UDC 004.8, 004.031.6, 004.932, 629.734 **doi: 10.32620/reks.2026.1.06**

Igor SINITSYN[1], Anatoliy DOROSHENKO[1,2], Ivan KYRYLOV[1], Iaroslav OMELIANENKO[1], Valentyn SMIRNOV[1], Olena YATSENKO[1]

[1] ***Institute of Software Systems of the National Academy of Sciences of Ukraine, Kyiv, Ukraine***
[2] ***National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine***

INTELLIGENT TECHNOLOGIES FOR TARGET DETECTION AND ENGAGEMENT BY UAVS BASED ON ARTIFICIAL INTELLIGENCE AND SIMULATION DATA

*The **subject matter** of the article is the methodology for developing integrated intelligent systems that enable automated detection, classification, tracking, and engagement of targets by unmanned aerial vehicles (UAVs) using artificial intelligence, computer vision, and simulation modeling technologies. The **goal** is to design an architecture and experimental environment for evaluating the efficiency of deep learning models, particularly the YOLO family, in real-time operation across different hardware platforms, considering resource and power constraints, as well as to quantitatively assess improvements in key system characteristics. The **tasks** to be solved are: developing a microservice-based system architecture using Kubernetes and ROS2; creating specialized datasets for model training; integrating the AirSim, ArduPilot SITL, and Mission Planner simulators into a unified testing environment; and conducting comparative performance analysis of various YOLO versions on platforms ranging from low-power ARM/NPU to high-end GPU. The applied **methods** are simulation modeling, automated neural network training, model quantization, inference optimization using TensorRT, and statistical analysis of the obtained metrics. The following **results** were obtained. A unified experimental methodology and a hardware–software platform were developed, providing a complete research cycle—from data generation and annotation to model performance evaluation. Experimental tests demonstrated that YOLOv8n and YOLOv11n offer the best balance of accuracy and speed on low-resource platforms. The most effective onboard solutions were identified as the Google Coral Dev Board Micro and NVIDIA Jetson Orin Nano. A simulation testbed was deployed that successfully models UAV flight, target acquisition, and engagement processes. **Conclusions.** The results confirmed the feasibility of using simulation data and a microservice-based approach for developing autonomous intelligent UAV control systems, demonstrated measurable improvements in performance, computational efficiency, and energy consumption, as well as provided practical guidelines for selecting the optimal "model–platform" configuration. The scientific novelty of the obtained results lies in the creation of a comprehensive research environment that integrates modern artificial intelligence tools, simulation modeling, and hardware optimization into a unified reproducible structure, enabling objective evaluation of target detection and engagement algorithms in real time.*

Keywords: *artificial intelligence; automated target detection and engagement; computer vision; deep learning; neural networks; simulation modeling; unmanned aerial vehicles.*

1. Introduction

Modern combat operations demonstrate the rapid growth of the role of unmanned aerial vehicles (UAVs) in reconnaissance, detection, tracking, and engagement of targets. The effectiveness of such systems largely depends on their level of autonomy, ability to operate in conditions of electronic warfare, as well as on the accuracy of target recognition in real time. The use of artificial intelligence (AI) and computer vision (CV) technologies opens up new opportunities for increasing the effectiveness of unmanned platforms in these tasks [1, 2]. At the same time, the issue of providing a reproducible, safe, and scalable environment for testing such systems remains relevant. In view of this, the role of simulation approaches and digital ranges is increasing, which allow training models, practicing algorithms for targeting and engaging targets without the risk of losing equipment and material resources [3, 4].

1.1. Motivation

The main factor in the development of intelligent UAV control systems is the need to reduce decision-making time and reduce the load on the operator when performing combat or monitoring tasks. Traditional target detection algorithms based on classical image

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

processing methods have shown their lack of flexibility and sensitivity to changes in observation conditions (lighting, weather factors, terrain type).

Systems based on deep learning, in particular, architectures such as YOLO (You Only Look Once) [5, 6], show significantly higher speed and accuracy of object detection, which makes them suitable for embedded systems and real-time applications. In addition, the integration of such models into the software environment of simulators (Microsoft AirSim [7], ArduCopter SITL [8], Mission Planner [9]) allows for preliminary testing of autonomous targeting and target engagement algorithms without using real equipment.

The use of AI/CV solutions in combination with simulation data allows for a significant reduction in the cycle of development, testing, and improvement of UAV control systems. This, in turn, contributes to increasing the level of technological independence of Ukraine's defense industry, in particular in terms of developing its own software and hardware solutions for autonomous platforms.

1.2. State of the Art

The problem of detecting and classifying objects in a UAV video stream is caused by a number of factors: low image quality, dynamic and complex backgrounds, changes in scale and lighting, and the need for real-time processing [10, 11]. Classical statistical methods of segmentation and background modeling provide a basic solution to the problem, but are not robust enough for high noise levels and rapid scene changes [12].

Several groups are conventionally distinguished among the approaches to solving the problem. The first group consists of traditional computer vision methods with selected features (SIFT [13], SURF [14], HOG [15]), which are useful for initial extraction of deterministic features, but their efficiency decreases under variable shooting conditions and frequent occlusions. The second group is deep convolutional neural networks: single-stage detectors (YOLO [5, 16], SSD [17]) provide high performance, critical for airborne systems, while two-stage approaches (R-CNN [18], Faster R-CNN [19, 20]) provide higher accuracy at the cost of higher latency. Recent modifications of the YOLO family (e.g., YOLOv7–YOLOv8) demonstrate improved performance with low-quality frames and complex backgrounds [11].

The third group is methods based on transformers (ViT, DETR), which better take into account the global context of the scene and show advantages in detecting small or heavily masked objects; at the same time, they require significant computational resources, which limits their use on board UAVs [21, 22]. The fourth direction is multimodal and hyperspectral approaches, which combine data from different sensors (RGB, IR, radar) and increase resistance to interference and masking, but complicate the onboard equipment and increase power consumption [23, 24].

To increase reliability, tracking algorithms (KLT [25], SORT [26], DeepSORT [27]) are used, which provide stable tracking of the target in the event of partial loss of observation or noise bursts and compensate for the limitations of single detection.

Among the current works, large-scale benchmarks and cross-platform studies are distinguished. In particular, in [28], YOLOBench is proposed—an extensive analysis of over 550 YOLO configurations on different hardware solutions (ARM, x86, GPU, NPU) with a unified training environment; the authors showed that with the help of modern architectures and techniques, even old models can occupy Pareto-optimal positions. In [29], models on embedded devices with different quantization modes are compared, evaluating mAP, power consumption, and latency, which highlights the problem of trade-off between accuracy and deployability.

At the same time, existing works have limitations: many studies focus on individual versions of YOLO or on a narrow class of hardware solutions, less often on a comprehensive performance assessment taking into account power consumption, latency, and optimization capabilities (FP16, INT8, TensorRT). There is also a lack of a generalized methodology for evaluating modern models (YOLOv10n, YOLOv11n, and other) on a wide range of platforms—from desktop GPUs to low-power ARM/NPU and specialized accelerators. Closing this gap will allow objectively selecting "model-platform" combinations for specific application scenarios in autonomous drones and related systems.

To automate the process of developing neural networks, implementations of the neuroevolutionary agglomerative topology (NEAT) method can be used [30]. The neuroevolutionary approach involves the automated creation of neural networks for recognizing certain classes of objects. Unlike traditional manual design, this method allows for simultaneous optimization of both the weights of connections between nodes and the network topology itself. In particular, in [31], the principles of creating a high-level distributed swarm intelligent system capable of quickly adapting to changes in environmental conditions through the use of a neuroevolutionary approach were studied. Combining heterogeneous agents with different sets of sensors and actuators (in particular, UAVs) into a swarm increases the system's ability to effectively assess environmental changes and adequately respond to them—both at the level of individual agents and the swarm as a whole.

