УДК 004.8:004.27

АНАЛІЗ МЕТОДІВ ОПТИМІЗАЦІЇ МОДЕЛЕЙ КОМП'ЮТЕРНОГО ЗОРУ ДЛЯ EDGE-ПРИСТРОЇВ ІЗ НЕЙРОМЕРЕЖЕВИМИ ПРИСКОРЮВАЧАМИ

Ю. Ф. Петяк

Національний університет «Львівська політехніка»,
вул. С. Бандери, 12, Львів, 79013, Україна

Стаття присвячена аналізу сучасних методів оптимізації моделей комп'ютерного зору для розгортання на периферійних пристроях із обмеженими ресурсами. Розглянуто адаптацію моделей YOLO для edge-платформ Orange Pi 5, Raspberry Pi 5 із застосуванням апаратних прискорювачів (NPU, TPU, VPU). Особливу увагу приділено методам компресії: квантизації (INT8, FP16), проріджению (pruning), передачі знань (knowledge distillation) та оптимізації графів обчислень. Проаналізовано системи делегування інференсу TensorRT, RKNN Toolkit та Edge TPU Delegate для детекції об›єктів у реальному часі. Запропоновано комплексний пайплайн оптимізації від підготовки датасету до розгортання моделі з урахуванням обмежень енергоспоживання. Результати демонструють зменшення обсягу моделі до 70% при збереженні точності понад 98%. Висновки створюють основу для впровадження детекторів у вбудованих системах, безпілотних літальних апаратах, системах відеомоніторингу та інших edge AI застосунках, де критичними є швидкодія та енергоефективність.

Ключові слова: *комп›ютерний зір, YOLO, edge AI, оптимізація нейронних мереж, квантизація, NPU/TPU, детекція об›єктів, вбудовані системи, SBC, делегування інференсу.*

Постановка проблеми. Розгортання систем комп'ютерного зору (CV-моделей) на периферійних обчислювальних платформах (edge computing) набуває дедалі більшого значення у контексті розвитку технологій штучного інтелекту [1]. Зростання інтенсивності використання безпілотних літальних апаратів (БПЛА), а також стрімке збільшення обсягів відеоданих, що потребують аналізу в режимі реального часу, зумовлюють необхідність створення ефективних систем детекції повітряних об'єктів.

Виконання обчислень безпосередньо на периферійних пристроях (edge inference) забезпечує низку переваг порівняно з традиційною архітектурою «клієнт-сервер», зокрема: мінімізацію мережевої затримки (latency), підвищення автономності системи, зниження навантаження на канали передачі даних та забезпечення конфіденційності інформації [3]. Водночас, архітектури глибинних нейронних мереж, що використовуються для детекції об'єктів — зокрема моделі сімейства YOLO (You Only Look Once) [4] — характеризуються високою обчислювальною складністю та значними енергетичними вимогами, що створює суттєві обмеження для їх застосування на ресурсообмежених платформах.

У зв'язку з цим актуальною є проблема адаптації та оптимізації моделей комп'ютерного зору для виконання на вбудованих системах із спеціалізованими апаратними прискорювачами, такими як Neural Processing Unit (NPU), Tensor Processing Unit (TPU) або Vision Processing Unit (VPU) [5]. Ці прискорювачі забезпечують паралельне виконання тензорних операцій з високою енергоефективністю, що дозволяє досягти балансу між швидкодією інференсу, точністю детекції та споживанням енергії. Таким чином, дослідження методів оптимізації deep learning моделей для edge-платформ із нейромережевими прискорювачами становить важливий напрямок для забезпечення практичної реалізації систем детекції та трекінгу об'єктів у режимі реального часу.

Аналіз останніх досліджень та публікацій. Останні дослідження у сфері оптимізації моделей комп'ютерного зору для периферійних обчислювальних платформ демонструють значну активність на рівні як апаратних, так і програмних рішень. Ряд робіт [6, 7] присвячено адаптації полегшених версій CV-моделей, зокрема YOLO, для одноплатних комп'ютерів (Single-board computers, SBC), що дозволяє скоротити час інференсу і зменшити енергоспоживання. В [8, 9], детально описуються методи конвертації та оптимізації моделей комп'ютерного зору для сумісності з спеціалізованими апаратними прискорювачами, включно з quantization та pruning для зменшенням розмірів моделей. Дослідження [10] демонструють, що вибір апаратної платформи суттєво впливає на швидкість обробки відеокадрів, затримку та енергоспоживання, причому конфігурації з NPU/TPU-прискорювачами значно відрізняються по ефективності в задачах детектування та трекінгу об'єктів від систем, що використовують лише універсальні процесори. Водночас зазначається, що існує ряд обмежень: відсутність стандартизованих метрик оцінки продуктивності, неповна документація для нових версій YOLO для edge-пристроїв і недостатня кількість експериментальних даних щодо практичних сценаріїв застосування.

Таким чином, сучасні дослідження створюють основу для подальшої систематизації методів оптимізації моделей комп'ютерного зору, інтеграції нових апаратних прискорювачів та формування рекомендацій для розробників систем реального часу.

