УДК 004.032.26 DOI: 10.14529/cmse250403

АНАЛИЗ ПРОИЗВОДИТЕЛЬНОСТИ ВЫВОДА МОДЕЛЕЙ ГЛУБОКОГО ОБУЧЕНИЯ НА ПЛАТЕ BANANA PI BPI-F3 НА ПРИМЕРЕ ЗАДАЧИ КЛАССИФИКАЦИИ ИЗОБРАЖЕНИЙ

© 2025 И.С. Мухин, В.Д. Кустикова

Нижегородский государственный университет им. Н.И. Лобачевского
(603022 Нижний Новгород, пр. Гагарина, 23)
E-mail: ismukhin03@gmail.com, valentina.kustikova@itmm.unn.ru

Поступила в редакцию: 18.08.2025

В работе выполняется анализ производительности вывода известных нейросетевых моделей ResNet-50 и MobileNetV2, обеспечивающих решение задачи классификации изображений, на плате Banana Pi BPI-F3, которая построена на базе архитектуры RISC-V. Вывод запускается средствами доступных фреймворков: PyTorch, TensorFlow Lite, Apache TVM и ExecuTorch. Предварительно модели конвертируются в формат каждого целевого фреймворка. Выполняется проверка корректности решения задачи с использованием полученных нейронных сетей. Демонстрируется, что показатели качества классификации изображений для этих моделей хорошо соотносятся с опубликованными значениями. Далее выполняется подбор оптимальных параметров запуска вывода для каждого фреймворка и модели. Сравнительный анализ производительности вывода показывает, что ExecuTorch (с XNNPACK-бэкендом) для обеих моделей демонстрирует лучшие результаты. Для модели ResNet-50 показатель количества кадров, обрабатываемых за секунду (Frames per Second, FPS), меняется от 2.649 до 3.339 fps при оптимальных параметрах запуска в зависимости от размера входного набора данных, обрабатываемого за один прямой проход по сети, для MobileNetV2 — от 11.26 до 29.96 fps. TensorFlow Lite уступает ExecuTorch в среднем в ∼2.1 раза. PyTorch и Apache TVM демонстрируют более низкие показатели производительности. Предположительно это связано с тем, что вывод в этих фреймворках не в полной мере оптимизирован для процессоров архитектуры RISC-V.

Ключевые слова: глубокое обучение, классификация изображений, производительность вывода, PyTorch, TensorFlow Lite, Apache TVM, ExecuTorch, Banana Pi BPI-F3, RISC-V.

ОБРАЗЕЦ ЦИТИРОВАНИЯ

Мухин И.С., Кустикова В.Д. Анализ производительности вывода моделей глубокого обучения на плате Banana Pi BPI-F3 на примере задачи классификации изображений // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2025. Т. 14, № 4. С. 40–60. DOI: 10.14529/cmse250403.

Введение

Модели и методы глубокого обучения являются эффективными инструментами для решения различных прикладных задач [1]. Применение глубокого обучения начинается с построения архитектуры нейронной сети. Далее сеть *обучается* на выделенном наборе данных, называемом *тренировочной выборкой*, и выполняется тестирование на данных, которые модель «не видела» в процессе обучения. При достижении приемлемых показателей качества модель внедряется в реальную программную систему, иначе выполняется настройка параметров обучения и/или модификация архитектуры сети. *Внедрение* предполагает многократное решение задачи на новых данных, называемое выводом. *Вывод* — это прямой проход по сети с целью получения и последующей обработки ее выхода.

Анализ производительности вывода нейронных сетей является важным этапом в процессе их внедрения в приложения, которые, как правило, должны функционировать в режи-

ме реального времени на маломощных устройствах. RISC-V [2] — активно развивающаяся архитектура, которая имеет большие перспективы широкого распространения в подобных устройствах. Проведение исследований, связанных с анализом производительности вывода на устройствах RISC-V, способствует дальнейшему развитию системного и прикладного программного обеспечения для таких устройств. Также оно имеет значение для планирования специализированных расширений набора команд, ориентированных на ускорение вывода нейронных сетей.

Цель настоящего исследования состоит в том, чтобы провести сравнительный анализ производительности вывода двух широко известных моделей ResNet-50 [3] и MobileNetV2 [4], обеспечивающих решение задачи классификации изображений, при запуске на плате Banana Pi BPI-F3, которая построена на базе архитектуры RISC-V, с использованием доступных фреймворков глубокого обучения. Исследования по аналогичной тематике проводились ранее для других архитектур [5–8]. Обзор литературы показывает, что для RISC-V подобные работы только начинают появляться [9–15]. При этом авторы используют разные тестовые модели, библиотеки для вывода нейронных сетей и устройства RISC-V для проведения экспериментов. Для платы Banana Pi BPI-F3 результаты бенчмаркинга вывода до настоящего момента не публиковались. Также в отличие от существующих работ в данном исследовании используется наиболее полный спектр фреймворков, доступных в настоящее время для вывода на устройствах RISC-V (PyTorch [16], TensorFlow Lite [17], Apache TVM [18], ExecuTorch [19]). Наряду с этим, реализация вывода и необходимой инфраструктуры разрабатывается в рамках открытой программной системы бенчмаркинга вывода Deep Learning Inference Benchmark (DLI) [20, 21], вследствие чего является общедоступной и расширяемой с точки зрения тестовых моделей, поддерживаемых фреймворков глубокого обучения и устройств, используемых для запуска.

Работа построена следующим образом. В разделе 1 дается обзор по тематике анализа производительности вывода нейронных сетей на процессорах архитектуры RISC-V, отмечаются отличия от существующих исследований. Раздел 2 содержит описание фреймворков глубокого обучения, доступных для запуска на устройствах RISC-V. В разделе 3 приводится формальная постановка задачи классификации изображений с большим числом категорий. Далее (раздел 4) описывается методика анализа производительности вывода глубоких моделей, которая предлагается в [5] и используется в экспериментальной части настоящей работы. В разделе 5 рассматривается архитектура программной системы DLI и основные изменения, которые внесены в рамках данного исследования. Раздел 6 посвящен анализу результатов экспериментов, полученных при запуске вывода моделей ResNet-50 и MobileNetV2 на плате Banana Pi BPI-F3 средствами доступных фреймворков. В заключении приводится краткая сводка результатов, формулируются выводы.

1. Обзор литературы

Внедрение нейронных сетей в реальные программные системы — сложный и важный этап жизненного цикла глубоких моделей. Он предполагает, что построена нейросетевая модель, которая решает поставленную задачу с высокими показателями качества. Далее необходимо проанализировать производительность ее вывода на конкретном устройстве, где будет выполняться многократный запуск. В зависимости от полученных результатов анализа может осуществляться оптимизация сети, либо изменение архитектуры с целью получения более «легковесной» и качественной модели. Один из типовых подходов к оп-

тимизации — *квантование весов* [22]. Квантование предусматривает понижение точности весов от формата FP32 к INT8 или UINT8 без существенной потери качества решения задачи. Технически реализуется с помощью встроенных функций библиотек глубокого обучения, либо специализированных фреймворков (например, NNCF [23]). При этом следует учитывать, что используемый для вывода инструмент должен поддерживать запуск квантованных моделей.

В настоящем разделе рассмотрим исследования, которые наиболее близки к тематике анализа производительности вывода на устройствах RISC-V. Авторы [9] описывают процедуру включения и оптимизации P-расширения RISC-V с целью выполнения квантованных нейронных сетей в тензорном компиляторе Apache TVM. В экспериментальной части приводятся результаты бенчмаркинга группы моделей MobileNet-v1 и Inception-v3 с весами в форматах FP32 и INT8 на симуляторе Spike с поддержкой P-расширения RISC-V. В [10] предлагается сравнение двух машин на базе RISC-V и Raspberry Pi для нейросетевого вывода на примере группы моделей MobileNet, и даются рекомендации по использованию этих машин. Это системы Sipeed Maixduino с ускорителем сверточных нейросетей и Raspberry Pi 4B в сочетании с USB-ускорителем Coral от Google. Авторы [11] оценивают производительность широкого спектра рабочих нагрузок машинного обучения на RISC-V с использованием архитектурного симулятора с открытым исходным кодом gem5. Работа направлена на бенчмаркинг вывода нейросетевых моделей, обеспечивающих решение различных прикладных задач: классификация изображений, детектирование объектов, семантическая сегментация изображений, оценка глубины сцены и других. Цель работы [12] — оценить производительность вывода языковых моделей BERT и GPT-2 на 64-ядерной архитектуре RISC-V SOPHON SG2042 с поддержкой векторных инструкций RVV v0.7.1. Проводится бенчмаркинг моделей с включением RVV-расширения и без него, при этом используется OpenBLAS и BLIS в качестве бэкендов BLAS для фреймворка PyTorch. В [13] анализируется производительность вывода кодировщика в составе модели трансформера на трех маломощных платформах с архитектурой RISC-V. Выполняется исследование вывода для двух репрезентативных представителей семейства моделей BERT, выявляются узкие места и возможности оптимизации на процессорах RISC-V: XuanTie C906, C908 и C910. Авторы [14] исследуют энергоэффективность и производительность квантованных нейросетевых моделей, развернутых на маломощных устройствах с различными аппаратными архитектурами, включая RISC-V, x86, ARM 64 и ARM 32. Рассматриваются два принципиально разных типа моделей: рекуррентные нейронные сети и большие языковые модели. При этом оценивается, как эти модели работают в практических сценариях. Измеряется точность, время выполнения и энергопотребление.