1.3. Objectives and Tasks

The aim of the work is to develop an integrated intelligent platform for automated detection, classification, and engagement of targets by unmanned aerial vehicles using artificial intelligence technologies and simulation data, as well as to quantitatively evaluate and determine improvements in key system characteristics such as processing speed, computational efficiency, and energy consumption.

To achieve the goal, within the framework of this publication, it is necessary to solve the following **tasks**:

1) develop a system architecture using a microservices approach based on Kubernetes and ROS2 to ensure scalability and flexible integration of components;

2) develop an automated experiment methodology that combines data collection, annotation, and augmentation modules, YOLO model training, mission simulation in the AirSim environment, and metrics analysis;

3) conduct experimental studies of the effectiveness of YOLO models of different versions (v4–v11) on hardware platforms of different performance levels—from low-resource to high-end Jetson Orin systems, as well as experiments on modeling UAV movement using simulators, including quantitative measurement and comparative analysis of system performance metrics and their improvement relative to a baseline configuration.

The article is structured as follows.

Section 2 presents the research methodology, dataset characteristics and composition employed in the study, and also performance metrics and improvement evaluation.

Section 3 describes the developed architecture of the target detection, classification, and tracking system.

Section 4 provides a case study analyzing the performance of YOLO models on various hardware platforms.

Section 5 discusses the results of simulation experiments conducted using the ArduPilot ArduCopter SITL, Microsoft AirSim, Mission Planner, and the high-level Autopilot BEE environment.

Section 6 provides a general discussion of the obtained results.

Finally, Section 7 concludes the paper by summarizing the main findings and outlining directions for future research.

2. Materials and Methods of Research

The study uses a comprehensive methodology that combines simulation modeling, the formation of specialized datasets, and testing of deep learning algorithms on various hardware platforms. The goal is to create an experimental environment that allows evaluating the performance and accuracy of target detection models in real time, taking into account the resource constraints inherent in on-board systems of unmanned aerial vehicles.

A unified environment should combine tools for data collection, model training, mission simulation, and metric collection. The proposed methodology assumes a modular approach, where each component performs clearly defined functions, and the interaction between them is automated using programmatic interfaces and scripts (Python, Bash). The integration of modules is implemented through a microservice architecture using ROS2 and Docker, which provides distributed execution of calculations and flexibility in the configuration of experiments.

To manage the components of the robotic system and integrate them into a common platform, the Robot Operating System (ROS) [32] is used, which is an open source software platform that includes a set of libraries and tools for developing robotic software. ROS provides life cycle management of the components of the robotic system and their orchestration, supporting message exchange using the publish/subscribe model. The platform facilitates the implementation of a microservice architecture, in which each microservice can be implemented using any of the major programming languages supported by ROS.

The methodology involves performing experiments in several sequential stages.

1. Data preparation. Formation of training samples based on video materials and images obtained from UAVs, satellite data, and simulations in the Microsoft AirSim environment. The CVAT (Computer Vision Annotation Tool) system is used to mark objects, which allows marking in YOLO and COCO formats. After that, data augmentation is performed (changing brightness, contrast, scaling, rotations, and reflections), which increases the size of the dataset and improves the generalization ability of the models. The formed sets are stored in YAML format with a clearly defined distribution structure into train/validation/test.

2. Model training and optimization. Training is performed for YOLOv5, YOLOv8, and YOLOv11 models implemented in the Ultralytics YOLO framework.

3. Experiment configuration. To simulate the flight and behavior of unmanned systems, the AirSim simulator is used, which provides highly realistic modeling of the physics of motion and visual perception; it is integrated with ArduPilot SITL, which simulates the operation of the autopilot in software-in-the-loop mode, and with Mission Planner, a ground

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

control station for planning, monitoring, and analyzing flight missions. In the AirSim environment, mission parameters are set: landscape type, flight altitude, speed, target position, and weather conditions. The autopilot should control the UAV via the MAVLink interface, and the system should record target coordinates, sensor signals, and recognition results.

The functional diagram of the interaction of the components of the automated target detection, classification, and engagement system is shown in Fig. 1.

The primary reason for selecting YOLO-based architectures (v4–v11) in this study is their consistently demonstrated balance between inference speed and detection accuracy on resource-constrained platforms. YOLO models are optimized for single-stage inference, which minimizes latency and makes them suitable for real-time embedded UAV systems. In contrast, architectures such as EfficientDet [33] and SSD [17, 34] typically require higher computational budgets to achieve comparable accuracy, especially when detecting small objects on complex backgrounds. Furthermore, YOLO benefits from extensive open-source support, active development, standardized training pipelines, and a large ecosystem of pretrained weights. These factors enable reproducible benchmarking across heterogeneous hardware platforms (ARM/NPU to GPU), which is essential for the objectives of this work.

Unlike EfficientDet, YOLO models require fewer FLOPs and memory resources, particularly in lightweight variants (n/s series), making them deployable on micro-accelerators such as Google Coral Dev Board

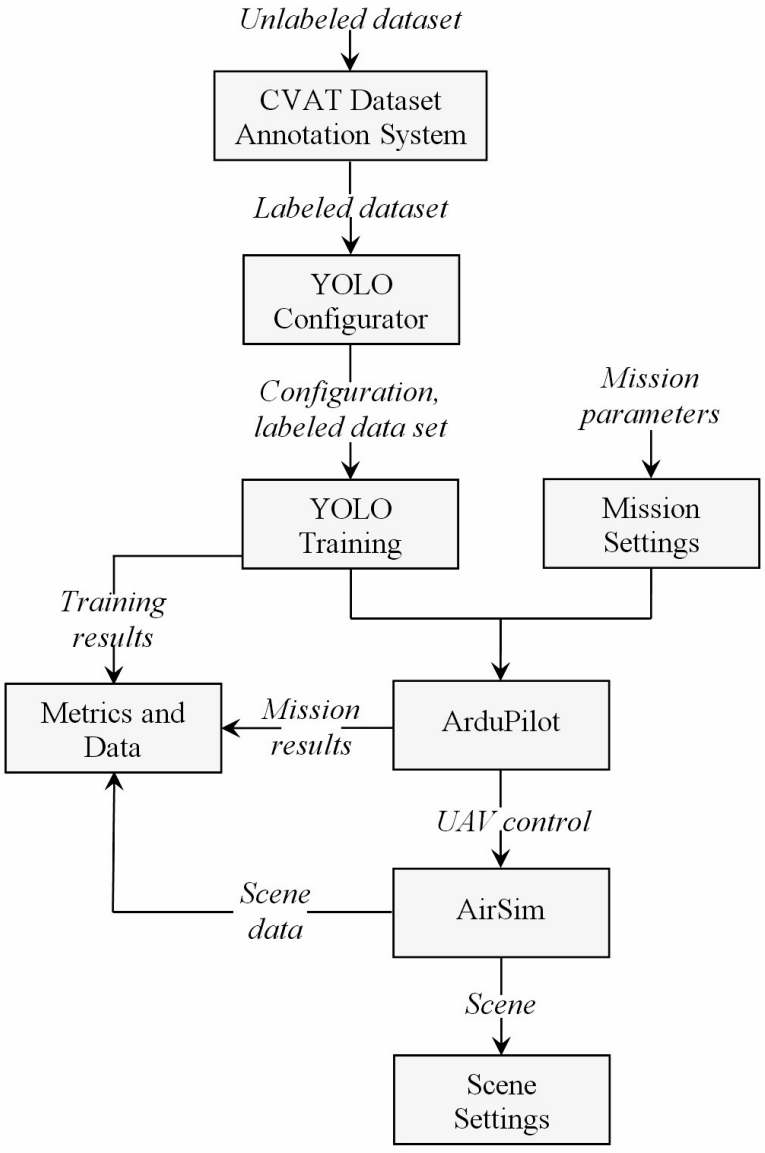

Fig. 1. Interaction of components of the experimental environment for automated target detection, classification and engagement

Micro and NVIDIA Jetson Orin Nano. This aligns directly with the scope of this research, which focuses on low-power onboard inference rather than server-class processing.

