

УДК 004.8:004.75:004.45

СРАВНИТЕЛЬНЫЙ АНАЛИЗ ПРОИЗВОДИТЕЛЬНОСТИ CUDA И WEBGPU В ЗАДАЧАХ МАШИННОГО ОБУЧЕНИЯ

Солдатов Андрей Тимофеевич, магистрант, направление подготовки 09.04.04 Программная инженерия, Московский авиационный институт (национальный исследовательский университет), Москва, Россия
e-mail: mastyfar@yandex.ru

Научный руководитель: **Султанов Наиль Закиевич**, доктор технических наук, профессор, профессор кафедры информационных технологий и систем, Российский государственный гуманитарный университет, Москва Россия
e-mail: sultanovnz@mail.ru

Аннотация. Современные задачи машинного обучения требуют высокопроизводительных вычислений, что актуализирует проблему зависимости от проприетарных решений, таких как CUDA. Цель работы – сравнительный анализ производительности и применимости универсального графического API WebGPU и специализированной связки PyTorch с CUDA. Использованы экспериментальные методы тестирования обучения свёрточной нейронной сети и инференса языковой модели Llama 3.2 1B с применением фреймворка Burn (wgpu) и PyTorch. Полученные результаты показывают, что WebGPU уступает CUDA по производительности в 2–3 раза, подтверждая гипотезу о компромиссе между переносимостью и эффективностью. Научная новизна заключается в систематическом сопоставлении современных инструментов. Практическая значимость связана с выбором технологий под ограниченные ресурсы. Дальнейшие исследования направлены на оптимизацию WebGPU и развитие соответствующих фреймворков.

Ключевые слова: машинное обучение, графические процессоры, CUDA, WebGPU, PyTorch, Burn, производительность вычислений, переносимость, нейронные сети, инференс моделей.

Для цитирования: Солдатов А. Т. Сравнительный анализ производительности CUDA и WebGPU в задачах машинного обучения // Шаг в науку. – 2026. – № 3. – С. 23–30.

COMPARATIVE PERFORMANCE ANALYSIS OF CUDA AND WEBGPU IN MACHINE LEARNING TASKS

Soldatov Andrey Timofeevich, postgraduate student, training program 09.04.04 Software Engineering, Moscow Aviation Institute, Moscow, Russia
e-mail: mastyfar@yandex.ru

Research advisors: **Sultanov Nail Zakievich**, Doctor of Technical Sciences, Professor, Professor of the Department of Information Technologies and Systems, Russian state university for the humanities, Moscow, Russia
e-mail: sultanovnz@mail.ru

Abstract. Modern machine learning tasks require high-performance computing, which makes the issue of dependence on proprietary solutions such as CUDA particularly relevant. The purpose of this study is to conduct a comparative analysis of the performance and applicability of the general-purpose graphics API WebGPU with the specialized PyTorch with CUDA stack. Experimental methods were used to test the training of a convolutional neural network and the inference of the Llama 3.2 1B language model using the Burn framework (wgpu) and PyTorch. The obtained results show that WebGPU is 2–3 times slower than CUDA, confirming the hypothesis of a trade-off between portability and efficiency. The scientific novelty of the study is in the systematic comparison of modern tools. Its practical significance is related to the selection of technologies under resource constraints. Further research is aimed at optimizing WebGPU and developing the corresponding frameworks.

Key words: machine learning, graphics processing units, CUDA, WebGPU, PyTorch, Burn, computational performance, portability, neural networks, model inference.

Cite as: Soldatov, A. T. (2026) [Comparative performance analysis of CUDA and WebGPU in machine learning tasks]. *Shag v nauku* [Step into science]. Vol. 3, pp. 23–30.

Введение

Современное машинное обучение в значительной степени опирается на высокопроизводительные вычисления, где доминирующую роль играют графические процессоры. Однако практическая реализация таких вычислений тесно связана с проприетарными программными стеками, прежде всего CUDA, что обеспечивает высокую эффективность, но одновременно формирует зависимость от конкретного аппаратного и программного вендора [1]. В результате переносимость решений между различными платформами оказывается ограниченной, а доступ к вычислительным ресурсам – неравномерным.