Данная работа является развитием [15], в которой анализируется производительность вывода сети DenseNet-121 на плате Lichee Pi 4A при запуске средствами инструментария OpenVINO, библиотеки TensorFlow Lite и компилятора машинного обучения Apache TVM. В отличие от [15] в данном исследовании OpenVINO не участвует в сравнении вследствие того, что до настоящего момента многие нейросетевые преобразования не оптимизированы для RISC-V. Наряду с TensorFlow Lite и Apache TVM здесь еще рассматриваются фреймворки PyTorch и ExecuTorch. В отличие от других работ, представленных в обзоре, в исследовании используется другой набор тестовых моделей (ResNet-50 и MobileNetV2). При этом вывод запускается на устройстве Banana Pi BPI-F3. Также разрабатываемое программное решение, обеспечивающее сбор результатов качества и производительности вывода нейросетей,

является открытым (лицензия Apache 2.0) и расширяемым, и может быть использовано для исследования производительности вывода других моделей, фреймворков глубокого обучения и устройств RISC-V.

2. Фреймворки глубокого обучения для вывода на устройствах RISC-V

Для сравнения производительности вывода используется несколько широко известных инструментов глубокого обучения, которые портированы и оптимизированы (частично или полностью) их разработчиками для запуска на процессорах архитектуры RISC-V.

1. PyTorch [16] — обширная экосистема с открытым исходным кодом для решения прикладных задач с использованием машинного обучения. Инструмент разработан на базе библиотеки Torch. В настоящее время PyTorch является стандартом де-факто для обучения и тестирования нейронных сетей вследствие удобства реализации и расширяемости возможностей. В процессе его разработки использован положительный опыт создания более ранних фреймворков глубокого обучения. Разработчики PyTorch предоставляют интерфейсы для языков C++ и Python, а также обертки для Java.
2. TensorFlow Lite (с сентября 2024 года LiteRT) [17] — библиотека для развертывания глубоких нейросетевых моделей на мобильных устройствах и микроконтроллерах. Имеются интерфейсы для C++, Python и некоторых других языков программирования.
3. Apache TVM [18] — активно развивающийся компилятор моделей машинного обучения с открытыми исходными кодами. Цель разработки состоит в предоставлении инженерам инструмента для оптимизации и последующего эффективного вывода нейросетевых моделей на разных устройствах. Обеспечивается широкий спектр программных интерфейсов, далее в работе используется Python API.
4. ExecuTorch [19] — относительно новый фреймворк для вывода глубоких нейросетевых моделей на мобильных и периферийных устройствах, а также на микроконтроллерах. Он является частью экосистемы PyTorch Edge и обеспечивает эффективное развертывание различных моделей в формате PyTorch на маломощных устройствах. Разработчики предоставляют программные интерфейсы для C++, Python и ряда других языков. При этом поддерживается значительное количество бэкендов, гарантирующих эффективное исполнение низкоуровневых операций, которые возникают в нейронных сетях, на конкретном аппаратном обеспечении.

Следует отметить, что существуют другие фреймворки, обеспечивающие запуск глубоких нейронных сетей на устройствах RISC-V (например, ncnn [24]), но их использование затруднительно вследствие ограниченных возможностей доступных конвертеров моделей.

3. Постановка задачи классификации изображений

Задача классификации изображений состоит в том, чтобы определить категорию, которой принадлежит изображение, из допустимого набора классов. При решении задачи на входе глубокой нейросетевой модели имеется изображение I, как правило, в формате RGB с разрешением $w \times h$, где w — ширина, h — высота изображения соответственно. Изображение I представляется в виде трехмерной матрицы интенсивностей с элементами I_{ijk} , где $i \in \{0, 1, \ldots, w-1\}$, $j \in \{0, 1, \ldots, h-1\}$, $k \in \{0, 1, 2\}$. Интенсивности принимают целые неотрицательные значения в диапазоне от 0 до 255, либо вещественные значения в диапазоне от 0 до 1, если они нормированы. Выход классификационной нейронной сети — это

вещественный вектор, каждый элемент которого содержит достоверность принадлежности изображения к одной из допустимых категорий. Размер вектора соответствует количеству этих категорий N. Таким образом, нейросетевая модель обеспечивает построение отображения $\phi : I \to \mathbb{R}^N$. Цель решения задачи состоит в том, чтобы построить нейронную сеть, которая для входного изображения формирует вектор достоверностей, где индекс максимального значения отвечает номеру искомого класса.

4. Методика анализа производительности вывода

Общая схема анализа производительности вывода состоит из нескольких этапов [5].

1. Обучение и/или конвертация исходной глубокой модели в форматы различных фреймворков, которые предполагается использовать для ее вывода на конечном устройстве.
2. Анализ и сравнение качества работы полученных моделей для проверки корректности предыдущего этапа.
3. Определение оптимальных параметров для запуска вывода.
4. Сжатие и оптимизация моделей.
5. Анализ и сравнение качества оптимизированных моделей.
6. Сравнение производительности вывода с использованием полученного набора моделей.

Сжатие и оптимизация нейронных сетей выходит за рамки данного исследования. Поэтому последовательность изложения результатов соответствует пунктам 1, 2, 3 и 6.

5. Программная реализация

Deep Learning Inference Benchmark (DLI) [20, 21] — программная система, разрабатываемая в ННГУ, позволяющая собирать показатели качества и производительности вывода глубоких моделей в автоматическом режиме. Система предоставляет программные интерфейсы для вывода на языках C++ и Python, поддерживает вывод с использованием значительного количества широко известных фреймворков глубокого обучения. DLI включает следующие основные компоненты (рис. 1).

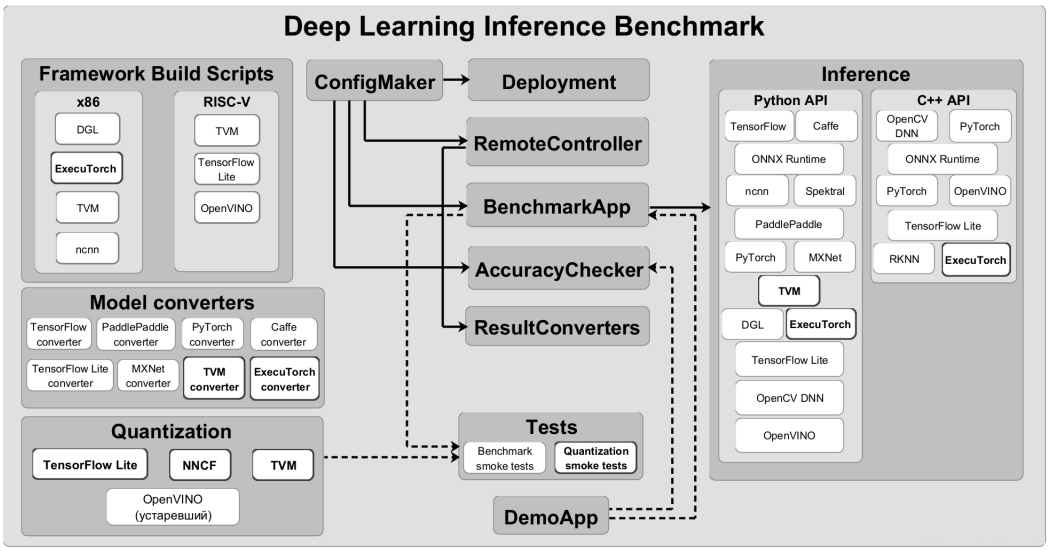

Рис. 1. Основные компоненты системы Deep Learning Inference Benchmark

1. **Framework Build Scripts** — независимый компонент, в состав которого входят скрипты для сборки различных фреймворков под архитектуры x86 и RISC-V.

