ISSN 2075-8545 (online)

Nanotechnologies in construction
Нанотехнологии в строительстве

2026; 18 (5):
673–680

Nanobuild.ru

SYSTEM SOLUTIONS FOR TECHNOLOGICAL PROBLEMS

CC BY 4.0

Original article
https://doi.org/10.15828/2075-8545-2026-18-5-673-680

Analysis of the OWLv2 model usage for real-time object detection

Konstantin Y. Ustyuzhanin, Fairuza T. Ziganshina*, Regina R. Sadykova
Ufa State Petroleum Technological University, Ufa, Kosmonavtov str. 1, 450064, Russian Federation
* Corresponding author: e-mail: fairusa85@mail.ru

ABSTRACT

The implementation of an AI-enabled video processing system for industry often faces a trade-off between accuracy, speed, cost, and safety. The authors present a plan and scheme of operation of an industrial AI system for searching objects in images in real time using only local budget solutions from the user segment. **Introduction.** This research describes the structure of an industrial scalable multi-camera system for detecting objects in a production environment based on text descriptors. The system is based on the OWLv2 (Open-World Localization Vision Transformer) model: a visual transformer for localizing objects in open space. **Methods and materials.** A video stream with various objects is used as objects, which must then be recognized and identified. The use of the open OWLv2 model and Nvidia's user-level GPU together with the microservices architecture allows you to create flexible software that can be scaled as the needs of the enterprise using a single local service that is completely isolated from the outside world. Such a service, among other things, will allow you to further implement advanced versions of models that solve complex problems, such as quality control of composites, without the need to significantly rebuild the service, while maintaining a single API. The system is designed to process real-time video streams and single images from multiple cameras in real time using graphics processing unit (GPU) acceleration, distributed task processing on Celery and RabbitMQ, centralized observability with Prometheus, Grafana and Loki. **Results and discussion.** The architecture prioritizes operational resilience, cost efficiency, and the ability to scale from two to a hundred cameras without a fundamental redesign. By adopting a decentralized, microservices approach from the outset, the project avoids the technical debt and refactoring burden that would arise from building a monolithic system and trying to distribute it later. Using MinIO, RabbitMQ, Prometheus, Loki, and Celery is a good choice for an industrial solution.

KEYWORDS: Artificial intelligence, OWLv2 model, object detection, system architecture, system requirements, Kubernetes system

SOURCES OF FUNDING FOR THE SCIENTIFIC WORK THAT RESULTED IN THE PUBLICATION: The study was carried out within the framework of the program of the Ministry of Science and Higher Education of the Russian Federation "PRIORITY 2030".

ACKNOWLEDGEMENTS: The authors express their gratitude for the financial support of the project.

FOR CITATION:
Ustyuzhanin K.Yu., Ziganshina F.T., Sadykova R.R. Analysis of the OWLv2 model usage for real-time object detection. *Nanotechnologies in Construction.* 2026;18(5):673–680. https://doi.org/10.15828/2075-8545-2026-18-5-673-680. – EDN: AXXVVG.

© Ustyuzhanin K.Yu., Ziganshina F.T., Sadykova R.R., 2026

ISSN 2075-8545 (online)

Nanotechnologies in construction
Нанотехнологии в строительстве

2026; 18 (5):
673–680

Nanobuild.ru

SYSTEM SOLUTIONS FOR TECHNOLOGICAL PROBLEMS

Анализ использования модели OWLv2 для обнаружения объектов в режиме реального времени

Константин Юрьевич Устюжанин, Файруза Тахваловна Зиганшина*, Регина Римовна Садыкова

Уфимский государственный нефтяной технический университет, 450064, Уфа, улица Космонавтов, 1, Российская Федерация