Отдельный интерес представляют инженерные задачи, в которых ML-модели используются для аппроксимации сложных физических зависимостей по экспериментальным данным. Например, для оценки параметров газовой смеси применяются регрессионные и древовидные методы машинного обучения [5]. В таких сценариях портативность вычислений становится особенно значимой: одна и та же модель может использоваться на исследовательской станции, промышленном сервере, локальном рабочем месте или встраиваемом устройстве, где доступны разные типы GPU и программных стеков. Таким образом, портативные API позволяют упростить развёртывание ML-решений на разнородном оборудовании, хотя и требуют оценки потерь производительности по сравнению со специализированными CUDA-реализациями.

Сходная проблема возникает в системах автоматизированного проектирования, где программные решения объединяют математические модели, обработку данных и пользовательский интерфейс. Работа Chernikova et al. демонстрирует применение такого подхода при автоматизации проектирования технологических процессов [3]. Для подобных систем портативные вычислительные API могут быть полезны при развёртывании на разнородном оборудовании, однако требуют оценки потерь производительности.

Альтернативой CUDA выступают универсальные графические и вычислительные API, такие как Vulkan, Metal, DirectX 12 и WebGPU, которые потенциально позволяют создавать платформонезависимые решения для задач машинного обучения. Тем не менее их практическое применение осложняется более низким уровнем абстракции, отсутствием зрелой экосистемы специализированных библиотек и неоднородностью драйверной поддержки.

В этой связи возникает конкретная проблема: несмотря на теоретическую универсальность графических API, остается неясным, в какой степени они

способны служить полноценной альтернативой специализированным решениям с точки зрения производительности, удобства разработки и охвата задач машинного обучения. Недостаток систематизированного анализа существующих подходов затрудняет оценку их применимости в реальных сценариях и выбор оптимальных инструментов.

Цель данной статьи – выполнить анализ подходов к использованию графических API в машинном обучении, рассмотреть существующие проекты и инструменты, а также сопоставить их возможности на примере базовых моделей с классическими системами. Особое внимание уделяется анализу компромисса между портативностью и производительностью, который определяет практическую применимость подобных решений.

В контексте поставленной цели формулируется следующая гипотеза исследования: несмотря на универсальность и портативность современных графических API, их использование в задачах машинного обучения приведёт к снижению производительности по сравнению со специализированными решениями, основанными на связке PyTorch и CUDA. Предполагается, что это обусловлено более низким уровнем оптимизации таких API под конкретное оборудование, а также ограниченным использованием специализированных аппаратных возможностей графических процессоров, таких как Tensor Cores [2]. Таким образом, ожидается наличие компромисса, при котором повышение переносимости и универсальности сопровождается частичной потерей вычислительной эффективности.

Для достижения данной цели необходимо выполнить ряд задач:

- определить используемый графический API для тестирования, а также тензорную библиотеку, использующую этот API;
- провести тестирование обучения модели свёрточной нейронной сети с помощью выбранной библиотеки и с помощью эталонной реализации на PyTorch;
- провести тестирование инференса большой языковой модели с помощью выбранной библиотеки и с помощью эталонной реализации на PyTorch;
- провести анализ результатов тестирования.

В качестве графического API для тестирования был выбран WebGPU, так как он обеспечивает максимальную портативность посредством трансляции в другие графические API, что видно в таблице 1, а также поддерживает вычислительные шейдеры, необходимые для моделирования тензорных вычислений [8].

Таблица 1. Сравнение аппаратных платформ, поддерживаемых CUDA и WebGPU

API	Nvidia	AMD	Apple	Intel	Qualcomm	Wasm*
CUDA	✓					
WebGPU	✓	✓	✓	✓	✓	✓

Примечания:

✓ – указанная аппаратная платформа поддерживается соответствующим API;

* – поддержка компиляции в набор инструкций виртуальной машины Wasm.

Источник: разработано автором

На практике взаимодействие с графическими процессорами редко осуществляется напрямую через низкоуровневые API. Вместо этого используются специализированные библиотеки тензорных вычислений, которые предоставляют высокоуровневые абстракции машинного обучения – тензоры, оптимизаторы, механизмы автоматического дифференцирования – и скрывают детали работы с GPU.

Среди современных проектов, реализующих подобный подход с использованием графических API, можно выделить два направления: lightweight-решения, ориентированные на простоту и прозрачность, и производительные фреймворки, нацеленные на эффективные вычисления. К первому типу относится `tinygrad` с бэкэндом WebGPU, где приоритет отдается компактности и понятности реализации. Ко второму – `burn`, поддерживающий бэкэнды `wgpu` (нативная реализация стандарта WebGPU) и `Vulkan` и изначально спроектированный с акцентом на производительность.

