку зміни mae на навчальних та валідаційних даних.

Дозволяє оцінити точність прогнозування моделі.

Допомагає виявити ознаки перенавчання (розбіжність кривих mae на навчальних та валідаційних даних).

3. Графік порівняння прогнозованих та фактичних значень:

Відображає відповідність між прогнозами моделі та фактичними значеннями на навчальних даних.

Дозволяє візуально оцінити якість апроксимації синусоїди моделлю.

Покращення моделі

Для покращення ефективності моделі, збільшимо її розмір, додавши додатковий шар нейронів. Це дозволить моделі вивчити більш складні залежності в даних.

Тестування моделі

Для оцінки здатності моделі до узагальнення на нових даних, використаємо тестовий набір даних. Важливо, щоб тестові дані не використовувалися під час навчання та валідації.

Конвертація моделі для TensorFlow Lite

Для розгортання моделі на мікроконтролері, необхідно конвертувати її у формат TensorFlow Lite. Це дозволить оптимізувати модель для роботи на пристроях з обмеженими ресурсами.

1. Квантування моделі:

Зменшує розмір моделі за рахунок зниження точності ваг та зсувів.

Прискорює виконання моделі на мікроконтролері.

Часто призводить до мінімальної втрати точності.

2. Конвертація у С файл:

Дозволяє вбудувати модель безпосередньо в код мікроконтролера.

Зменшує обсяг пам'яті, необхідний для зберігання моделі.

Цей розділ описує процес навчання, оцінки та оптимізації моделі машинного навчання для подальшого розгортання на мікроконтролері. Використання метрик навчання, графічного аналізу та тестування дозволяє оцінити ефективність моделі та виявити можливі проблеми. Конвертація моделі у формат TensorFlow Lite та квантування дозволяють оптимізувати її для роботи на пристроях з обмеженими ресурсами.

На цьому етапі завершено процес створення моделі машинного навчання. Ми навчили, оцінили та конвертували глибоку нейронну мережу TensorFlow, яка здатна приймати число в діапазоні від 0 до 2π та видавати достатньо точне наближення його синуса.

ВИСНОВОК

У цій дипломній роботі було проведено дослідження та аналіз можливостей застосування технологій On-Device Learning та TinyML в системах розумного будинку та промислової автоматизації. Дослідження охопило теоретичні аспекти обох технологій, їхні переваги та обмеження, а також практичні аспекти їх інтеграції в реальні системи.

Було встановлено, що On-Device Learning дозволяє здійснювати навчання та обробку моделей машинного навчання безпосередньо на пристроях, що забезпечує швидкість, конфіденційність та енергоефективність. TinyML, як напрямок машинного навчання, спеціалізується на розробці та оптимізації моделей для малопотужних пристроїв, що робить його ідеальним для використання в IoT-пристроях та системах з обмеженими ресурсами.

Аналіз показав, що інтеграція On-Device Learning та TinyML в системи розумного будинку та промислової автоматизації має значний потенціал. Ці технології дозволяють створювати автономні системи, здатні приймати рішення в реальному часі без необхідності передавати дані на віддалені сервери. Це особливо важливо для застосувань, де швидкість та конфіденційність є критичними факторами.

В рамках роботи було розглянуто фреймворки для TinyML, такі як TensorFlow Lite Micro, що дозволяють розробникам ефективно створювати моделі для малопотужних пристроїв. Було також проведено аналіз переваг та обмежень цих технологій, що включає обмежені обчислювальні ресурси пристроїв та необхідність оптимізації моделей для роботи в таких умовах.

За результатами дослідження можна зробити висновок, що On-Device Learning та TinyML є перспективними напрямками для розвитку інтелектуальних автономних систем. Їх застосування в системах розумного будинку та промислової автоматизації дозволяє значно покращити ефективність, знизити затримки у прийнятті рішень та підвищити рівень автономності систем.

Подальші дослідження можуть бути спрямовані на розробку та тестування більш складних прототипів систем, а також на вивчення можливостей оптимізації моделей для ще більш обмежених ресурсів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

https://www.edgeimpulse.com/

https://www.tensorflow.org/lite/micro

https://www.oreilly.com/library/view/tinyml/9781492052036/ch04.html

https://ieeexplore.ieee.org/

https://arxiv.org/

https://ai.googleblog.com/

https://www.hackster.io/

Building a TinyML Application with TF Micro and SensiML. *tensorflow.org*. URL: https://blog.tensorflow.org/2021/05/building-tinyml-application-with-tf-micro-and-sensiml.html(date of access: 09.03.2025).

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інформаційних систем та технологій

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

«ON-DEVICE LEARNING ТА ЙОГО ЗАСТОСУВАННЯ У СИСТЕМИ "РОЗУМНОГО БУДИНКУ" ТА ПРОМИСЛОВОЇ АВТОМАТИЗАЦІЇ»

Виконав: студент групи ІСД-43
Артем НЕГОДА
Керівник: к.т.н., доцент
Андрій АРОНОВ

Київ-2025

Мета роботи – дослідження та аналіз можливостей застосування TinyML і On-Device Learning в системах розумного будинку та промислової автоматизації, а також розробка прототипу системи, що використовує ці технології для автономного управління і оптимізації процесів у реальному часі.