* Автор, ответственный за переписку: e-mail: fairusa85@mail.ru

АННОТАЦИЯ

Внедрение системы обработки видеоинформации с помощью ИИ для промышленности часто сталкивается с компромиссом между точностью, скоростью, дороговизной и безопасностью. Авторы представляют план и схему работы промышленной ИИ-системы поиска объектов на изображениях в реальном времени с использованием только локальных бюджетных решений из пользовательского сегмента. **Введение.** Работа описывает структуру промышленной масштабируемой многокамерной системы детектирования объектов в производственном окружении на основе текстовых дескрипторов. В основе системы лежит модель OWLv2 (англ. Open-World Localization Vision Transformer): визуальный трансформер локализации объектов в открытом пространстве. **Методы и материалы.** В качестве объектов используется видеопоток с различными объектами, которые в дальнейшем должны быть распознаны и определены. Использование открытой модели OWLv2 и ГП Nvidia пользовательского уровня вместе с архитектурой микросервисов позволяет создать гибкое ПО, которое можно масштабировать по мере нужд предприятия, используя единый локальный сервис, полностью изолированный от внешнего мира. Такой сервис, в том числе, позволит в дальнейшем внедрять продвинутые версии моделей, решающих сложные задачи, такие как контроль за качеством композитов без необходимости существенной перестройки сервиса, при этом сохраняя единый API. Система разработана для обработки видеопотока и единичных изображений в реальном времени со множества камер в реальном времени с использованием ускорения на графическом процессоре (ГП), распределенной обработкой задач на Celery и RabbitMQ, централизованной наблюдаемостью с Prometheus, Grafana и Loki. **Результаты и обсуждения.** Архитектура ставит приоритетом операционную устойчивость, эффективность по стоимости и возможность масштабирования с двух до сотни камер без фундаментального редизайна. Применяя с самого начала децентрализованный, микросервисный подход, проект избегает технического долга и бремени рефакторинга, которые возникли бы при создании монолитной системы и попытке ее последующего распределения. Гибкая облачная система на базе Kubernetes, где каждый компонент независим, масштабируется по потребности и легко поддерживается. Использование MinIO, RabbitMQ, Prometheus, Loki и Celery – хороший выбор для промышленного решения.

КЛЮЧЕВЫЕ СЛОВА: Искусственный интеллект, модель OWLv2, детектирование объектов, архитектура системы, системные требования, система Kubernetes

ИСТОЧНИКИ ФИНАНСИРОВАНИЯ НАУЧНОЙ РАБОТЫ, РЕЗУЛЬТАТОМ КОТОРОЙ СТАЛА ПУБЛИКАЦИЯ: Исследование выполнено в рамках программы Министерства науки и высшего образования Российской Федерации «ПРИОРИТЕТ 2030».

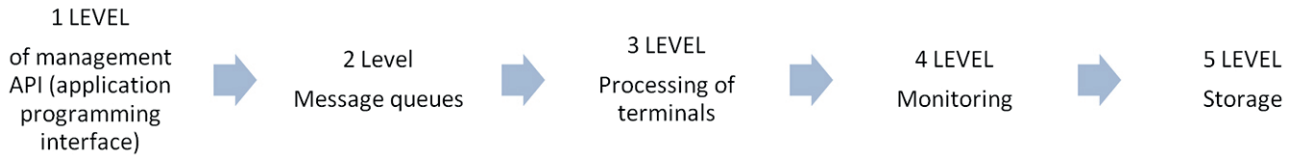
БЛАГОДАРНОСТИ: Авторы выражают благодарность за финансовую поддержку проекта.

ДЛЯ ЦИТИРОВАНИЯ:

Устюжанин К.Ю., Зиганшина Ф.Т., Садыкова Р.Р. Анализ использования модели OWLv2 для обнаружения объектов в режиме реального времени. *Нанотехнологии в строительстве.* 2026;18(5):673–680. https://doi.org/10.15828/2075-8545-2026-18-5-673-680. – EDN: AXXVVG.

INTRODUCTION

This paper offers an overview of the system and the justification of design solutions for the creation of a machine vision module [1]. The proposed system operates on the principle of separation of duties, where each component performs a separate task within the output pipeline. The architecture consists of five main layers:



ISSN 2075-8545 (online)

This approach reduces latency and does not lead to system-wide errors.

When implementing artificial intelligence (AI) in production, two related problems arise: the cost of implementing a unit of the system in a separate production area and the limitation of horizontal scaling, which in turn requires the implementation of an expensive AI system in each area separately. The increased cost is primarily due to the limited software licenses and the high cost of graphics accelerators when implementing isolated AI. In many industries, security standards do not allow the movement of company data outside its network, which makes it impossible to use third-party computing services and rented facilities. Production AI should be located inside the network and be isolated from the outside world, especially when its work is related to the management and support of production robots and monitoring the state of production areas. In this paper, the authors propose an architecture for a scalable computer vision system based on microservices and Kubernetes orchestration, the cost of which will grow linearly, not exponentially, and which will allow for the implementation of updates and improvements without stopping work.