Comparative characteristics of object detection architectures relevant to real-time UAV deployment are given in Table 1.

2.1. Dataset Characteristics and Composition

The experimental research was conducted using a curated dataset constructed specifically for UAV-based detection and targeting tasks. The primary training dataset consisted of 16,847 annotated images retained after quality control procedures from an initial collection of 18,352 samples. During preprocessing, corrupted files, duplicates, and low-quality examples were removed, and manual verification was applied to ensure annotation accuracy and sufficient visual clarity. The source material combined real UAV-captured imagery (7,581 images, ~45%), synthetic data generated in the AirSim simulator under varying environmental conditions (5,896 images, ~35%), and publicly available satellite imagery (3,370 images, ~20%). The original resolutions ranged from 640×640 to 1920×1080 pixels; however, all samples were standardized to 640×640 for training.

The dataset included 11 object classes relevant to reconnaissance and target engagement. These comprised four subclasses of military ground vehicles, two subclasses of aircraft and helicopters, three categories of ground installations and infrastructure, and several additional tactical object classes. The detailed taxonomy is partially restricted due to defense-related confidentiality.

The initial class distribution exhibited moderate imbalance, with the most frequent category accounting for 18.3% of samples and the least frequent for 4.7%. To address this issue, a combination of dataset balancing strategies was applied, including oversampling of underrepresented categories through targeted augmentation, the use of weighted loss functions based on inverse class frequency, and the generation of additional synthetic examples (1,200 samples) for rare classes in AirSim. As a result, the final class distribution ranged from 8.1% to 12.4%, with a standard deviation of 1.8%, indicating a sufficiently balanced dataset for neural network training.

To increase robustness and improve the generalization capability of the models, extensive data augmentation techniques were applied. These included brightness and contrast adjustments, full-angle rotations, horizontal and vertical flips, scaling transformations, Gaussian noise injection, and simulated weather effects in synthetic imagery. Each image was used together with two augmented variants, resulting in an effective training set size of 50,541 samples per epoch.

A stratified splitting strategy was employed to maintain the overall class proportions across the dataset subsets. The training, validation, and test portions comprised 70% (11,793 images), 20% (3,369 images), and 10% (1,685 images), respectively. A chi-square test ($p > 0.05$) confirmed that class distributions remained statistically consistent across all splits.

Annotation quality was ensured using CVAT, producing YOLO-format labels with normalized bounding box coordinates and class identifiers. All critical classes underwent double-checking by experienced annotators, resulting in an inter-annotator agreement score exceeding IoU > 0.85 for 94% of all annotations. In total, 42,318 annotated object instances were recorded, with an average of 2.51 objects per image. Approximately 38% of samples contained a single annotated object, while the remaining 62% contained multiple targets.

Additionally, an external validation dataset of 2,150 images from previously unseen sources was compiled to evaluate generalization performance. This subset included varied brightness levels, object sizes, occlusion conditions, and background textures. Small objects represented 31.2% of all instances, medium objects 48.7%, and large objects 20.1%. Regarding occlusion, 58.3% of samples exhibited no visual obstruction,

Table 1

Comparative characteristics of object detection architectures relevant to real-time UAV deployment

Architecture	**Detection Stage**	**Inference Speed on Embedded Platforms**	**Accuracy (mAP)**	**Suitability for UAV Real-Time Tasks**	**Community Support**
YOLOv4–v11	One-stage	High (30–85 FPS)	High	Excellent	Very strong
EfficientDet	One-stage (compound scaling)	Medium–Low	High	Limited (latency sensitive)	Moderate
SSD	One-stage	Medium	Medium	Moderate	Moderate
Faster R-CNN	Two-stage	Low	Very high	Poor (high latency)	Strong

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

32.1% partial occlusion, and 9.6% severe occlusion. Background complexity analysis indicated 24% of simple scenes, 51% of moderate complexity environments, and 25% of highly complex scenes such as urban or forested areas.

2.2. Performance Metrics and Improvement Evaluation

To quantitatively evaluate the improvement of system characteristics, a set of normalized performance metrics was introduced based on a baseline configuration corresponding to the YOLOv4 model.

The following indicators were used:

1) speed improvement coefficient

$$I_{FPS} = \frac{FPS_{model}}{FPS_{baseline}},$$

where FPS_{model} is the inference speed of the evaluated model and $FPS_{baseline}$ corresponds to the baseline YOLOv4 performance;

2) computational efficiency which characterizes the efficiency of using computational resources

$$C = \frac{FPS}{GFLOPS},$$

where FPS is frames per second;

3) energy efficiency

$$E = \frac{FPS}{P},$$

where P is the power consumption (in watts). This metric is critical for onboard UAV systems with limited energy supply.

These metrics allow a unified comparison of different model-platform configurations and provide a quantitative basis for determining improvements in system characteristics, including processing speed, computational efficiency, and energy consumption.

3. The Developed Architecture of the Target Detection, Classification, Tracking, and Engagement System

The general architecture of the server environment of the developed system is shown in Fig. 2, where *pub* and *sub* denote publish and subscribe, respectively. It is based on the Kubernetes platform (for managing the deployment and scaling of microservices), namely, its minikube variant [35].