Мета статті. Задача дослідження полягає у узагальненні сучасних підходів та технічних рішень у сфері оптимізації моделей комп'ютерного зору для розпізнавання об'єктів на пристроях із вбудованими або зовнішніми нейронними процесорними модулями (NPU/TPU). Огляд охоплює ключові методи та інструменти, включаючи підходи до прискорення обчислень, конвертації моделей у легкі формати, а також оцінювання ефективності на різних апаратних платформах — таких як Raspberry Pi, Orange Pi, Jetson та Coral Edge TPU. Результати аналізу дозволяють оцінити практичну користь існуючих рішень, систематизувати та сформулювати основні напрямки майбутніх досліджень у сфері оптимізації моделей комп'ютерного зору для систем виявлення та трекінгу об'єктів у відеопотоці в режимі реального часу.

Виклад основного матеріалу дослідження. Використання deep neural networks (DNN) мереж на процесорах SBC пристроїв, таких як Raspberry Pi чи Orange Pi, або використання вбудованих систем (embedded systems), є неефективним через обмежену

обчислювальну потужність CPU, низьку паралельність обробки тензорних операцій та високі затрати енергії при тривалому навантаженні. Також процесори SBC схильні до тротлінгу під час інтенсивних обчислень через перегрів процесора, що призводить до зниження продуктивності в режимі реального часу. З огляду на це, для ефективної детекції та трекінгу об'єктів на периферійних пристроях застосовуються різні апаратні прискорювачі, які дозволяють зменшити затримки інференсу та підвищити продуктивність моделей комп'ютерного зору, забезпечуючи безперервну роботу систем у польових умовах застосування, наприклад у автономних дронах або системах виявлення та трекінгу БПЛА. Серед таких рішень можна виділити як інтегровані в системи-на-чіпі (SoC) NPU, так і спеціалізовані зовнішні прискорювачі.

До таких пристроїв відносяться, наприклад, Neural Processing Unit (NPU) — спеціалізовані апаратні прискорювачі, оптимізовані для виконання обчислень DNN мереж. NPU дозволяють значно підвищити швидкість інференсу за рахунок ефективної паралельної обробки операцій матричних та векторних обчислень, що становлять основу сучасних моделей комп'ютерного зору. Використання NPU на периферійних платформах, таких як NVIDIA Jetson Nano, Jetson Xavier NX, Google Coral Edge TPU або Intel Movidius Neural Compute Stick, забезпечує виконання складних моделей у реальному часі без необхідності передачі даних на віддалені спеціалізовані сервери з GPU та суттєво зменшує енергоспоживання, що критично для мобільних та автономних систем.

Вбудовані NPU в SBC пристроях, які створюються на базі процесора Rockchip RK3588, наприклад Orange Pi 5, забезпечують апаратне прискорення інференсу завдяки виділеним обчислювальним блокам для матричних операцій та тензорних обчислень [11]. Проте процесор Broadcom BCM2712, що застосовується в Raspberry Pi 5, не має вбудованих апаратних прискорювачів для моделей комп'ютерного зору, що обмежує продуктивність при запуску моделей для детекції та трекінгу об'єктів і потребує використання зовнішніх NPU/TPU модулів для ефективного інференсу.

До категорії таких модулів відносяться спеціалізовані периферійні прискорювачі, призначені для edge AI застосунків. Прикладами таких модулів є:

- Google Coral Edge TPU це апаратний прискорювач на базі TPU (Tensor Processing Unit) для inference моделей машинного навчання на периферійних пристроях. Coral Edge TPU спеціалізується на виконанні моделей у низькій точності Int8, що дозволяє досягати високої продуктивності при мінімальному енергоспоживанні [12]. Пристрій підтримує фреймворк TensorFlow Lite та сумісний з готовими моделями для детекції і класифікації об'єктів та широко використовується в embedded-системах, де критично важливі енергоефективність та швидкість інференсу.
- NVIDIA Jetson — це сімейство комп'ютерів на базі SoC для edge AI, оснащених GPU з архітектурою CUDA, оптимізованих для інференсу нейронних мереж та комп'ютерного зору. Jetson підтримує виконання моделей у FP16 та Int8, що забезпечує баланс продуктивності та точності [13]. Платформа сумісна з популярними AI фреймворками для оптимізації та прискорення інференсу.

– Intel Movidius реалізує VPU (Vision Processing Unit), спеціалізований апаратний блок для прискорення інференсу комп'ютерного зору та DNN мереж на edge пристроях [14]. VPU Movidius оптимізований для виконання операцій матричного множення та згортання, характерних для CNN (Convolutional Neural Networks), при значно нижчому енергоспоживанні порівняно з CPU або GPU. Архітектура VPU включає багатоядерні процесори SHAVE (Streaming Hybrid Architecture Vector Engine) для паралельної обробки даних та спеціальні блоки для оптимізації роботи з тензорами.
– Hailo-8 (Hailo Technologies) є високопродуктивним NPU, оптимізованим для виконання DNN мереж у режимі реального часу на edge-платформах. Архітектура Hailo-8 передбачає багаторівневу конвеєрну структуру з високою паралельністю для CNN та DNN моделей, що дозволяє ефективно реалізовувати завдання детекції об'єктів, розпізнавання облич та дорожніх знаків на периферійних пристроях [15].
– Kendryte K210 (Canaan Creative) являє собою інтегрований SoC з NPU для edge- та IoT-застосувань із низьким енергоспоживанням. Пристрій підтримує inference моделей типу CNN та MLP у 8-бітовій точності (Int8). Архітектура K210 включає dual-core RISC-V CPU та KPU (AI Accelerator), що забезпечує виконання базових CV та ML завдань на недорогих та компактних пристроях. В таблиці 1 представлено порівняння спеціалізованих прискорювачів.

