DOI: 10.15514/ISPRAS-2025-37(5)-12

Improving Image Analysis and Processing Performance on the RISC-V Platform with Lichee Pi 4A

N.I. Cherepanov, ORCID: 0009-0001-9135-9654 <cherepanov.ni@edu.spbstu.ru>
N.O. Stepina, ORCID: 0009-0001-4740-637X <gubenko_no@spbstu.ru>
I.V. Nikiforov, ORCID: 0000-0003-0198-1886 <nikiforov_iv@spbstu.ru>

Peter the Great St. Petersburg Polytechnic University,
29, Polytechnicheskaya st., St. Petersburg, 195251, Russia.

Abstract. The study explores optimization methods for improving image processing performance on the RISC-V platform with Lichee Pi 4A. The research focuses on real-time video processing within a microservice-based self-service system. Several existing optimization strategies are considered and evaluated, including neural network model optimization, hardware acceleration using RVV vector instructions and leveraging the built-in Neural Processing Unit (NPU). The profiling results on existing strategies indicate that object detection and feature extraction consume the most computation resources. In order to eliminate the performance gap, the model quantization to INT8 format is implemented, that allows to reduce memory usage and inference latency. Additionally, a modified ONNX Runtime version is deployed to support NPU acceleration. These improvements led to 75% reduction in model size and a 35% decrease in inference latency. The study concludes that hardware-aware optimizations significantly enchase performance on the RISC-V (Lichee Pi 4A) platform. The main issue encountered is the low processing speed on Lichee Pi 4A, with a current frame rate of only 0.05 FPS, which in unsuitable for practical usage.

Keywords: RISC-V; Lichee Pi 4A; image processing; neural network; vectorization; NPU; ONNX Runtime; performance optimization; real-time processing.

For citation: Cherepanov N. I., Stepina N. O., Nikiforov I. V. Improving image analysis and processing performance on the RISC-V platform with Lichee Pi 4A, Proceedings of the Institute for System Programming of the RAS, vol. 37, issue 5, 2025, pp. 157-172. DOI: 10.15514/ISPRAS-2025-37(5)-12.

Повышение производительности анализа и обработки изображений на платформе RISC-V с помощью Lichee Pi 4A

Н.И. Черепанов, ORCID: 0009-0001-9135-9654 <cherepanov.ni@edu.spbstu.ru>
Н.О. Степина, ORCID: 0009-0001-4740-637X <gubenko_no@spbstu.ru>
И.В. Никифоров, ORCID: 0000-0003-0198-1886 <nikiforov_iv@spbstu.ru>

Санкт-Петербургский политехнический университет Петра Великого,
195251, Россия, Санкт-Петербург, Политехническая улица, д. 29.

Аннотация. В исследовании изучаются методы оптимизации для повышения производительности обработки изображений на платформе RISC-V с использованием Lichee Pi 4A. Исследование сосредоточено на обработке видео в режиме реального времени для системы самообслуживания, которая реализована в виде микросервисного приложения. Рассматриваются и оцениваются стратегии оптимизации, включая оптимизацию модели нейронной сети, аппаратное ускорение с использованием векторных инструкций RVV и использование встроенного ускорителя для нейронных сетей (NPU). Результаты профилирования существующих стратегий показывают, что обнаружение объектов и извлечение признаков потребляют большую часть вычислительных ресурсов. Чтобы устранить разрыв в производительности, реализовано квантование модели в формат INT8, что позволяет сократить использование памяти и задержку вывода. Кроме того, развернута модифицированная версия ONNX Runtime для поддержки ускорения NPU. Эти улучшения привели к уменьшению размера модели на 75% и уменьшению задержки вывода на 35%. В исследовании делается вывод, что аппаратно-ориентированные оптимизации значительно повышают производительность на платформе RISC-V (Lichee Pi 4A). А также определена основная проблема практического применения разработанного решения на Lichee Pi 4A, связанная с низкой скоростью обработки данных: текущая частота кадров составляет всего 0,05 FPS.

Ключевые слова: RISC-V; Lichee Pi 4A; обработка изображений; нейронная сеть; векторизация; NPU; ONNX Runtime; оптимизация производительности; обработка в реальном времени.

Для цитирования: Черепанов Н. И., Степина Н. О., Никифоров И. В. Повышение производительности анализа и обработки изображений на платформе RISC-V с помощью Lichee Pi 4A, Труды ИСП РАН, том 37, вып. 5, 2025 г., стр. 157–172 (на английском языке). DOI: 10.15514/ISPRAS–2025–37(5)–12.

1. Introduction

Modern and young open RISC-V [1] architecture is widely used in embedded systems and high-performance computing. However, when it comes to computer vision [2] and image processing, the platforms, that implements the RISC-V architecture, face several challenges. Well-established architectures such as x86 and ARM [3] are free of those challenges because of years of development and thousands of researchers and developers involved.

One of the main challenges of using RISC-V (especially on Lichee Pi 4A) for image and video processing is low framerate (FPS) when processing video streams, which is critical for object detection and classification.

For production lines and environments, where, for example, robotic arms are used, that are equipped with vision systems, video processing plays a crucial role in object recognition (Fig. 1). Computer vision relies heavily on video stream processing [4] as working with dynamic scenes requires real-time object recognition and rapid system response to changes. This is particularly important in fields such as retail, medical diagnostics and autonomous systems, where the accuracy and speed of frame analysis directly impact decision-making. Transitioning from standard processors to RISC-V platforms could significantly reduce manufacturing costs due to their open-source nature and hardware flexibility in comparison to traditional hardware and software design.