2. **Model converters** — независимый компонент, содержащий программный интерфейс для конвертации моделей из формата одного фреймворка в формат другого.
3. **Quantization** — независимый компонент, в котором определяется интерфейс для квантования весов моделей с помощью встроенных в различные фреймворки инструментов.
4. **ConfigMaker** — графическое приложение для автоматизации процедуры формирования конфигурационных файлов для разных запускаемых компонент системы.
5. **Deployment** — компонент, обеспечивающий автоматическое развертывание тестовой инфраструктуры на вычислительных узлах средствами технологии Docker. Информация об узлах содержится в конфигурационном файле компонента.
6. **RemoteController** — компонент, выполняющий удаленный запуск экспериментов для сбора показателей производительности и качества глубоких моделей на вычислительных узлах с использованием компонент **BenchmarkApp** и **AccuracyChecker** соответственно и последующую агрегацию результатов экспериментов.
7. **BenchmarkApp** — компонент, реализующий сбор показателей производительности вывода набора моделей с использованием различных инструментов глубокого обучения. Информация о моделях и параметрах вывода описывается в конфигурационном файле.
8. **AccuracyChecker** — программная обертка над аналогичным инструментом из состава OpenVINO, который обеспечивает сбор показателей качества набора моделей. Информация о моделях содержится в конфигурационном файле.
9. **ResultConverters** — вспомогательный компонент, содержащий скрипты для конвертации результатов производительности и качества, которые агрегируются компонентом **RemoteController**, в форматы html и xlxs.
10. **Inference** — компонент, содержащий реализацию вывода моделей глубокого обучения средствами различных фреймворков. Присутствует поддержка программных интерфейсов для языков C++ и Python.
11. **Tests** — компонент, обеспечивающий автоматическую проверку корректности разработанных реализаций вывода с помощью различных фреймворков. На текущий момент проверяется возможность запуска и вывода показателей производительности.
12. **DemoAPP** — консольное приложение, демонстрирующее работу системы DLI в правильной последовательности. Использует явно **BenchmarkApp**, **AccuracyChecker** и неявно оставшиеся компоненты системы.

На схеме (рис. 1) полужирными рамками выделены части компонент, которые разработаны или модифицированы в рамках выполнения настоящего исследования: конвертеры моделей для TVM и ExecuTorch (компонент **Model Converters**), реализация вывода средствами TVM (Python API) и ExecuTorch (C++ и Python APIs) (компонент **Inference**). Отметим, что также добавлены обертки для квантования весов глубоких моделей (компонент **Quantization**). Эксперименты показывают, что в настоящее время разные версии некоторых фреймворков работают нестабильно с квантованными нейросетями на устройствах RISC-V, поэтому вопросы производительности таких моделей далее не затрагиваются.

6. Вычислительные эксперименты

6.1. Показатели качества

Для оценки качества классификации используется показатель точности top-k (top-k accuracy). Пусть N — количество допустимых категорий изображений, тогда выход модели — вектор достоверностей $y^j = (y_1^j, y_2^j, ..., y_N^j)$ для каждого изображения I^j в выборке,

где $j = \overline{1,S}$, S — общее количество изображений, y_i^j — достоверность того, что изображение I^j принадлежит классу i. Если среди k наибольших достоверностей $y_{i_1^j}^j, y_{i_2^j}^j, ..., y_{i_k^j}^j$, присутствует достоверность, соответствующая искомому классу, то изображение считается проклассифицированным корректно. Тогда *точность top-k* определяется как отношение числа правильно проклассифицированных изображений к общему их количеству:

$$topk = \frac{\sum_{j=1}^{N} I(l_j \in \{i_1^j, i_2^j, ..., i_k^j\})}{S}, \tag{1}$$

где l_j — номер искомого класса, $I(l_j \in \{i_1^j, i_2^j, ..., i_k^j\})$ — индикаторная функция, которая принимает значение 1, если $l_j \in \{i_1^j, i_2^j, ..., i_k^j\}$, и 0, в противном случае. Далее в экспериментах рассматриваются точности top-1 и top-5.

6.2. Показатели производительности

Эксперимент предполагает, что набор обрабатываемых данных разбивается на *пачки (batch)* равного размера. Решение задачи классификации для пачки данных — это прямой проход по обученной нейронной сети и вычисление финального вектора достоверностей. Количество пачек, на которых запускается прямой проход, определяет число таких проходов — *итераций*. Итерации выполняются последовательно, следующая итерация запускается после завершения предыдущей. Для каждого прямого прохода измеряется продолжительность его выполнения t_i, $i = \overline{1,L}$, где L — количество итераций. Для оценки производительности вывода используется показатель *числа кадров, обрабатываемых в секунду (Frames per Second, FPS)*, который вычисляется как отношение размера входного набора данных (общего числа изображений) S к суммарному времени выполнения всех итераций:

$$FPS = \frac{S}{\sum_{i=1}^{L} t_i}. \tag{2}$$

Отметим, что время вывода с использованием разных фреймворков для одной и той же модели может отличаться в зависимости от размера входной пачки данных. Чтобы прогнозировать время завершения экспериментов, выполняются пробные запуски и для каждого размера пачки фиксируется разное количество итераций (табл. 1). Выбор максимального размера пачки обусловлен размерами оперативной памяти устройства.

Таблица 1. Выполняемое количество итераций для каждого размера входной пачки данных

Размер пачки	1	2	4	8	16	32	64	128
Количество итераций	100	100	85	70	55	40	25	20

6.3. Наборы данных

Для проведения экспериментов используется подмножество изображений валидационной выборки набора данных ImageNet [25]. Поскольку доступные на сегодняшний день образцы процессоров архитектуры RISC-V пока существенно отстают от высокопроизводительных устройств, то вывод глубоких моделей работает относительно медленно. В связи с этим для проверки точности нейронных сетей используются первые 1 000 изображений выборки. В случае корректности работы модели такой подход позволяет получить значение точности классификации, близкое к опубликованному авторами обученной сети. Для анали-

за производительности вывода выбраны 32 произвольных изображения из той же выборки. Следует отметить, что для определения показателей производительности также можно использовать синтетические входные данные, поскольку количество операций, выполняемых на прямом проходе, не зависит от контекста изображений.

6.4. Тестовые модели

Анализ производительности вывода выполняется для широко известных классификационных моделей — ResNet-50 [3] и MobileNetV2 [4]. Эти модели являются сверточными нейронными сетями с остаточными связями. Принципиальное отличие MobileNetV2 от ResNet-50 состоит в использовании сверток, отделимых по глубине. Классическая свертка предполагает проход трехмерным ядром слева направо и сверху вниз по входной трехмерной матрице (тензору) и вычисление суммы произведений соответствующих компонент. Свертка, отделимая по глубине, предусматривает последовательное применение точечной и пространственной свертки. *Точечная свертка* включает вычисление набора одномерных сверток (скалярных произведений) вдоль размерности, соответствующей каналам входной трехмерной матрицы. *Пространственная свертка* предполагает применение двумерных ядер к каждому каналу входного тензора.

В процессе анализа используются обученные модели ResNet-50 и MobileNetV2 из репозитория torchvision [26]. Модели предварительно загружаются, сериализуются и сохраняются с помощью встроенной функции ‘torch.jit.save’ библиотеки PyTorch. Затем они конвертируются и/или компилируются в форматы TensorFlow Lite, Apache TVM и ExecuTorch (рис. 2).

1. *Конвертация в формат TensorFlow Lite.* Выполняется с помощью фреймворка TensorFlow Backend for ONNX [27] в формат ONNX, далее в формат TensorFlow Lite.
2. *Конвертация в формат TVM.* Обеспечивается посредством вызова встроенной функции ‘torch.onnx.export’, которая преобразует модель в формат ONNX, после чего полученное представление передается на вход конвертеру моделей в формат TVM, реализованному в рамках системы DLI.
3. *Конвертация в формат ExecuTorch.* Реализуется в два этапа. Вначале выполняется оптимизация PyTorch-модели с использованием заданного бэкенда с помощью функции ‘to_edge_transform_and_lower’ библиотеки ExecuTorch. Далее оптимизированная модель сериализуется в формат ExecuTorch. В работе в качестве бэкенда применяется библиотека XNNPACK [28], поскольку она оптимизирована для устройств RISC-V.

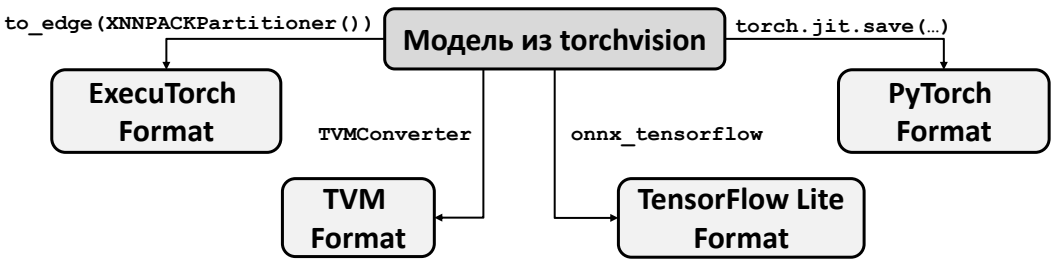

Рис. 2. Последовательность конвертации тестовых моделей в форматы целевых фреймворков

6.5. Тестовая инфраструктура

Ниже (табл. 2) приведены параметры тестовой инфраструктуры, использованной для проведения экспериментов. Фреймворки глубокого обучения собираются из исходных кодов с помощью компиляторов gcc и g++ версии 13.2.0-23ubuntu4bb3.