Таблиця 1

Популярні SBC пристрої та зовнішні модулі з підтримкою NPU/TPU

Назва модуля	Продуктивність (TOPS)	Виконання	Підтримувані бібліотеки / фреймворки	Точність	Застосування
Hailo-8	26	PCIe, M.2, USB	Hailo Compiler, TensorFlow, ONNX	Int8, Int16	Дрони, розумні камери, робототехніка, детекція об'єктів
Kendryte K210	0.3	SoC	KendryteNN, TensorFlow Lite	Int8	Міні-дрони, смарт-датчики, розумні камери, IoT
Google Coral Edge TPU	4	USB, PCIe	TensorFlow Lite	Int8	Інференс CNN на периферії, розпізнавання образів
NVIDIA Jetson Xavier NX	21	SoC	TensorRT, PyTorch, ONNX, CUDA	FP16, Int8	Дрони, робототехніка, автономні системи, відеоспостереження
Intel Movidius Myriad X	1	USB, PCIe	OpenVINO	FP16, Int8	Edge-детекція об'єктів, CV-моделі на низькобюджетних пристроях

Застосування апаратних прискорювачів не є єдиним шляхом для пришвидшення інференсу та покращення якості детектування та трекінгу об'єктів. Ще одним є оптимізація моделей машинного навчання з метою зниження обчислювальних витрат та енергоспоживання без суттєвої втрати точності. Сучасні підходи включають кілька взаємодоповнюючих методик: квантизацію (Quantization), обрізання параметрів і розрідження (Pruning і Sparsity), передавання знань між моделями (Knowledge Distillation) та оптимізацію графа обчислень (Graph Optimization).

Квантизація передбачає зменшення розрядності параметрів моделі та проміжних значень під час інференсу. Найпоширенішими варіантами є Int8 та FP16, що дозволяють зменшити обсяг використаної пам'яті та обчислювальні ресурси, одночасно зберігаючи допустимий рівень точності. Наприклад, Int8-квантизація дозволяє виконувати інференс на egde пристроях із значним прискоренням, водночас зменшуючи енергоспоживання в 2–4 рази порівняно з FP32 [16]. FP16-квантизація є компромісом між продуктивністю та точністю, особливо для моделей із великою кількістю параметрів, таких як YOLOv5–v8.

Прийманням параметрів і розрідження (Pruning і Sparsity) є ще одним підходом для зменшення обчислювальної складності моделей. Pruning полягає у видаленні незначущих ваг нейронної мережі, що дозволяє скоротити кількість операцій множення та додавання під час інференсу. Розрідження створює структуру моделей із високою кількістю нульових значень, яка може ефективно оброблятися апаратними прискорювачами, оптимізованими для sparse-формату, що особливо актуально для SoC із обмеженою пропускною здатністю пам'яті [17].

Knowledge Distillation передбачає перенесення знань від великої teacher model до меншої student model, що дозволяє зберегти високу точність при значно менших розмірах та обчислювальних вимогах. Цей підхід ефективний у контексті edge AI, де ресурсні обмеження перешкоджають використанню великих моделей у реальному часі [18].

Оптимізація графа обчислень включає використання спеціалізованих інструментів та фреймворків, таких як ONNX, TensorRT, RKNN та TensorFlow Lite. Вони дозволяють перетворювати та адаптувати моделі для конкретної апаратної платформи, виконувати додаткові оптимізації (Fuse, Layer Merging, Operator Fusion) та генерувати прискорений код для інференсу. Наприклад, TensorRT оптимізує графи для NVIDIA Jetson, RKNN для Rockchip SoC, а TFLite для широкого спектра мобільних і периферійних пристроїв, включно з Coral Edge TPU [19].

Таким чином, поєднання квантизації, pruning, knowledge distillation та графових оптимізацій дозволяє значно підвищити продуктивність та енергоефективність детекторів об'єктів на edge платформах, роблячи можливим їхнє застосування у режимі реального часу на таких пристроях, як Raspberry Pi 5, Orange Pi 5, а також у вбудованих системах із підтримкою VPU або NPU.

У контексті edge-AI особливу увагу привертають моделі сімейства You Only Look Once (YOLO), які завдяки високій швидкодії та ефективності стали стандартом для задач детектування й трекінгу об'єктів. YOLO є одноетапним детектором, що формалізує задачу виявлення як єдине регресійне завдання для всього

зображення, поєднуючи локалізацію та класифікацію об'єктів в одному проході нейронної мережі. Такий підхід забезпечує мінімальну затримку під час інференсу, що критично для роботи у реальному часі на пристроях з обмеженими ресурсами.

Сучасні реалізації, зокрема YOLOv8 від Ultralytics, інтегрують оптимізовані архітектури та механізми прискорення інференсу, що робить їх придатними для периферійних платформ, таких як Jetson Nano, Jetson Xavier NX, Coral Edge TPU або Movidius Neural Compute Stick [20]. Для платформ із обмеженими обчислювальними ресурсами розроблено легкі версії, наприклад YOLO-Tiny, які зменшують глибину мережі та спрощують блоки, забезпечуючи прийнятну точність при зниженому енергоспоживанні та затримках обробки.