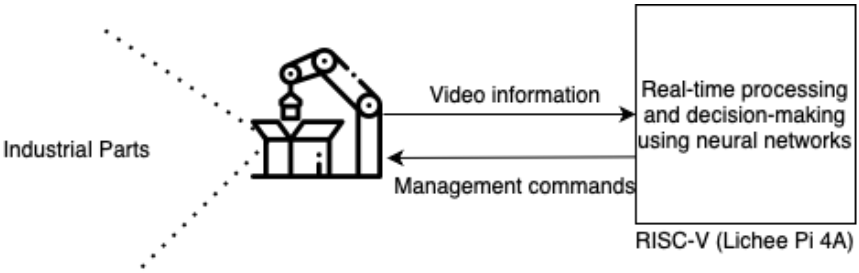

Fig. 1. Testing environment - computer vision system.

There are the following existing implementations of RISC-V on the market: Lichee Pi 4A, Mango pi MPI-MQ1, Milk-V Pioneer, Banana Pi BPI-K1, VisionFive 2, GiFive Unmatched. Each of these platforms varied in terms of performance, available features and suitability for machine vision applications (Table 1).

The Lichee Pi 4A board served as the hardware platform for this project, following a task proposed by an industrial partner. The goal of the work includes evaluation of the performance characteristics and evaluating if Lichee Pi 4A is suitable for practical applicability of this specific RISC-V implementation in real-time machine vision scenarios. Compares to other boards, Lichee Pi 4A offered a balanced combination of high CPU frequency, a powerful GPU and a dedicated NPU, making it suitable for neural inference tasks such feature extraction.

Table 1. Comparison of characteristics of single RISC-V Boards.

Model	CPU	CPU Freq.	GPU	NPU	RAM	Price
Mango Pi	Allwinner D1 (C906, RISC-V)	1.0GHz	-	-	1GB DDR3	~$20
Lichee Pi	T-Head TH1520 (4xC910)	2.0GHz	Imagination BXM-4-64	4 TOPS	up to 16GB LPDDR4X	~$119
Mikl-V	SOPHON SG2042 (64xC920)	up to 2Ghz	-	-	up to 128GB DDR4	~$1000
Banana Pi	SpacemiT K1 (6xX60)	-	IMG BXE-2-32	2 TOPS	up to16 GB LPDDR4	~$100
VisionFive 2	StarFive JH7110 (4xU74)	1.5GHz	IMG NXE-4-32	-	up to 8GB LPDDR3	~$70
HiFive	SiFive U740 (4xU74 +S7 core)	1.2GHz	-	-	16GB DDR4	~$665

As a result of the testing and evaluating the performance in the article it is concluded that Lichee Pi 4A lags in performance, especially in real-time processing. This is not due to RISC-V flaws in the architecture itself, but rather its relative novelty: high-performance chips are still in development and many essential software tools have not been ported yet.

As far as there is no direct access to industrial systems, article authors created a development environment for retail domain. There is a microservise application [5] developed, where video processing serves as the functionality. Based on this system, various optimization approaches are considered and evaluating. The system consists of three main microservices [6]:

- backend service - responsible for video stream processing, object detection and managing the consumers requests;

- frontend service - provides the user interface and displays the video stream;
- database service - stores product data, including names, prices and categories.

The main goal of testing system, that is used for performance evaluation, is to automatically identify the products taken by the customer and generate a shopping cart for checkout. However, its current implementation, the video processing speed is only 0.05 FPS, making the system unsuitable for practical use. To ensure successful, the processing speed must reach 30 FPS [7].

Thus, the key objective of this study is to increase the performance of the computer vision system on the Lichee Pi 4A platform to 30 FPS. To achieve this, the following steps are necessary [8]:

- optimizing the neural network model for object detection;
- improving frame processing while considering the capabilities of the RISC-V platform;
- utilizing hardware accelerators such as NPU, SIMD and RISC-V Vector Extensions (RVV) [9].

To evaluate the system's real-time performance, the frames per second (FPS) metric is measured using Python high-resolution timer. The procedure includes the following steps:

- at the beginning of each frame-processing cycle, the start timestamp is recorder;
- the frame undergoes all stags of processing, including acquisition, processing, neural network inference and postprocessing;
- upon completion, the end timestamp is recorded;
- the time taken for a single frame is computed as the difference between the end and start time;
- instantaneous FPS is calculated as the reciprocal of the frame time;
- this process is repeated for a large number of frames and the average FPS is derived by averaging the collected values.

To assess the computational load of operations, CPU usage is analyzed at each stage of processing. The 15-20% allocation for preprocessing is determined by comparing the total processing time with the time spent specifically on this stage across several experiments.

In order to understand hardware and software design (co-design) of the experiment stand, that is critical for performance evaluation, let's consider every part separately.

2. Research

Modern research it the field of image processing on the RISC-V platform demonstrates a growing interest in optimizing performance and energy efficiency, especially for embedded systems and devices with limited resources. This chapter examines the key work on this topic, as well as highlights their main achievements and limitations.

In [10], a hardware accelerator for YOLOv3-Tiny using RISC-V SoC was proposed. The authors achieve a bandwidth of 21.6 GOPS/s, but note limitations associated with frequent memory access.

The article [11] compares various models (SOLO, SSD, Faster RUN) on the SiFive U540 platform. YOLOv3 and SSD-MobileNet showed the best results, which confirms the importance of choosing a model for a specific hardware platform.

The work [12] demonstrates the advantages of vector instructions to speed up CNN operations. The authors note that increasing the length of the vector (VLEN) does not always lead to a proportional increase in performance due to memory limitations.

In [13], the use of TVM for quantized RISC-V models with the P extension is investigated. The results show an acceleration of 2.7 – 7.0 times compared to FP32, which highlights the potential of quantization for RISC-V.

3. Platform's hardware equipment

The project is implemented using the Lichee Pi 4A - a single-board computer based on the T-Head TH1520 processor. Its key specifications include:

- processor – 4-core RISC-V C920 (up to 1.85 GHz) with SIMD and RVV 0.7.1 support;
- graphics – 50 GFLOPS Imagination BXM-4-64 GPU (supports OpenGL ES 3.x and Vulcan);
- NPU – 4 TOPS performance for accelerating AI computations;
- RAM – up to 16 GB LPDDR4/4x.