Таблица 2. Тестовая инфраструктура

CPU, RAM	Spacemit (R) X60, 1.6GHz, 8 ядер, 8 потоков, 16GB
Операционная система	Bianbu 2.1
Фреймворки	PyTorch v2.6.0 (OpenBLAS 0.3.26+ds-1) TensorFlow Lite v2.14.0 (XNNPACK бэкенд) Apache TVM v0.14.0 ExecuTorch v0.5.0 (XNNPACK бэкенд)

6.6. Параметры экспериментов

В табл. 3 приведены перебираемые параметры экспериментов. Вывод запускается с использованием программных интерфейсов для языков C++ и Python. При подборе оптимальных параметров для каждого размера входной пачки данных обеспечивается поиск оптимального числа потоков, которое устанавливается для обеспечения параллелизма. Отметим, что в Python API для TVM и ExecuTorch используется значение количества потоков по умолчанию, равное числу физических ядер. Также TVM позволяет установить уровень оптимизации модели opt_level в процессе ее предварительной компиляции для вывода.

Таблица 3. Параметры запуска вывода с использованием разных фреймворков

Фреймворк	**Программный интерфейс**	**Параметры**		
		Размер пачки	**Количество потоков**	**Внутренние параметры**
PyTorch	C++	+	+	—
	Python	+	+	—
TensorFlow Lite	C++	+	+	—
	Python	+	+	—
Apache TVM	Python	+	по умолчанию	уровень оптимизации opt_level
ExecuTorch	C++	+	+	—
	Python	+	по умолчанию	—

6.7. Результаты экспериментов

Качество классификации изображений. Качество решения поставленной задачи с использованием доступных фреймворков верифицируется средствами компонента **Accuracy Checker** системы DLI, который является оберткой над соответствующим инструментом в составе OpenVINO [29]. Возможности указанного инструмента расширены авторами статьи, поскольку исходная версия не поддерживает вывод глубоких моделей с помощью TVM и ExecuTorch. Реализация выложена в открытый доступ [30]. Результирующие показатели качества представлены ниже (рис. 3). Из гистограмм можно сделать вывод, что для обеих моделей полученные значения близки к опубликованным. Отличие для модели ResNet-50 по метрике top-1 составляет 0.758, по top-5 — 1.166; для модели MobileNetV2 по top-1 — 0.154, по top-5 — 0.778. Отклонения от заявленных значений точности могут

варьироваться при выборе разных подмножеств валидационной выборки ImageNet. Если распределение данных в процентном соотношении для подмножества совпадет с распределением всей валидационной выборки, то значения метрики top-1 должны быть практически равными. Также отличие может быть обусловлено тем, что некоторые фреймворки в процессе конвертации модели в собственное внутреннее представление изменяют порядок выполнения операций, в связи с чем на данных, где модель дает близкие выходные значения достоверностей, может изменяться порядок классов при вычислении top-1 или top-5.

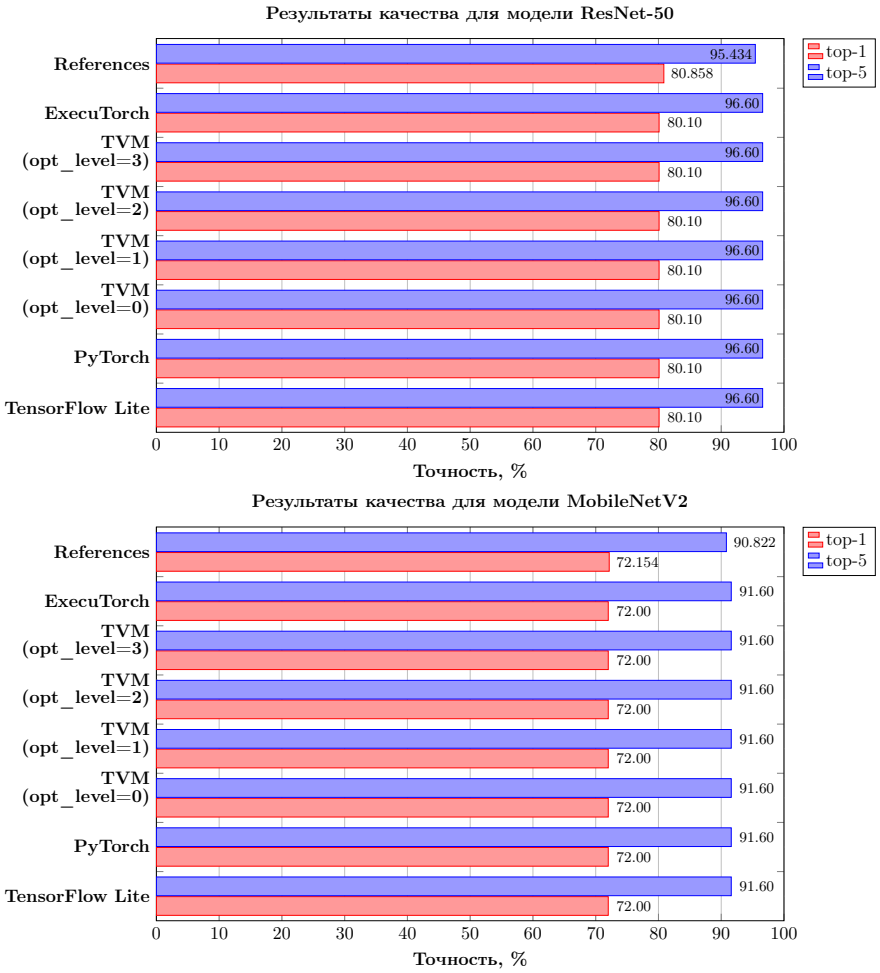

Рис. 3. Качество классификации на множестве из первых 1 000 изображений валидационной выборки ImageNet

Подбор оптимальных параметров. На данном этапе для каждого допустимого размера входной пачки данных необходимо подобрать оптимальные параметры запуска. Фреймворк *PyTorch* в программных интерфейсах C++ и Python позволяет перебирать количество потоков. Из полученных результатов (рис. 4) можно видеть, что для обеих тестовых моделей поведение показателя производительности нестабильно. Для модели ResNet-50 лучшие значения показателя FPS достигаются на размере пачки, равном 1, как для программного интерфейса языка C++, так и для Python. Отличие во втором знаке после запятой ($\sim$0.044). Такой сценарий получения входных данных характерен для многих приложений, обеспечивающих обработку видео с низкой частотой, либо не требующих запуска вывода глубокой модели на каждом кадре видеопотока. Для MobileNetV2 ситуация кардинально противоположная. В целом с увеличением размера пачки повышается про-

изводительность, при этом максимальная — наблюдается на наибольшем размере входной пачки изображений. Согласно открытым источникам (официальный форум [31]) PyTorch собирается для запуска на устройствах RISC-V, но фреймворк не оптимизирован под данную архитектуру. К сожалению, в доступных процессорах архитектуры RISC-V не хватает привычных для x86-64 датчиков производительности, и средства профилировки обладают скромными возможностями. Поэтому сложно указать точные причины подобных артефактов. Следует отметить, что анализ результатов производительности вывода на устройствах x86-64 для тех же тестовых моделей показывает хорошую масштабируемость.

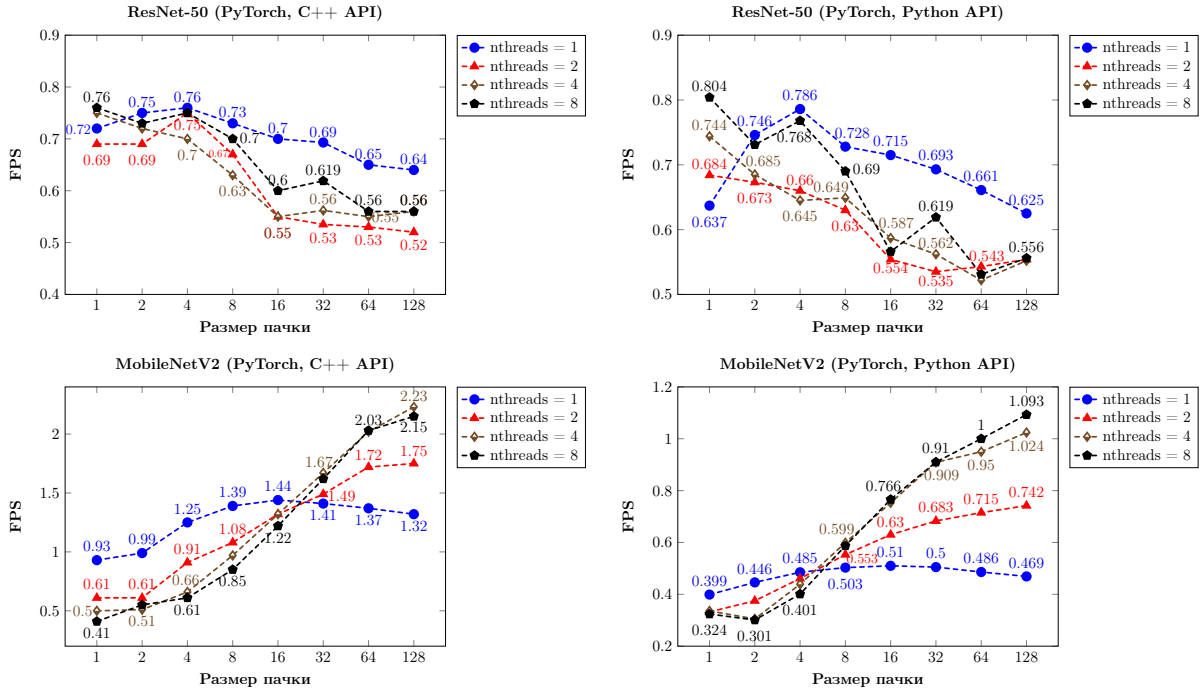

Рис. 4. Зависимости числа кадров, обрабатываемых за секунду, от размера входной пачки данных при разном количестве потоков для фреймворка PyTorch. Каждый график соответствует определенной тестовой модели, запущенной с использованием C++ или Python API

TensorFlow Lite по аналогии с PyTorch позволяет перебирать количество потоков для параллельного исполнения вывода. Результаты экспериментов (рис. 5) показывают хорошую масштабируемость с ростом числа потоков для модели ResNet-50. При этом увеличение размера пачки данных не приводит к падению производительности. Поэтому выбор оптимального размера пачки в основном зависит от скорости получения входных данных в приложении.