Об’єкт дослідження – системи розумного будинку та промислової автоматизації, процеси обробки даних та навчання моделей на пристроях з обмеженими ресурсами.

Предмет дослідження – технології On-Device Learning та TinyML, методи оптимізації моделей машинного навчання для вбудованих систем, фреймворки для розробки TinyML-рішень (наприклад, TensorFlow Lite Micro) та їх інтеграція в автономні системи.

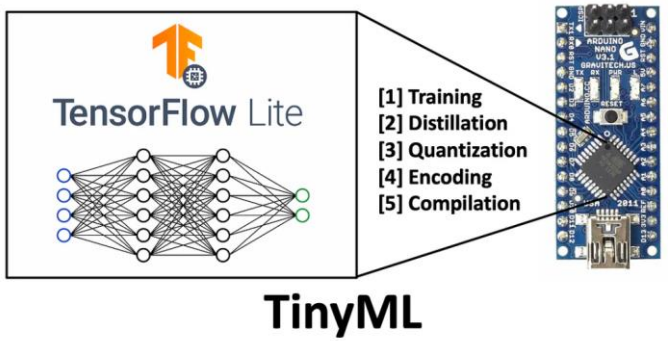
Завдання дослідження включають:

- Ознайомлення з основними принципами On-Device Learning та TinyML;
- Аналіз можливостей інтеграції цих технологій у системи розумного будинку та промислової автоматизації;
- Оцінка переваг та обмежень застосування TinyML на вбудованих пристроях;
- Розробка та тестування прототипу системи на основі TinyML для однієї з зазначених сфер.

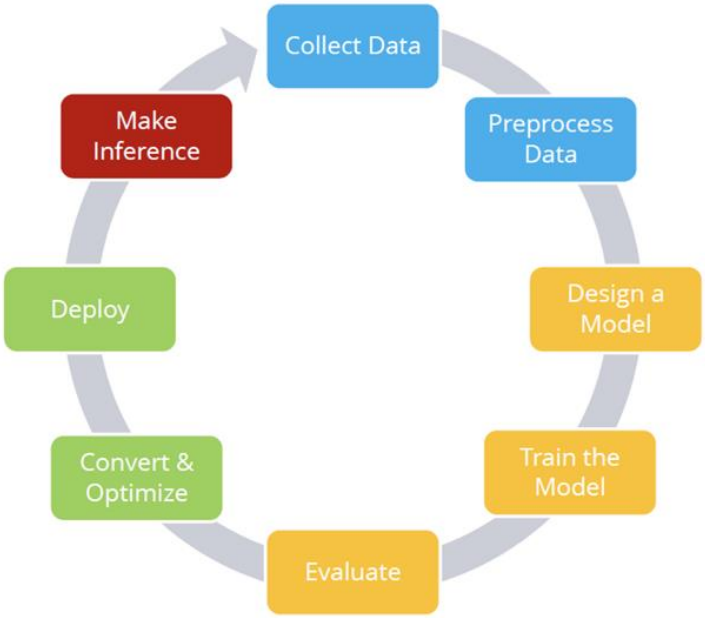
ON-DEVICE LEARNING TA TINYML

On-Device Learning — це підхід до машинного навчання, при якому всі етапи обробки даних, навчання та адаптації моделей здійснюються безпосередньо на пристрої, без потреби відправляти дані на віддалені сервери або в хмару. Це означає, що модель машинного навчання працює локально на самому пристрої, наприклад, на мобільному телефоні, розумному годиннику чи іншому IoT-пристрої, а не в хмарі чи на сервері.

TinyML — це напрямок машинного навчання, який спеціалізується на розробці та оптимізації моделей машинного навчання для використання на малопотужних пристроях з обмеженими ресурсами, таких як мікроконтролери, датчики, розумні годинники, фітнес-браслети та інші пристрої Інтернету речей (IoT).