The T-Head TH1520 processor, developed by Alibaba Group's semiconductor division, is designed for embedded systems with high computational demands. It features an optimized L1 and L2 cache hierarchy, which plays a crucial role in processor performance. The L1 cache is split into separate instruction and data caches, allowing for faster access to frequently used data and reducing latency. The L2 cache, being larger and shared among cores, helps mitigate memory bottlenecks by storing recently accessed data, reducing the need for frequent main memory accesses. This cache structure significantly improves processing speed, particularly in image analysis and video processing tasks, where rapid data retrieval is essential. The BXM-4-64 GPU provides hardware-accelerated rendering and supports 4K displays. However, for machine learning tasks and other algorithms that require massive parallel computing, it is recommended to use NPU, since its performance higher than the GPU capabilities in similar workloads.

4. Software architecture

ONNX Runtime is a high-performance inference engine designed to execute machine learning models in the ONNX (Open Neural Network Exchange) format [14]. It provides hardware acceleration and optimization techniques, making it suitable for deployment across various platforms, including CPU, GPU and specialized accelerators.

The project uses ONNX Runtime for model execution because TensorFlow, PyTorch and other major ML libraries are not officially ported to RISC-V. TensorFlow Lite for Microcontrollers has been ported to RISC-V architecture, but this is just a lightweight version. Porting the full version of TensorFlow to RISC-V requires the use of cross-compilers and additional settings, which is confirmed by the documentation of the RISE project. PyTorch also has no official support for the RISC-V architecture. There are initiatives to port PyTorch to RISC-V, such as the pythorch-riscv64 project, which provides pre-built packages for RISC-V. However, these solutions are experimental and are not part of the official PyTorch release. In addition, discussions on the PyTorch forums confirm that official support for RISC-V is in plans but has not yet been implemented. Since there is no built-in support for these platforms in RISC-V, ONNX provides a universal solution that allows you to export models trained in various environments (for example, PyTorch or TensorFlow) to ONNX format and then efficiently execute them on RISC-V hardware.

Key reasons for choosing ONNX on Lichee Pi 4A are listed below.

1. Cross-platform compatibility – ONNX models can be exported from multiple ML frameworks.
2. Hardware acceleration – ONNX Runtime optimizes inference through quantization, graph optimizations and hardware-specific execution provides.
3. Lack for TensorFlow/PyTorch support – since these frameworks are not available on RISC-V, ONNX is the best alternative.
4. Support for custom execution providers – while ONNX Runtime does not native support TH1520 NPU, it allows experimentation with custom providers like ShlExecutionProvider for potential acceleration.

YOLOv8n (You Only Look Once, version 8, nano model) is a deep learning model designed for real-time object detection [15]. It balances accuracy and speed, making it suitable for embedded systems like the Lichee Pi 4A. The model is exported in ONNX format for compatibility with ONNX Runtime.

Key features of YOLOv8n:

- single-stage detection – the model predicts object location and classifications in a single pass, ensuring fast inference;
- optimized for edge devices – the "small" version is designed for efficiency, making it suitable for resource – limit platforms;
- flexibility – it can be quantized to INT8 for acceleration on NPU, though additional steps are needed for TH1520 support.

YOLOv8n followed a CSP-based architecture [16] and included three main components:

- backbone (C2f + CBS) – extracted features at multiple scales using convolutional layers with residual connections;
- neck (PAN-FPN) – aggregated multi-scale feature maps using anchor-free detection;
- head – prediction object classes and bounding boxes directly from feature maps using anchor-free detection.

This lightweight design allowed the model to maintain good detection accuracy with reduced latency and memory usage.

To train the object detection model, a custom dataset is created. The dataset consists of N products categories, each containing 500 images, a total of 2800 images are used for training and validation of the model, approximately 85MB on disk, collected from various online sources [17]. The dataset is prepared in the YOLO format, which includes:

- images – the raw images containing objects of interest;
- annotation files – each image has a corresponding text file with bounding box coordinates and class labels in YOLO format.

The annotation process involved:

- collecting images – downloading diverse product images to cover different angles, lighting conditions, and backgrounds;
- manually labeling objects – using LabalImg and other annotation tools to draw bounding boxes around objects and assign category labels.

The dataset images vary in resolution. All images are stored in 24-bit RGB color format with a DPI of 72. This dataset is used to train YOLOv8n, optimizing it for real-world object detection in the system.

As the metrics below show, this amount of data is enough to detect objects, but for more important tasks, for example in the field to medicine, where accuracy should be close to 1, an order magnitude more images are needed [18].

After training, the model achieved high accuracy. The average reached 0.993, indicating an almost perfect match between predicated and actual objects.

On the Precision-Recall Curve [19] (Fig. 2), the curve for most classes stayed close to the upper-right corner, confirming high precision along with excellent recall.

On the Recall-Confidence Curve [20] (Fig. 3), all classes maintained high recall up to a confidence threshold of 0.85-0.9, meaning the model detected almost all object even at high confidence levels.

The system follows a structured pipeline for image processing and object detection, that is described step by step below.

1. Preprocessing – normalization, resizing, and noise reduction.
2. Embedding extraction – converting the image into a vector representation.
3. Inference – running the neural network for detection and classification.
4. Postprocessing – interpreting and visualizing the results.

