УДК 004.8:004.312.466 **doi: 10.32620/aktt.2022.6.07**

А. Є. ПЕРЕПЕЛИЦИН, Є. В. КАСАПЄН, Г. В. ФЕСЕНКО, В. С. ХАРЧЕНКО

Національний аерокосмічний університет ім. М. Є. Жуковського «Харківський авіаційний інститут», Харків, Україна

ТЕХНОЛОГІЇ РЕАЛІЗАЦІЇ ШТУЧНОГО ІНТЕЛЕКТУ ЯК СЕРВІСУ НА ОСНОВІ АПАРАТНИХ ПРИСКОРЮВАЧІВ

Предметом *вивчення в даній статті є сучасні технології, інструменти та способи побудови систем штучного інтелекту (ШІ) як послуги з використання FPGA як платформи.* ***Метою*** *роботи є аналіз сучасних технологій та інструментів, які використовуються для розроблення FPGA проєктів для систем, які реалізують штучний інтелект як послугу, і підготовка практичного прототипу сервісу ШІ.* ***Завдання:*** *проаналізувати еволюцію змін у продуктах провідних виробників програмованих логічних пристроїв та експериментально-практичні приклади впровадження парадигми безперервного перепрограмування програмованої логіки; проаналізувати динаміку змін інструментальних засобів розроблення FPGA систем для ШІ; проаналізувати базові елементи побудови проєктів для систем ШІ з використанням програмованої логіки. Відповідно до поставлених завдань, були отримані наступні* ***результати****. Проаналізовано процес програмування FPGA прискорювачів для проєктів ШІ. Виконано аналіз можливостей мікросхем FPGA із швидкою HBM пам'яттю для побудови проєктів, які вимагають високу швидкодію та об'єм пам'яті. Детально проаналізовано мови опису, фреймворки, ієрархію інструментальних засобів для побудови апаратних прискорювачів проектів штучного інтелекту. Обговорюється парадигма опису таких рішень. Проаналізовані етапи побудови проєктів штучного інтелекту з використанням нових засобів розробки програмованої логіки, базові блоки DPU для реалізації алгоритмів ШІ та можливі параметри таких блоків. Запропоновано практичні кроки побудови і оптимізації таких систем. Надані рекомендації з оптимізації нейронної мережі для реалізації з використанням FPGA. На базі цих рішень було підготовлено, навчено та протестовано практичний кейс сервісу ШІ.* ***Висновки:*** *Головний внесок цього дослідження полягає в тому, що на підставі запропонованих ідей і рішень є зрозумілими наступні кроки щодо створення гетерогенних систем на основі поєднання трьох елементів: ШІ як послуги, прискорювачів FPGA як технології підвищення продуктивності, надійності та безпеки та хмарних або Edge ресурсів для створення інфраструктури FPGA та ШІ як послуги. Розвиток цього методологічного і технологічного базису є напрямом подальших досліджень і розробок.*

Ключові слова: *штучний інтелект; FPGA; ШІ як сервіс; гетерогенні проєкти ШІ систем; апаратні прискорювачі ШІ; DPU; інструментальні засоби розробки ШІ; XRT.*

Вступ

Використання FPGA як платформи для побудови систем штучного інтелекту викликало значний інтерес завдяки появі спеціалізованих програмованих чіпів з великим об’ємом динамічної пам’яті, високою пропускною здатністю та швидкими каналами зв’язку з головними програмами. Такі чіпи виробляються як компоненти карт PCIe і встановлюються в дата-центрах [1]. Окрім центрів оброблення даних, такі рішення також використовуються у вбудованих системах [2]. Перехід до розумних автомобілів і поява автомобілів з бортовою логікою, що дозволяє реалізувати функції автопілоту, приведе до їх масового виробництва у наступному десятилітті. Це також з великою ймовірністю спричинить зміни законодавчих норм щодо використання ШІ в повсякденному житті [3].

В аерокосмічної галузі ШІ знаходить застосування як в бортових системах, так і для вирішення завдань обробки даних моніторингу, аналітикою, конфігурацією програмного забезпечення і низкою інших процесів [4]. Останніми роками такі компанії, як Raytheon, General Dynamics і Northrop Grumman оголосили про ініціативи розробки на основі ШІ та запуски продуктів.

Сьогодні в галузях промисловості з’являються два різних класи застосунків ШІ: аналітичний та операційний. Аналітичний ШІ можна використовувати для відстеження, планування та керування техобслуговуванням, для допомоги в розробці різноманітних застосунків, які моніторять паливо, допомагають в управлінні повітряним рухом або при створенні нових деталей літаків, вдосконалюючи існуючі конструкції, щоб вони були більш аеродинамічними та мали меншу вагу.

ISSN 1727-7337 (print)
ISSN 2663-2217 (online)

Операційний ШІ може керувати фабричними процесами, літаком або іншим транспортним засобом та реагувати на непередбачувані події.

Має бути зрозуміло, що для побудови таких систем ШІ потрібні спеціалізовані апаратні рішення з продуктивністю, яка перевищує продуктивність класичних комп'ютерів.

При цьому енергоспоживання зменшується, і є можливість досить компактного виконання для розміщення «під капотом». Є багато прикладів, коли такі системи допомагали автопілоту реагувати набагато швидше, ніж може реагувати людина, і це, на нашу думку, важливий тренд автоматизації.

Технології програмованої логіки з використанням сучасних засобів розроблення можуть допомогти швидко її виконувати за допомогою різнорідних обчислень. Це обумовлює можливість оперативного прототипування та моделювання алгоритмів та компонентів під час створення складних систем штучного інтелекту/машинного навчання [5].

Метою дослідження є аналіз сучасних технологій та інструментів, які використовуються для розроблення проєктів на основі FPGA для систем, які реалізують штучний інтелект як послугу, і підготувати практичний прототип служби ШІ.