METHODS AND MATERIALS

This paper describes the integration of several technologies and tools for processing video streams from multiple cameras. Such technologies are: Kubernetes, Docker, RabbitMQ, Redis, Prometheus, Grafana, Loki, FastAPI, and ONNX Runtime [2–6].

As a basis, the principle of dividing the task into subtasks is used, which create their environment for the convenience of storing and transferring data, that is, the principle of microservices architecture. This technology is achieved using Kubernetes and Docker. To control the operation of the system, collect and analyze, the following tools are used: Prometheus, Grafana and Loki, and for message transfer and data storage in RAM, RabbitMQ and Redis are used. This principle of operation can be presented in the form of a diagram shown in Figure 1.

In this case, the AI is launched as a Celery job service based on the created job in RabbitMQ with access to data via links to saved objects in Redis. The data itself is stored in MinIO. FastAPI serves as a layer for managing system flows, creating and managing work status, and providing access to information for external enterprise services. The video stream from the cameras goes directly to MinIO, the AI works independently, on demand of the task, and consumes resources only during inference directly, when the AI was created for image processing. The AI itself can be any model that can be run on the selected hardware, but the authors suggest using OWLv2 as the basis for solving most computer vision production problems based on a text prompt. At the same time, OWLv2 with an accuracy of FP16 takes up only 0.35–0.4 Gb along with weights, activations, and overhead (360 MB of scales, 155 million parameters).

RESULTS AND DISCUSSION

Below are the justifications for choosing a particular system and technology.

CPU and GPU Acceleration: Performance and Cost Analysis. The inclusion of the NVIDIA RTX 5060 Ti graphics card is a critical decision in system design.

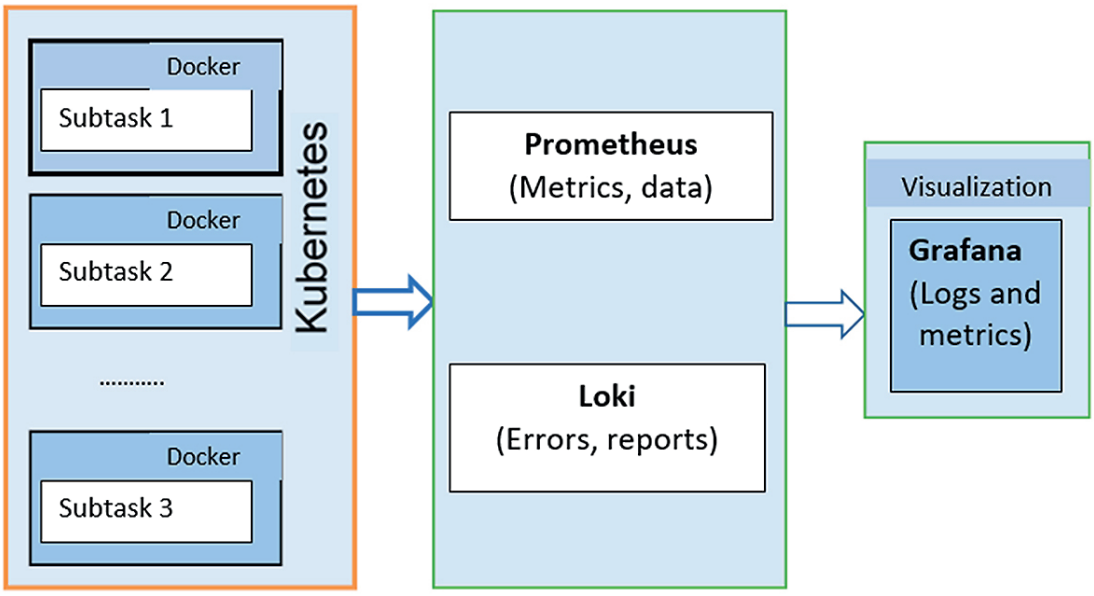


Fig. 1. Approximate scheme of video stream processing

ISSN 2075-8545 (online)

OWLv2 is a graphics data conversion program with approximately 1.5 billion parameters [9]. Table 1 below shows the characteristics of a single stage of data processing using the CPU and the RTX 5060 Ti [10].