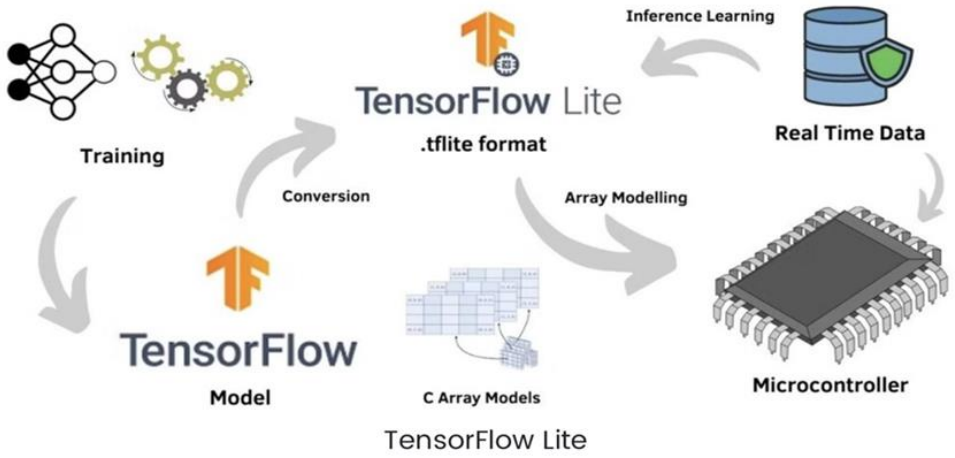


Цикл розробки TinyML

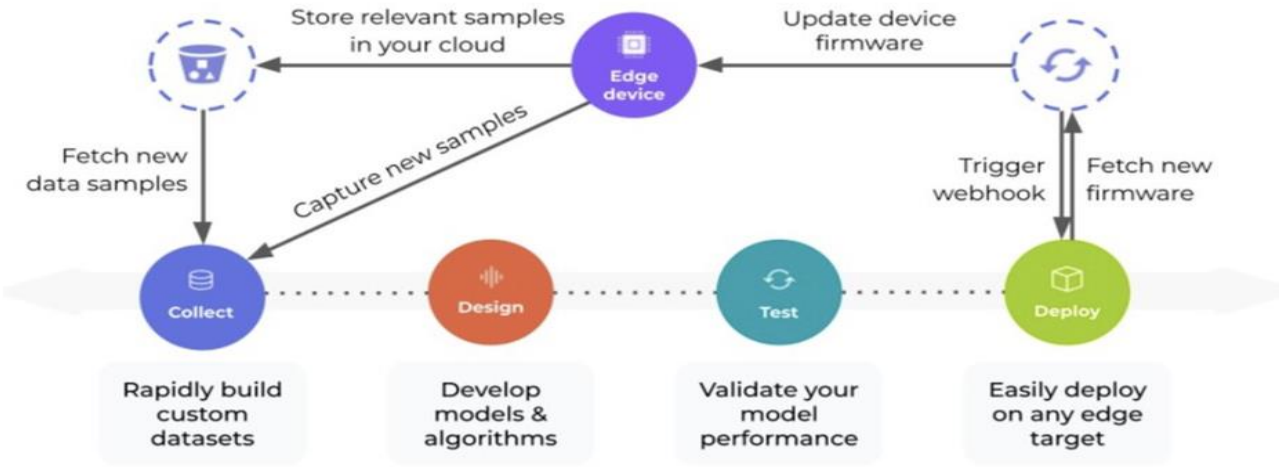


TinyML App Design Flow

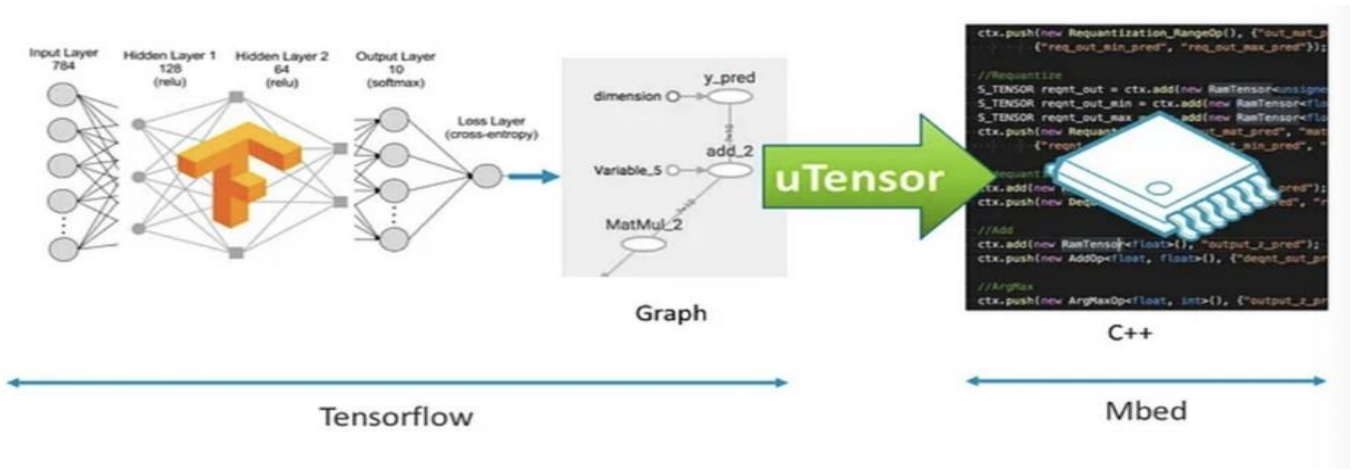
Фреймворки для TinyML



Розробка моделі TinyML



Приклад роботи TensorFlow



ІНТЕГРАЦІЯ TINYML У СИСТЕМИ РОЗУМНОГО БУДИНКУ ТА ПРОМИСЛОВОЇ АВТОМАТИЗАЦІЇ

Розумний будинок — це будинок, оснащений різноманітними технологіями для автоматизації повсякденних процесів.