Для MobileNetV2 ситуация несколько хуже, в особенности при использовании максимального числа потоков на малых размерах пачки данных (менее четырех изображений). На пачках, размер которых превышает 4, проблемы с масштабируемостью постепенно уходят, но начиная с 32 изображений происходит спад. При этом на 16 изображениях увеличение числа потоков вдвое приводит к ускорению, близкому к 1.9. Предположительно такой результат обусловлен особенностями параллельной реализации пространственных сверток, которые составляют основное преобразование в сети MobileNetV2 и требуют нерегулярного обхода тензоров. Таким образом, при запуске указанной модели имеет смысл подавать на

вход пачку данных размера 16 и устанавливать число потоков, равное 8. Если же приложение не позволяет сформировать пачку, размер которой превышает 2 изображения, то при параллельном исполнении следует задействовать только 4 потока.

Отдельно следует отметить, что Python API в целом демонстрирует скорость обработки изображений близкую или немного ниже по сравнению с C++ API для обеих тестовых моделей вследствие наличия накладных расходов на вызов функций библиотеки C++.

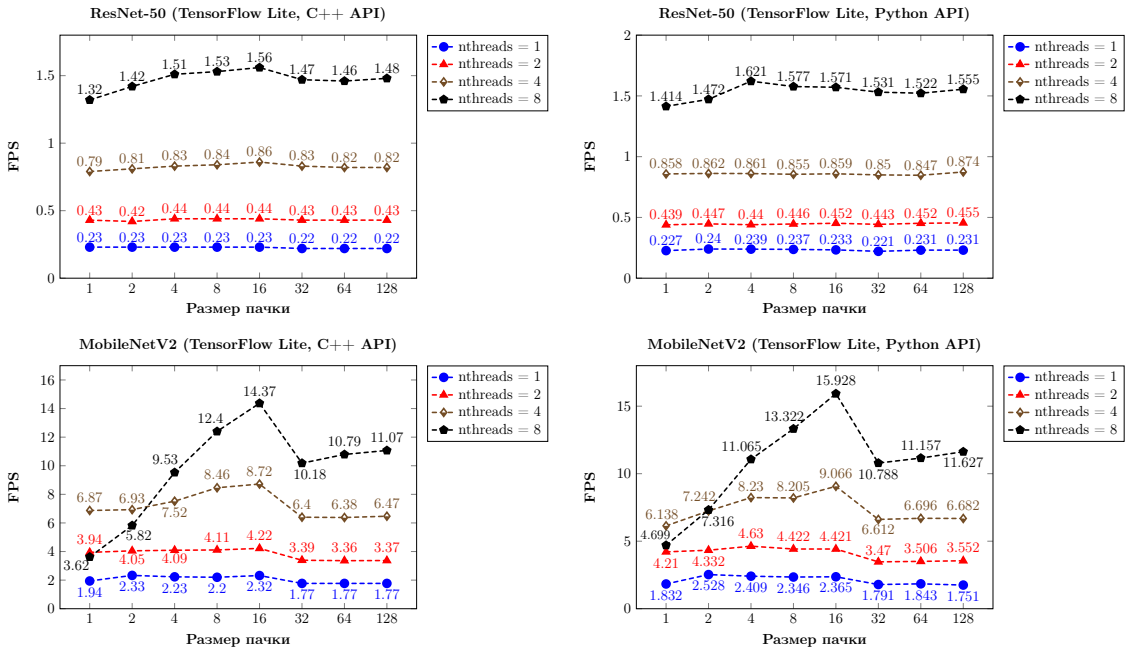

Рис. 5. Зависимости числа кадров, обрабатываемых за секунду, от размера входной пачки данных при разном количестве потоков для фреймворка TensorFlow Lite. Каждый график соответствует определенной тестовой модели, запущенной с использованием C++ или Python API

TVM позволяет установить уровень оптимизации модели. Ниже приведены (рис. 6) графики зависимости показателя числа обрабатываемых за секунду кадров от размера входной пачки данных. Теоретически чем выше уровень оптимизации (opt_level), тем быстрее должен работать вывод. На практике это утверждение не всегда справедливо. Для модели ResNet-50, например, на пачках 2, 4 и 8 показатели производительности отличаются незначительно (второй знак после запятой), а для MobileNetV2 на всех размерах пачки, за исключением 1 и 128, при opt_level=3 наблюдаются лучшие значения FPS. Отсутствие эффекта от оптимизации модели ResNet-50 объясняется тем, что основная операция в сети — это классическая свертка, которая изначально хорошо оптимизирована во фреймворке. Небольшое улучшение производительности вывода MobileNetV2 при изменении уровня оптимизации, вероятнее всего, связано с применением более эффективных стратегий обхода входных тензоров при реализации сверток, отделимых по глубине. При этом следует отметить «провал» производительности на пачке в 32 изображения и opt_level=1, возникший предположительно из-за простоев при работе с памятью. Таким образом, для первой тестовой модели при каждом размере пачки нельзя однозначно рекомендовать оптимальное значение параметра opt_level, в то время как для второй — на больших размерах пачки максимальный уровень оптимизации гарантирует лучшую производительность вывода.

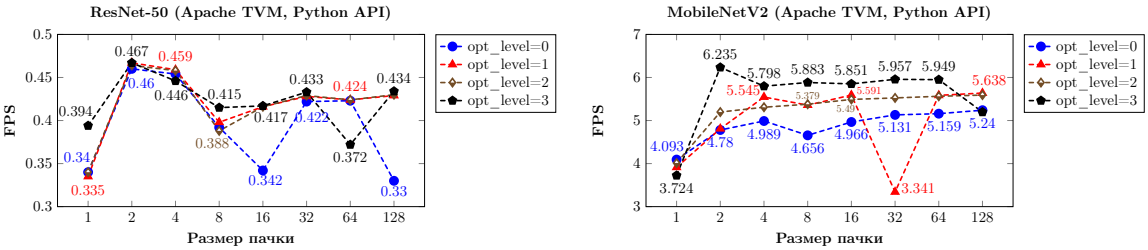

Рис. 6. Зависимости числа кадров, обрабатываемых за секунду, от размера входной пачки данных при разном уровне оптимизации модели (opt_level) для фреймворка Apache TVM

ExecuTorch позволяет перебирать количество потоков для параллельного запуска вывода только в C++ API, в Python API данный параметр устанавливается в значение по умолчанию, равное количеству физических ядер. Ниже приведены результаты определения оптимального числа потоков для C++ API (рис. 7) и сравнение лучших результатов C++ API с Python API для каждого допустимого размера пачки данных (рис. 8).

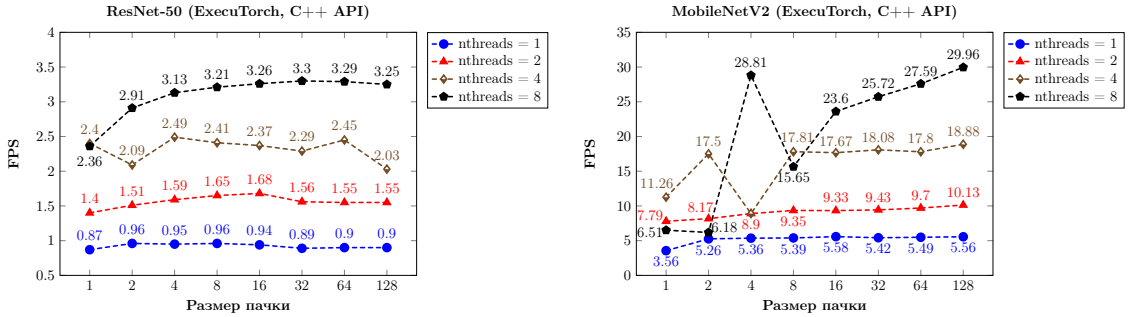

Рис. 7. Зависимости числа кадров, обрабатываемых за секунду, от размера входной пачки данных для фреймворка ExecuTorch (C++ API)