Для досягнення мети планується розв'язати наступні завдання: проаналізувати еволюцію змін у продуктах провідних виробників програмованих логічних пристроїв; дослідити експериментально-практичні приклади впровадження парадигми безперервного перепрограмування програмованої логіки; проаналізувати динаміку змін в середовищі розроблення програмованих логічних систем для ШІ; проаналізувати базові елементи побудови проєктів для систем штучного інтелекту з використанням програмованої логіки.

1. Аналіз процесу програмування карт прискорювачів на основі FPGA для проєктів ШІ

Досвідчені розробники звикли до того, що FPGA у складі проекту розглядається як центральний об'єкт системи в цілому. По суті, при переході на FPGA прискорювачі у форматі карт з інтерфейсом PCIe та засоби розроблення для них необхідно змінити парадигму розробки FPGA-рішень.

При створенні масиву прискорювачів FPGA можна представити найскладнішу програмовну систему або найскладніше завдання просто як функцію на С-подібній мові, яку можна викликати з хост-програми на хост-комп'ютері.

Фактично, це один рядок коду з функцією з параметрами. Синхронізація під час виклику здійснюється аналогічно синхронізації між потоками.

З точки зору програміста, весь процес запуску проєкту в FPGA виглядає просто як виклик функції та очікування прапора результату для отримання даних або читання даних з буфера. Початок розроблення з використанням таких FPGA рішень вимагає зміни парадигми та розуміння місця прискорювача у всій системі. Для багатьох розробників FPGA цей перехід на новий спосіб роботи з FPGA вимагає певних зусиль. Це пов'язано з попереднім досвідом завантаження програми в FPGA через повільні інтерфейси, що зазвичай є тривалим процесом.

Програмування FPGA реалізацій з PCIe відбувається миттєво завдяки безпосередньому завантаженню бінарного представлення проєкту із системної пам'яті головного комп'ютера. Хост-застосунок надсилає бінарний файл із конфігурацією FPGA за допомогою інфраструктури зв'язку PCIe та драйвера. Після цього відбувається безперервний запис конфігурації в чіп FPGA на основі SRAM.

Потім дані, що завантажені в пам'ять високої пропускної здатності (High Bandwidth Memory – HBM) всередині FPGA, і сигнал запуску передаються до ядра (kernel під керуванням XRT фреймворку). Усі ці кроки приховані від розробника, а запуск ядра виглядає як виклик функції мовою програмування.

Коли всі обчислення в FPGA виконані, ядро записує результат у вихідний буфер і формує сигнали завершення всіх операцій. Цей процес виконується ядром MicroBlaze в захищеній частині мікросхеми FPGA, яка називається оболонкою. Це попередньо встановлене мікропрограмне забезпечення від Xilinx для підтримки зв'язку за допомогою XRT фреймворку. Ядро сигналізує про закінчення операцій, надсилаючи спеціальні сигнали головній програмі.

Для програм штучного інтелекту це не довше кількох секунд. Наприкінці цього процесу хост-застосунок виявляє системний об'єкт, який виконує синхронізацію з операційною системою. Для хост- застосуноку це означає, що дані в спільних буферах мікросхеми FPGA готові до читання. На цьому робота ядра всередині FPGA завершена.

За допомогою описаного методу програмування FPGA можна створити попередньо скомпільовану колекцію бінарних файлів для виконання певних обчислень. Для прискорення в FPGA доцільно використовувати кілька десятків попередньо скомпільованих ядер із найпопулярнішими ресурсомісткими обчисленнями ШІ. Середній розмір бінарного файлу становить 20-100 Мб.

Такі комбінації ядер можна просто зініціювати в головній програмі крок за кроком і миттєво викликати, щоб отримати апаратний прискорювач з потрібними функціями. Можна викликати багато різних ядер з різних бінарних файлів, виконати їх у FPGA та очистити чіп протягом однієї секунди.

2. Апаратні прискорювачі для побудови ШІ як сервіс

Виклик прискорювача як функції у мові програмування дозволяє дуже динамічно змінювати проєкт в FPGA. Це допомагає надати ресурси FPGA або запустити проєкт як послугу, і кожен із цих конкретних прискорювачів може бути частиною реалізованого FaaS. Для виконання конкретних завдань штучного інтелекту доступ до прискорювача можна організувати у вигляді черги на основі часу.

Гетерогенні проєкти – це потужний спосіб спростити реалізацію штучного інтелекту на основі FPGA. Всередині одного чіпу FPGA можна розмістити велику кількість ядер. Таким ядрам властива гетерогенна природа і бути реалізовані за допомогою різних мов програмування. В одному бінарному контейнері ядра можуть мати різні властивості. Основна вимога до їх розміщення разом – відповідати загальному споживанню ресурсів мікросхеми FPGA. Це важливо як частина компіляції проєкту на етапі маршрутизації. Також це залежить від версії обраного середовища розробки.

Однією з найважливіших сфер, де використовуються рішення PCIe FPGA, є сфера, пов’язана з впровадженням штучного інтелекту. Виробники процесорів додають спеціальні регістри та спеціальну апаратну оптимізацію, щоб спростити реалізацію програми та підтримувати базові примітиви для конкретних типів обчислень ШІ.

Високорівневий синтез (HLS) дозволяє скоротити зусилля, необхідні для розробки проєкту ШІ з FPGA. На відміну від ЦП, FPGA набагато краще підходять для створення унікальних реалізацій архітектури [6]. Середовища розробки FPGA вже дозволяють використовувати існуючі бібліотеки для прототипування рішень ШІ. Складність проєкту для розробників значно спрощується, коли фреймворк XRT використовується для зв’язку з FPGA через PCIe. З іншого боку, час виходу на ринок проєктів з використанням HLS значно менший, ніж у випадку класичної розробки RTL. Таке поєднання дозволяє відповідати вимогам і закономірностям необхідної динаміки розробки проєктів з ШІ.

Слід зазначити, що на вході середовища розробки є набір описів проєкту, представлених мовами високого рівня. Під час тривалого процесу компіляції ці описи перетворюються в остаточний бінарний файл за допомогою бібліотек ШІ, компіляторів, оптимізаторів і пакувальників.