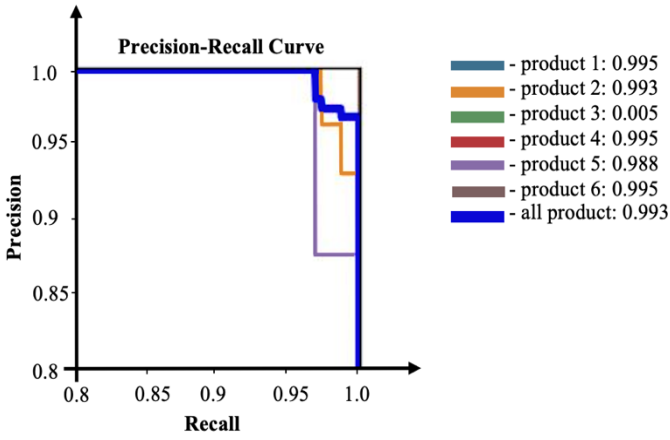

Fig. 2. Precision-Recall Curve for testing dataset of object recognition.

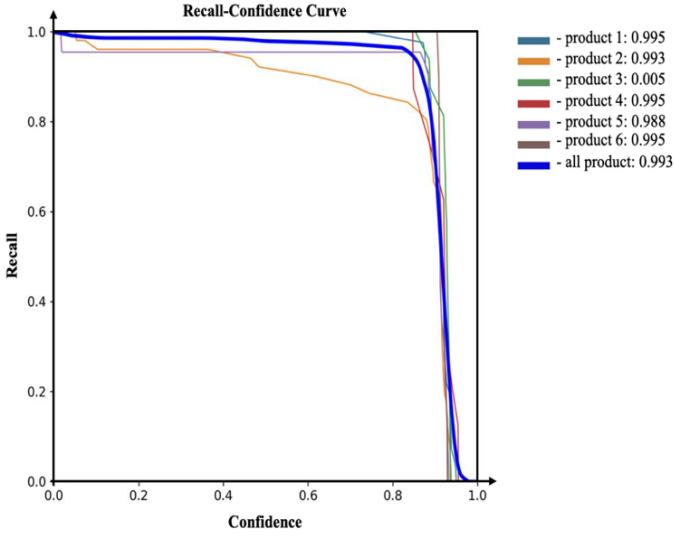

Fig. 3. Recall-Confidence Curve for testing dataset of object recognition.

5. Bottleneck analysis

Let's consider video processing steps and how they are implemented.

The OpenCV library is utilized for video stream capture and preprocessing [21], providing user-friendly interfaces for handling video sources and image processing. Object detection is performed using the ONNX version of YOLO [22], executed via ONNX Runtime.

Video acquisition is handled using OpenCV through the cv2.VideoCapture object. The resolution parameters for the video stream are defined this loop as listing 1:

Listing 1. Loop video stream capture

```
cap = cv2.VideoCapture(1)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
```

This configuration allows capturing frames at a resolution of 640x480 pixels in real time. The value 1 in VideoCapture(1) specifies that an external camera is being used. However, the resolution of 640x480 indicated in the article formally falls under the category of "low" according to GOST 51558-2014, where the threshold is considered to be a resolution of up to 756x576 pixels. In addition, the choice of this resolution in the article is not due to an attempt to achieve an industrial level of quality, but to the desire to demonstration the operability of the entire system at a prototype level with low hardware capabilities. For industrial implementation, the solution can be adapted to a higher camera resolution that meets requirements of GOST with more efficient hardware at the same time. With our current experiment we see, that even for low picture resolution the recognition speed is not enough for industrial tasks.

Object detection is performed using YOLO model in ONNX format. The ONNX Runtime library in used for inference and preprocessing steps include below items.

1. Conversion to a Pillow object.
2. Resizing to 640x640 pixels (matching the YOLO model input format).
3. Pixel value normalization and array reshaping.

A typical video processing pipeline consists of the following key stages [23] listed below.

1. Video capture (from a camera or disk).
2. Preprocessing (video pipeline).
3. Object detection (detector).
4. Main data processing (post processing).
5. Database search (search).

Analyzing the workload at each stage helps identify the most resource-intensive operations and determine bottlenecks that affect system performance. The percentage values in the table are obtained through profiling and benchmarking of each processing stage. These are relative values from the total time. The time share is measured by running the video processing pipeline on the Lichee Pi 4A and recording the execution time for each step. Profiling and logging tools are used to analyze performance bottlenecks, and running tests multiple times ensures consistency of the results shown in Table 2.

Table 2. Time distribution and main limitations for video processing stages.

Processing stage	Time share	Main limitations
Video capture	0.10%	Depends on camera I/O speed
Preprocessing	36.27%	Resize, normalization CPU-bound
Object detection	61.24%	Low CPU processing speed
Main data processing	1.80%	Decode CPU-bound
Database search	0.04%	Scale with database size

To identify bottleneck in system performance, a detailed analysis is conducted using the gprof tool. The primary focus is on the following aspects:

- function execution time – measuring the time spent on key image processing stages;
- CPU and NPU workload distribution – analyzing hardware accelerator utilization;
- cache efficiency – evaluating the impact of caching on data processing speed.

Profiling is performed on real video streams with a resolution of 640x480 pixels. The backend service used for testing is ran on the Lichee Pi 4A.

5.1. Video capture and pre-processing

The video capture and pre-processing stage involves reading and decoding the video stream [24]. The main workload comes from continuous data writing and reading, which can quickly fill the cache memory. A limited cache size may cause additional delays due to frequent access to RAM.

5.2. Object detection

Detection is the first stage where neural network algorithms are applied [25]. It runs faster on a GPU, but if GPU acceleration is unavailable, the CPU must handle the workload, creating significant pressure on processor cores. On the Lichee Pi 4A, the built-in IPU can be used, but integrating it with ONNX Runtime presents certain challenges:

- manual model conversion and low-level integration are required;
- hardware support is limited;
- the lack of documentation and stable tools (HHB, SDK) complicates debugging.

5.3. Inference

This stage involves running deep neural network models [26]. The main workload is typically handled by the GPU, but on the RISC-V platform with RVV, some vector processing operations can be offloaded to hardware, reducing dependence on the GPU. However, the lack of stable OpenCL/Vulkan drivers for the GPU remains an issue.