Промислова автоматизація — це використання технологій для автоматизації виробничих процесів, управління підприємствами та контролю за різними процесами.

Як TinyML допомагає у розумних будинках

Інтелектуальна автоматизація - TinyML дає змогу пристроям в будинку адаптуватися до звичок і поведінки користувачів.

Розпізнавання голосу та звуків-використовуючи TinyML, розумні колонки або інші аудіопристрої можуть обробляти звукові сигнали безпосередньо на місці, розпізнаючи голоси або звуки.

Моніторинг стану здоров’я - TinyML дозволяє використовувати носимі пристрої для моніторингу здоров’я, такі як фітнес-браслети або розумні годинники, для збору даних про пульс, кроки, рівень кисню в крові та інші показники.

Розпізнавання обличчя та безпека - пристрої відеоспостереження з TinyML можуть безпосередньо на місці здійснювати розпізнавання обличчя для ідентифікації осіб.

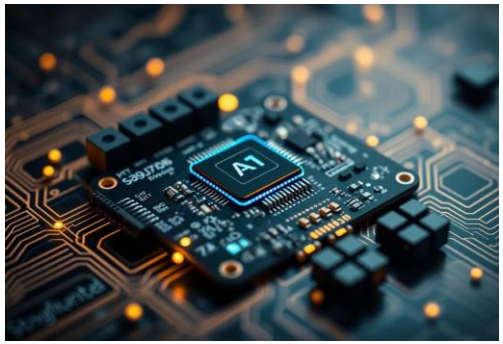
Інтелектуальне освітлення - розумне освітлення може використовувати TinyML для автоматичного налаштування яскравості світла в залежності від часу доби або рівня природного освітлення в приміщенні.

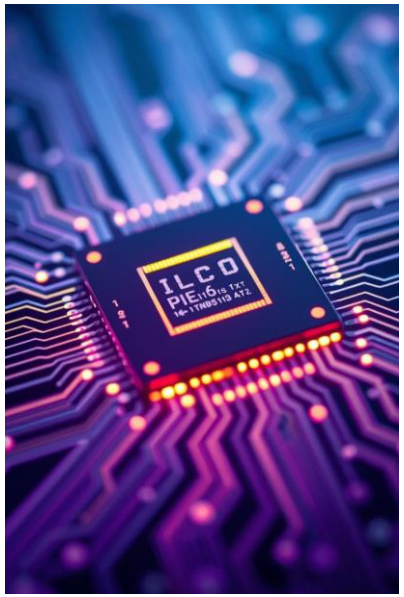
Переваги інтеграції TinyML у системи

Інтеграція TinyML у промислові автоматизовані системи має низку значних переваг, які можуть суттєво підвищити ефективність, безпеку та економічність виробничих процесів.

Ось деякі з основних переваг:

- Енергоефективність
- Автономність
- Швидка обробка даних у реальному часі
- Зниження витрат на інфраструктуру
- Покращена безпека і конфіденційність даних
- Підвищення продуктивності
- Масштабованість
- Прогнозування та профілактика поломок
- Миттєве виявлення аномалій та збоїв
- Покращення якості продукції

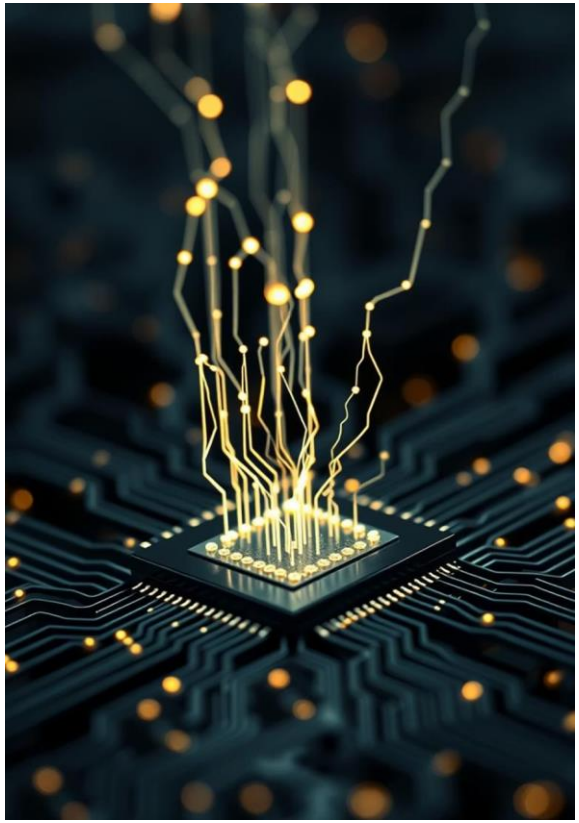




ТЕХНІЧНА РЕАЛІЗАЦІЯ ТА МОДЕЛЮВАННЯ ON-DEVICE LEARNING ТА TINYML

Основні методи і алгоритми, які можуть бути використані для навчання на пристроях:

1. Лінійна регресія
2. Дерева рішень
3. Логістична регресія
4. Нейронні мережі (включаючи компактні моделі)
5. Метод опорних векторів (SVM)
6. K-найближчих сусідів (K-NN)
7. Кластеризація (наприклад, K-середніх)
8. Підсилення моделей (Boosting та Bagging)
9. Періодичні та часові моделі (RNN, LSTM)



Прототип системи розумного будинку з TinyML

Метою цього прототипу є створення системи розумного будинку, яка використовує технології TinyML для автоматизації управління температурою та освітленням. Система буде приймати рішення на основі даних, зібраних сенсорами, і передбачень, зроблених за допомогою моделей машинного навчання, щоб забезпечити комфорт і енергоефективність у будинку. Вона має здатність адаптуватися до змін в середовищі, забезпечуючи оптимальне керування будинковими пристроями.

Для реалізації цієї системи вибрано кілька ключових компонентів. Одним з них є мікроконтролер, наприклад, Arduino Nano 33 BLE Sense або ESP32. Arduino Nano 33 BLE Sense оснащений вбудованими сенсорами для збору даних, таких як температура, вологість та освітленість, що робить його зручним для простих завдань.

Прототип для промислової автоматизації з TinyML

У промисловій автоматизації TinyML може бути використано для моніторингу обладнання та виявлення аномалій. Наприклад: вибираємо ESP32 з вбудованими Wi-Fi і Bluetooth для підключення до мережі, використовуємо вібраційні датчики для моніторингу робочих механізмів, навчаємо модель на великих даних про нормальну роботу обладнання, щоб вона могла виявляти відхилення (наприклад, надмірні вібрації).Оптимізована модель завантажується на пристрій, який локально аналізує вібрації і, у разі виявлення аномалії, сповіщає оператора про необхідність перевірки або ремонту.

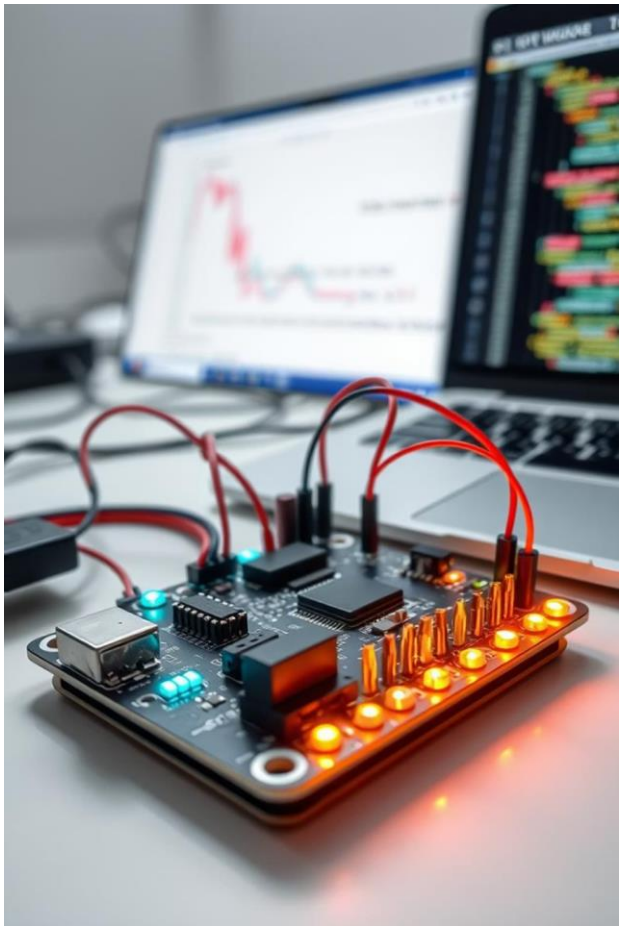
СТВОРЕННЯ ТА НАВЧАННЯ МОДЕЛІ TINYML

Необхідне обладнання:

Для повноцінного розгортання моделі TinyML потрібні вбудовані пристрої, такі як Arduino Nano 33 BLE Sense, SparkFun Edge або ST Microelectronics STM32F746G Discovery kit. Тестування коду можливе на ПК.

Програмні засоби:

Ми використовуватимемо Python, TensorFlow та Google Colaboratory – хмарне середовище для експериментів з кодом Python. Всі ці інструменти є загальнодоступними та безкоштовними.