Створені бінарні файли можна завантажувати, використовувати для програмування мікросхеми FPGA з головної програми та використовувати як частину набору бінарних файлів для програмування під час виконання всередині FPGA.

Пам'ять високої пропускної здатності (HBM) це технологія вбудованої динамічної пам'яті, яка дозволяє пряме підключення до FPGA. Це корисна функція, яка забезпечується шляхом передачі параметрів і даних за допомогою спільної високошвидкісної пам’яті.

Алгоритми та рішення ШІ вимагають великого обсягу пам’яті. Можливість використання такої швидкої пам'яті спрощує обробку та дозволяє успішно реалізовувати алгоритми, яким треба багато пам’яті.

FPGA для штучного інтелекту та дата-центри перейшли на ще більш швидку пам’ять під назвою High Bandwidth Memory (HBM). Це загальний тренд перенесення всередину кристала всіх можливих перемичок і підкладок, щоб це все було в одному корпусі. Результатом такої інтеграції є зниження енергоспоживання, підвищення продуктивності та надійності таких рішень.

Один чіп UltraScale+ HBM FPGA використовує 8 або 16 ГБ динамічної пам'яті, пропускна здатність такої пам'яті становить до 460 ГБ/с [7]. Ця швидкість є теоретичним максимумом для більшості практичних реалізацій, оскільки вона розраховується на основі пропускної здатності інтерфейсу та логіки читання, що працює на частоті 460 МГц у пакетному режимі. На практиці досяжна швидкість в лінійному режимі читання може досягати 400 гігабайт в секунду і близько 260-280 гігабайт в секунду у випадку довільного читання [8].

Тим не менш, цієї пропускної здатності достатньо для побудови великого класу систем штучного інтелекту. У зв'язку з цим новий клас таких інтенсивних щодо швидкодії пам'яті рішень можна класифікувати як мікросхеми FPGA з HBM. Це тренд для великої кількості пристроїв. Значення 8-16 ГБ – це не обмеження, а лише поточний розмір для HBM2e.

Дата-центри застосовують FPGA прискорювачі для ШІ для побудови спеціалізованих обчислювальних кластерів. FPGA дозволяє створити для штучного інтелекту власну архітектуру, яка використовує цю динамічну HBM пам’ять для зберігання даних і обміну ними з головною програмою, а самі обчислення та операції реалізуються безпосередньо як апаратна логіка.

Для підвищення ефективності роботи з картами Alveo FPGA вони згруповані в одне шасі для монтажу в дата-центрах і повинні бути добре та централізовано охолоджені. При цьому одна хост-машина містить велику кількість слотів з платами FPGA.

Усі вони підключаються до цієї хост-машини й усі разом стають суперлогікою, яку можна програмувати незалежно або виконувати одне завдання в межах усього центру обробки даних. Застосовувати такі рішення можна в більшості областей, де потрібна обробка даних [9].

ISSN 1727-7337 (print)
ISSN 2663-2217 (online)

Для центрів обробки даних це дуже цікаво, оскільки ефективність FPGA набагато вища, ніж у графічних процесорів або процесорів загального призначення. Перевага продуктивності FPGA для штучного інтелекту базується на здатності створювати власну архітектуру всередині FPGA. Це допомагає набагато швидше вирішувати типові завдання з однаковим або меншим енергоспоживанням.

Саме спеціалізовані прискорювачі забезпечують істотний приріст продуктивності за рахунок паралельного виконання великої кількості операцій. Така реалізація дозволяє знизити енергоспоживання в центрах обробки даних (ЦОД), що є суттєвим через високе навантаження та обсяг обробки даних, і відповідно зменшує виробництво як теплової енергії, так і вуглекислого газу під час роботи ЦОД.

3. Аналіз засобів FPGA розробки для ШІ

Набір інструментів середовища розробки є інформативним, пропонує розробнику набір вікон і нагадує більшість середовищ розробки. Але слід зазначити, що є деякі додаткові параметри, включаючи кілька режимів компіляції. Один із них називається системним рівнем у SDAccel та апаратним рівнем у Vitis [10]. Цей режим дозволяє генерувати бінарні файли на основі проєкту для виконання програмування FPGA на ходу.

Процес компіляції проєкту в такому режимі дуже тривалий. Мінімальний час компіляції близько 1 години. Максимальна тривалість може досягати кількох днів. Середній час отримання бінарних файлів для чіпів UltraScale+ FPGA займає 5-6 годин для проєкту ШІ і залежить від обсягу оперативної пам’яті на машині Linux, де відбувається розробка.

Процес розробки з використанням SDAccel або потоку проєктування Vites вимагає додаткових зусиль для отримання кінцевого результату. У розробці апаратного забезпечення ми повинні чекати до 2 годин, щоб перевірити певну ідею, але в розробці програмного забезпечення можна протестувати модель після кількох хвилин компіляції.

Вимоги до продуктивності ПК для розробки у разі використання таких FPGA набагато вище, ніж для звичайної розробки програмного забезпечення. Виробник (Xilinx) рекомендує використовувати 64 ГБ оперативної пам'яті через механізм технічної підтримки для усунення помилок. Середовище розробки використовує Java, і цей набір інструментів споживає багато ресурсів, включаючи пам'ять. З 32 ГБ оперативної пам’яті компіляція великого проєкту для ШІ вимагає інтенсивного використання диска підкачки. Однак періодично виникають помилки, коли середовище розробки намагається інтенсивно використовувати пам’ять.

Процес компіляції таких рішень дуже тривалий, навіть якщо є достатній обсяг оперативної пам'яті. Це пов'язано зі ступенем інтеграції сучасних мікросхем FPGA. При компіляції проєктів їм необхідно враховувати безліч можливих варіантів трасування великої кількості ресурсів і для внутрішньої оптимізації проєктів. Під час компіляції існує багато варіантів розміщення, які слід розглянути, щоб отримати бажану кінцеву тактову частоту (до 500 МГц, якщо використовується XRT) [11, 12].