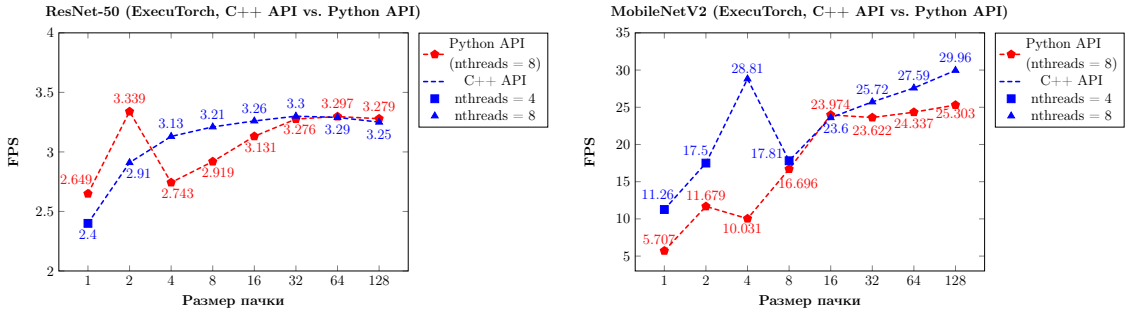

Рис. 8. Сравнение результатов производительности вывода для Python и C++ APIs фреймворка ExecuTorch

Для модели ResNet-50 фреймворк демонстрирует рост производительности при увеличении числа потоков независимо от размера входной пачки данных (рис. 7, слева). Так для пачки в 32 изображения, на которой достигается максимальная производительность, с увеличением числа потоков вдвое ускорение в среднем составляет ∼1.55 раза. При этом для каждого фиксированного числа потоков с увеличением размера пачки данных FPS либо не изменяется, либо растет (во втором знаке после запятой). Для сети MobileNetV2

при запуске в 2 потока также наблюдается увеличение FPS в среднем ∼1.77 раза, а на 4 и 8 потоках (рис. 7, справа) — нестабильное поведение показателя производительности. При запуске в 4 потока очевиден «провал» на пачке в 4 изображения, а при запуске в 8 потоков на размере пачки, равном 8, возникает пик в 28.81 fps, близкий к максимальному значению производительности (29.96 fps на пачке 128). Как отмечалось ранее, в настоящее время на RISC-V отсутствуют технические возможности, чтобы достоверно объяснить причины недостаточного ускорения от параллелизма и других артефактов. Предположительно это связано с тем, что плата Banana Pi BPI-F3 оснащена не самой быстрой подсистемой памяти, также в процессоре отсутствует внеочередное исполнение инструкций, в результате чего меньше возможностей для уменьшения простоев из-за работы с памятью. Пик на 8 потоках вероятнее всего достигается вследствие удачного расположения данных в памяти. Подробнее подобные вопросы обсуждаются в [32].

Если сравнивать лучшие результаты производительности, полученные при использовании C++ API, с показателями Python API (рис. 8), то можно сделать следующие выводы. Для модели ResNet-50 показатели в целом отличаются незначительно, Python API на большинстве размеров входных пачек уступает из-за наличия накладных расходов на вызов функций библиотеки C++. Аналогичное утверждение справедливо и для модели MobileNetV2, но эта разница выглядит более существенной для размеров пачки 1, 2 и 4. В связи с этим в приложениях предпочтительно использовать для вывода C++ API. Обе тестовые модели имеет смысл запускать на максимально возможном размере пачки данных 128 в 8 потоков.

Сравнительный анализ. Сравнение показателей производительности, полученных при оптимальных параметрах запуска вывода (рис. 9), показывает, что ExecuTorch обеспечивает лучшую скорость обработки изображений независимо от размера входной пачки данных. Для модели ResNet-50 скорость обработки меняется от 2.649 до 3.339 fps при оптимальных параметрах запуска в зависимости от размера входной пачки данных, для MobileNetV2 — от 11.26 до 29.96 fps. Вывод средствами TensorFlow Lite уступает ExecuTorch в среднем в ∼2.1 раза. Допускаем, что использование более новой версии библиотеки TensorFlow Lite позволит улучшить показатели производительности. Остальные фреймворки показывают существенно худшую производительность. Предположительно данный факт связан с тем, что хотя для PyTorch и Apache TVM официально заявлена поддержка RISC-V, но они не в полной мере оптимизированы под указанную архитектуру.

Заключение

Активное применение глубокого обучения для решения прикладных задач неизбежно поднимает вопросы, связанные с анализом производительности вывода моделей на конечных устройствах, которые используются для многократного прямого прохода. Темпы развития архитектуры RISC-V и обзор различных источников свидетельствуют об интересе сообщества к подобным вопросам, что говорит об актуальности результатов исследования.

В работе для классификационных моделей ResNet-50 и MobileNetV2 демонстрируется, что среди известных фреймворков глубокого обучения, обеспечивающих запуск вывода на устройствах RISC-V, на текущий момент на плате Banana Pi BPI-F3 лучшую производительность показывает фреймворк ExecuTorch (с XNNPACK-бэкендом). Для сети ResNet-50, где преобладают классические свертки с трехмерным ядром и одномерные свертки вдоль размерности, соответствущей каналам, наблюдается хорошая масштабируемость независи-

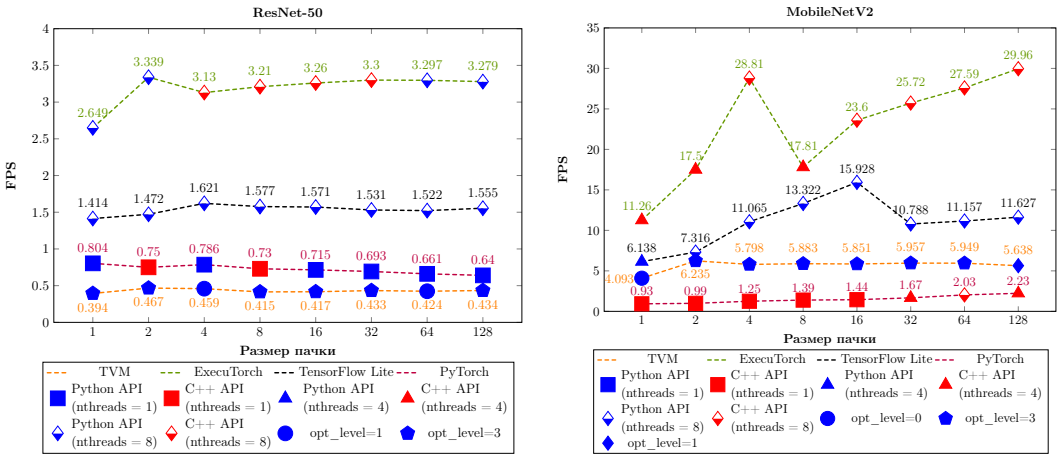

Рис. 9. Лучшие результаты производительности, полученные при использовании оптимальных параметров запуска вывода. Цвет и тип линий соответствует фреймворку, маркеры — программному интерфейсу и оптимальным параметрам запуска

мо от размера входной пачки данных. Лучшие показатели производительности для обеих тестовых моделей достигаются при запуске в 8 потоков. При этом MobileNetV2 на пачках 4, 32, 64 и 128 изображений работает в режиме реального времени (более 25 fps). На практике выбор оптимального размера входной пачки данных зависит не только от результатов анализа производительности, но от скорости поступления входных данных в конкретном приложении (изображений в системах видеоанализа).

Несмотря на то, что Banana Pi BPI-F3 относится к числу маломощных, вывод нейронных сетей на таких устройствах имеет смысл. Эксперименты показывают, что даже на классическом примере известной «легковесной» сети MobileNetV2 удается обеспечить обработку данных в режиме реального времени. Данный факт говорит в поддержку тезиса о перспективности внедрения процессоров RISC-V в мобильные устройства. Полученные результаты исследования позволяют сделать вывод, что при развитии архитектуры RISC-V полезно обратить внимание на возможность аппаратного ускорения операции свертки, поскольку она является наиболее вычислительно-трудоемкой для большого класса сетей.

Разработанная инфраструктура в рамках программной системы Deep Learning Inference Benchmark позволяет автоматизировать запуск вывода и агрегацию результатов бенчмаркинга вывода глубоких моделей. Система является открытой и расширяемой, поэтому может быть использована для анализа производительности вывода в аналогичных исследованиях на новых образцах процессоров архитектуры RISC-V.

Литература

1. Noor M.H.M., Ige A.O. A survey on state-of-the-art deep learning applications and challenges // Engineering Applications of Artificial Intelligence. 2025. Vol. 159, Part B. P. 111225. DOI: 10.1016/j.engappai.2025.111225.

2. Mezger B.W., *et al.* A Survey of the RISC-V Architecture Software Support // IEEE Access. 2022. Vol. 10. P. 51394–51411. DOI: 10.1109/ACCESS.2022.3174125.

3. He K., Zhang X., Ren S., Sun J. Deep Residual Learning for Image Recognition // 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016. P. 770–778. DOI: 10.1109/CVPR.2016.90.