The analysis points to the advantage of GPU acceleration [11, 12]. A CPU-only system would require eight to ten high-performance cores to achieve comparable throughput, which means a larger, more power-hungry server at a significantly higher cost [12].

Thus, the use of a GPU, despite the cost, allows you to reduce operating and capital costs. At the same time, it becomes possible to move to real-time data processing with high resolution, which will increase the accuracy of detection [13]. The main limitation of the RTX 5060 Ti is 4Ti is 8 gigabytes of video memory, which limits the packet size, image resolution, and the number of streams from cameras that can be processed simultaneously.

To solve this problem, you can delegate control of Kubernetes resources [14], allocating them explicitly to the pods responsible for running the model. If you restrict access to the GPU only to inference pods, and run APIs, databases, and support services pods strictly on CPU resources, then the limited GPU resources will be spent strictly on the most difficult and costly task of the system.

If a monolithic Docker container [15] with a single FastAPI application and ONNX Runtime [16] were used, then when moving from 1 camera to 2–3, it would be necessary to scale the entire instance, which would require a separate video processor for each additional instance, each camera [17]. The microservice architecture allows you to move all resource-intensive tasks into a separate container, which will run the model and take up GPU resources alone, but which will be able, firstly, to process many camera streams simultaneously, and secondly, will be connected to the rest of the system only through the API. The advantages of this approach are emphasized in Newman's book "Creating Microservices". The use of message queues will allow you to receive video streams from many cameras and process them using limited resources. Table 2 shows a comparison of the two main tools for organizing queues: RabbitMQ and Redis.

If you need to provide horizontal scaling as in the described system, then RabbitMQ is the best choice. It will allow, among other things, to create queues with different priorities and tasks for different cameras, which can be important for large production with many sites. It will work as a StatefulSet [26] to ensure a constant state during reboots. In the future, this choice will allow you to expand the system to several models running on separate GPUs performing different tasks within a single service. This approach allows you to provide resource management at the nanotechnology level and solve priority problems and flexibly redistribute the load.

At the same time, it would be rational to provide centralized storage of camera frames, work results, and system artifacts in the MinIO object storage [18], which is compatible with S3 [19]. Direct transfer of large objects through the internal network will introduce undue delay and increase the load on limited system resources. MinIO can work locally even in Docker Compose [20] and provide a compatible S3 API [21], which makes it possible to unify the protocols for storing and retrieving information in the system, linking all services to a single local storage of large objects. It is very popular in the

Table 1. Characteristics of the data processing stage using a CPU and an RTX 5060 Ti

Characteristics	CPU Processing Stage (720p)	RTX 5060 Ti (c 8GB VRAM)
Frame rate (frames per second)	5–10 frames	20–30 frames
Impact on system requirements	Real-time limitation	Real-time application support
Example of human tracking (at 5 frames per second)	Difficult to track at 5 frames	125 Frames – Smooth Motion and Analysis

Table 2. A comparison of two tools: RabbitMQ and Redis

Criterion	RabbitMQ	Redis
Ease of Use	More difficult to set up and operate	Easy to use, minimal setup
Performance	Suitable for complex scenarios, high throughput	Very high, especially for basic operations
Message routing support	Advanced Routing, Priority Queues	Limited routing options
Handling dead-letter messages	There are mechanisms for handling errors and retries	No built-in error handling mechanisms
Priority queue support	Yes	No
Suitable for multidimensional systems	Better, thanks to complex processing and prioritization	Fewer opportunities for complex scenarios

ISSN 2075-8545 (online)