Додавання параметрів дозволяє збільшувати чи зменшувати цей час в залежності від бажаних параметрів і обраного плану оптимізації.

Є можливість поєднувати різні ядра в складі рішення без переробки розробленого проєкту. Потрібні ядра можна додати до бінарного контейнера, що дозволяє додати набір певних комбінацій існуючих ядер та їх комбінацій. І якщо їх загальне споживання ресурсів дозволяє розмістити їх в одному чіпі, зазначеному як цільовий пристрій у налаштуваннях проєкту, тоді такий проєкт можна скомпілювати.

4. Аналіз технологій реалізації сервісів штучного інтелекту в FPGA

Класичні технології проєктування FPGA також підтримуються сучасними засобами розробки для завдань ШІ. Середовища розробки підтримують широкий спектр мов і технологій проєктування [13].

Блок-дизайн – це частина класичного опису схеми, яка підтримується для прототипування проєктів FPGA. Використовуючи цей метод опису, можна будувати серйозні продукти. Використання стандартних блоків із заздалегідь визначених шаблонів дозволяє оптимально описати ці рішення. Оптимально в даному випадку означає точне значення, а саме, використання не більше 60 відсотків ресурсів мікросхеми. Таку рекомендацію надають постачальники мікросхем FPGA для підтримки максимально можливої частоти з низьким енергоспоживанням.

RTL – це найефективніший спосіб розробки проєкту FPGA. Використання стандартних мов опису апаратного забезпечення часто називається в інструментах розробки FPGA потоком RTL. Список мов включає System Verilog, Verilog і VHDL. Вони дозволяють описати проєкт FPGA на максимально детальному рівні, аж до точного елемента або реєстру. Цей опис близький до опису мовою низького рівня, наприклад С.

DCP – це найнижчий можливий рівень програмування проєктів FPGA. Цей рівень опису близький до асемблера в мовах програмування. DCP означає Design CheckPoint. Цей спосіб прототипування не входить у звичайний список методів опису, оскільки для його використання потрібні дуже специфічні

знання. Оптимізація DCP використовується для оптимізації деяких рішень, які надходять у масове виробництво та передаються з FPGA на ASIC.

Мови та технології для створення ШІ проєктів для FPGA також включають мови програмування. Фреймворки і мови програмування ШІ використовуються для створення прототипів FPGA. Описати проєкт можна мовами, які в класичному розумінні непридатні для опису рішень FPGA, а саме мовою OpenCL, мовою C, C++, мовою Python, TensorFlow і Caffe.

Насправді Python, TensorFlow і Caffe були додані безпосередньо в середовищі розробки Vitis. Більшість із цих мов в першу чергу орієнтовані на застосунки в системах ШІ, оскільки більшість бібліотек для вирішення завдань ШІ написані на тих же мовах.

Немає нічого принципово неможливого у використанні таких мов для проєктів FPGA. Насправді існують деякі вимоги до використання цих мов для FPGA. Якщо проєкт описано на одній із цих мов відповідно до наведених форматів і вимог, тоді результатом є проміжний рівень HLS, який дозволяє перетворити це представлення в код SystemVerilog, який буде достатньо обфускований, оскільки він призначений для використання самим середовищем розробки.

Це внутрішній проміжний файл, створений Vivado, після перекладу він буде скомпільований у звичайний спосіб для проєктів RTL. Проєкти на таких мовах зазвичай використовують 50-60 відсотків ресурсів, що в принципі дуже добре, оскільки для написання проєктів на цих мовах і технологіях від розробника не потрібні якісь спеціальні знання.

Deep Learning Processing Unit (DPU) – базовий блок для обробки часто використовуваних обчислень в алгоритмах ШІ. Він також є базовим еквівалентом процесора, який представляє потік даних як потік інструкцій і обробку цих даних.

Xilinx має намір використовувати блоки DPU як частину своїх FPGA, які, по суті, є окремими IP-ядрами, які можна параметризувати для використання великої кількості популярних нейронних мереж, включаючи VGG, ResNet, GoodLeNet, YOLO, SSD, MobileNet, FNP та інші. Важливою сферою використання DPU є обробка графічних даних, у тому числі обробка графічних даних у режимі реального часу. Кожен екземпляр DPU дозволяє вибрати необхідні параметри на етапі проєктування; після компіляції проєкт FPGA включає рішення, вже спеціалізоване для вибраного завдання.

Необхідно розуміти ієрархію всіх технологій і те, що в кінцевому підсумку все, що робиться, виконується на самому базовому рівні, яким є DPU.

5. Рекомендації з оптимізації нейронної мережі для апаратної FPGA реалізації

Оптимізація нейронної мережі для використання в FPGA починається з модифікації початкової моделі програмного забезпечення. Треба враховувати, як це працює. Під час оптимізації такої реалізації мережі для апаратних рішень необхідно трансформувати деякі частини відповідно до загальних вимог до апаратної реалізації, які також використовуються під час розгортання циклів.

У більшості випадків мережа – це кілька рівнів вузлів, які здійснюють голосування або голосування по порогу залежно від типу мережі.

Прунінг гілок із низьким рівнем пріоритету є важливою частиною оптимізації апаратного забезпечення. Деякі з гілок просто істотно не впливають на кінцевий результат. Цю оптимізацію можна здійснити, відрізавши деякі з'єднання та, можливо, деякі вузли.

Це дозволяє оптимізувати описи та моделі нейронної мережі для перенесення на апаратну реалізацію FPGA. Середовище розробки обіцяє від 5 до 50 разів прискорити впровадження FPGA. Це суттєва оптимізація обсягу ресурсів, які можна використовувати, оскільки пряма передача досить ресурсомістка.