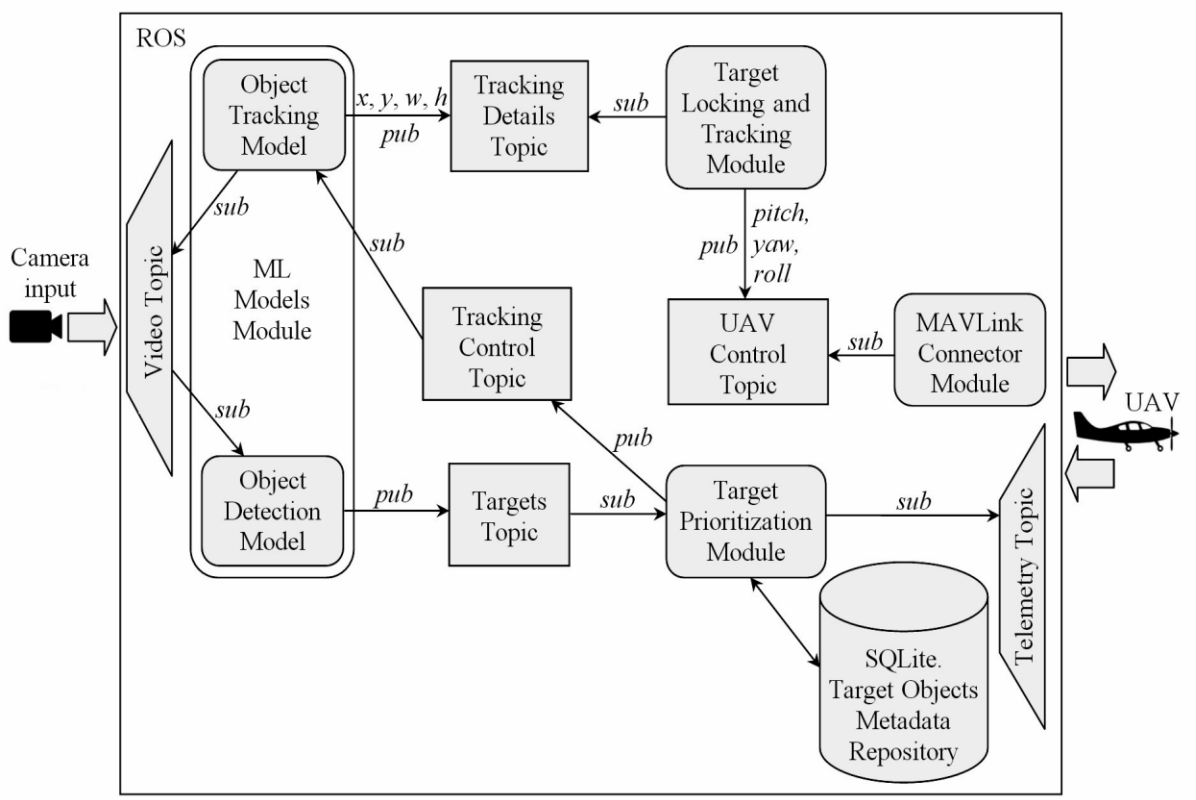

Fig. 2. Architecture of the system for detecting, classifying, and defeating targets based on artificil intelligence

The proposed software architecture is implemented as a set of microservices deployed in lightweight container instances orchestrated by Kubernetes. Each functional module—sensor data ingestion, neural network inference, target tracking, navigation, and telemetry—is encapsulated into a separate ROS2 node, packaged as a container image. Kubernetes schedules these containers across available compute resources, monitors their health, and restarts failed components automatically, which increases overall system resilience.

Microservices are orchestrated using Kubernetes Deployments, which maintain the desired number of replicas for critical components such as the inference engine and tracking modules. Stateless services are exposed via Kubernetes Services, allowing dynamic load balancing and runtime scaling. Stateful components, including mission telemetry logging, are deployed via StatefulSets backed by persistent volumes. Kubernetes health probes (liveness and readiness checks) ensure that malfunctioning containers are restarted with minimal downtime.

Communication between components is provided by ROS2 DDS (Data Distribution Service) middleware, which supports publish–subscribe messaging semantics. DDS automatically performs message discovery, serialization, delivery prioritization, and quality-of-service (QoS) negotiation. Critical data streams such as bounding box coordinates and velocity vectors are transmitted using QoS profiles with reliability guarantees, while non-critical telemetry uses a best-effort mode to reduce bandwidth consumption. Inter-service communication between Kubernetes pods uses virtual networking (CNI), ensuring isolation and low-latency message routing.

This architecture offers several key advantages for UAV-based systems:

– fault tolerance: failed components are automatically restarted by Kubernetes without interrupting higher-level behavior;

– scalability: inference workloads can be horizontally scaled across multiple nodes depending on mission complexity;

– modularity: perception, navigation, planning, and actuation remain decoupled, enabling independent updates;

– deployment flexibility: the same microservice configuration can be deployed on embedded hardware, cloud simulators, or hybrid environments;

– QoS-aware communication: ROS2 DDS ensures deterministic delivery of time-critical messages even under network congestion;

– resource efficiency: Kubernetes schedules workloads based on CPU/GPU availability, minimizing idle compute time.

The general services deployed on the platform are as follows:

1) GitLab for storing configurations, settings, source code, etc.;

2) Minio for storing files with datasets, machine learning (ML) models, etc.;

3) PostgreSQL for storing relational data;

4) QuestDB for storing unstructured data and metrics in the form of time series, etc.;

5) Prometheus for monitoring the operation of all platform components;

6) Grafana for graphical display of the state of the system as a whole and the state of individual components.

The experimental environment for implementing the methodology described in Section 2 includes the following key subsystems:

1) data acquisition and preparation subsystem. Uses the CVAT tool [36] for labeling video and image data. Generates structured datasets in YOLO format. Augmentation methods are used to increase the variability of training samples;

2) model training subsystem. Prepares configurations for YOLO family architectures (YOLOv3–YOLO11). Runs training using GPU/TPU for acceleration. Automatically collects intermediate results and metrics (mAP, precision, recall, F1-score) [37];

3) simulation and control subsystem. The following are used: ArduPilot (SITL) [8, 38]—emulation of an autopilot for multicopters in a software simulation environment; AirSim [7]—a virtual environment simulator that simulates the physics of motion, sensors, and visual scenes; Autopilot BEE [39]—a high-level mission control module capable of performing automated target search, tracking, and attack scenarios;

4) configuration and automation subsystem. Includes the development of scripts to launch all components (ArduPilot, AirSim, Mission Planner [9], Autopilot [39]) in automatic mode, setting up a client-server model to run simulations in a distributed environment, and unifying mission and scene parameters to ensure reproducibility of experiments;

5) subsystem for collecting and analyzing results. Registration of model parameters (hyperparameters, version, dataset configuration), logging of mission results (trajectory, guidance accuracy, task completion), saving of scenes and simulation parameters, as well as generation of reports with visualization of metrics and comparison of models.

Given the defense-oriented application domain, particular attention was paid to ensuring the security of data transfer between components of the proposed system. Inter-service communication inside the Kubernetes cluster is protected using mutual TLS (mTLS), ensuring

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

authentication and encryption of all internal traffic. Role-based access control (RBAC) policies are enforced to restrict access to critical microservices, preventing unauthorized invocation of mission-critical endpoints. In addition, Kubernetes Network Policies limit lateral movement by isolating namespaces and restricting traffic flows between services.

For ROS2-based communication, the DDS Security specification is enabled, providing message authentication, tamper resistance, and cryptographic protection against spoofed topics. All external network interfaces are protected through whitelisting and port-level filtering, with telemetry multiplexed over encrypted channels using Mavlink2 signing to mitigate injection attacks.

To ensure system safety in the event of intermittent communication loss, the high-level autopilot module maintains a local fallback behavior set, automatically switching to predefined mission profiles (e.g., orbiting, loitering, or return-to-home). This reduces operational risk and prevents uncontrolled UAV behavior when remote control is degraded or absent. Finally, logging and auditing mechanisms continuously collect metadata about component interactions to enable post-incident analysis and early anomaly detection.

4. Case Study 1: Experiments Analyzing the Performance of YOLO Models

We analyzed the performance of current versions of YOLO (from YOLOv4 to YOLOv11n), which differ in architectural complexity, hardware requirements, and support for optimizations (FP16, INT8, TensorRT, etc.). Particular attention was paid to platforms with limited power consumption, such as Orange Pi, Raspberry Pi, Rockchip, Google Coral, as well as the NVIDIA Jetson Orin.