Nanotechnologies in construction
Нанотехнологии в строительстве

2026; 18 (5):
673–680

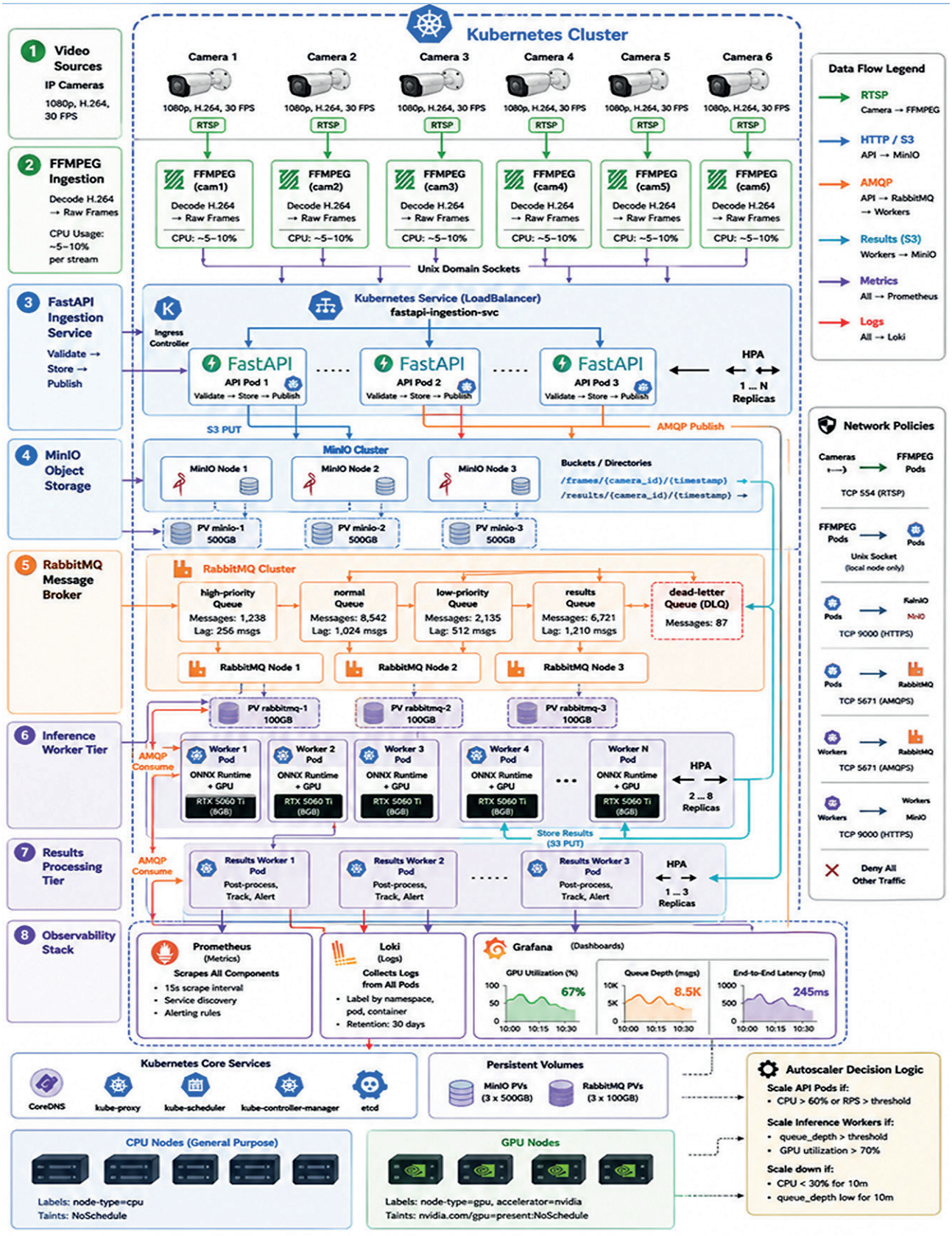


Fig. 2. Architecture of the industrial computer vision system

ISSN 2075-8545 (online)

DevOps community [22] due to its ease of operation. In this scheme, the stream from cameras will be directly stored in MinIO, and access to them will be strictly via the frame URI in RabbitMQ [23]. Similarly, the results will go directly to the storage without being transferred between other components of the system. Even in the event of a storage failure, the system will be able to keep messages in the queue until access to MinIO is restored. It is advantageous to run MinIO itself as a StatefulSet with persistent volumes to preserve data in the event of a crash [27].

To ensure control over everything that happens in the system, it is also necessary to add centralized services for collecting statistics, monitoring and collecting logs. Prometheus [28] can centrally collect metrics from the entire system at regular intervals, and Loki will take care of centralized log collection. Their display can be given to the Grafana dashboard. In the future, Jaeger traces can be added [24]. Unlike traditional logging systems [25], with Loki you can index only median characteristics and access all other data through filtering and queries.