Один з варіантів оптимізації моделей сімейства YOLO для вбудованих та SBC платформ, зокрема квантизацію, проріджения (pruning) та knowledge distillation, представлено в [21]. Виділяється два основних підходи до квантизації:

1. Post-training quantization (PTQ) — застосовується після тренування моделі та дозволяє знизити точність обчислень з FP32 до INT8, скорочуючи розмір моделі приблизно у 4 рази та пришвидшуючи інференс на 50–70 %, проте іноді призводить до невеликого зниження mAP (2–3 %).

2. Quantization-aware training (QAT) — інтегрує ефекти квантизації безпосередньо під час тренування, дозволяючи зберегти майже еквівалентну точність FP32-моделі при значно меншому розмірі. Експерименти показали, що YOLOv8s-INT8 після QAT зменшила точність лише на 0.6 % та забезпечила продуктивність ≈ 80 FPS на Jetson Nano.

Ще одним напрямом оптимізації є структурне та неструктурне проріджения (pruning). При неструктурному pruning видаляються окремі ваги з малими значеннями, створюючи розріджену (sparse) матрицю параметрів та скорочуючи обчислювальні операції, хоча для повного використання переваг sparsity потрібні спеціалізовані бібліотеки (наприклад, TensorRT 8 Sparsity API). Структурне проріджения видаляє цілі канали або блоки, зберігаючи сумісність з фреймворками. Експерименти показали, що видалення до 40 % каналів підвищувало FPS на 35 % при втраті точності не більше 1.2 %.

Knowledge distillation (KD) дозволяє компактній student model імітувати вихід великої teacher model. Для YOLOv8 такий підхід дозволив зменшити модель із 27 M параметрів до 8 M при втраті точності лише 0.8 %. KD особливо ефективний у поєднанні з pruning та квантизацією, формуючи послідовний ланцюг оптимізацій, що дозволяє значно скоротити розмір моделі перед розгортанням на edge-пристроях з критичними вимогами до латентності (відеоаналітика, дрони, робототехніка).

Таким чином, сучасні методи оптимізації моделей YOLO дозволяють досягати балансу між продуктивністю, точністю та енергоспоживанням, роблячи їх ефективними для застосування в системах детекції та трекінгу об'єктів у реальному часі на периферійних платформах.

Оскільки одноетапні детектори, такі як YOLOv8, характеризуються значними обчислювальними потребами, застосування апаратних прискорювачів та делегування інференсу дозволяє забезпечити реальну швидкодію при обмежених

ресурсах. Найбільш поширеними підходами для делегування інференсу є використання TensorRT, RKNN Toolkit та Edge TPU Delegate, які реалізують оптимізацію моделей відповідно для GPU, NPU та TPU.

TensorRT (NVIDIA) є високопродуктивним оптимізатором та рушієм інференсу для графічних процесорів NVIDIA. Він дозволяє прискорювати виконання нейронних мереж шляхом попередньої оптимізації графу обчислень, управління пам'яттю та використання обчислювальних ядер GPU. TensorRT підтримує моделі у форматах ONNX та TensorFlow SavedModel, забезпечує квантизацію в FP16 та INT8, ф'южн операцій, автоматичний підбір ядра обчислень та динамічне управління пам'яттю. Для реалізацій YOLOv8 застосування TensorRT дозволяє скоротити час інференсу у 2–4 рази порівняно з базовим виконанням на FP32 у PyTorch, що робить його доцільним для серверних або вбудованих платформ на базі GPU, таких як NVIDIA Jetson.

RKNN Toolkit розроблено компанією Rockchip для оптимізації та виконання моделей на процесорах із інтегрованим Neural Processing Unit (NPU), зокрема на RK3399Pro, RK3588 та інших одноплатних комп'ютерах типу Orange Pi. RKNN забезпечує конвертацію моделей з форматів ONNX або TensorFlow Lite у власний формат .rknn, що підтримує виконання на NPU. Під час конвертації застосовуються методи post-training quantization до INT8, об'єднання шарів, оптимізація пам'яті та вибір найбільш ефективних операцій для NPU. Делегування інференсу через RKNN дозволяє прискорити виконання моделей на 5–20 разів порівняно з CPU, що є критичним для задач edge AI у відеоспостереженні та робототехніці.

Edge TPU Delegate, розроблений Google, призначений для прискорення виконання моделей у форматі TensorFlow Lite на апаратних модулях Edge TPU, таких як Coral USB Accelerator та Coral Dev Board. Основною вимогою для використання є повна квантизація моделі до INT8, оскільки TPU підтримує лише операції з цим форматом чисел. Він перенаправляє сумісні операції на TPU, залишаючи решту обчислень CPU, що дозволяє досягти високої енергоефективності та продуктивності. За рахунок цього час обробки кадру відеопотоку значно зменшується, що робить Edge TPU оптимальним для застосувань у IoT, системах розумного дому та автономних пристроях. Специфікація систем делегування інференсу представлена в таблиці 2.

Таблиця 2

Системи делегування інференсу

Параметр	TensorRT	RKNN	Edge TPU
Платформа	NVIDIA GPU	Rockchip NPU	Google Coral TPU
Тип моделі	ONNX, TensorFlow	ONNX, TFLite	TFLite (INT8)
Прискорення	2–6	5–20	4–10
Квантизація	FP16/INT8	INT8	Обов'язково INT8
Основне застосування	Сервери, ПК, Jetson	Edge AI, SBC	IoT, Smart Home
API / Delegate	TensorRT Engine, trt-py	RKNN Toolkit / rknn-runtime	TFLite Delegate

Усі три технології реалізують принцип делегування обчислень, тобто передачу обчислювально інтенсивних операцій з центрального процесора на спеціалізовані апаратні модулі. Це забезпечує значне зниження затримки інференсу, зменшення енергоспоживання та підвищення частоти обробки кадрів, що є критично важливим для детектування та трекінгу об'єктів у реальному часі. Основні відмінності полягають у типі апаратного прискорювача та підтримуваних форматах моделей.