In order to enable comparison of results across platforms and architectures, a system of universal performance units was developed. The baseline unit characteristics were established through a rigorous experimental protocol designed to ensure reproducibility and statistical significance. All measurements were conducted within a 90–95% confidence interval, calculated using Student's t-distribution based on the variance of repeated measurements. The primary characteristics are as follows:

– Computing Power (50–70 GFLOPS): the theoretical computational complexity of the YOLOv4 architecture was calculated through layer-by-layer analysis, accounting for floating-point operations in convolutional layers, batch normalization, and activation functions. The theoretical FLOPs per inference were multiplied by the achieved inference rate (in frames per second, FPS) on the reference platform to obtain effective GFLOPS. These calculations were verified using NVIDIA profiling tools (nvprof and Nsight Compute), which provided kernel-level performance metrics and confirmed the correspondence between theoretical and measured computational load. The reported range (50–70 GFLOPS) reflects systematic variations across different input resolutions (416 × 416 vs. 640 × 640 pixels) and batch sizes (1–4), with all values falling within the established confidence interval;

– GPU Memory (4–6 GB): active memory consumption during inference was monitored using the NVIDIA System Management Interface (nvidia-smi) and CUDA profiling utilities. Peak memory allocation was measured during continuous video processing under sustained workload conditions. The range accounts for batch size variations and framework overhead, with measurements repeated according to the protocol described in Table 1 to ensure statistical reliability;

– Reference Platform Selection: the NVIDIA GTX 1070 (8 GB) was selected as the baseline reference platform due to its widespread availability, well-documented specifications, and representative mid-range GPU performance characteristics. This choice enables objective cross-platform comparisons and provides a stable reference point for relative performance assessment;

– Power Consumption (15–20 W): power draw was measured using NVIDIA System Management Interface (nvidia-smi) during sustained inference workload, following a 10-second warm-up period to stabilize GPU operating temperature (65–75°C) and eliminate initialization effects. Measurements were sampled at 1-second intervals and averaged over the complete test duration. Background processes were minimized to isolate GPU inference power consumption. The reported range represents the 90-95% confidence interval across all tested configurations;

– Measurement Protocol Consistency: all baseline measurements strictly adhered to the unified experimental parameters specified in Table 2, including 50–100 inference runs per configuration, controlled environmental conditions, and standardized test sequences. Each measurement was repeated three times, as stated in our methodology, with statistical analysis performed to ensure that all reported values satisfy the 90–95% confidence criterion.

This baseline unit serves as a normalization factor, allowing for relative performance comparison across all tested platforms (ARM, NPU, various GPU configurations) by providing a common reference point. The confidence interval approach ensures that performance variations observed across different hardware platforms represent genuine architectural differences rather than measurement artifacts.

Table 2

Measurement parameters for experiments

Parameter	Value	Note
Test resolution	640 × 640 / 416 × 416	For ARM platforms
Batch size	1–4	Selected taking into account memory limitations
Precision	FP32 / FP16 / INT8	The optimal one for the platform is used
Number of runs	50–100	Ensuring statistical significance
Confidence interval	90–95 %	Determined based on the variance of measurements

In all experiments, the models were tested under conditions closely resembling real-world application scenarios: inference was performed on real video frames or images, taking into account delays arising from memory limitations, frame rates, and hardware access to the NPU, GPU, or CPU. To reduce the impact of background processes, all measurements were performed in dedicated environments or at minimal system load. In this work, the term "dedicated environments" refers to isolated and preconfigured hardware–software setups intended exclusively for benchmarking deep learning models. These environments use controlled CUDA and driver versions and contain no unnecessary software components that could introduce resource interference. The term "minimal system load" denotes operating conditions in which no other compute-intensive tasks are executed, and background processes are reduced to the lowest possible level, ensuring that measured performance reflects the behavior of the model rather than unrelated system activity.

Performance on desktop GPUs. On the NVIDIA GTX 1070 reference platform, the YOLOv11n and YOLOv8n models demonstrated the best real-time data processing speed, reaching 65–85 FPS and 60–80 FPS, respectively, at a batch size of 2–4 (see Table 3). YOLOv11n turned out to be the most efficient due to its compact NMS-free architecture, which reduces inference latency.

At the same time, the YOLOv5x and YOLOv8l models, despite their high accuracy (mAP), have a significant computational load (124.8 ± 9.3 GFLOPS and 165.2 ± 12.1 GFLOPS, respectively), which makes them less suitable for resource-constrained systems. These results indicate the feasibility of using lighter models (YOLOv8n, YOLOv11n) for tasks requiring high processing speed.

Table 3

Baseline performance testing results of YOLO models

YOLO version	FPS	Batch size	Recommendation
YOLOV5s	45–60	1–2	Perfect
YOLOV5m	30–40	1–2	Good
YOLOV8n	60–80	2–4	Ideal
YOLOV8s	40–50	2–4	Optimal
YOLOV11n	65–85	2–4	The best choice

Performance on mobile ARM platforms. Testing on Orange Pi, Raspberry Pi, and Rockchip platforms showed that YOLOv11n consistently outperforms other models in terms of FPS and power efficiency. For example, on Orange Pi 5 Plus (RK3588) (see Table 4), the model achieved 18–22 FPS at 8–12 W power consumption, which is optimal for autonomous systems with a limited power budget. On Raspberry Pi 5 without an AI accelerator, performance was limited (7–11 FPS), but adding the Hailo-8L accelerator increased FPS to 45–55 at 12–15 W power consumption. Rockchip platforms, in particular the RV1106, showed high power efficiency (2–4 W at 22–28 FPS), which makes them promising for microdrones and continuous video surveillance systems.

Performance on Google Coral. Google Coral has proven to be one of the most efficient solutions for low-resource edge devices due to its support for INT8-quantized models and low power consumption (1.5–3 W). Mini PCIe and M.2 A+E modules achieved 52–62 FPS and 48–58 FPS, respectively, with 16–21 ms latency. However, the platform has limitations such as a fixed resolution (300 × 300 pixels) and a maximum model size (8 MB), which precludes the use of more complex architectures such as YOLOv5x or YOLOv8l. For drones, the Google Coral Dev Board Micro is the optimal choice due to its compactness (40 g) and low power consumption.

NVIDIA Jetson Orin Performance. NVIDIA Jetson Orin platforms demonstrated exceptional performance for high-performance tasks. The AGX Orin 64 GB model delivered 250–290 FPS for YOLOv11n with 3.4–4.0 ms latency, the best result among all tested platforms.

Optimization using TensorRT in FP16 mode increased FPS for YOLOv8s by 2.4 times (from 45–55 to 108–128). The Orin NX 16 GB proved to be optimal in terms of cost/performance ($5.3/FPS), making it attractive for mid-range drones and industrial applications. The Orin Nano 4GB, despite its lower performance (58–72 FPS), remains an effective solution for compact systems thanks to its 10 W power consumption and 70 g weight.

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

Table 4

Performance measurement results on the Orange Pi platform

Device	YOLO version	GFLOPS	FPS	Power consumption
Orange Pi 5 Plus (RK3588), Mali-G610 MP4, 8 GB LPDDR4X, 1000 MHz	YOLOv5n	3.6 ± 0.4	13–17	8–12 W
	YOLOv8n	3.3 ± 0.4	15–19	8–12 W
	YOLOv11n	3.1 ± 0.3	18–22	8–12 W
Orange Pi 5 (RK3588S), Mali-G610 MP4, 4 GB LPDDR4X, 1000 MHz	YOLOv8n	2.9 ± 0.3	14–18	6–10 W
	YOLOv11n	2.7 ± 0.3	16–20	6–10 W

Table 5

Evaluation criteria for drones

Criterion	Weight (%)	Requirements	Substantiation
Performance	25	> 30 FPS	Implementing real-time navigation and detection
Energy efficiency	30	< 20 W	Ensuring long battery life (more than 30 minutes)
Dimensions/weight	20	< 200 g	Minimizing the impact on aerodynamics
Price	15	< $500	Accessibility for mass solutions
Ease of integration	10	Ready-made solutions	Reduction of development time

Drone application. The choice of a hardware platform for drones should consider not only computational performance, but also dimensions, weight, power consumption, and integration capabilities. We evaluated platforms based on weighting factors relevant to unmanned aerial vehicles.

The specifics of the YOLO application on drones require special attention to the parameters listed in Table 5.

The optimal solutions for drones include the Google Coral Dev Board Micro and the NVIDIA Jetson Orin Nano 4GB. The measurement results are listed in Table 6.