В данной работе рассматривается именно `burn`, поскольку он ориентирован на построение высокоэффективных вычислительных систем. `Burn` представляет собой тензорный фреймворк, разработанный на языке Rust, предназначенный для обучения и инференса нейронных сетей. Он предоставляет инструменты для работы с тензорами, автоматического дифференцирования и аппаратного ускорения, при этом особое внимание уделяется безопасности памяти, модульности и переносимости [11].

Несмотря на наличие полноценного функционала для обучения нейронных сетей, `Burn` менее удобен для быстрого прототипирования по сравнению с решениями на интерпретируемых языках. Это обусловлено использованием Rust, который предъявляет более строгие требования к типизации, управлению памятью и структуре кода, что увеличивает время разработки и снижает гибкость при проведении быстрых экспериментов. В отличие от экосистем, ориентированных на Python, где интерактивные среды и динамическая типизация ускоряют итеративный процесс, `Burn` требует более формального подхода к реализа-

ции моделей, что делает его менее предпочтительным инструментом на ранних этапах исследований.

Для сравнения производительности тестирование разделено на два этапа: оценка скорости обучения и скорости инференса. На каждом из этапов проводится замер производительности: PyTorch на центральном процессоре, PyTorch с бэкэндом `cuda` и `burn` с бэкэндом `wgpu`.

Такой формат тестирования был выбран, так как он позволяет оценить эффективность тензорного фреймворка с графическим API в целевых задачах, а именно в построении и обучении небольших моделей, и инференс больших моделей на портативных устройствах.

Стоит уточнить, что на данный момент различие производительности будет обусловлено не только влиянием API для вычислений на видеокарте, но и фреймворком тензорных вычислений, так как `burn` является молодым фреймворком, с низким уровнем зрелости, находящимся в стадии активной разработки, в то время как PyTorch долгое время является де-факто стандартом для ML исследований, поэтому результаты эксперимента отражают не только различия API, но и зрелость фреймворков, однако, как было сказано в разделе введения, использование API для задач ML возможно только с помощью тензорного фреймвока, поэтому подобное тестирование будет в большей мере отражать текущую картину.

Тестирование проводилось на видеокарте NVIDIA GeForce RTX 5070 и центральном процессоре AMD Ryzen 7 5800X.

В процессе эксперимента использовались следующие версии программного обеспечения:

- PyTorch 2.11.0;
- CUDA 13.0;
- `burn` 0.21.0-pre3.

Репозиторий с экспериментом доступен по ссылке: <https://github.com/AndreySoldatov/pytorch-burn-compare>.

Репозиторий с моделью llama 3.2, реализованной на `burn`, доступен по ссылке: <https://github.com/tracel-ai/models/tree/main/llama-burn>.

Обучение и инференс всех моделей проводились с типом данных `fp32`.

На этапе обучения использовалась свёрточная нейронная сеть, обучаемая на датасете MNIST. Архитектура модели включает несколько свёрточных ядер и два линейных слоя. Общее число параметров составляет 531178, что, с одной стороны, позволяет наглядно сравнить производительность обучения на

центральной процессоре и видеокарте, а с другой – обеспечивает возможность запуска на потребительских GPU [4].

Для обучения использовался оптимизатор Adam и `batch_size = 128`. Замер времени проводился по эпохам.

Архитектура модели представлена на рисунке 1.