Моделі YOLO звичайно тренують в PyTorch (Ultralytics), але для запуску на різних пристроях потрібні спеціальні формати. Бібліотека Ultralytics надає можливість експорту YOLO у ONNX, TensorRT, CoreML, TFLite тощо. Зокрема, перенесення моделі в TensorRT забезпечує до 5-кратного пришвидшення на NVIDIA GPU, а ONNX/OpenVINO дає до 3-кратного приросту швидкодії на CPU. Rockchip NPU потребує власного формату RKNN: за допомогою RKNN Toolkit мережа перетворюється у файл .rknn, оптимізований під NPU платформи RK3588, RK3566 тощо. Іншою альтернативою є NCNN – фреймворк від Tencent для мобільних CPU. Ultralytics також підтримує експорт у NCNN: конвертація в цей формат дозволяє отримати прискорений інференс при зниженні споживання пам'яті без суттєвої втрати точності. Отже, широка екосистема фреймворків (ONNX, TFLite, TensorRT, OpenVINO, RKNN, NCNN тощо) дає змогу перенести YOLO-проєкти на різноманітні пристрої, але вимагає врахування обмежень (розрядність, підтримка операцій) кожного середовища.

Незважаючи на високу продуктивність YOLO, його запуск на edge-пристроях пов'язаний із низкою складнощів:

– Втрати точності через квантизацію: перехід у INT8 може знизити mAP на 5–10 %.
– Перегрів апаратури: на Raspberry Pi 5 без охолодження температура під навантаженням піднімалась до ~80 °C, що призводить до троттлінгу під час тривалого інференсу.
– Відносно довгі затримки: усі версії YOLOv8 (особливо Medium) мають більші часи інференсу порівняно з EfficientDet або SSD на тих же пристроях. Наприклад, у тестах Jetson Orin показав мінімальне вікно обробки 16 мс для YOLOv8_n, тоді як Raspberry Pi 3/4/5 демонстрував значно гірші результати [22].
– Обмеження апаратного забезпечення: доступна пам'ять, тепловіддача та підтримка специфічних операцій (конволюції, нормалізацій) залишаються критичними для розгортання великих моделей

Таким чином, навіть сучасні NPU та GPU значно прискорюють YOLO, проте на edge-пристроях потрібен системний комплексний підхід для оптимізації виконання, що поєднує апаратні прискорювачі, методи компресії моделей та делегування інференсу. Наведемо основні етапи такого підходу:

1. Вибір та підготовка апаратного прискорювача:

– Orange Pi 5 оснащено Rockchip RK3588 з інтегрованим NPU, що прискорює критичні для DNN операції (матричні множення, згортки).

– На цьому етапі визначають характеристики NPU: продуктивність у TOPS, підтримувані формати чисел, обмеження енергоспоживання.

2. Оптимізація моделі YOLO. Модель підлягає кільком методам оптимізації:

– Post-Training Quantization (PTQ) — перетворення ваг та активацій у INT8/FP16 після тренування для зменшення розміру моделі та прискорення інференсу.
– Quantization-Aware Training (QAT) — тренування з урахуванням квантизаційних ефектів, що дозволяє зберегти точність FP32 при меншому розмірі.
– Pruning та Sparsity — видалення незначущих ваг або каналів для скорочення обчислювальної складності.
– Knowledge Distillation (KD) — передача знань від великої «вчительської» моделі до компактної «студентської» з метою збереження високої точності при менших ресурсних вимогах.

3. Конвертація та експорт. Після оптимізації модель експортується у сумісні формати:

– ONNX — універсальний формат для оптимізації графів та виконання на різних платформах.
– RKNN (.rknn) — специфічний формат Rockchip для запуску на NPU через RKNN Toolkit.
– TensorFlow Lite (TFLite) — для використання стандартних делегатів TPU або тестування на інших пристроях.

4. Делегування інференсу на NPU:

– Виконання на NPU через RKNN Runtime забезпечує:
– автоматичне відображення сумісних операцій,
– ефективне використання пам’яті та обчислювальних ядер,
– підвищення FPS та зниження latency.
– При необхідності можна інтегрувати додаткові делегати для GPU або TPU.

5. Верифікація та тестування:

– Вимірювання latency та FPS для різних розмірів зображень.
– Перевірка точності після квантизації та pruning.
– Оцінка енергоспоживання та температурних режимів для запобігання тротлінгу.

6. Ітеративне вдосконалення

– Додаткове тонке налаштування моделі.
– Коригування рівня квантизації.
– Перерозподіл операцій між CPU та NPU для оптимального балансу продуктивність/точність.

Нижче наведено запропонований пайплайн для оптимізації детекції об’єктів із використанням моделей YOLO для edge-платформи Orange Pi 5 (таблиця 3).