Evaluation of system performance improvement. The quantitative improvement of key system characteristics is summarized in Table 7. (Note that speed-up values are calculated relative to the baseline YOLOv4 model on the NVIDIA GTX 1070 platform. For embedded platforms, speed-up is not reported due to the absence of directly comparable baseline measurements.) Modern lightweight models such as YOLOv8n and YOLOv11n demonstrate a consistent increase in processing speed compared to the baseline YOLOv4 model, achieving up to 1.6 times higher FPS on the reference platform. At the same time, a significant reduction in computational complexity leads to a 5–7 times improvement in computational efficiency (FPS/GFLOPS).

The most pronounced effect is observed in terms of energy efficiency. For example, embedded platforms such as Google Coral Dev Board Micro achieve up to 25–32 FPS/W, which significantly exceeds the corresponding values for desktop GPU configurations.

These results confirm that the proposed combination of optimized YOLO architectures, quantization techniques, and hardware-specific acceleration provides measurable and practically significant improvements in system characteristics, which are critical for real-time UAV applications under resource constraints.

5. Case Study 2: Simulation Experiments

Experiments on deploying a simulation bench for testing UAV systems using ArduPilot ArduCopter SITL, Microsoft AirSim, Mission Planner, and the high-level autopilot Autopilot BEE showed successful integration of components and their interaction in the simulation environment. The tests performed covered network configuration, component startup, basic operations (takeoff, guided control, autopilot activation), and system performance monitoring. The results of the experiments and their analysis are presented below, taking into account data from a previous paper on the performance of YOLO neural networks used in computer vision tasks for UAVs.

Table 6

Optimal solutions for drones

Solution	Parameter	Value	Score	Recommendations
Google Coral Dev Board Micro – overall score 8.7/10	Performance (YOLOv8n)	11.4 ± 0.8 GFLOPS, 38–48 FPS	7/10	YOLO version YOLOv8n (quantized INT8), resolution 300 × 300. For small drones (less than 2 kg), patrol tasks, real-time image monitoring.
	Power consumption	1.5 W	10/10	
	Dimensions/weight	65 × 30 mm, 40 g	10/10	
	Price	$130	8/10	
	Integration	TensorFlow Lite ready-made models	9/10	
NVIDIA Jetson Orin Nano 4GB – overall score 8.2/10	Performance (YOLOv8n)	7.4 ± 0.6 GFLOPS, 58–72 FPS	8/10	YOLO version YOLOv8n/YOLOv11n with TensorRT. Resolution 640 × 640. For medium drones (2–10 kg), particularly for search and rescue or industrial applications.
	Power consumption	10 W	8/10	
	Dimensions/weight	70 × 45 mm, 70 g	9/10	
	Price	$400	6/10	
	Integration	Full CUDA and TensorRT support	10/10	

Table 7

Quantitative improvement of system characteristics

Model	Platform	FPS	GFLOPS	Power (W)	Speed-up vs YOLOv4	Computational efficiency (FPS/GFLOPS)	Energy efficiency (FPS/W)
YOLOv4 (baseline)	GTX 1070	50–60	65.2 ± 4.8	15–20	1.0	0.77–0.92	2.5–4.0
YOLOv8n	GTX 1070	60–80	14.9 ± 1.6	15–20	1.2–1.4	4.0–5.4	3.0–5.3
YOLOv11n	GTX 1070	65–85	11.7 ± 1.2	15–20	1.3–1.6	5.6–7.3	3.3–5.7
YOLOv11n	Raspberry Pi 5	8–11	1.1 ± 0.2	4–6	—	7.3–10.0	1.3–2.8
YOLOv11n + Hailo-8L	Raspberry Pi 5	45–55	7.2 ± 0.7	12–15	4–6	6.3–7.6	3.0–4.6
YOLOv11n	Google Coral Dev Board Micro	38–48	11.4 ± 0.8	1.5	—	3.3–4.2	25–32
YOLOv11n	Jetson Orin Nano 4GB	58–72	7.4 ± 0.6	10	—	7.8–9.7	5.8–7.2

Setting up and running the simulation environment. The simulation testbed was successfully deployed on a local PC (Windows 11 with WSL 2.5.9.0) and in a local network configuration (Windows 10). Using WSL Mirrored Networking in Windows 11 significantly simplified the network configuration, eliminating the need to manually configure ports via the vEthernet virtual adapter (WSL).

In Windows 10, the adapter IP addresses were used for correct operation (for example, 172.22.240.1 for Windows and 172.22.252.194 for Linux), which required additional checks using netstat and tcpdump.

The launch of the components (ArduCopter SITL, AirSim, Mission Planner, Autopilot BEE) was stable with the correct port configuration (UDP 9002/9003 for AirSim, TCP 5762 for MAVLink, UDP 14550 for

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

Mission Planner). In all tests, the components successfully established communication, as evidenced by the appearance of the text "Initializing autopilot" in AirSim after launching Autopilot BEE.

Autopilot and simulator performance. Testing showed that ArduCopter SITL in conjunction with AirSim provides a realistic simulation of multicopter flight. In Guided mode, the drone stably performed a takeoff to a height of 4 meters after activation via Mission Planner. The transition to automatic mode with Autopilot BEE (target search and destruction) occurred after turning on the first Servo/Relay position to the "High" position. In the AirSim simulation, messages about selected targets were displayed, and the drone performed the specified maneuvers.

Startup automation. The developed scripts air_run.cmd and air_prepare_to_fly.py successfully automated the launch of all components on the local PC. The script air_kill.cmd ensured the correct termination of processes, avoiding conflicts. For network configurations, the client-server application Air Server/Air Client was proposed, which in tests showed the potential for automating deployment on different network nodes via RabbitMQ. This greatly simplifies the scaling of the simulation environment.

Debugging and diagnostics. Using the tools netstat, socat, tcpdump (Linux), and Wireshark (Windows) allowed us to quickly identify connection problems. For example, checking open ports (9002, 9003, 5762, 14550) using netstat and the air_tcpview.py script confirmed the correctness of the network connections. Wireshark with the MAVLink plugin successfully decoded packets on port 5762, which allowed us to analyze MAVLink messages between ArduCopter and Autopilot BEE. Filtering unnecessary packets through Wireshark (for example, excluding ports 80, 443, 5353) increased the efficiency of diagnostics.

6. Discussion

The methodology proposed in Section 2 allows accelerating the research of target detection systems, reducing the influence of the human factor, and ensuring the uniformity of experiments. The use of simulation data allows to reduce the cost of field tests and to expand the range of studied scenarios without the risk of damage to the equipment.

The experimental results described in Section 4 confirm that the choice of the YOLO model and hardware platform depends on the specific requirements of the task. For drones with tight weight and power constraints (e.g., microdrones up to 2 kg), the optimal ones are Google Coral Dev Board Micro and YOLOv8n/YOLOv11n in INT8 quantization, which provide a balance between performance (38–48 FPS) and power efficiency (1.5 W). For medium-sized drones (2–10 kg) with search and rescue or industrial operations tasks, the NVIDIA Jetson Orin Nano 4 GB with YOLOv11n and TensorRT optimization is recommended, which provides 58–72 FPS at a resolution of 640 × 640 and a power consumption of 10 W. The key factor in improving performance is the application of optimization techniques such as quantization (FP16, INT8) and the use of TensorRT. For example, TensorRT has provided significant speedups (up to 2.4x) on NVIDIA platforms, allowing for more efficient use of computing resources. At the same time, Google Coral, despite its high energy efficiency, is limited to supporting only INT8 models, which requires additional efforts to convert models to formats compatible with TensorFlow Lite. Benchmarking also revealed bottlenecks. On ARM platforms such as the Raspberry Pi without an accelerator, performance is limited by low computing power (1.1–1.2 GFLOPS), making them unsuitable for real-time tasks without additional AI accelerators. Google Coral, despite its advantages, does not support more complex models due to limitations on model size and operations, which narrows the range of possible applications.