Використання представлення чисел із фіксованою точкою також допомагає зменшити складність на наступному етапі оптимізації. Важливо використовувати типи даних, які можна легко реалізувати в FPGA. У класичних випадках нейронна мережа використовує представлення дійсних значень від 0 до 1. Для його представлення використовується тип даних float. Це число з плаваючою точкою одинарної точності, яке складається з 4 байтів.

Для передачі такого опису моделі нейронної мережі на апаратне забезпечення необхідно використовувати представлення числа з фіксованою. Він може бути представлений на основі діапазонного цілочисельного типу даних з різною кількістю бітів. Точна кількість бітів залежить від можливої кількості розпізнаваних рівнів. За допомогою рівня квантування таке перенесення можна здійснити з необхідною точністю. Це можна зробити вручну під час створення прототипу.

Отримання фіксованої архітектури є останнім кроком оптимізації. На цьому етапі відбувається оптимізація таких рішень. Він включає розмежування потоку команд і потоку даних. Оптимізація залученої конструкції з попередніми кроками дозволяє отримати остаточну оптимізовану архітектуру.

У результаті виникає потік даних і набір інструкцій, які виконуються всередині DPU.

ISSN 1727-7337 (print)
ISSN 2663-2217 (online)

6. Прототип медичного застосунку на основі ШІ

Рентген грудної клітки відіграє вирішальну роль у діагностиці пневмонії, але виявлення пневмонії на рентгенограмі грудної клітки є складним завданням, яке залежить від досвіду рентгенологів. Штучний інтелект може допомогти лікарям вчасно діагностувати пневмонію та визначити її тип, щоб призначити оптимальний курс лікування пацієнта. Якщо раніше технологія FPGA використовувалася лише для обробки томографічних зображень [14], то тепер вона також може бути залучена до процесу аналізу контенту зображень.

Наданий прототип медичного застосунку є сервісом аналізу рентгенівських зображень для виявлення пневмонії та її типу.

Було обрано наступні інструменти розробки: середовище розробки VS Code, мова програмування Python, фреймворк розробки нейронної мережі PyTorch, інструмент відстеження процесу навчання TensorBoard, фреймворк розробки веб-застосунків Streamlit і Streamlit sharing для розгортання застосунків, система контролю версій GitHub.

Основою програми є згорткова нейронна мережа, оскільки цей тип мережі найкраще підходить для аналізу зображень [15]. Була використана мережа DenseNet-121 з модифікованим вихідним шаром, відповідальним за класифікацію. В офіційній версії мережа має 1000 виходів, що відповідає 1000 класам. Оскільки необхідно класифікувати наявність пневмонії та її тип, було вирішено використовувати для класифікації 3 виходи, які означають: відсутність пневмонії, вірусну пневмонію, бактеріальну пневмонію. Для навчання мережі було обрано загальнодоступний набір даних «Chest X-Rays – Pneumonia Detection». Для вивчення та тестування нейронної мережі потрібні три набори даних: тестувальний, навчальний та валідаційний. На навчальному наборі мережа навчалася, тобто на ній вибиралися найбільш оптимальні параметри.

Валідаційний набір потрібен для перевірки мережі під час навчання, щоб побачити, наскільки вона точна щодо даних, яких раніше не бачала. Точність мережі була перевірена на тестовому наборі. Існуючий набір даних не мав валідаційного набору, тому його було створено шляхом переміщення кожного п'ятого зображення з навчального набору до валідаційного. Таким чином було отримано рівномірно розподілену кількість зображень для кожного класу мережі (бактеріальна пневмонія, вірусна пневмонія, відсутність пневмонії) в кожному з наборів даних (тренувального, валідаційного та тестувального). Розподіл кількості зображень кожного класу для кожного набору даних показано на рисунку 1.

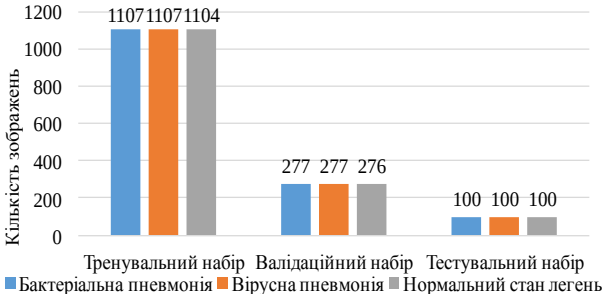

Рис. 1. Порівняння кількості зображень у кожному наборі після поділу навчального набору

Моніторинг результатів мережі під час навчання мережі за допомогою валідаційного набору є важливим процесом, оскільки він допомагає уникнути перенавчання мережі. Перенавчання – це процес, коли мережа просто запам'ятовує дані замість навчання, тобто значення функції втрат залежно від епохи навчання зменшується на тренувальному наборі та збільшується на валідаційному.

Щоб навчити мережу, потрібно вибрати:

– кількість епох (епоха – це одна ітерація навчального процесу, що містить в собі представлення всіх прикладів із тренувального набору та перевірку якості навчання);

– розмір пакету даних – гіперпараметр, який визначає кількість навчальних прикладів, які необхідно опрацювати перед оновленням внутрішніх параметрів моделі, тобто за одну ітерацію;

– функція втрат – показник того, наскільки точно ваша модель здатна передбачити очікуваний результат (чим менше значення функції втрат, тим краще передбачення моделі);

– оптимізатор, що оновлює параметри мережі.

Для навчання мережі вибрано такі параметри:

– міні-розмір пакету даних, що дорівнює 32 (останній пакет відкидається якщо був неповним);

– 200 епох із застосуванням методу ранньої зупинки, якщо значення функції втрат на валідаційному наборі не змінювалося протягом 5 епох;

– функція втрат – крос-ентропія, оскільки вона добре підходить для багатокласової класифікації;

– оптимізатор мережі з використанням алгоритму Адам з еталонною швідкістю навчання 10-3;

– виходом мережі є значення ймовірності для кожного класу. Оскільки можна мати лише один із класів для кожного зображення, була використана функція softmax, яка обробляє вихідні дані мережі таким чином, щоб усі значення були в діапазоні від 0 до 1 і сумою 1;

– для оцінки точності мережі використовувалася метрика точності (accuracy), оскільки не було дисбалансу в кількості зображень для кожного класу в заданому тренувальному наборі.