5.4. Vectorization and embedding

After inference, feature extraction is required. This stage demands intensive computation. The use of SIMD and RVV vector instructions could speed up the process [27], but current implementations do not always take full advantage of these capabilities.

5.5. Database search

This stage places a load on memory and storage. If the database is stored on a device with limited memory, frequent disk access can cause delays. However, in the current project, this is not a critical issue that needs immediate resolution.

Future following improvements are possible:

- implementation of multi-level caching;
- use of memory-mapped files;
- optimization of data structures for RISC-V architecture;
- vector quantization to reduce memory footprint.

5.5. Diagnosing problems

Video processing challenges on the Lichee Pi 4A stem from both hardware limitations and software inefficiencies. Optimizations such as SIMD/RVV utilization, hardware NPU acceleration, and advanced memory management algorithms can significantly improve system performance.

Diagnostics can be performed at different levels:

- hardware level – the Lichee Pi 4A is based on a relatively new architecture and its processor implementation differs from more established platforms. This leads to potential inefficiencies due to immature compiler optimizations, incomplete hardware support, and

limited documentation;

- software level – many libraries and frameworks have not yet been fully ported to RISC-V, leading to compatibility issues and suboptimal performance. Additionally, the software stack itself can often be optimized further, reducing redundant computations and improving overall efficiency.

6. Optimization methods

Optimizing image processing is an important aspect that helps reduce computation time [28], lower CPU load and improve overall system performance. This section discusses various optimization techniques, from using hardware instructions to implementing multithreaded data processing.

6.1. Using RVV for preprocessing and preprocessing

The RVV vector instruction extension enables parallel computing (Fig. 4), which is particularly useful for matrix and tensor operations is neural network models [29] This method significantly speeds up tasks involving large amounts of data, such as image transformation, normalization, and convolution.

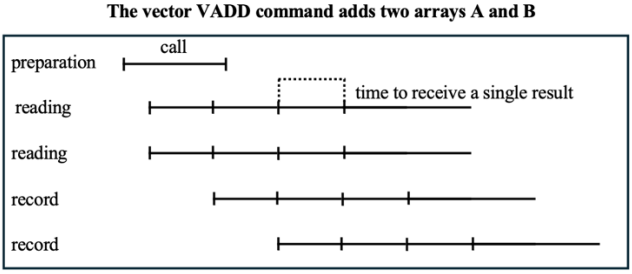

Fig. 4. Time diagram of vector addition of two arrays.

Applying RVV (RISC-V Vector Extension) can notably accelerate both data preprocessing (such as normalization, filtering and image transformations) and the inference stage of deep learning models. For example, operations like matrix multiplication within convolutional layers benefit greatly from vectorized execution [30]. Unlike scalar processing (Fig. 5), which handles data elements one at a time, vector registers in RVV enable simultaneous execution of multiple operations within a single CPU cycle. It allows you to load a bunch of values into a vector register and simultaneously perform operations on them.

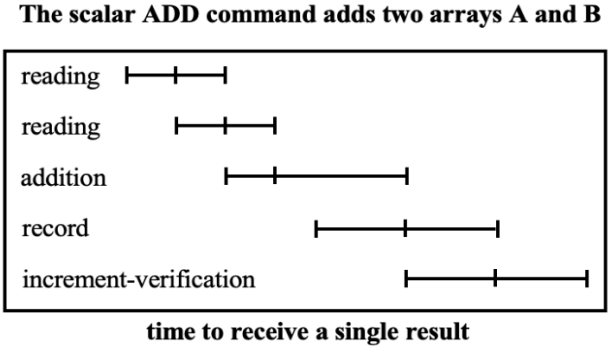

Fig. 5. Time diagram of vector addition of two arrays.

Since the analysis of the video processing pipeline from section IV showed that it is necessary to speed up the work of not only the inference model, but also postprocessing and preprocessing, since they also significantly load the system. To do this, we use vectorization of calculations. On Fig. 6

the color of rectangle represents the load factor of the module. Green color – is low level of load, yellow color – is medium load and red color – is the highest loaded modules, that requires a lot of hardware resources.

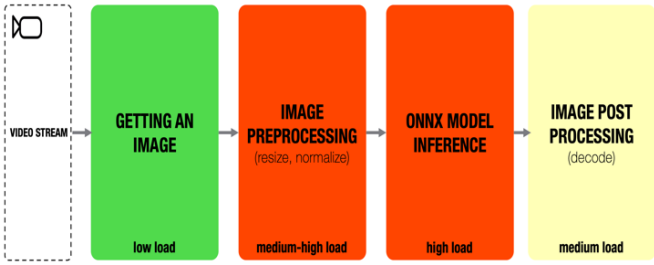

Fig. 6. Video processing pipeline.

To apply RVV in an experimental application, functions for preprocessing and postprocessing are written in C using an intrinsic. An important clarification is that the Lichee Pi 4A has the RVV 0.7.1 standard, which does not have auto-vectoring. Therefore, the RVV code is cross compiled into an executable file and functions are inserted into Python code using the Cpython library.

At the preprocessing stage (Fig. 7), image scaling, normalization and formatting operations are performed to feed into the neural network model. These steps include:

- resizing the image (cv2.resize);
- normalization of pixel values to the range [0.0, 0.1];
- channel rearrangement (CHW).

Using RVV allows to vectorize channel normalization and transformation operations. Instead of sequentially processing each pixel, RVV loads a vector of pixels and applies division and transpose operations in parallel.