4. Sandler M., *et al.* MobileNetV2: Inverted Residuals and Linear Bottlenecks // 2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, June 18–22, 2018. P. 4510–4520. DOI: 10.1109/CVPR.2018.00474.

5. Алибеков М.Р. и др. Методика анализа производительности вывода глубоких нейронных сетей на примере задачи классификации изображений // Вычислительные методы и программирование. 2024. Т. 25, № 2. С. 127–141. DOI: 10.26089/NumMet.v25r211.

6. Demidovskij A., *et al.* OpenVINO Deep Learning Workbench: Comprehensive Analysis and Tun-ing of Neural Networks Inference // ICCV Workshop, 2019. URL: `https://openaccess.thecvf.com/content_ICCVW_2019/papers/SDL-CV/Gorbachev_OpenVINO_Deep_Learning_Workbench_Comprehensive_Analysis_and_Tuning_of_Neural_ICCVW_2019_paper.pdf` (дата обращения: 25.07.2025).

7. Arya M., Simmhan Y. A Preliminary Performance Analysis of LLM Inference on Edge Accelerators // 2024 IEEE 31st International Conference on High Performance Computing, Data and Analytics Workshop (HiPCW), Bangalore, India, 2024. P. 183–184. DOI: 10.1109/HiPCW63042.2024.00069.

8. Verma G., *et al.* Performance Evaluation of Deep Learning Compilers for Edge Inference // 2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Portland, OR, USA, 2021. P. 858–865. DOI: 10.1109/IPDPSW52791.2021.00128.

9. Chen Y.-R., *et al.* Experiments and optimizations for TVM on RISC-V Architectures with P Extension // 2020 International Symposium on VLSI Design, Automation and Test (VLSI-DAT), Hsinchu, Taiwan, 2020. P. 1–4. DOI: 10.1109/VLSI-DAT49148.2020.9196477.

10. Christofas V., *et al.* Comparative Evaluation between Accelerated RISC-V and ARM AI Inference Machines // 2023 6th World Symposium on Communication Engineering (WSCE), Thessaloniki, Greece, 2023. P. 108–113. DOI: 10.1109/WSCE59557.2023.10365853.

11. Bhattacharjee D., *et al.* Full-Stack Evaluation of Machine Learning Inference Workloads for RISC-V Systems // RISC-V Summit 2024. URL: `https://riscv-europe.org/summit/2024/media/proceedings/posters/58_poster.pdf` (дата обращения: 12.10.2025).

12. Garcia A.M., *et al.* Assessing Large Language Models Inference Performance on a 64-core RISC-V CPU with Silicon-Enabled Vectors // BigHPC2024: Special Track on Big Data and High-Performance Computing, co-located with the 3rd Italian Conference on Big Data and Data Science, ITADATA2024, Pisa, Italy, September 17–19, 2024. URL: `https://ceur-ws.org/Vol-3785/paper110.pdf` (дата обращения: 19.07.2025).

13. Martinez H., *et al.* Performance Analysis of BERT on RISC-V Processors with SIMD Units // High Performance Computing. ISC High Performance 2024 International Workshops. Vol. 15058 / eds. by M. Weiland, S. Neuwirth, C. Kruse, T. Weinzierl. Springer, 2024. P. 325–338. Lecture Notes in Computer Science. DOI: 10.1007/978-3-031-73716-9_23.

14. Suárez D., *et al.* Energy-Efficient Inference on RNN and LLM networks: A Quantized Evaluation on RISC-V, ARM, and x86 Devices // 16th ACM International Conference on Future and Sustainable Energy Systems (E-Energy '25). Association for Computing Machinery, New York, NY, USA, 2025. P. 882–889. DOI: 10.1145/3679240.3735101.

15. Mukhin I., *et al.* Benchmarking Deep Learning Inference on RISC-V CPUs // Supercomputing. RuSCDays 2024. Vol. 15406 / eds. by V. Voevodin, A. Antonov, D. Nikitenko. Springer, 2025. P. 331–346. Lecture Notes in Computer Science. DOI: 10.1007/978-3-031-78459-0_24.

16. Официальная страница фреймворка PyTorch. URL: `https://pytorch.org` (дата обращения: 19.07.2025).

17. Официальная страница фреймворка TensorFlow Lite. URL: `https://www.tensorflow.org/lite` (дата обращения: 19.07.2025).

18. Chen T., *et al.* TVM: An Automated End-to-End Optimizing Compiler for Deep Learning // 13th USENIX Conference on Operating Systems Design and Implementation, Carlsbad, CA, USA, 2018. P. 579–594. DOI: 10.5555/3291168.3291211.

19. Официальная страница фреймворка ExecuTorch. URL: `https://pytorch.org/executorch-overview` (дата обращения: 19.07.2025).

20. DLI: Deep Learning Inference Benchmark. Репозиторий проекта. URL: `https://github.com/itlab-vision/dl-benchmark` (дата обращения: 19.07.2025).

21. Kustikova V., *et al.* DLI: Deep Learning Inference Benchmark // Supercomputing. RuSCDays 2019. Vol. 1129 / eds. by V. Voevodin, S. Sobolev. Springer, 2019. P. 542–553. Communications in Computer and Information Science. DOI: 10.1007/978-3-030-36592-9_44.

22. Lin D., *et al.* Fixed Point Quantization of Deep Convolutional Networks // 33rd International Conference on Machine Learning, New York, USA, 2016. P. 2849–2858. URL: `https://proceedings.mlr.press/v48/linb16.pdf` (дата обращения: 19.07.2025).

23. Kozlov A., *et al.* Neural network compression framework for fast model inference // Intelligent Computing 2021. Vol. 285 / ed. by K. Arai. Springer, 2021. P. 213–232. Lecture Notes in Networks and Systems. DOI: 10.1007/978-3-030-80129-8_17.

24. Официальная страница фреймворка ncnn. URL: `https://github.com/Tencent/ncnn` (дата обращения: 19.07.2025).

25. Deng J., *et al.* ImageNet: A Large-Scale Hierarchical Image Database // IEEE Computer Vision and Pattern Recognition (CVPR), 2009. URL: `https://www.image-net.org/static_files/papers/imagenet_cvpr09.pdf` (дата обращения: 19.07.2025).

26. TorchVision: PyTorch's Computer Vision library. 2016. URL: `https://github.com/pytorch/vision` (дата обращения: 19.07.2025).

27. TensorFlow Backend for ONNX. URL: `https://github.com/onnx/onnx-tensorflow` (дата обращения: 19.07.2025).

28. XNNPACK. High-efficiency floating-point neural network inference operators for mobile, server, and Web. URL: `http://github.com/google/XNNPACK` (дата обращения: 19.07.2025).

29. OpenVINO. OpenVINO is an open source toolkit for optimizing and deploying AI inference. URL: `https://github.com/openvinotoolkit/openvino` (дата обращения: 19.07.2025).

30. OpenVINO Toolkit — Open Model Zoo repository. Fork (branch '24.3.0/tvm' for Apache TVM, branch 'omz_executorch' for ExecuTorch). URL: `https://github.com/itlab-vision/open_model_zoo_tvm` (дата обращения: 19.07.2025).

31. PyTorch RISC-V support. URL: `https://discuss.pytorch.org/t/pytorch-risc-v-support/212065` (дата обращения: 19.07.2025).

32. Pirova A., *et al.* Performance optimization of BLAS algorithms with band matrices for RISC-V processors // Future Generation Computer Systems. 2026. Vol. 174. P. 107936. DOI: 10.1016/j.future.2025.107936.

Мухин Иван Сергеевич, студент, кафедра высокопроизводительных вычислений и системного программирования, Институт информационных технологий, математики и механики, Нижегородский государственный университет им. Н.И. Лобачевского (Нижний Новгород, Российская Федерация)

Кустикова Валентина Дмитриевна, к.т.н., доцент, кафедра высокопроизводительных вычислений и системного программирования, Институт информационных технологий, математики и механики, Нижегородский государственный университет им. Н.И. Лобачевского (Нижний Новгород, Российская Федерация)

DOI: 10.14529/cmse250403

PERFORMANCE ANALYSIS OF DEEP LEARNING INFERENCE ON THE BANANA PI BPI-F3 BOARD USING THE IMAGE CLASSIFICATION PROBLEM AS AN EXAMPLE

© 2025 I.S. Mukhin, V.D. Kustikova

Lobachevsky State University of Nizhny Novgorod

(pr. Gagarina 23, Nizhny Novgorod, 603022 Russia)

E-mail: ismukhin03@gmail.com, valentina.kustikova@itmm.unn.ru

Received: 18.08.2025

The paper analyzes the inference performance of the well-known neural networks ResNet-50 and MobileNetV2, which provide a solution for the problem of image classification, on the Banana Pi BPI-F3 board, which is built on the RISC-V architecture. The inference is launched by available frameworks: PyTorch, TensorFlow Lite, Apache TVM and ExecuTorch. The models are converted to the format of each target framework. The correctness of the problem solving is checked using the obtained neural networks. It is demonstrated that the accuracy indicators of image classification using these models correlate well with the published ones. Then, the optimal parameters for launching the inference for each framework and model are selected. A comparative analysis of the inference performance shows that ExecuTorch demonstrates the best results for both models. For the ResNet-50 model, the number of frames processed per second (FPS) varies from 2.649 to 3.339 fps with optimal parameters depending on the batch size of images processed in one forward pass through the network, for MobileNetV2 – from 11.26 to 29.96 fps. TensorFlow Lite is inferior to ExecuTorch by an average of ∼2.1 times. PyTorch and Apache TVM demonstrate lower performance indicators. Probably, this is due to the fact that they are not fully optimized for the RISC-V architecture.