The system architecture is shown in Figure 2. The cameras transmit RTSP video in H.264 format to the FFMPEG decoder, after which they are stored in the MinIO repository. FastAPI automatically publishes messages to RabbitMQ for the model with a hint that corresponds to the current task or the objects being searched for for each of the cameras. Under the inference, it loads images and hints into the OWLv2 model and stores the result in MinIO. It runs on Celery with sessions on NVIDIA Multi-Process Service. The results are post-processed in a separate pod independent of the model, after which they are ready to be displayed in the system's GUI or transferred.

This architecture is capable of scaling linearly from two cameras to a hundred with minimal investment. When adding a new camera, you only need to update the camera configuration in the API, after which the stream will be automatically added to the data pipeline without further involvement of the API itself. Additional workflows will be created automatically if the current ones are not enough. If the GPU itself becomes a bottleneck, then it is enough to add another GPU to the cluster without changing the configurations of existing threads or reconfiguring the system.

The OWLv2 model can find any objects in the image with a high level of confidence from a text prompt, making it the best choice for the implementation of the computing core of an industrial vision system, where the final object of interest is unknown, can change, or has a complex, variable structure, poor lighting, and complex, volatile backgrounds. When the model is exported to ONNX format using standard pytorch, with optimum optimizing for FP16 floating-point accuracy, the use becomes cost-effective even in the absence of a message queue and a direct video stream over the Docker network.

NVIDIA GTX 4060 8Gb VRAM Laptop GPU, 32GB RAM, Intel Core i9-13900H graphics card used in the test. Figures 3–7 show the results of testing the model on a set of objects.

Search for "tool", "hose", "glasses" (Figs. 3–5) individually at a confidence level of at least 0.2, frame-by-frame script, stationary objects are shown in Figures 3–5.

Figure 6 shows the search result for "powerbank" at a confidence level of at least 0.2 for a frame-by-frame script, an example of a classification error, this is clearly not "powerbank", a static batch. Figure 7 shows the search result for "metal dish, tool, hose, glasses" at a confidence level of at least 0.2 for a thread-based ensemble script, dynamic batch. You can see that the model is basically able to adapt to the recognition of very different objects in noise conditions. Even on miniature and low-power hardware, the OWLv2 model showed an FPS of 5–30 frames with direct continuous image streaming. With the use of OpenCV trackers and periodic updates, the simplified version of the service consistently produces 60 FPS on a single camera.

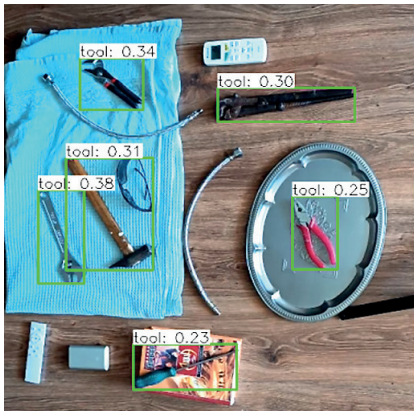


Fig. 3. Search for "tool"

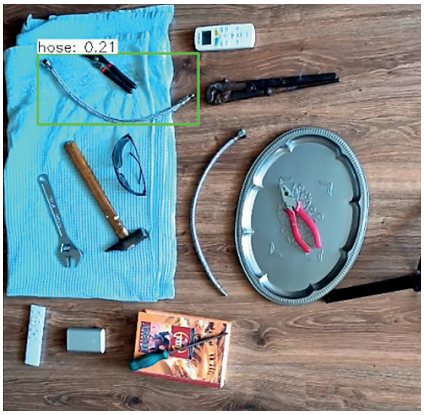


Fig. 4. Search for "hose"

Fig. 5. Search for "glasses"

ISSN 2075-8545 (online)

Fig. 6. Search for "powerbank" (classification error)