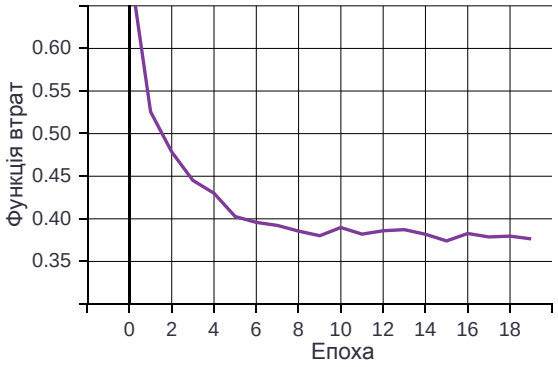

Рис. 2. Графік зміни функції втрат
під час навчання моделі для тренувального набору

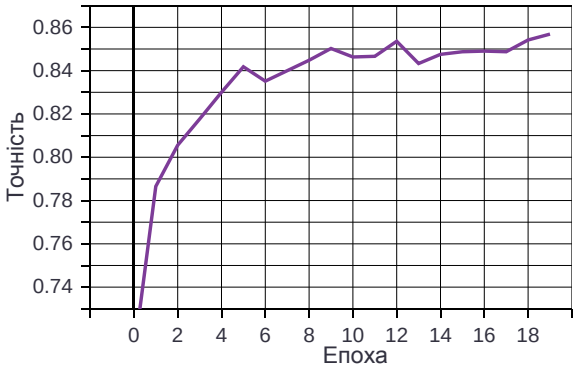

Рис. 4. Графік зміни точності моделі
під час навчання для тренувального набору

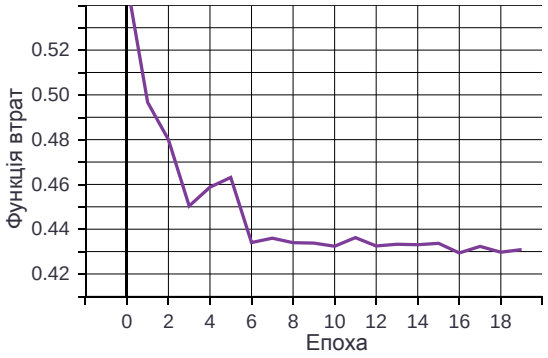

Рис. 3. Графік зміни функції втрат
під час навчання моделі для валідаційного набору

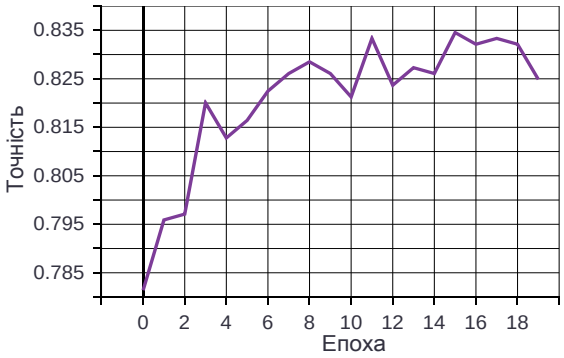

Рис. 5. Графік зміни точності моделі
під час навчання для валідаційного набору

Було використано TensorBoard для відстеження функції втрат і точності під час навчання мережі. Графік функції втрат під час навчання мережі для тренувального та валідаціного наборів відповідно показані на рисунках 2-3. Графік зміни точності під час навчання мережі для тренувального та валідаціного наборів відповідно показано на рисунках 4-5.

З рисунків 2-5 видно, що найбільше спадання функції втрат (рис. 2-3) та найбільше зростання точності моделі (рис. 4-5) спостерігається при збільшенні кількості епох від 0 до 13 для тренувального та від 0 до 15 – для валідаційного наборів. Подальше збільшення кількості епох не призводить ні до подальшого спадання функції втрат (рис. 2-3), ні до зростання точності (рис. 4-5) моделі під час навчання.

Після навчання на тестовому наборі була перевірена точність передбачення мережі, яка склала 86%. Для простоти використання був створений вебзастосунок, який розгорнуто на Streamlit sharing.

Висновки

Розвиток інтегральних схем, перехід на 5-нм технологічний процес і 1,5 нм протягом найближчих років вказують на перспективу постійного зростання технологічного удосконалення таких рішень.

Проаналізовано процес програмування FPGA прискорювачів для проєктів ШІ. Виконано аналіз можливостей мікросхем FPGA із швидкою HBM пам'яттю для побудови проєктів, які вимагають високу швидкодію та об'єм пам'яті. Детально проаналізовано мови опису, фреймворки, ієрархію інструментальних засобів для побудови апаратних прискорювачів проектів штучного інтелекту.

Проаналізовані етапи побудови проєктів штучного інтелекту з використанням нових засобів розробки програмованої логіки, базові блоки DPU та можливі параметри цих блоків надали змогу запропонувати рекомендації та етапи оптимізації нейронної мережі для реалізації з використанням FPGA.

На базі цих рішень було підготовлено, навчено та протестовано практичний кейс сервісу ШІ. Аналіз цієї реалізації показав точність прогнозу 86% на основі вибірки з 1000 випадків.

Головний внесок цього дослідження полягає в тому, що на підставі запропонованих ідей і рішень є зрозумілими наступні кроки щодо створення гетерогенних систем на основі поєднання трьох елементів:

– ШІ як послуга, що дозволяє використовувати функції штучного інтелекту як COTS продукт;

– прискорювачів FPGA як технології підвищення продуктивності, надійності та безпеки;

– хмарних або Edge (граничних) ресурсів для створення інфраструктури FPGA та ШІ як послуги.

Розвиток методологічного і технологічного базису є напрямом подальших досліджень і розробок.