Keywords: deep learning, image classification, inference performance, PyTorch, TensorFlow Lite, Apache TVM, ExecuTorch, Banana Pi BPI-F3, RISC-V.

FOR CITATION

Mukhin I.S., Kustikova V.D. Performance Analysis of Deep Learning Inference on the Banana Pi BPI-F3 Board Using the Image Classification Problem as an Example. Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering. 2025. Vol. 14, no. 4. P. 40–60. (in Russian) DOI: 10.14529/cmse250403.

References

1. Noor M.H.M., Ige A.O. A survey on state-of-the-art deep learning applications and challenges. Engineering Applications of Artificial Intelligence. 2025. Vol. 159, Part B. P. 111225. DOI: 10.1016/j.engappai.2025.111225.

2. Mezger B.W., *et al.* A Survey of the RISC-V Architecture Software Support. IEEE Access. 2022. Vol. 10. P. 51394–51411. DOI: 10.1109/ACCESS.2022.3174125.

3. He K., Zhang X., Ren S., Sun J. Deep Residual Learning for Image Recognition. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016. P. 770–778. DOI: 10.1109/CVPR.2016.90.

4. Sandler M., *et al.* MobileNetV2: Inverted Residuals and Linear Bottlenecks. 2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, June 18–22, 2018. P. 4510–4520. DOI: 10.1109/CVPR.2018.00474.

5. Alibekov M.R., *et al.* Performance analysis methodology of deep neural networks inference on the example of an image classification problem. Numerical Methods and Programming. 2024. Vol. 25, no. 2. P. 127–141. (in Russian) DOI: 10.26089/NumMet.v25r211.

6. Demidovskij A., *et al.* OpenVINO Deep Learning Workbench: Comprehensive Analysis and Tun-ing of Neural Networks Inference. ICCV Workshop, 2019. URL: `https://openaccess.thecvf.com/content_ICCVW_2019/papers/SDL-CV/Gorbachev_OpenVINO_Deep_Learning_Workbench_Comprehensive_Analysis_and_Tuning_of_Neural_ICCVW_2019_paper.pdf` (accessed: 25.07.2025).

7. Arya M., Simmhan Y. A Preliminary Performance Analysis of LLM Inference on Edge Accelerators. 2024 IEEE 31st International Conference on High Performance Computing, Data and Analytics Workshop (HiPCW), Bangalore, India, 2024. P. 183–184. DOI: 10.1109/HiPCW63042.2024.00069.

8. Verma G., *et al.* Performance Evaluation of Deep Learning Compilers for Edge Inference. 2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Portland, OR, USA, 2021. P. 858–865. DOI: 10.1109/IPDPSW52791.2021.00128.

9. Chen Y.-R., *et al.* Experiments and optimizations for TVM on RISC-V Architectures with P Extension. 2020 International Symposium on VLSI Design, Automation and Test (VLSI-DAT), Hsinchu, Taiwan, 2020. P. 1–4. DOI: 10.1109/VLSI-DAT49148.2020.9196477.

10. Christofas V., *et al.* Comparative Evaluation between Accelerated RISC-V and ARM AI Inference Machines. 2023 6th World Symposium on Communication Engineering (WSCE), Thessaloniki, Greece, 2023. P. 108–113. DOI: 10.1109/WSCE59557.2023.10365853.

11. Bhattacharjee D., *et al.* Full-Stack Evaluation of Machine Learning Inference Workloads for RISC-V Systems. RISC-V Summit 2024. URL: `https://riscv-europe.org/summit/2024/media/proceedings/posters/58_poster.pdf` (accessed: 12.10.2025).

12. Garcia A.M., *et al.* Assessing Large Language Models Inference Performance on a 64-core RISC-V CPU with Silicon-Enabled Vectors. BigHPC2024: Special Track on Big Data and High-Performance Computing, co-located with the 3rd Italian Conference on Big Data and Data Science, ITADATA2024, Pisa, Italy, September 17–19, 2024. URL: `https://ceur-ws.org/Vol-3785/paper110.pdf` (accessed: 19.07.2025).

13. Martinez H., *et al.* Performance Analysis of BERT on RISC-V Processors with SIMD Units. High Performance Computing. ISC High Performance 2024 International Workshops. Vol. 15058 / eds. by M. Weiland, S. Neuwirth, C. Kruse, T. Weinzierl. Springer, 2024. P. 325–338. Lecture Notes in Computer Science. DOI: 10.1007/978-3-031-73716-9_23.

14. Suarez D., *et al.* Energy-Efficient Inference on RNN and LLM networks: A Quantized Evaluation on RISC-V, ARM, and x86 Devices. 16th ACM International Conference on Future and Sustainable Energy Systems (E-Energy '25). Association for Computing Machinery, New York, NY, USA, 2025. P. 882–889. DOI: 10.1145/3679240.3735101.

15. Mukhin I., *et al.* Benchmarking Deep Learning Inference on RISC-V CPUs. Supercomputing. RuSCDays 2024. Vol. 15406 / eds. by V. Voevodin, A. Antonov, D. Nikitenko. Springer, 2025. P. 331–346. Lecture Notes in Computer Science. DOI: 10.1007/978-3-031-78459-0_24.

16. The official web site of the framework PyTorch. URL: `https://pytorch.org` (accessed: 19.07.2025).

17. The official web site of the framework TensorFlow Lite. URL: `https://www.tensorflow.org/lite` (accessed: 19.07.2025).

18. Chen T., *et al.* TVM: An Automated End-to-End Optimizing Compiler for Deep Learning. 13th USENIX Conference on Operating Systems Design and Implementation, Carlsbad, CA, USA, 2018. P. 579–594. DOI: 10.5555/3291168.3291211.

19. The official web site of the framework ExecuTorch. URL: `https://pytorch.org/executorch-overview` (accessed: 19.07.2025).

20. DLI: Deep Learning Inference Benchmark. GitHub Repo. URL: `https://github.com/itlab-vision/dl-benchmark` (accessed: 19.07.2025).

21. Kustikova V., *et al.* DLI: Deep Learning Inference Benchmark. Supercomputing. RuSCDays 2019. Vol. 1129 / eds. by V. Voevodin, S. Sobolev. Springer, 2019. P. 542–553. Communications in Computer and Information Science. DOI: 10.1007/978-3-030-36592-9_44.

22. Lin D., *et al.* Fixed Point Quantization of Deep Convolutional Networks. 33rd International Conference on Machine Learning, New York, USA, 2016. P. 2849–2858. URL: `https://proceedings.mlr.press/v48/linb16.pdf` (accessed: 19.07.2025).

23. Kozlov A., *et al.* Neural network compression framework for fast model inference. Intelligent Computing 2021. Vol. 285 / ed. by K. Arai. Springer, 2021. P. 213–232. Lecture Notes in Networks and Systems. DOI: 10.1007/978-3-030-80129-8_17.

24. The official web site of the framework ncnn. URL: `https://github.com/Tencent/ncnn` (accessed: 19.07.2025).

25. Deng J., *et al.* ImageNet: A Large-Scale Hierarchical Image Database. IEEE Computer Vision and Pattern Recognition (CVPR), 2009. URL: `https://www.image-net.org/static_files/papers/imagenet_cvpr09.pdf` (accessed: 19.07.2025).

26. TorchVision: PyTorch's Computer Vision library. 2016. URL: `https://github.com/pytorch/vision` (accessed: 19.07.2025).

27. TensorFlow Backend for ONNX. URL: `https://github.com/onnx/onnx-tensorflow` (accessed: 19.07.2025).

28. XNNPACK. High-efficiency floating-point neural network inference operators for mobile, server, and Web. URL: `http://github.com/google/XNNPACK` (accessed: 19.07.2025).

29. OpenVINO. OpenVINO is an open source toolkit for optimizing and deploying AI inference. URL: `https://github.com/openvinotoolkit/openvino` (accessed: 19.07.2025).

30. OpenVINO Toolkit – Open Model Zoo repository. Fork (branch '24.3.0/tvm' for Apache TVM, branch 'omz_executorch' for ExecuTorch). URL: `https://github.com/itlab-vision/open_model_zoo_tvm` (accessed: 19.07.2025).

31. PyTorch RISC-V support. URL: `https://discuss.pytorch.org/t/pytorch-risc-v-support/212065` (accessed: 19.07.2025).

32. Pirova A., *et al.* Performance optimization of BLAS algorithms with band matrices for RISC-V processors. Future Generation Computer Systems. 2026. Vol. 174. P. 107936. DOI: 10.1016/j.future.2025.107936.