Таблиця 3

Етапи оптимізації та розгортання моделі YOLO на edge-платформі Orange Pi 5

Етап	Інструменти / Фреймворки	Формат моделі / Даних	Очікуваний результат
Збір та розмітка датасету (Data Collection & Annotation)	CVAT, Label Studio, Roboflow	COCO JSON, YOLO TXT	Створений та структурований набір даних для навчання
Аугментація даних (Data Augmentation)	Albumentations, Ultralytics augmentations	Розширений датасет	Підвищення варіативності даних та стійкості моделі
Стратифікація вибірки (Data Stratification)	Scikit-learn, custom scripts	train / val / test	Збалансовані підвибірки з репрезентативними класами
Навчання базової моделі (Model Training)	PyTorch, Ultralytics YOLOv8	.pt (PyTorch)	Отримання навченої базової моделі
Оцінка точності (Model Evaluation)	PyTorch metrics, mAP@50/95, confusion matrix	звіт / графіки	Кількісна оцінка продуктивності моделі
Квантизація (Quantization)	PyTorch QAT/PTQ, TensorFlow Lite Converter	INT8 / FP16	Зменшення розміру моделі та пришвидшення інференсу
Експорт у проміжні формати (Model Export)	PyTorch → ONNX / TFLite exporter	ONNX, TFLite	Модель, готова до оптимізації під NPU
Делегування інференсу (Inference Delegation)	RKNN Toolkit, TensorRT, Edge TPU Compiler	RKNN, TensorRT Engine	Оптимізована модель для конкретного апаратного прискорювача
Тестування на пристрої (Edge Deployment Testing)	Orange Pi 5 (RK3588 NPU), Python API	RKNN Runtime	Перевірка швидкодії та стабільності інференсу
Моніторинг продуктивності (Performance Profiling)	RKNN Profiler, top, perf, custom logger	звіти, метрики	Аналіз FPS, latency, споживання пам’яті та енергії

Висновки. У статті проведено систематичний аналіз методів оптимізації моделей YOLO для запуску на периферійних платформах із обмеженими обчислювальними ресурсами, таких як Orange Pi 5 та Raspberry Pi 5, із застосуванням апаратних прискорювачів (NPU, Edge TPU, VPU). Розглянуто ключові підходи до оптимізації, включаючи квантизацію (INT8, FP16), pruning, knowledge distillation та графові оптимізації через ONNX, TensorRT, RKNN і TensorFlow Lite. Сформульовано такі рекомендації:

– для систем без GPU (наприклад, SBC на ARM Cortex-A) — INT8 PTQ або QAT є оптимальним балансом швидкодії та точності;
– для платформ із NPU/TPU — mixed-precision quantization (FP16 + INT8) забезпечує найкращу продуктивність;
– комбіновані методи (pruning + QAT + KD) дозволяють зменшити обсяг пам'яті до 70 % при збереженні точності понад 98 % від базової.

Аналіз показав, що поєднання цих методів значно підвищує продуктивність моделей, зменшує затримку інференсу та знижує енергоспоживання при збереженні прийнятного рівня точності детекції.

Особливу увагу приділено делегуванню інференсу на апаратні прискорювачі, зокрема через RKNN Toolkit, TensorRT та Edge TPU delegate, що дозволяє ефективно використовувати обчислювальні ресурси та забезпечує стабільну роботу моделей у режимі реального часу на edge-пристроях. На основі проведеного аналізу автор статті запропонував інтегрований пайплайн оптимізації, який включає підготовку моделі (квантизація, pruning, knowledge distillation), конвертацію в сумісні формати (.rknn, .tflite, .onnx) та делегування інференсу на апаратний прискорювач. Цей пайплайн слугує практичним керівництвом для розгортання YOLO на периферійних платформах і забезпечує системний підхід до підвищення продуктивності.

Зараз існує широкий спектр рішень для прискорення YOLO на edge пристроях, але питання балансування швидкодії та точності лишаються відкритими. Потрібні подальші експерименти з новими архітектурами та прискорювачами, аби охопити весь діапазон практичних задач.

Отримані результати підкреслюють, що запропонований автором підхід відкриває перспективи для широкого впровадження детекторів об'єктів у вбудованих системах, автономних дронах, системах відеоспостереження та інших edge-AI застосунках. Подальші дослідження можуть бути спрямовані на вдосконалення пайплайнів оптимізації, адаптацію моделей під різні апаратні платформи та підвищення енергоефективності, що створює основу для практичних рішень із високою продуктивністю та точністю детекції на периферійних пристроях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Shi W., Cao J., Zhang Q., Li Y., Xu L. Edge Computing: Vision and Challenges // IEEE Internet of Things Journal. — 2016. — Vol. 3, No. 5. — C. 637–646.
2. Satyanarayanan M. The Emergence of Edge Computing // Computer. — 2017. — Vol. 50, No. 1. — C. 30–39.
3. Shi W., Dustdar S. The Promise of Edge Computing // Computer. — 2016. — Vol. 49, No. 5. — C. 78–81.
4. Redmon J., Divvala S., Girshick R., Farhadi A. You Only Look Once: Unified, Real-Time Object Detection // Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR). — 2016. — C. 779–788.
5. Jouppi N. P., Young C., Patil N., et al. In-Datacenter Performance Analysis of a Tensor Processing Unit // Proc. 44th Annual Int. Symposium on Computer Architecture (ISCA). — 2017. — C. 1–12.