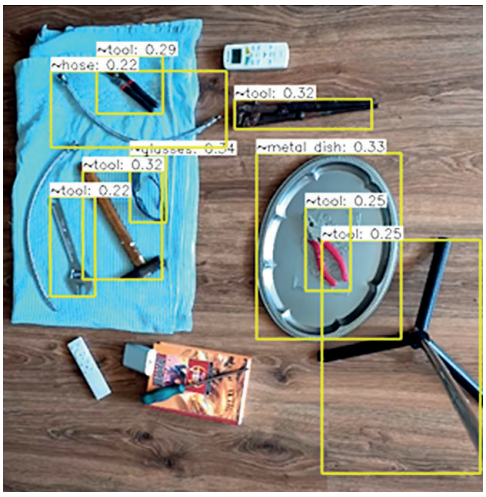


Fig. 7. Search for «metal dish, tool, hose, glasses»

CONCLUSION

Finding a sustainable and scalable solution for computer vision comes down to the task of moving from the exponential growth of the cost of horizontal scaling to linear scaling. The use of modern AI requires the use of graphics accelerators and the division of the task between loosely coupled services within a single architecture.

Moving on to the real applications of the described AI system, its application is not field-specific and is not limited to one type of work. If necessary, the system can monitor the movement of personnel, foreign objects and the position of workpieces in the working area of the manipulator, or even detect defects in composite materials, including those reinforced with nanofills, where it is necessary to quickly detect defects or foreign objects to supplement the standard OWLv2 model with an extended context, or implement a satellite model to solve this specific task without significant changes in the system and linear cost increase.

REFERENCES

1. *Amazon Simple Storage Service Documentation*. Amazon S3; 2026. URL: https://docs.aws.amazon.com/s3/ (accessed: 04.05.2026).
2. Burns K., Beda K. *Kubernetes Up and Running: Dive into the Future of Infrastructure*. 2nd ed. O'Reilly Media; 2019
3. *Celery – Distributed Task Queue*; 2023. URL: https://docs.celeryq.dev/ (accessed: 04.05.2026).
4. DevOps. Humble J., Farley D. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Boston: Addison-Wesley Professional; 2010.
5. Docker Compose. *Docker Compose Documentation*; 2026. (Accessed: 04.05.2026). URL: https://docs.docker.com/compose/ (accessed: 04.05.2026)
6. *Docker Documentation*; 2026. URL: https://docs.docker.com/ (accessed: 04.05.2026)
7. *FastAPI Documentation*; 2026. URL: https://fastapi.tiangolo.com/ (accessed: 04.05.2026)
8. FFmpeg Documentation; 2024. *FFmpeg Protocols*. URL: https://ffmpeg.org/ffmpeg-protocols.html (accessed: 04.05.2026)
9. Grafana Labs. *Grafana Documentation*; 2026. URL: https://grafana.com/docs/ (accessed: 04.05.2026)
10. *Grafana Loki Documentation*; 2026. URL: https://grafana.com/docs/loki/latest/ (accessed: 04.05.2026)
11. Hänsel M. Real-Time Streaming Protocol (RTSP): Architecture and Design Considerations. *IEEE Communications Surveys & Tutorials*. 2020. Vol. 22, no. 3. P. 1234–1256. https://doi.org/10.1109/COMST.2020.2987258
12. Jaeger: open source, end-to-end distributed tracing; 2026. URL: https://www.jaegertracing.io/docs/ (accessed: 04.05.2026)
13. Kubernetes Documentation; 2026. URL: https://kubernetes.io/docs/home/ (accessed: 04.05.2026)Load balancer. Kubernetes. *Service – Type LoadBalancer*; 2026. URL: https://kubernetes.io/docs/concepts/servicesnetworking/service/#loadbalancer (accessed: 04.05.2026)
14. MinIO Documentation; 2026. URL: https://min.io/docs/minio/kubernetes/upstream/index.html (accessed: 04.05.2026)
15. Newman S. *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media; 2015.