In addition to qualitative analysis, the obtained results allow a quantitative assessment of improvements in key system characteristics. As was summarized in Table 7 (Section 4), modern lightweight models (YOLOv8n, YOLOv11n) provide up to 1.6 times increase in processing speed compared to the baseline YOLOv4 model, while simultaneously reducing computational complexity, resulting in a 5–7 times improvement in computational efficiency (FPS/GFLOPS). Furthermore, the use of embedded AI accelerators significantly enhances energy efficiency. For example, platforms such as Google Coral Dev Board Micro achieve 25–32 FPS/W, which is an order of magnitude higher than traditional GPU-based solutions. These quantitative results confirm that the proposed approach ensures measurable improvements in system performance, making it suitable for real-time UAV applications with strict constraints on power consumption, weight, and onboard computational resources.

Based on the results obtained, the following recommendations have been formulated:

1) for tasks with strict requirements for power consumption (less than 3 W) and compactness (weight up to 40 g), it is recommended to use the Google Coral Dev Board Micro with YOLOv8n or YOLOv11n in INT8 quantization;

2) for medium-sized drones with high resolution (640 × 640) and performance (over 50 FPS) requirements, the NVIDIA Jetson Orin Nano 4GB with TensorRT optimization is optimal;

3) for high-demand tasks such as industrial analytics or complex navigation, AGX Orin 64GB with YOLOv8n/YOLOv11n is recommended for maximum performance;

4) TensorRT and quantization (FP16, INT8) are mandatory for all NVIDIA platforms to achieve maximum inference speed;

5) for ARM platforms without AI accelerators (e.g., Raspberry Pi) it is advisable to use additional modules such as Hailo-8L to ensure an acceptable FPS level.

While FPS is an important metric for evaluating neural network inference performance, real-world target engagement systems require assessment of the total end-to-end latency. In this context, end-to-end latency represents the cumulative delay between frame acquisition by the onboard sensor, detection and classification by the neural network, transmission of the decision to the guidance subsystem, and the actuation of control signals. The integration of a latency measurement pipeline is planned as part of future work, in order to provide timing guarantees required for time-critical engagement scenarios.

The experimental results given in Section 5 confirm that the simulation testbed based on ArduPilot ArduCopter SITL, AirSim, Mission Planner and Autopilot BEE is an effective tool for testing UAV systems. The integration of components provides realistic flight simulation, control, and execution of autonomous missions such as target search and destruction.

The main advantages of the testbed are:

– configuration flexibility: support for both local startup (Windows 11 with WSL Mirrored Networking) and network startup (Windows 10), which makes the testbed versatile for various hardware configurations;

– automation: scripts for local launch and the proposed client-server approach simplify system deployment and management;

– diagnostics: netstat, socat, tcpdump, and Wireshark tools provide reliable debugging of network connections.

The resilience of the proposed system to real operational conditions was partially evaluated within the simulation environment. In scenarios with temporary communication loss between the guidance subsystem and the neural network inference module, the UAV switches to a failsafe mode based on the last known valid target coordinates and continues tracking using onboard visual feedback. Communication channels are monitored through heartbeat signals, and short-term disconnections (up to 2–3 seconds) do not interrupt the engagement process. Extended loss of communication triggers a controlled return-to-home procedure based on ArduPilot failsafe mechanisms.

Robustness against electronic interference was not evaluated experimentally in this work due to infrastructure limitations. However, preliminary mitigation strategies are included in the system design, such as diversity of communication channels (2.4 GHz telemetry + optional LTE uplink), automatic frequency hopping, and fallback to mission-level autonomy through locally executed guidance logic. These aspects will be addressed in future field experiments by integrating RF noise generators in controlled test ranges.

The implemented UAV perception and engagement subsystem can be integrated into the Defence Resource Micro Services (DRMS) platform [40, 41], developed at the Institute of Software Systems of the National Academy of Sciences of Ukraine, to provide real-time mapping capabilities and target engagement control. In this configuration, the subsystem functions as part of a unified situational awareness and threat management environment. At the architectural level, integration is achieved through standardized message exchange mechanisms, enabling the transfer of detected target coordinates, classification labels, bounding box trajectories, and confidence scores. These data streams are consumed by the DRMS Threat Object Identification Module, which evaluates whether the detected object is known, previously observed, or belongs to a recognized threat class. Once ingested, the DRMS Behavior Prediction Module utilizes extrapolation and approximation algorithms to forecast the future location of aerial targets based on their observed motion dynamics, enabling proactive threat assessment.

The proposed UAV module also complements the DRMS Threat Assessment Module by providing target-specific metadata such as movement direction, velocity, and bounding box dynamics, which can be used to compute risk indicators for critical infrastructure assets. Integration benefits from the existing DRMS message exchange subsystem, which routes internal messages across functional modules in a scalable microservice environment. Target state information additionally synchronizes with the DRMS Object Database, ensuring persistent storage and historical inference.

Integration further offers operational advantages for decision support. Through the DRMS Visualization Module, threat trajectories and predicted positions can be rendered on GIS maps (based on QGIS), allowing operators to visually inspect the evolving air picture and evaluate potential impact zones. In scenarios of rapidly changing operational conditions, this enhances situational awareness and shortens operator reaction time.

From a system perspective, the integration provides three notable advantages:

1) multi-source fusion—UAV optical detection results can be combined with other sensor modalities already ingested by DRMS;

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

2) mission-level autonomy—even in degraded communication environments, DRMS can reason about predicted threat motion;

3) scalability—Kubernetes orchestration supports horizontal scaling of neural inference and data ingestion services, enabling parallel processing of multiple UAV data streams.

7. Conclusions

An experimental environment has been developed and implemented for researching intelligent systems for detecting, classifying, tracking, and engaging targets with unmanned aerial vehicles using artificial intelligence, computer vision, and simulation modeling technologies.

The conducted research allowed us to obtain the following main results:

1) an integrated system architecture has been created that combines data collection and labeling modules (CVAT), model training (Ultralytics YOLO), simulation modeling (AirSim, ArduPilot SITL, Mission Planner, Autopilot BEE), and monitoring (Prometheus, Grafana) in the Kubernetes/ROS2 microservice environment;

2) a unified methodology for evaluating the effectiveness of YOLO models of different versions on different hardware platforms is proposed, which takes into account performance, latency, and energy efficiency;

3) according to the results of experiments, it was found that the YOLOv8n and YOLOv11n models provide the best ratio between speed and accuracy when deployed on low-resource ARM/NPU platforms. The most effective for use on board drones were determined to be Google Coral Dev Board Micro and NVIDIA Jetson Orin Nano 4 GB;

4) successful integration of simulation components (AirSim, ArduPilot SITL, Mission Planner, Autopilot BEE) into a single testbed, providing realistic simulation of flight, navigation, and mission execution with autonomous target engagement;

5) the proposed software scripts and client-server architecture provide automation of simulation launch and monitoring, which increases the reproducibility and scalability of experiments.

Based on the conducted experiments, several practical recommendations can be formulated for different application scenarios. For UAVs with strict constraints on energy consumption (below 3 W) and weight (under 50 g), the most suitable configuration is YOLOv8n or YOLOv11n in INT8 quantization deployed on Google Coral Dev Board Micro, which provides a balanced trade-off between accuracy and inference speed in real-time operation. For medium-class UAV platforms (2–10 kg) requiring higher-resolution inference at 640×640 pixels and throughput above 50 FPS, NVIDIA Jetson Orin Nano 4 GB combined with TensorRT optimization is recommended. For complex operational environments involving multiple simultaneous visual streams or advanced onboard analytics, NVIDIA Jetson Orin AGX 64 GB provides sufficient computational headroom, achieving up to 250–290 FPS for YOLOv11n. On low-power ARM platforms without dedicated accelerators, real-time inference cannot be ensured; therefore, external AI accelerators such as Hailo-8L are advised.