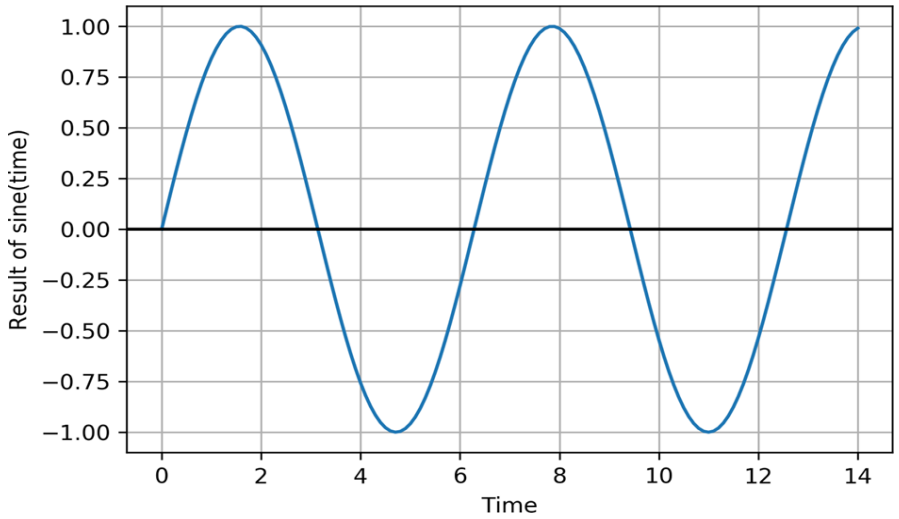
Процес створення моделі та її застосування

Отримання даних: генерація 1000 випадкових значень x від 0 до 2π та обчислення їх синуса, з додаванням випадкового шуму для реалістичності.

Навчання моделі: створення простої нейронної мережі за допомогою Keras для передбачення значень синуса. Модель навчається на 60% даних, 20% для валідації, 20% для тестування.

Оцінка продуктивності: аналіз метрик loss та mae для оцінки точності моделі та виявлення перенавчання.

Конвертація для пристроїв: конвертація моделі у формат TensorFlow Lite та квантування для оптимізації під мікроконтролери.



Візуалізація та розгортання

Візуалізація результатів

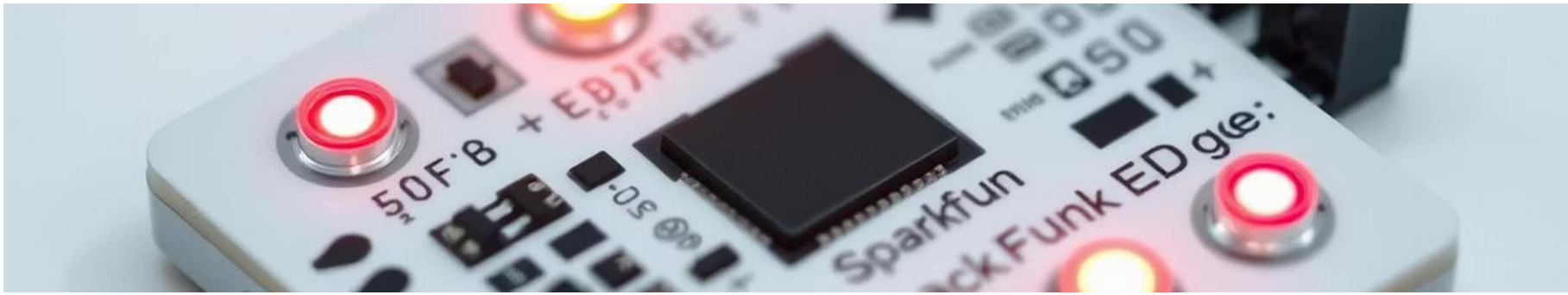
Вихідні дані моделі використовуються для керування світлодіодами або графічною анімацією, демонструючи взаємодію ML та апаратного забезпечення.

Розгортання на пристроях

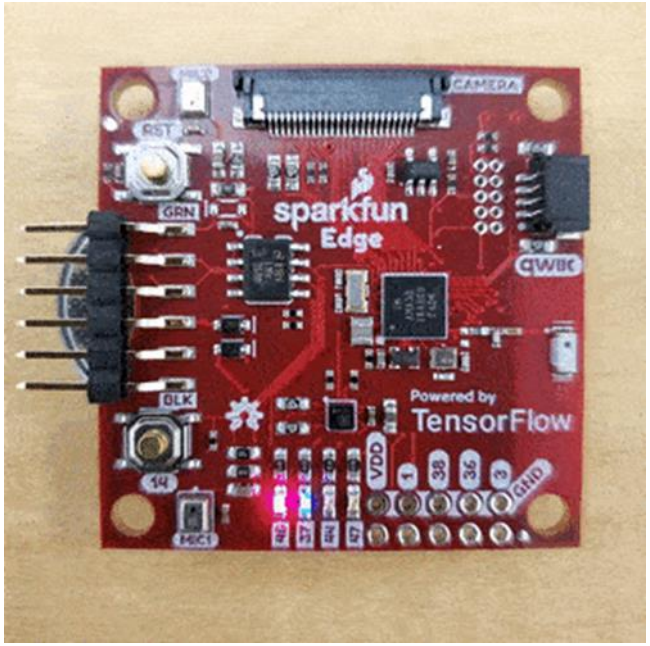
Після перевірки базового коду модель розгортається на SparkFun Edge, Arduino Nano 33 BLE Sense та ST Microelectronics STM32F746G Discovery kit.