6. Bochkovskiy A., Wang C.-Y., Liao H.-Y. M. YOLOv4: Optimal Speed and Accuracy of Object Detection. — arXiv:2004.10934. — 2020. — Електронний ресурс. — Режим доступу: https://arxiv.org/abs/2004.10934 (дата звернення: 20.10.2025).
7. Wang C.-Y., Bochkovskiy A., Liao H.-Y. M. YOLOv7: Trainable bag-of-freebies and aggregation of reparameterized modules. — arXiv:2207.02696. — 2022. — Електронний ресурс. — Режим доступу: https://arxiv.org/abs/2207.02696 (дата звернення: 20.10.2025).
8. Jacob B., Kligys S., Chen B., Zhu M., Tang M., Howard A., Adam H., Kalenichenko D. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference // Proc. IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR). — 2018.
9. Han S., Mao H., Dally W. J. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding // International Conference on Learning Representations (ICLR). — 2016. — Електронний ресурс. — Режим доступу: https://arxiv.org/abs/1510.00149 (дата звернення: 20.10.2025).
10. Ignatov A., Timofte R., et al. AI Benchmark: Running Deep Neural Networks on Android Smartphones // Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW). — 2019.
11. Orange Pi. Orange Pi 5 User Manual — Rockchip RK3588 Specifications. — 2023. — Електронний ресурс. — Режим доступу: http://www.orangepi.org/html/hardWare/computerAndMicrocontrollers/service-and-support/Orange-Pi-5.html (дата звернення: 20.10.2025).
12. Google. Coral Edge TPU Technical Specifications and Performance Guide. — 2023. — Електронний ресурс. — Режим доступу: https://coral.ai/docs/edgetpu/inference/ (дата звернення: 20.10.2025).
13. NVIDIA Corporation. Jetson Modules — Technical Specifications. — 2024. — Електронний ресурс. — Режим доступу: https://developer.nvidia.com/embedded/jetson-modules (дата звернення: 20.10.2025).
14. Intel Corporation. Intel Movidius VPU — Vision Processing Unit Architecture. — 2022. — Електронний ресурс. — Режим доступу: https://www.intel.com/content/www/us/en/products/details/processors/movidius-vpu.html (дата звернення: 20.10.2025).
15. Hailo Technologies. Hailo-8 AI Processor Datasheet. — 2023. — Електронний ресурс. — Режим доступу: https://hailo.ai/products/hailo-8-ai-accelerator/ (дата звернення: 20.10.2025).
16. Krishnamoorthi R. Quantizing deep convolutional networks for efficient inference: A whitepaper. — arXiv:1806.08342. — 2018. — Електронний ресурс. — Режим доступу: https://arxiv.org/abs/1806.08342 (дата звернення: 20.10.2025).
17. Molchanov P., Tyree S., Karras T., Aila T., Kautz J. Pruning Convolutional Neural Networks for Resource Efficient Inference // International Conference on Learning Representations (ICLR). — 2017. — Електронний ресурс. — Режим доступу: https://arxiv.org/abs/1611.06440 (дата звернення: 20.10.2025).
18. Hinton G., Vinyals O., Dean J. Distilling the Knowledge in a Neural Network. — arXiv: 1503.02531. — 2015. — Електронний ресурс. — Режим доступу: https://arxiv.org/abs/1503.02531 (дата звернення: 20.10.2025).
19. NVIDIA Corporation. TensorRT Developer Guide — Optimizing Inference Performance. — 2024. — Електронний ресурс. — Режим доступу: https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/ (дата звернення: 20.10.2025).

20. Ultralytics. YOLOv8 Documentation — Export and Deployment. — 2024. — Електронний ресурс. — Режим доступу: https://docs.ultralytics.com/modes/export/ (дата звернення: 20.10.2025).
21. Jani M. Optimization of YOLOv5 for Embedded Platforms: Quantization, Pruning and Knowledge Distillation // International Journal of Computer Vision and Machine Learning. — 2023. — Vol. 12, No. 3. — C. 45–62.
22. Alqahtani F., Al-Makhadmeh Z., Tolba A. Energy-Efficient Object Detection on Edge Devices: A Comparative Study of YOLO Variants on Raspberry Pi and NVIDIA Jetson // Sensors. — 2024. — Vol. 24, No. 8. — Article 2517.

REFERENCES

1. Shi, W., Cao, J., Zhang, Q., Li, Y., & Xu, L. (2016). Edge Computing: Vision and Challenges. IEEE Internet of Things Journal, 3(5), 637–646.
2. Satyanarayanan, M. (2017). The Emergence of Edge Computing. Computer, 50(1), 30–39.
3. Shi, W., & Dustdar, S. (2016). The Promise of Edge Computing. Computer, 49(5), 78–81.
4. Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 779–788.
5. Jouppi, N. P., Young, C., Patil, N., et al. (2017). In-Datacenter Performance Analysis of a Tensor Processing Unit. Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA), 1–12.
6. Bochkovskiy, A., Wang, C.-Y., & Liao, H.-Y. M. (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection. arXiv:2004.10934. Retrieved from https://arxiv.org/abs/2004.10934.
7. Wang, C.-Y., Bochkovskiy, A., & Liao, H.-Y. M. (2022). YOLOv7: Trainable bag-of-freebies and aggregation of reparameterized modules. arXiv:2207.02696. Retrieved from https://arxiv.org/abs/2207.02696.
8. Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., & Kalenichenko, D. (2018). Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).
9. Han, S., Mao, H., & Dally, W. J. (2016). Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. International Conference on Learning Representations (ICLR). Retrieved from https://arxiv.org/abs/1510.00149.
10. Ignatov, A., Timofte, R., et al. (2019). AI Benchmark: Running Deep Neural Networks on Android Smartphones. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW).
11. Orange Pi. (2023). Orange Pi 5 User Manual — Rockchip RK3588 Specifications. Retrieved from http://www.orangepi.org/html/hardWare/computerAndMicrocontrollers/service-and-support/Orange-Pi-5.html.
12. Google. (2023). Coral Edge TPU Technical Specifications and Performance Guide. Retrieved from https://coral.ai/docs/edgetpu/inference/.
13. NVIDIA Corporation. (2024). Jetson Modules — Technical Specifications. Retrieved from https://developer.nvidia.com/embedded/jetson-modules.