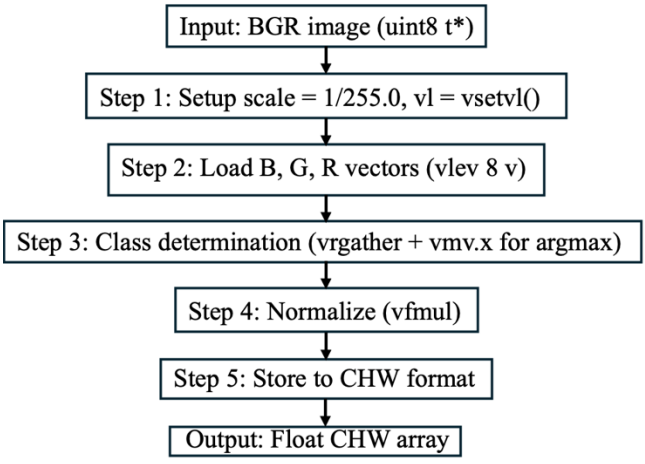

Fig. 7. Block diagram of the preprocessing function.

Postprocessing includes (Fig. 8) processing the output tensor of the model: threshold filtering, coordinate recalculation and preparation on the final list of objects. RVV is used to vectorize the following operations:

- extracting and converting coordinates of boxes (x1, x2y, x2, y2);
- calculation of confidence (np.max);
- finding the class (np.argmax);

- filtering by threshold;
- converting coordinates to pixels.

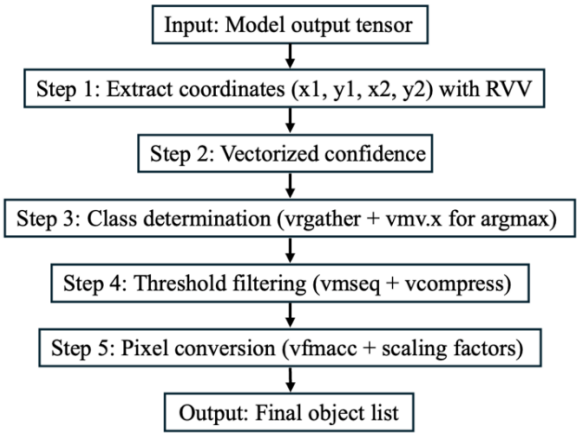

Fig. 8. Block diagram of the postprocessing function.

The following RVV intrinsics were used:

- vsetvl_e32m4 – sets the length of the vector for operations with 32-bit elements using 4 registers (m4). Automatically determining the maximum possible length for the remaining data.
- vsle_v_u8m1 – page loading of 8-bit unsigned integers in 3-byre increments. Allows you to load color components from an alternating format.
- vwaddu_vx_u16m2/vwaddu_vx_u32m4 – is an unsigned bit depth extension with a zero extension. What you need to convert to float.
- vfcvt_f_xu_v_f_32m4 – conversion of unsigned 32-bit integers to float32. It is necessary to maintain accuracy during normalization.
- vfmul_vf_32m4 – vector multiplication for normalization. Multiplies each element of the vector by 1/255 in one operation.
- vse_v_f32m4 – batch saving of 32-bit float values. This is necessary for the correct location of the data in the CHW format.

The postprocessing function has been optimized in a similar way.

By applying RVV instruction image processing, it was possible to achieve some speed improvements (Table 3). Thus, RVV becomes an excellent optimization tool both for processing the data preparation stages before launching the neural network and for subsequent processing of the results. Moreover, compared OpenCV vector methods the speed increases by about 2 times after using RVV. And by an order of magnitude compared to scalar methods.

Table 3. RVV application results.

	Time using OpenCV scalar functions (sec)	**Time using OpenCV vector functions (sec)**	**Out optimization option (sec)**
Preprocessing	45.4798	0.0437	0.0222
Postprocessing	0.0093	0.0026	0.0005

6.2. Optimizing Inference Using NPU

The Neural Processing Unit (NPU) is designed for operations related to neural networks. Using the NPU can significantly speed up inference by offloading computations from the CPU and running them in a dedicated hardware block, which is especially useful for real-time processing of large datasets [31].

However, not all models automatically support hardware acceleration, requiring adaptation. Optimization includes replacing unsupported operations with equivalent ones that work efficiently on the NPU and using quantized models to reduce computational load [32].

To use the NPU, follow these steps:

- environment preparation;
- converting the trained model to onnx format;
- quantification of the model in INT8 format, using special HHB tool;
- cross-compilation of the model into an executable program on the CPU/NPU.

This method efficiently processes images, leveraging parallel computing to accelerate embedding extraction.

Using multithreading to distribute computational tasks across CPU cores can improve image processing performance [33]. For example, separate threads can handle preprocessing and inference, allowing them to run un parallel. After using the NPU, good improvements were obtained (Table 4). The launches were carried out on the CPU and NPU.

Table 4. Comparison of data processing time on a neuroprocessor and a central processing unit.

Device	Time using (sec)
NPU	0.063
CPU	67

As one can see from the results, running the model without using an NPU has no practical application, since the execution speed will be too low. Using an NPU significantly speeds up execution.

6. Conclusion

The highest computational load in image processing in our experimental stand comes from the inference and image processing. Therefore, these should be rewritten in C using RVV instructions and, if possible, the NPU accelerator. Using optimized libraries and multithreading can also significantly improve performance.

After applying the NPU, we got quite good improvements (Table 5). For comparison, another YOLOv5n model was used, which is optimized for our NPU. The launches were carried out on the CPU and NPU.

Table 5. Comparing the performance and accuracy of YOLO models on different computing devices.

Model	Device	FPS	Accuracy
YOLOv8n	NPU	5-10	0.967
YOLOv8n	CPU	<1	0.993
YOLOv5n	NPU	>30	0.962
YOLOv5n	CPU	<1	0.991

As one can see from the lest results, the speed increased significantly. Running the model without a NPU does not make sense, since the processing speed will be too low for practical use. But these results do not give us an accurate understanding of whether this board can be used for industrial

applications, since all the steps taken have a lot of pitfalls. However, after solving the problems associated with the development of software tools, it will give a better understanding.