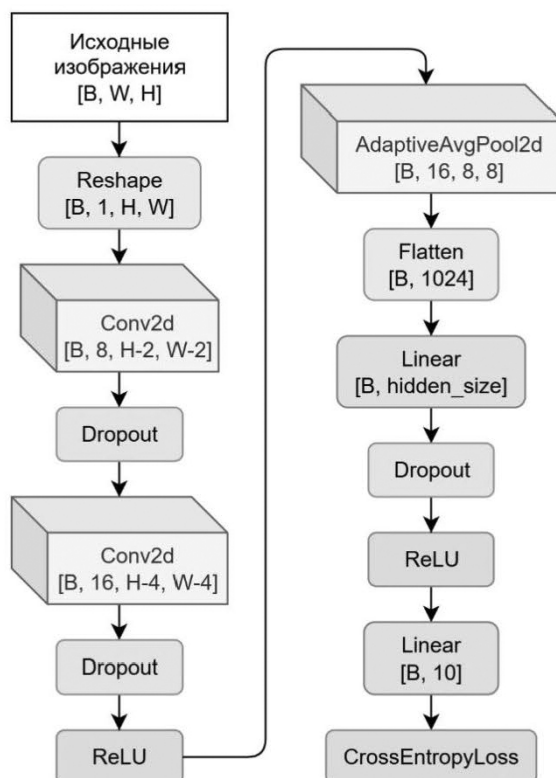


Рисунок 1. Архитектура свёрточной нейронной сети

Источник: разработано автором

В результате эксперимента были получены данные о том, сколько времени каждая реализация тратит на одну эпоху обучения.

Результаты эксперимента представлены в таблице 2 и на рисунке 2.

Таблица 2. Результаты обучения свёрточной нейронной сети

Показатель	PyTorch CPU	PyTorch GPU	Burn wgpu
Среднее время обучения (секунд/эпоху)	20.82	5.63	11.13

Источник: разработано автором

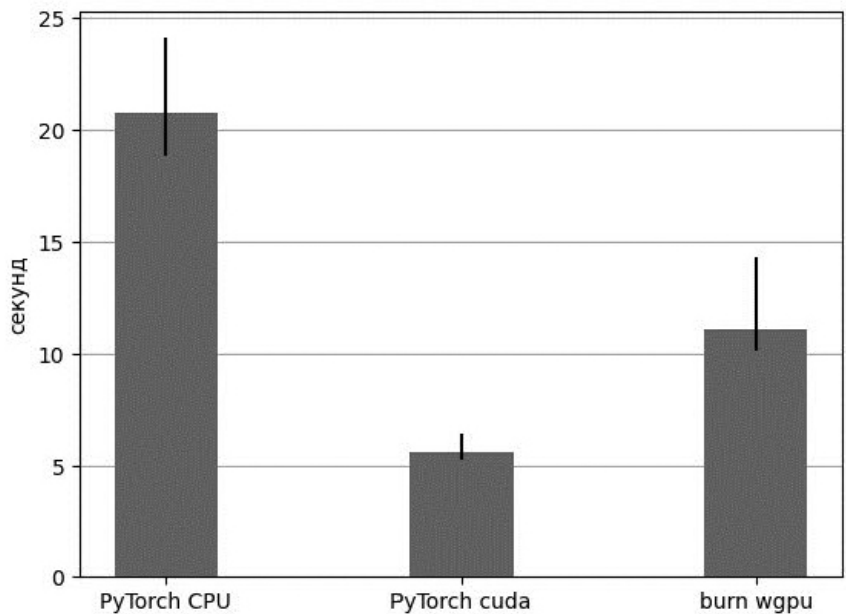


Рисунок 2. Результат замеров производительности обучения свёрточной нейронной сети (чем меньше, тем лучше)

Источник: разработано автором

График показывает, что обучение на CPU в PyTorch происходит существенно медленнее, чем на GPU, что ожидаемо из-за высокой параллельности вычислений на видеокарте. Бэкенд Burn с использованием WebGPU занимает промежуточное положение: он примерно в 2 раза быстрее CPU, но также в 2 раза медленнее Cuda.

Для оценки скорости инференса выбрана языковая модель Llama 3.2 1B Instruct. Интерес к обучению и инференсу больших языковых моделей в настоящее время высок, поэтому их включение в эксперимент является обоснованным. Архитектура трансформера позволяет эффективно выявить различия в производительности, поскольку операции attention и связанные с ними вычисления хорошо оптимизированы в CUDA и PyTorch. Выбор конкретной модели обусловлен её применимостью к сценарию локального инференса

на клиентских и edge-устройствах с ограниченными вычислительными ресурсами, таких как смартфоны, встраиваемые системы и IoT-устройства. Для таких сценариев характерны жёсткие ограничения по производительности, энергопотреблению и доступной памяти, что делает выбор эффективного вычислительного стека особенно значимым [10].

В данном случае тестирование проводилось с максимальной, поддерживаемой моделью длиной контекста (131072 токена) и длина генерируемого потока токенов ограничена 128.

Архитектура модели Llama 3.2 1b приведена на рисунке 3.