14. Intel Corporation. (2022). Intel Movidius VPU — Vision Processing Unit Architecture. Retrieved from https://www.intel.com/content/www/us/en/products/details/processors/movidius-vpu.html.
15. Hailo Technologies. (2023). Hailo-8 AI Processor Datasheet. Retrieved from https://hailo.ai/products/hailo-8-ai-accelerator/.
16. Krishnamoorthi, R. (2018). Quantizing deep convolutional networks for efficient inference: A whitepaper. arXiv:1806.08342. Retrieved from https://arxiv.org/abs/1806.08342.
17. Molchanov, P., Tyree, S., Karras, T., Aila, T., & Kautz, J. (2017). Pruning Convolutional Neural Networks for Resource Efficient Inference. International Conference on Learning Representations (ICLR). Retrieved from https://arxiv.org/abs/1611.06440.
18. Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the Knowledge in a Neural Network. arXiv:1503.02531. Retrieved from https://arxiv.org/abs/1503.02531.
19. NVIDIA Corporation. (2024). TensorRT Developer Guide — Optimizing Inference Performance. Retrieved from https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/.
20. Ultralytics. (2024). YOLOv8 Documentation — Export and Deployment. Retrieved from https://docs.ultralytics.com/modes/export/.
21. Jani, M. (2023). Optimization of YOLOv5 for Embedded Platforms: Quantization, Pruning and Knowledge Distillation. International Journal of Computer Vision and Machine Learning, 12(3), 45–62.
22. Alqahtani, F., Al-Makhadmeh, Z., & Tolba, A. (2024). Energy-Efficient Object Detection on Edge Devices: A Comparative Study of YOLO Variants on Raspberry Pi and NVIDIA Jetson. Sensors, 24(8), Article 2517.

doi: 10.32403/1998-6912-2025-2-71-140-154

ANALYSIS OF COMPUTER VISION MODEL OPTIMIZATION METHODS FOR EDGE DEVICES WITH NEURAL NETWORK ACCELERATORS

Yu. F. Petiak

Lviv Polytechnic National University,
12 S. Bandery Str., Lviv, 79013, Ukraine
e-mail: yurii.f.petiak@lpnu.ua

This article is devoted to the analysis of modern methods for optimizing computer vision models for deployment on edge devices with limited computational resources. The key approaches to adapting YOLO (You Only Look Once) family models for edge platforms such as Orange Pi 5 and Raspberry Pi 5 using hardware neural network accelerators (NPU, TPU, VPU) are examined. Special attention is paid to model compression methods, including quantization (INT8, FP16), structural and non-structural pruning, knowledge distillation, and computational graph optimization. Inference delegation systems are analyzed, including TensorRT, RKNN Toolkit, and Edge TPU Delegate, which ensure efficient use of specialized accelerators for real-time object detection.

The study demonstrates that the application of combined optimization methods allows achieving model size reduction up to 70% while maintaining detection accuracy above 98% of the baseline level. A comprehensive optimization pipeline is proposed, covering stages from dataset preparation to model deployment on the target platform, taking into account energy consumption and computational power constraints. The pipeline includes data collection and annotation, augmentation, stratified sampling, baseline model training, accuracy evaluation, quantization (QAT/PTQ), export to intermediate formats (ONNX, TFLite), inference delegation through specialized toolkits (RKNN, TensorRT, Edge TPU Compiler), edge deployment testing, and performance profiling.

Comparative analysis of hardware accelerators reveals significant differences in performance, supported precision formats, and application areas. Google Coral Edge TPU provides up to 4 TOPS with mandatory INT8 quantization, making it optimal for IoT and smart home systems. NVIDIA Jetson Xavier NX delivers 21 TOPS with FP16/INT8 support, suitable for drones, robotics, and video surveillance. Hailo-8 offers 26 TOPS for detection tasks in smart cameras and UAVs. Intel Movidius Myriad X (1 TOPS) is designed for low-budget CV applications, while Kendryte K210 (0.3 TOPS) targets mini-drones and IoT sensors.

The research findings create a foundation for practical implementation of object detectors in embedded systems, autonomous unmanned aerial vehicles, airspace video monitoring systems, and other edge-AI applications where performance, autonomy, and energy efficiency are critical. Future research directions include refining optimization pipelines, adapting models for diverse hardware platforms, and enhancing energy efficiency to enable high-performance and accurate detection solutions on edge devices.

Keywords: *computer vision, YOLO, edge AI, neural network optimization, quantization, NPU/TPU, object detection, embedded systems, SBC, inference delegation.*

Стаття надійшла до редакції 10.10.2025.
Received 10.10.2025.