ISSN 1727-7337 (print)
ISSN 2663-2217 (online)

Література

1. Perepelitsyn, A. FPGA as a Service Solutions Development Strategy [Text] / A. Perepelitsyn, I. Zarizenko, V. Kulanov // Proceedings 2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies, DESSERT 2020. – 2020. – P. 376–380. DOI: 10.1109/DESSERT50317.2020.9125017

2. Vitis AI User Guide, Xilinx, UG1414 (v2.5) [Online]. – Available at: https://docs.xilinx.com/r/en-US/ug1414-vitis-ai. – 15.06.2022.

3. Moskalenko, V. Neural network based image classifier resilient to destructive perturbation influences – architecture and training method [Text] / V. Moskalenko, A. Moskalenko // Radioelectronic and Computer Systems. – 2022. – No. 3. – P. 95-109. DOI: 10.32620/reks.2022.3.07.

4. Bouraou, N. Applications of neural networks for crack initiation and propagation monitoring in aircraft structures [Text] / N. Bouraou, S. Yutskevych, A. Kompanets // Aerospace Technic and Technology. – 2021. – No. 4sup2. – P. 99-103. DOI: 10.32620/aktt.2021.4sup2.13.

5. AI Engine Kernel Coding Best Practices, Xilinx, UG1079 (v2022.2) [Online]. – Available at: https://docs.xilinx.com/r/en-US/ug1079-ai-engine-kernel-coding. – 19.10.2022.

6. Vitis HLS User Guide, Xilinx, UG1399 (v2021.1) [Online]. – Available at: https://docs.xilinx.com/r/en-US/ug1399-vitis-hls. – 5.08.2021.

7. Alveo U280 Data Center Accelerator Card User Guide, Xilinx, UG1314 [Online]. – Available at: sandycast.com/support/documentation/boards_and_kits/accelerator-cards/ug1314-u280-reconfig-accel.pdf. – 27.02.2020.

8. UltraFast Design Methodology Guide for the Vivado Design Siute, Xilinx, UG949 (v2019.2) [Online]. – Available at: https://docs.xilinx.com/v/u/2019.2-English/ug949-vivado-design-methodology. – 6.12.2019.

9. What is an AI accelerator? [Online]. – Available at: https://www.synopsys.com/ai/what-is-an-ai-accelerator.html. – 27.02.2022.

10. Vitis Unified Software Platform Documentation: Application Acceleration Development, Xilinx, UG1393 (v2019.2) [Online]. – Available at: https://docs.xilinx.com/r/en-US/ug1393-vitis-application-acceleration. – 28.02.2020.

11. UltraScale Architecture Memory Resources. User Guide, Xilinx, UG573 (v1.10) [Online]. – Available at: https://docs.xilinx.com/v/u/en-US/ug573-ultrascale-memory-resources. – 4.02.2019.

12. 7 Series DSP48E1 Slice. User Guide, Xilinx, UG479 (v1.10) [Online]. – Available at: https://docs.xilinx.com/v/u/en-US/ug479_7Series_DSP48E1. – 27.03.2018.

13. Intel® FPGA Development tools. Intel. [Online]. – Available: https://intel.de/content/www/de/de/software/programmable/soc-eds/overview.html. – 27.02.2022.

14. Perepelitsyn, A. FPGA technologies in medical equipment: Electrical impedance tomography [Text] / A. Perepelitsyn, D. Shulga // Proceedings of IEEE East-West Design & Test Symposium (EWDTS 2013). – 2013. – P. 1-4. DOI: 10.1109/EWDTS.2013.6673157.

15. Densely Connected Convolutional Networks [Online] / G. Huang, Z. Liu, L. Van Der Maaten and K. Q. Weinberger. – Available at: https://doi.org/10.48550/arXiv.1608.06993. – 27.02.2022.

References

1. Perepelitsyn, A., Zarizenko, I., Kulanov, V. FPGA as a Service Solutions Development Strategy. *Proceedings 2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies, DESSERT 2020*, 2020, pp. 376–380. DOI: 10.1109/DESSERT50317.2020.9125017.

2. *Vitis AI User Guide, Xilinx, UG1414 (v2.5)*. Available at: https://docs.xilinx.com/r/en-US/ug1414-vitis-ai. (accessed June 15, 2022).

3. Moskalenko, V., Moskalenko, A. Neural network based image classifier resilient to destructive perturbation influences – architecture and training method. *Radioelectronic and Computer Systems*, 2022, no. 3, pp. 95-109. DOI: 10.32620/reks.2022.3.07.

4. Bouraou, N., Yutskevych, S., Kompanets, A. Applications of neural networks for crack initiation and propagation monitoring in aircraft structures. *Aerospace Technic and Technology*, 2021, no. 4sup2, pp. 99-103. DOI: 10.32620/aktt.2021.4sup2.13.

5. *AI Engine Kernel Coding Best Practices, Xilinx, UG1079 (v2022.2)*. Available at: https://docs.xilinx.com/r/en-US/ug1079-ai-engine-kernel-coding. (accessed October 19, 2022).

6. *Vitis HLS User Guide, Xilinx, UG1399 (v2021.1)*. Available at: https://docs.xilinx.com/r/en-US/ug1399-vitis-hls. (accessed August 5, 2021).

7. *Alveo U280 Data Center Accelerator Card User Guide, Xilinx, UG1314 (v1.3)*. Available at: https://www.sandycast.com/support/documentation/boards_and_kits/accelerator-cards/ug1314-u280-reconfig-accel.pdf. (accessed February 27, 2020).

8. *UltraFast Design Methodology Guide for the Vivado Design Siute, Xilinx, UG949 (v2019.2)*. Available at: https://docs.xilinx.com/v/u/2019.2-English/ug949-vivado-design-methodology. (accessed December 6, 2019).

9. *What is an AI accelerator?* Available: https://www.synopsys.com/ai/what-is-an-ai-accelerator.html. (accessed February 27, 2022).