Критерием производительности в данном случае является среднее количество сгенерированных токенов в секунду. Результат замеров производительности отражён на рисунке 4.

Таблица 3. Результаты инференса большой языковой модели

Показатель	PyTorch CPU	PyTorch GPU	Burn wgpu
Среднее количество токенов в секунду	0.45	35.59	11.23

Источник: разработано автором

Llama 3.2 1B

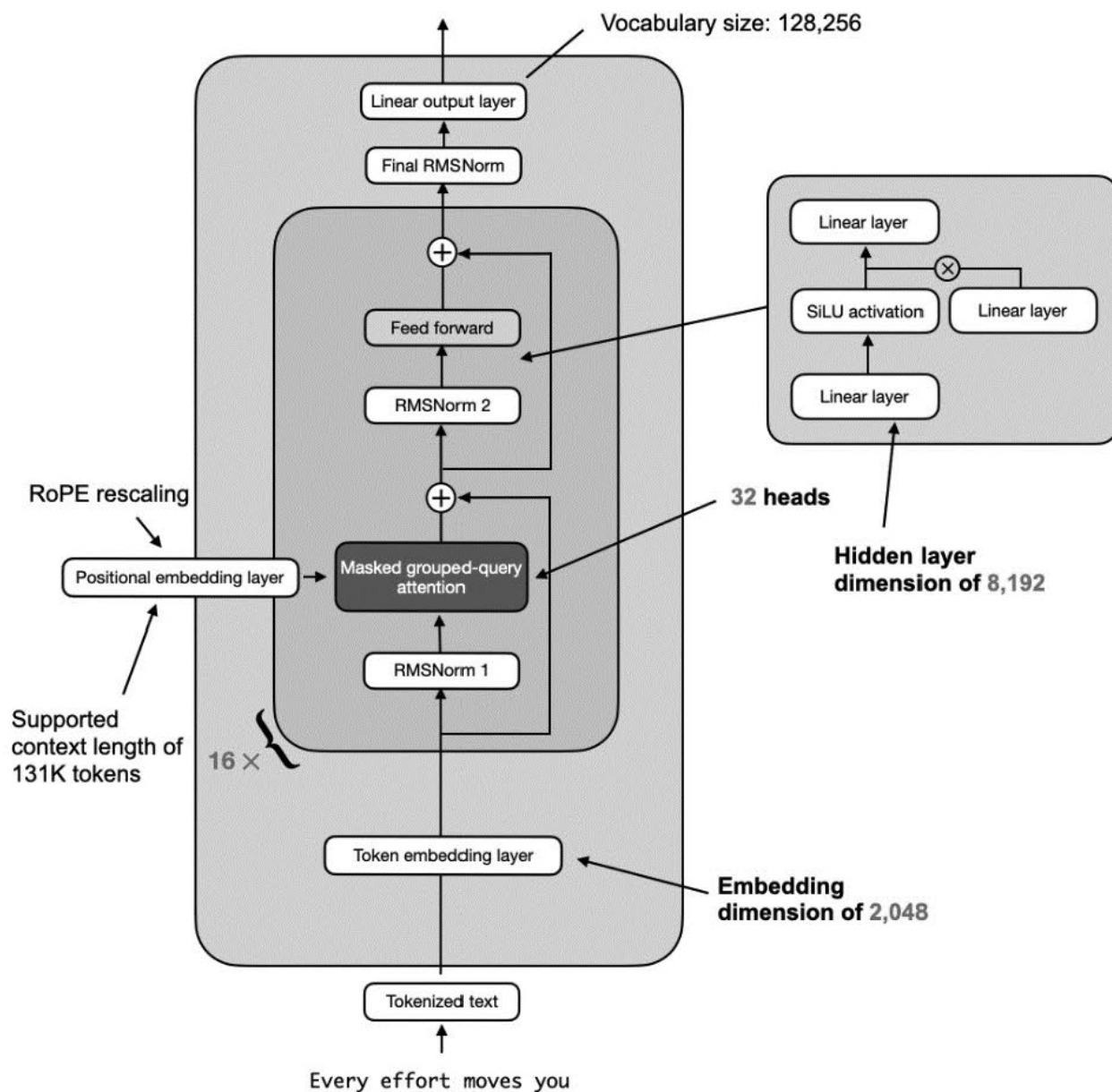


Рисунок 3. Архитектура модели Llama 3.2 1b

Источник: взято из работы [9]