ISSN 2075-8545 (online)

16. NVIDIA Corporation; 2023. *CUDA C++ Programming Guide*. URL: https://docs.nvidia.com/cuda/cuda-c-programming-guide/ (accessed: 04.05.2026)
17. *ONNX Runtime Documentation*; 2026. URL: https://onnxruntime.ai/docs/ (accessed: 04.05.2026)
18. OWLv2 (Open-World Localization Vision Transformer). Minderer M., Gritsenko A., Stone A. et al. Scaling Open-Vocabulary Object Detection. *Advances in Neural Information Processing Systems (NeurIPS 2023)*. 2023;36:72983–73007. URL: https://arxiv.org (accessed: 04.05.2026). https://doi.org/10.5555/3666122.3669313
19. *Prometheus – Monitoring system & time series database*; 2026. URL: https://prometheus.io/docs/ (accessed: 04.05.2026)
20. *RabbitMQ Documentation;* 2026. URL: https://www.rabbitmq.com/docs (accessed: 04.05.2026)
21. RabbitMQ StatefulSet. Kubernetes. *StatefulSets*; 2026. URL: https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/ (accessed: 04.05.2026)
22. *RabbitMQ Documentation: Message Queues and Priority Queues*; 2024. URL: https://www.rabbitmq.com/priority.html (accessed: 04.05.2026)
23. Tannenbaum A. S., Van Steen M. *Distributed Systems: Principles and Paradigms*. 2nd ed. Prentice Hall; 2006.
24. WebRTC W3C Working Group; 2023. *WebRTC 1.0: Real-time Communication Between Browsers*. W3C Recommendation. URL: https://www.w3.org/TR/webrtc/ (accessed: 04.05.2026)
25. Architectural storage pattern. Kleppmann M. *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. Boston: O'Reilly Media; 2017.
26. Horizontal autoscaling by metrics Prometheus. Kubernetes. *Horizontal Pod Autoscaler*. URL: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/ (accessed: 04.05.2026).
27. *Prometheus Adapter for Kubernetes Metrics*. URL: https://github.com/kubernetes-sigs/prometheus-adapter (accessed: 04.05.2026)
28. Power consumption: 150 W; NVIDIA Corporation. *GeForce RTX 5060 Ti Graphics Card – Specifications*; 2026. URL: https://www.nvidia.com/en-us/geforce/graphics-cards/rtx-5060-ti/ (accessed: 04.05.2026).
29. 20–30 FPS performance on the RTX 5060 Ti. *ONNX Runtime Performance Tuning*. Microsoft. URL: https://onnxruntime.ai/docs/performance/ (accessed on 04.05.2026). Note: Exact measurements for this map are not available in open sources; the link is to the optimization documentation to extrapolate the expected performance.
30. Traditional log aggregation systems (ELK). *Elastic Stack Documentation*; 2026. URL: https://www.elastic.co/guide/index.html (accessed 04.05.2026).

ADDITIONAL INFORMATION

The authors state that generative artificial intelligence technologies and technologies based on artificial intelligence were not used in the preparation of the article.

INFORMATION ABOUT THE AUTHORS

Konstantin Yu. Ustyuzhanin – Senior Lecturer, Department of Gas Chemistry and Modeling of Chemical and Technological Processes, Ufa State Petroleum Technological University, Ufa, Kosmonavtov str. 1, 450064, Republic of Bashkortostan, Russian Federation, ustyuzhanin.ky@gmail.com, https://orcid.org/0000-0003-0256-155X

Fairuza T. Ziganshina – Cand. Sci. (Phys.-Math.), Associate Professor, Department of Integrated Engineering and Computer Graphics, Ufa State Petroleum Technological University, Ufa, Kosmonavtov str. 1, 450064, Republic of Bashkortostan, Russian Federation, fairusa85@mail.ru, https://orcid.org/0009-0004-8627-5714

Regina R. Sadykova – Lecturer, Department of Integrated Engineering and Computer Graphics, Ufa State Petroleum Technological University, Ufa, Kosmonavtov str. 1, 450064, Republic of Bashkortostan, Russian Federation, umbaru@mail.ru, https://orcid.org/0000-0003-2967-882X

CONTRIBUTION OF THE AUTHORS

Konstantin Yu. Ustyuzhanin – scientific supervision; development of the concept and methodology of the research, analysis of the research results, writing the initial text of the article, drawing up the final conclusions.

Fayruza T. Ziganshina – processing and analysis of experimental data using machine learning methods, graphical and tabular presentation of the research results, writing the initial text of the article, drawing up the final conclusions.

Regina R. Sadykova – literature review, conducting experimental work, collecting and systematizing experimental data.

The authors declare no conflicts of interest.

The article was submitted 23.06.2026; approved after reviewing 01.10.2026; accepted for publication 06.10.2026.