References

[1]. Cui E., Li T. Wei Q. RISC-V instruction set architecture extensions. A survey. IEEE Access 11, 2023, 24696–24711. DOI: 10.1109/ACCESS.2023.3246491.

[2]. Shen Y. Computer Vision: Technologies and Applications. Applied and Computational Engineering, vol. 163, no. 1, pp. 35–41, Jun. 2025, DOI: 10.54254/2755-2721/2025.23817.

[3]. Ali W. Exploring Instruction Set Architectural Variations: x86, ARM, and RISC-V in Compute-Intensive Applications, Aug. 2023, DOI: 10.36227/techrxiv.24026736.

[4]. Han C., Chang C., Srivastava S., Lu Y. Scalable Complex Event Processing on Video Streams. Proc. ACM on Management of Data, vol. 3, no. 3, pp. 1–29, Jun. 2025, DOI: 10.1145/3725419.

[5]. Borysenko V., Borysenko T. Modern approaches of design software applications based on microservice architecture in computer and information systems and technologies. Apr. 2020, DOI: 10.30837/IVcsitic2020201441.

[6]. Bhatnagar S., Mahant R. Designing Microservices in The Art of Decoding Microservices: An In-Depth Exploration of Modern Software Architecture. Launch IT, 2025, pp. 135-192.

[7]. Domenech-Asensi G., Garrigos J., Lopez P., Brea V., Cabello D. Real time architectures for the Scale Invariant Feature Transform algorithm. CNNA 2016; 15th International Workshop on Cellular Nanoscale Networks and their Applications. Dresden, Germany, 2016, pp. 1-2.

[8]. Obukhov A., Dedov D., Volkov A., Rybachok M. Technology for Improving the Accuracy of Predicting the Position and Speed of Human Movement Based on Machine Learning Models. Technologies, vol. 13, no. 3, p. 101, Mar. 2025, DOI: 10.3390/technologies13030101.

[9]. Qin X., Liu X., Han J. A CNN Hardware Accelerator Designed for YOLO Algorithm Based on RISC-V SoC. Proc. IEEE Int. Conf. ASIC, Kunming, China, 2021, pp. 1-4, DOI: 10.1109/ASICON52560.2021.9620500.

[10]. Srivastava S. K, Srivastava A. K., Allam S., Lilaramani D. Comparative analysis on Deep Convolution Neural Network models using Pytorch and OpenCV DNN frameworks for identifying optimum fruit detection solution on RISC-V architecture. IEEE Mysore Sub Section International Conference (MysuruCon), Hassan, India, 2021, pp. 738-743, DOI: 10.1109/MysuruCon52639.2021.9641594.

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

[12]. Yu M.-S., Chang H.-C., Wang C.-T., Tien Y.-W. Optimizing computer vision algorithms with TVM on VLIW architecture based on RVV. The Journal of Supercomputing, vol. 81, no. 1, Nov. 2024, DOI: 10.1007/s11227-024-06530-x.

[13]. Jajal P, Jiang W, Tewari A, Kocinare E, Woo J, Sarraf A. Interoperability in deep learning: a user survey and failure analysis of ONNX model converters. In: Proc. 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis. New York: ACM; 2024. p. 1466–1478, DOI: 10.1145/3650212.3680374.

[14]. Fusaomi N., Shingo S., Ryoma A., Keigo W., Maki K. H. Evaluation of Interoperability of CNN Models between MATLAB and Python Environments Using ONNX Runtime Model. AI, Computer Science and Robotics Technology 3(1), 1–13. 2024, DOI: 10.5772/acrt.20240043.

[15]. Sohan M., Ram T. S., Ch V. R. R. A Review on YOLOv8 and Its Advancements. Data Intelligence and Cognitive Informatics, Jan. 2024, pp. 529–545, DOI: 10.1007/978-981-99-7962-2_39.

[16]. Almeyda S., Davila A.: Process Improvement in Software Requirements Engineering: A Systematic Mapping Study. Programming and Computer Software, 48, Aug. 2022, pp. 513–533. DOI: 10.1134/S0361768822080084.

[17]. Lunev D., Poletykin S., Kudryavtsev D. Brain-computer interfaces: Technology overview and modern solutions. Modern Innovations, Systems and Technologies, vol. 2, no. 3, Jul. 2022, pp. 0117-0126, DOI: 10.47813/2782-2818-2022-2-3-01170126.

[18]. Tsekhmystro R., Rubel O., Prysiazhniuk O., Lukin V. V. Impact of distortions in UAV images on quality and accuracy of object localization. radioelectronic and computer systems, Jan. 2025, DOI: 10.32620/reks.2024.4.05.

[19]. Fischer L., Wollstadt P. Precision and Recall Reject Curves for Classification. Aug. 2023, DOI: 10.48550/arXiv.2308.08381.

[20]. Boyd K., Eng K. H., Page C. D. Area under the Precision-Recall Curve: Point Estimates and Confidence Intervals. Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Lecture Notes in Computer Science, vol. 8190, pp. 451–466, Sep. 2013, DOI: 10.1007/978-3-642-40994-3_29.

[21]. Sinha E., Kumar A., Tyagi A. OpenCV for Computer Vision Applications. International Journal For Multidisciplinary Research, vol. 7, no. 3, May 2025, DOI: 10.36948/ijfmr.2025.v07i03.44280.

[22]. Chinnaraju A. Benchmarking cross-platform AI: Web Assembly, ONNX Runtime and TVM for Real-Time Web, Mobile, and IoT Deployment. World Journal of Advanced Research and Reviews, vol. 26, no. 2, pp. 1937–1963, May 2025, DOI: 10.30574/wjarr.2025.26.2.1832.