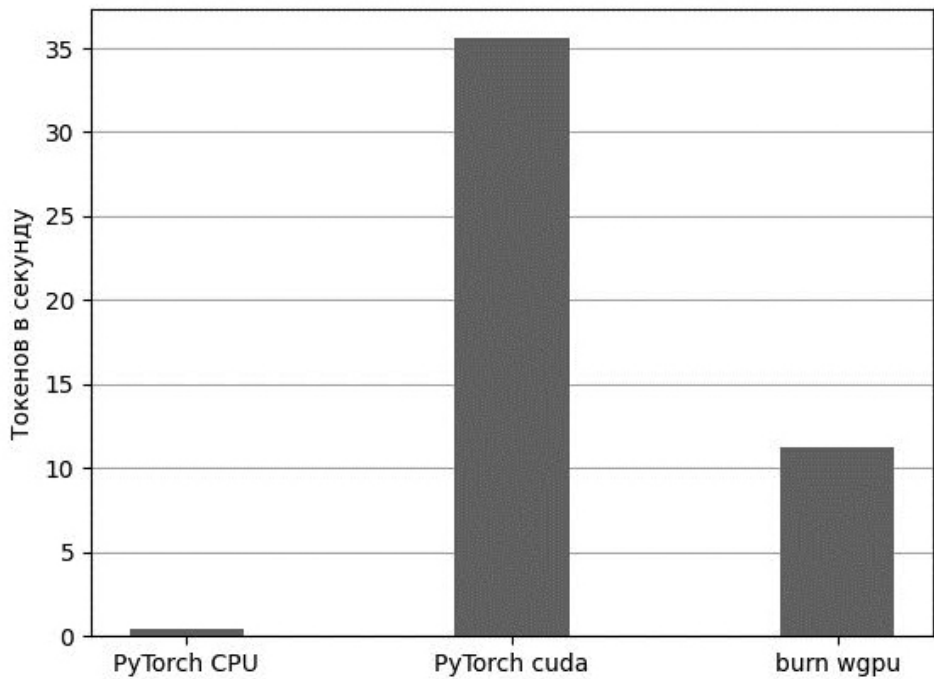


Рисунок 4. Результат замеров производительности инференса большой языковой модели (чем больше, тем лучше)

Источник: разработано автором

Из графика следует, что реализация на CUDA по-прежнему демонстрирует более высокую производительность по сравнению с WebGPU и генерирует токены примерно в 3 раза быстрее, однако обе реализации значительно опережают реализацию на CPU.

Тесная интеграция PyTorch с экосистемой NVIDIA, включая использование высокооптимизированных библиотек cuBLAS и cuDNN, а также тензорных ядер, обеспечивает значительное преимущество в производительности на соответствующем оборудовании. Однако такая оптимизация в определённой степени снижает переносимость решений, поскольку максимальная эффективность достигается преимущественно на GPU NVIDIA [7].

В противоположность этому, фреймворк Burn с бэкендом WebGPU реализует более универсальный подход: он жертвует частью производительности ради высокой портативности, позволяя выполнять вычисления на любом устройстве с поддержкой современного графического API [11].

Экспериментальные результаты показывают, что с ростом числа параметров модели и степени параллелизма вычислений преимущество GPU над CPU становится более выраженным. При этом разрыв в производительности между CUDA-реализацией и WebGPU также увеличивается, что указывает на более высокий уровень низкоуровневой оптимизации вычислений в случае PyTorch, что можно видеть в таблице 4.

Таблица 4. Сравнительная таблица производительности, относительно реализации на CPU

Задача	PyTorch CPU	Pytorch Cuda	Burn WebGPU
Обучение CNN	1.0	3.7	1.9
Инференс LLM	1.0	79.1	25.0

Источник: разработано автором

Выводы

Обучение моделей с миллиардами параметров требует значительных вычислительных ресурсов и времени, что напрямую связано с финансовыми затратами. В этом контексте использование WebGPU в настоящее время ограничено: потенциальное снижение производительности в 2–3 раза по сравнению с CUDA-программами приводит к пропорциональному увеличению времени обучения, энергопотребления и тепловыделения, что делает такой подход экономически неэффективным для крупных задач, таких как обучение больших языковых моделей.