Відкритий репозиторій

Всі програмні компоненти доступні у відкритому репозиторії GitHub TensorFlow, що забезпечує прозорість та повторюваність експерименту.



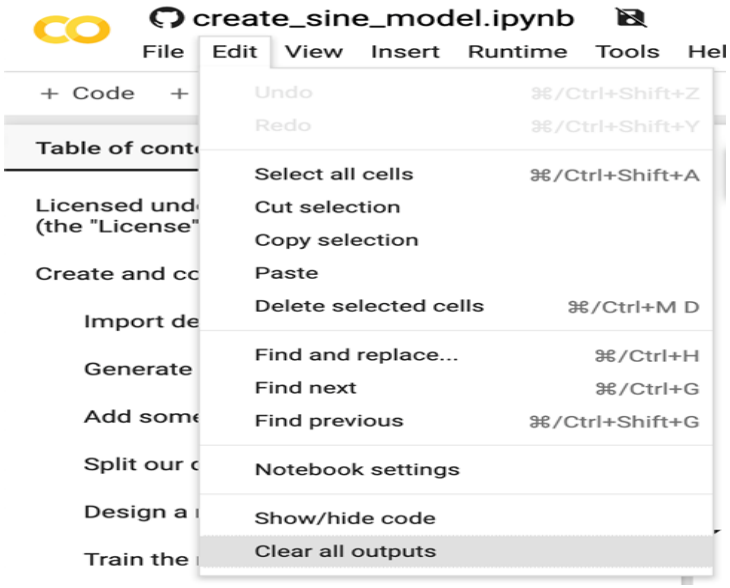
Код, що виконується на SparkFun Edge



Кнопка "Запустити в Google Colab"

Run in Google Colab	View source on GitHub

Опція "Очистити всі вихідні дані"



Комірка "Імпорт залежностей" у вибраному стані

Import dependencies

Our first task is to import the dependencies we need. Run the following cell to do so:

```
# TensorFlow is an open source machine learning library
# Note: The following line is temporary to use v2
!pip install tensorflow==2.0.0-beta0
import tensorflow as tf
# Numpy is a math library
import numpy as np
# Matplotlib is a graphing library
import matplotlib.pyplot as plt
# math is Python's math library
import math
```

Комірка "Імпорт залежностей" в стані виконання

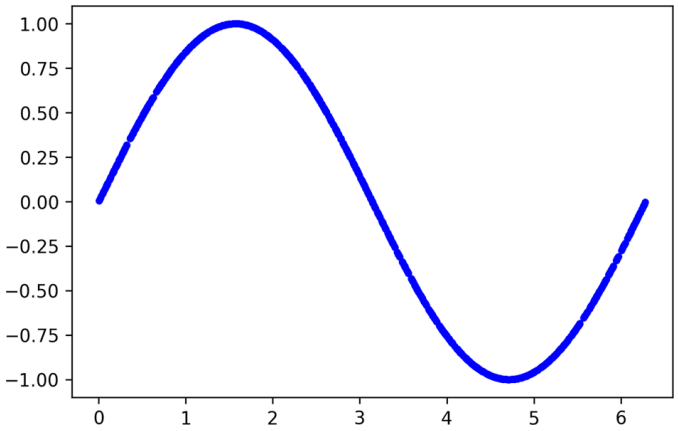
```
# TensorFlow is an open source machine learning library
# Note: The following line is temporary to use v2
!pip install tensorflow==2.0.0-beta0
import tensorflow as tf
# Numpy is a math library
import numpy as np
# Matplotlib is a graphing library
import matplotlib.pyplot as plt
# math is Python's math library
import math
```

```
Collecting tensorflow==2.0.0-beta0
```

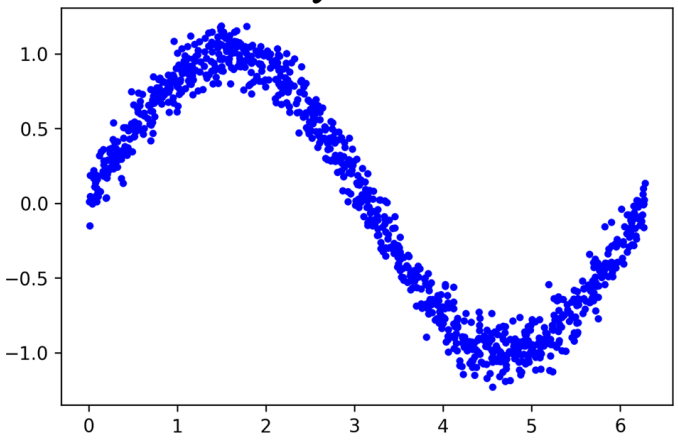
Лічильник запусків осередку в верхньому лівому куті

```
[1] # TensorFlow is an open source machine learning library
    # Note: The following line is temporary to use v2
    !pip install tensorflow==2.0.0-beta0
    import tensorflow as tf
    # Numpy is a math library
    import numpy as np
    # Matplotlib is a graphing library
    import matplotlib.pyplot as plt
    # math is Python's math library
    import math
```