[23]. Yang S., Lu T. T3 SOC design flow case study: Design a video processing pipeline. ASIC, ASICON '07. 7th International Conference, Nov. 2007, DOI: 10.1109/ICASIC.2007.4415551.

[24]. Jindal K. Design and Implementation of an Embedded Image Processing System on Zynq ZedBoard: A VLSI Perspective. International Journal for Research in Applied Science and Engineering Technology, vol. 13, no. 5, pp. 5141–5145, May 2025, DOI: 10.22214/ijraset.2025.71372.

[25]. Smirnov E., Timoshenko D., Andrianov S. Comparison of Regularization Methods for ImageNet Classification with Deep Convolutional Neural Networks. AASRI Procedia, vol. 6, pp. 89–94, Dec. 2014, DOI: 10.1016/j.aasri.2014.05.013.

[26]. Pujari S. D., Pawar M. M., Wadekar M. Multi-Classification of Breast Histopathological Image Using Xception: Deep Learning with Depthwise Separable Convolutions Model. Techno-Societal, pp. 539–546, May 2021, DOI: 10.1007/978-3-030-69921-5_54.

[27]. Wang S., Wang X., Xu Z., Chen B. Optimizing CNN Computation Using RISC-V Custom Instruction Sets for Edge Platforms. IEEE Trans. Comput, May 2024, pp. 1-14, DOI: 10.1109/TC.2024.3362060.

[28]. Titopoulos V., Alexakis G., Nicopoulos C., Dimitrakopoulos G. Efficient Implementation of RISC-V Vector Permutation Instructions. *arXiv:2505.07112*, May 2025, DOI: 10.48550/arXiv.2505.07112.

[29]. Yuan T., Liu W., Han J., Lombardi F. High Performance CNN Accelerators Based on Hardware and Algorithm Co-Optimization. IEEE Trans. Circuits Syst. I, Reg. Papers, Oct. 2020, pp. 1-14, DOI: 10.1109/TCSI.2020.3030663.

[30]. Jin S., Qi S., Dai Y., Hu Y. Design of Convolutional Neural Network Accelerator Based on RISC-V. Proc. 10th Int. Conf. Appl. Tech. Cyber Intell. (ICATCI 2022), 2023, pp. 446-454. DOI: 10.1007/978-3-031-29097-8_53.

[31]. Cono D'Elia D., Demetrescu C. Ball-Larus Path Profiling across Multiple Loop Iterations. SIGPLAN Not. 48, 10 (oct 2013), pp. 373-390, DOI :10.1145/2544173.2509521.

[32]. Agarwal R., Deshmukh R., Borhade P., Murarka S. Image Classification using Parallel CPU and GPU Computing. Int. J. Eng. Adv. Technol., vol. 9, no. 4, Apr. 2020, pp. 5, DOI: 10.35940/ijeat.D7870.049420.

[33]. Shanthi M., Anthony Irudhayaraj A. Multithreading - An Efficient Technique for Enhancing Application Performance. International Journal of Recent Trends in Engineering, Vol 2, No. 4, Nov. 2009, pp. 165-167, DOI: 10.22146/ijccs.57594.

Информация об авторах / Information about authors

Никита Иванович ЧЕРЕПАНОВ– студент магистратуры высшей школы программной инженерии Санкт-Петербургского политехнического университета Петра Великого. В 2025 получил квалификацию бакалавра в Санкт-Петербургском политехническом университете Петра Великого по специальности "Технология разработки и сопровождения качественного программного продукта". Сфера научных интересов: программные архитектуры, RISC-V, машинное обучение, компьютерное зрение, искусственный интеллект.

Nikita Ivanovich CHEREPANOV is a master's student at the Higher School of Software Engineering at Peter the Great St. Petersburg Polytechnic University. In 2025, he got bachelor degree by graduating from Peter the Great St. Petersburg Polytechnic University with a specialty in "Technology for developing and maintaining a high-quality software product". Research interests: software architectures, RISC-V, machine learning, computer vision, artificial intelligence.

Надежда Олеговна СТЕПИНА – ассистент высшей школы программной инженерии Санкт-Петербургского политехнического университета Петра Великого. В 2023 году окончила Санкт-Петербургский государственный политехнический университет по специальности

«Программная инженерия». В 2024 году стала аспирантом по специальности 05.13.11 – «Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей». Область научных интересов – разработка программного обеспечения, машинное обучение, высокопроизводительные вычисления, IoT и embedded-системы.

Nadegda Olegovna STEPINA is an assistant at the Higher School of Software Engineering at Peter the Great St. Petersburg Polytechnic University. In 2023, she graduated from the St. Petersburg State Polytechnic University with a degree in Software Engineering. In 2024, she became a postgraduate student in the field of Mathematical and Software Support for Computing Machines, Complexes, and Computer Networks. Her research interests include software development, machine learning, high-performance computing, IoT, and embedded systems.

Игорь Валерьевич НИКИФОРОВ – доцент высшей школы программной инженерии Санкт-Петербургского политехнического университета Петра Великого. В 2011 году окончил Санкт-Петербургский государственный политехнический университет по специальности «Программное обеспечение вычислительной техники и автоматизированных систем». В 2014 году защитил диссертацию на соискание ученой степени кандидата технических наук по специальности 05.13.11 – «Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей». Является автором 100 научных публикаций. Область научных интересов – разработка программного обеспечения, имитационное моделирование, аналитика больших данных, распределенные вычисления.

Igor Valerievich NIKIFOROV. In 2011, he graduated from St. Petersburg State Polytechnic University with a degree in «Computer Science and Automated Systems Software». He got his Cand. Sci. (Tech.) degree in Mathematical and software support for computers, complexes and computer networks in 2014. He is an Associate Professor at the Higher School of Software Engineering at Peter the Great St. Petersburg Polytechnic University. He is the author of more than 100 scientific publications. Research interests – software engineering, simulation modeling, big data analytics, distributed computing.