В то же время WebGPU может быть оправдан для небольших исследовательских групп с ограниченными ресурсами, а также для задач инференса и локального развёртывания моделей на устройствах без дис-

кретных GPU NVIDIA – например, на смартфонах или встраиваемых системах.

Следует учитывать, что приведённые ограничения относятся к текущему уровню зрелости WebGPU. В рамках развития стандарта в Google и связанных рабочих группах рассматриваются возможности расширения функциональности, включая поддержку тензорных ядер, SIMD-подгрупп и других механизмов, направленных на повышение эффективности тензорных вычислений на широком спектре аппаратных платформ [6].

Кроме того, сам фреймворк Burn находится в стадии активной разработки и пока уступает PyTorch по уровню зрелости оптимизаций, таких как слияние операций (kernel fusion) и других низкоуровневых техник повышения производительности [11].

Литература

1. Козлов С. О., Медведев А. А. Использование технологии CUDA при разработке приложений для параллельных вычислительных устройств // Вестник Курганского государственного университета. Серия: Естественные науки. – 2015. – № 4(38). – С. 106–112. – EDN: WKOYTP.
2. Кузьминский М. Б. Новое поколение GPGPU и сопутствующего оборудования: микроархитектура и производительность вычислительных систем от серверов до суперкомпьютеров // Программные системы: теория и приложения. – 2024. – Т. 15, № 2(61). – С. 139–473. – <https://doi.org/10.25209/2079-3316-2024-15-2-139-473>. – EDN: OJEKRO.
3. Chernikova N. N., et al. (2018) Preproduction Using Automated Process Design. *Russian Engineering Research*. – Vol. 38. – No. 5, pp. 384–386. – <https://doi.org/10.3103/S1068798X18050052>. – EDN: YBTQTB.
4. D.-y. Ge, et al. (2020) *Design of High Accuracy Detector for MNIST Handwritten Digit Recognition Based on Convolutional Neural Network*, 2019 12th International Conference on Intelligent Computation Technology and Automation, ICICTA, 2019. Xiangtan, China, 26–27 October. New York: IEEE. – <https://doi.org/10.1109/ICICTA49267.2019.00145>. (In Eng.).
5. Dayev Z., et al. (2022) Modeling the flow rate of dry part in the wet gas mixture using decision tree/kernel/non-parametric regression-based soft-computing techniques. *Flow Measurement and Instrumentation*. – Vol. 86, pp. 102195. – <https://doi.org/10.1016/j.flowmeasinst.2022.102195>. – EDN: VBFWZG.
6. Eng A., Gandluri D., Beaufort F. (2024) WebAssembly and WebGPU enhancements for faster Web AI, part 2. *Chrome for developers*. – URL: <https://developer.chrome.com/blog/io24-webassembly-webgpu-2> (accessed: 27.04.2026).
7. Hanindhito B., John L. K. (2024) *Accelerating ML Workloads using GPU Tensor Cores: The Good, the Bad, and the Ugly*, proceedings of the 15th ACM/SPEC International Conference on Performance Engineering, ICPE, 2024. London, United Kingdom, 7–11 May. New York: Association for Computing Machinery. – <https://doi.org/10.1145/3629526.3653835>. (In Eng.).
8. Kenwright B. (2022) *Introduction to the WebGPU API*, proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference, SIGGRAPH 2022 Courses, 2022. Vancouver, BC, Canada, 7–11 August. New York: Association for Computing Machinery. – <https://doi.org/10.1145/3532720.3535625>. (In Eng.).
9. Raschka S. (2024). *Build a Large Language Model (From Scratch)*. New York: Manning Publications Co. – URL: <https://djuv.online/file/ulhD8PI5UQ9o1> (accessed: 27.04.2026).
10. Shuvo Md. M. H. (2023) Efficient Acceleration of Deep Learning Inference on Resource-Constrained Edge Devices: A Review. *Proceedings of the IEEE*. – Vol. 111. – No. 1, pp. 42–91. – <https://doi.org/10.1109/jproc.2022.3226481>. – EDN: IIFMBT.
11. Why Burn? *The Burn Book*. – URL: <https://burn.dev/books/burn/> (accessed: 27.04.2026). (In Eng.).

Статья поступила в редакцию: 28.04.2026; принята в печать: 05.08.2026.

Автор прочитал и одобрил окончательный вариант рукописи.