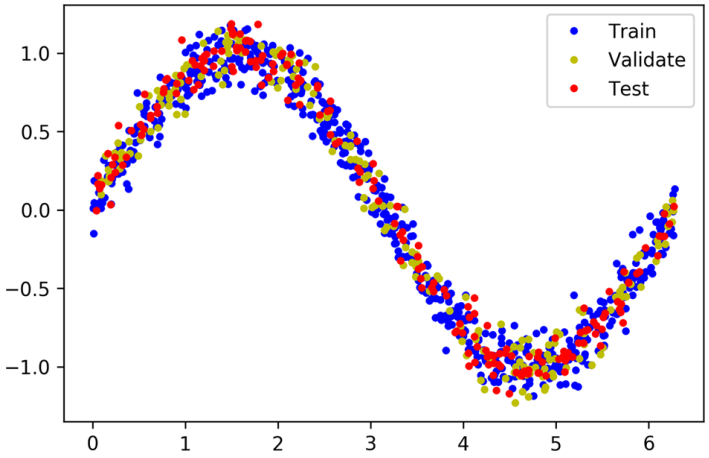
Графік наших згенерованих даних



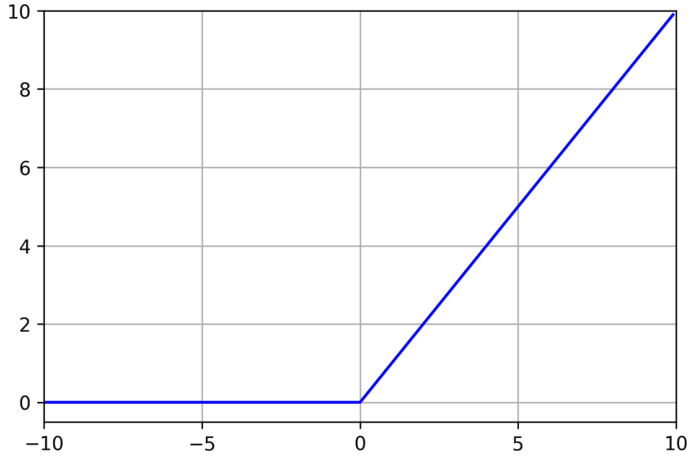
Графік наших даних з доданим шумом



Графік наших даних, розділених на навчальний, перевірочний та тестовий набори



Графік ReLU для вхідних даних від -10 до 10



ВИСНОВКИ

У цій дипломній роботі було проведено дослідження та аналіз можливостей застосування технологій On-Device Learning та TinyML в системах розумного будинку та промислової автоматизації. Дослідження охопило теоретичні аспекти обох технологій, їхні переваги та обмеження, а також практичні аспекти їх інтеграції в реальні системи.

Було встановлено, що On-Device Learning дозволяє здійснювати навчання та обробку моделей машинного навчання безпосередньо на пристроях, що забезпечує швидкість, конфіденційність та енергоефективність. TinyML, як напрямок машинного навчання, спеціалізується на розробці та оптимізації моделей для малопотужних пристроїв, що робить його ідеальним для використання в IoT-пристроях та системах з обмеженими ресурсами.

Аналіз показав, що інтеграція On-Device Learning та TinyML в системи розумного будинку та промислової автоматизації має значний потенціал. Ці технології дозволяють створювати автономні системи, здатні приймати рішення в реальному часі без необхідності передавати дані на віддалені сервери. Це особливо важливо для застосувань, де швидкість та конфіденційність є критичними факторами.

ПРОДОВЖЕННЯ ВИСНОВКІВ

В рамках роботи було розглянуто фреймворки для TinyML, такі як TensorFlow Lite Micro, що дозволяють розробникам ефективно створювати моделі для малопотужних пристроїв. Було також проведено аналіз переваг та обмежень цих технологій, що включає обмежені обчислювальні ресурси пристроїв та необхідність оптимізації моделей для роботи в таких умовах.

За результатами дослідження можна зробити висновок, що On-Device Learning та TinyML є перспективними напрямками для розвитку інтелектуальних автономних систем. Їх застосування в системах розумного будинку та промислової автоматизації дозволяє значно покращити ефективність, знизити затримки у прийнятті рішень та підвищити рівень автономності систем.

Подальші дослідження можуть бути спрямовані на розробку та тестування більш складних прототипів систем, а також на вивчення можливостей оптимізації моделей для ще більш обмежених ресурсів.