10. *Vitis Unified Software Platform Documentation: Application Acceleration Development, Xilinx, UG1393 (v2019.2)*. Available at: https://docs.xilinx.com/r/en-US/ug1393-vitis-application-acceleration. (accessed February 28, 2020).

11. *UltraScale Architecture Memory Resources. User Guide, Xilinx, UG573 (v1.10)*. Available at: https://docs.xilinx.com/v/u/en-US/ug573-ultrascale-memory-resources. (accessed February 4, 2019).

12. *7 Series DSP48E1 Slice. User Guide, Xilinx, UG479*. Available at: https://docs.xilinx.com/v/u/en-US/ug479_7Series_DSP48E1. (accessed March 27, 2018).

13. *Intel® FPGA Development tools. Intel*. Available: https://intel.de/content/www/de/de/software/programmable/soc-eds/overview.html. (accessed February 27, 2022).

14. Perepelitsyn, A., Shulga, D. FPGA technologies in medical equipment: Electrical impedance tomography. *Proceedings of IEEE East-West Design & Test Symposium (EWDTS 2013)*, 2013, pp. 1–4. DOI: 10.1109/EWDTS.2013.6673157.

15. Huang, G., Liu, Z., L. Van Der Maaten, Weinberger, K. Q. *Densely Connected Convolutional Networks*. Available at: https://doi.org/10.48550/arXiv.1608.06993. (accessed February 27, 2022).

Надійшла до редакції 10.10.2022, розглянута на редколегії 20.11.2022

TECHNOLOGIES FOR IMPLEMENTING OF ARTIFICIAL INTELLIGENCE AS A SERVICE BASED ON HARDWARE ACCELERATORS

Artem Perepelitsyn, Yelyzaveta Kasapien, Herman Fesenko, Vyacheslav Kharchenko

The subject of study in this article is modern technologies, tools and methods of building AI systems as a service using FPGA as a platform. The **goal** is to analyze modern technologies and tools used to develop FPGA-based projects for systems that implement artificial intelligence as a service and to prepare a practical AI service prototype. **Task**: to analyze the evolution of changes in the products of leading manufacturers of programmable logic devices and experimental and practical examples of the implementation of the paradigm of continuous reprogramming of programmable logic; analyze the dynamics of changes in the development environment of programmable logic systems for AI; analyze the essential elements of building projects for AI systems using programmable logic. According to the tasks, the following **results** were obtained. The area of application of hardware implementation of artificial intelligence for onboard and embedded systems including airspace industry, smart cars and medical systems is analyzed. The process of programming FPGA accelerators for AI projects is analyzed. The analysis of the capabilities of FPGA with HBM for building projects that require enough of high speed memory is performed. Description languages, frameworks, the hierarchy of tools for building of hardware accelerators for AI projects are analyzed in detail. The stages of prototyping of AI projects using new FPGA development tools and basic DPU blocks are analyzed. The parameters of the DPU blocks were analyzed. Practical steps for building such systems are offered. The practical recommendations for optimizing the neural network for FPGA implementation are given. The stages of neural network optimization are provided. The proposed steps include pruning of branches with low priority and the use of fixed point computations with custom range based on the requirements of an exact neural network. Based on these solutions, a practical case of AI service was prepared, trained and tested. **Conclusions.** The main contribution of this study is that, based on the proposed ideas and solutions, the next steps to create heterogeneous systems based on the combination of three elements are clear: AI as a service, FPGA accelerators as a technology for improving performance, reliability and security, and cloud or Edge resources to create FPGA infrastructure and AI as service. The development of this methodological and technological basis is the direction of further R&D.

Keywords: Artificial Intelligence; FPGA; AI as a Service; Heterogeneous AI System Design; Hardware AI Accelerators; DPU; AI development tools; XRT.

Перепелицин Артем Євгенович – канд. техн. наук, доц., доц. каф. комп'ютерних систем, мереж і кібербезпеки, Національний аерокосмічний університет ім. М. Є. Жуковського «Харківський авіаційний інститут», Харків, Україна.

Касапєн Єлизавета Вячеславівна – студ. каф. комп'ютерних систем, мереж і кібербезпеки, Національний аерокосмічний університет ім. М. Є. Жуковського «Харківський авіаційний інститут», Харків, Україна.

Фесенко Герман Вікторович – д-р техн. наук, проф., проф. каф. комп'ютерних систем, мереж і кібербезпеки, Національний аерокосмічний університет ім. М. Є. Жуковського «Харківський авіаційний інститут», Харків, Україна.

Харченко Вячеслав Сергійович – д-р техн. наук, проф., зав. кафедри комп'ютерних систем, мереж і кібербезпеки, Національний аерокосмічний університет ім. М. Є. Жуковського «Харківський авіаційний інститут», Харків, Україна.

Artem Perepelitsyn – PhD, Associate Professor of Computer Systems, Networks and Cybersecurity Department, National Aerospace University «Kharkov Aviation Institute», Kharkiv, Ukraine,
e-mail: a.perepelitsyn@csn.khai.edu, ORCID: 0000-0002-5463-7889.

Yelyzaveta Kasapien – student at the Department of Computer Systems, Networks and Cybersecurity, National Aerospace University «Kharkov Aviation Institute», Kharkiv, Ukraine,
e-mail: e.kasapen@student.csn.khai.edu, ORCID: 0000-0002-2891-466X.

Herman Fesenko – Doctor of Technical Science, Professor, Professor at the Department of Computer Systems, Networks and Cybersecurity, National Aerospace University «Kharkov Aviation Institute», Kharkiv, Ukraine,
e-mail: h.fesenko@csn.khai.edu, ORCID: 0000-0002-4084-2101.

Vyacheslav Kharchenko – Doctor of Technical Science, Professor, Head of the Department of Computer Systems, Networks and Cybersecurity, National Aerospace University «Kharkov Aviation Institute», Kharkiv, Ukraine,
e-mail: v.kharchenko@csn.khai.edu, ORCID: 0000-0001-5352-077X.