At the model level, YOLOv8n and YOLOv11n demonstrated the best accuracy-to-throughput ratio across all tested platforms, while larger variants (e.g., YOLOv5x, YOLOv8l) showed insufficient performance for onboard deployment due to increased latency. Thus, lightweight YOLO architectures optimized with FP16 or INT8 quantization are recommended for real-time target detection and engagement on UAVs operating under resource constraints.

The practical significance of the results is to create a basis for building a digital testing ground for autonomous guidance and target engagement systems. The proposed approach allows reducing the cost of field experiments, increasing test safety, and accelerating the development cycle of AI solutions for UAVs.

Further research is planned to expand the simulation environment for multi-drone missions, improve cooperative target detection and engagement algorithms, and integrate trajectory prediction algorithms and decisions based on reinforcement learning. A promising direction for further development of intelligent target detection and engagement system is also the use of neuroevolutionary methods for automated design and optimization of neural network architectures. The use of NEAT-type algorithms can ensure the adaptation of deep learning models to specific environmental conditions—for example, to changes in lighting, terrain type, or the presence of interference in the video stream from a UAV. Future work will include hardware-in-the-loop (HIL) experiments with induced RF interference, packet loss injection, and adversarial signal degradation to quantitatively evaluate robustness under hostile electronic warfare conditions.

Contributions of authors: conceptualization, methodology – **Igor Sinitsyn, Anatoly Doroshenko**; formulation of tasks, analysis – **Igor Sinitsyn, Anatoly Doroshenko**; architecture and technology selection – **Iaroslav Omelianenko**; development of model, software, verification – **Ivan Kyrylov, Valentyn Smirnov**; analysis of results, visualization – **Ivan Kyrylov, Valentyn Smirnov**; writing – original draft preparation, writing – review and editing – **Olena Yatsenko**.

Conflict of Interest

The authors declare that they have no conflict of interest in relation to this research, whether financial, personal, authorship or otherwise, that could affect the research and its results presented in this paper.

Financing

This study was conducted with the financial support of the NAS of Ukraine within the framework of the following projects: "Development of methods and tools for a software and technological platform for automating resource management and information and analytical activities in the security and defense sector of Ukraine", state registration number 0124U000686, being implemented in accordance with the Resolution of the Bureau of the Informatics Department of the NAS of Ukraine dated July 13, 2023, protocol no. 1; "Development of methods and software tools for target detection, classification, and engagement based on artificial intelligence technologies", carried out in accordance with the Resolution of the Presidium of the National Academy of Sciences of Ukraine dated January 8, 2025, no. 11, "On the implementation by the institutions of the NAS of Ukraine in 2025 of R&D projects within the Targeted Scientific and Technical Defense Research Program of the NAS of Ukraine".

Data Availability

The manuscript has no associated data.

Use of Artificial Intelligence

The authors confirm that they did not use artificial intelligence technologies when creating the current work.

All the authors have read and agreed to the published version of this manuscript.

References

1. Tang, G., Ni, J., Zhao, Y., Gu, Y., & Cao, W. A survey of object detection for UAVs based on deep learning. *Remote Sensing*, 2024, vol. 16, iss. 1, pp. 1-29. DOI: 10.3390/rs16010149.

2. Valaboju, R., Vaishnavi, Harshitha, C., Kallam, A. R., & Babu, B. S. Drone detection and classification using computer vision. *Proceedings of the 7th International Conference on Trends in Electronics and Informatics (ICOEI)*. IEEE, Piscataway, 2023, pp. 1320-1328. DOI: 10.1109/ICOEI56765.2023.10125737.

3. Dimmig, C. A., Silano, G., McGuire, K., Gabellieri, C., Hönig, W., Moore, J., & Kobilarov, M. Survey of simulators for aerial robots: an overview and in-depth systematic comparisons. *IEEE Robotics & Automation Magazine*, 2025, vol. 32, iss. 2, pp. 153-166. DOI: 10.1109/MRA.2024.3433171.

4. Chan, J. H., Liu, K., Chen, Y., Sagar, A. S. M. S., & Kim Y.-G. Reinforcement learning-based drone simulators: survey, practice, and challenge. *Artificial Intelligence Review*, 2024, vol. 57, pp. 1-47. DOI: 10.1007/s10462-024-10933-w.

5. Gallagher, J. E., & Oughton, E. J. Surveying You Only Look Once (YOLO) multispectral object detection advancements, applications and challenges. *IEEE Access*, 2025, vol. 13, pp. 7366-7395. DOI: 10.1109/access.2025.3526458.

6. Bochkovskiy, A., Wang, C.-Y., & Liao, H.-Y. M. *YOLOv4: optimal speed and accuracy of object detection*. Available at: https://doi.org/10.48550/arXiv.2004.10934 (accessed 17 October 2025).

7. Shah, S., Dey, D., Lovett, C., & Kapoor, A. AirSim: high-fidelity visual and physical simulation for autonomous vehicles. *Proceedings of the 11th International Conference on Field and Service Robotics (FSR)*. *Springer Proceedings in Advanced Robotics*, 2018, vol. 5, pp. 621-635. DOI: 10.1007/978-3-319-67361-5_40.

8. *SITL Simulator (Software in the Loop)*. Available at: https://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html (accessed 17 October 2025).

9. *Mission Planner Overview*. Available at: https://ardupilot.org/planner/docs/mission-planner-overview.html (accessed 17 October 2025).

10. Zhao, J., Zhang, J., Li, D., & Wang, D. Vision-based anti-UAV detection and tracking. *IEEE Transactions on Intelligent Transportation Systems*, 2022, vol. 23, iss. 12, pp. 25323-25334. DOI: 10.1109/TITS.2022.3177627.

11. Bo, C., Wei, Y., Wang, X., Shi, Z., & Xiao, Y. Vision-based anti-UAV detection based on YOLOv7 GS in complex backgrounds. *Drones*, 2024, vol. 8, iss. 7, pp. 1-21. DOI: 10.3390/drones8070331.

12. Li, L., Huang, W., Gu, I. Y. H., & Tian, Q. Statistical modeling of complex backgrounds for foreground object detection. *IEEE Transactions on Image Processing*, 2004, vol. 13, iss. 11, pp. 1459-1472. DOI: 10.1109/TIP.2004.836169.

13. Lowe, D. G. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 2004, vol. 60, pp. 91-110. DOI: 10.1023/B:VISI.0000029664.99615.94.

14. Bay, H., Tuytelaars, T., & Van Gool, L. SURF: speeded up robust features. *Proceedings of the 2006 European Conference on Computer Vision (ECCV)*. *Lecture Notes in Computer Science*, 2006, vol. 3951, pp. 404-417. DOI: 10.1007/11744023_32.

15. Dalal, N., & Triggs, B. Histograms of oriented gradients for human detection. *Proceedings of the 2005*

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, Piscataway, 2005, pp. 886-893. DOI: 10.1109/CVPR.2005.177.

16. Jocher, G., & Qiu, J. *Ultralytics YOLO11*. Available at: https://github.com/ultralytics/ultralytics (accessed 17 October 2025).

17. Ni, J., Shen, K., Chen, Y., & Yang, S. X. An improved SSD-Like deep network-based object detection method for indoor scenes. *IEEE Transactions on Instrumentation and Measurement*, 2023, vol. 72, pp. 1-15. DOI: 10.1109/TIM.2023.3244819.

18. Girshick, R., Donahue, J., Darrell, T., & Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the 2014 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE, Piscataway, 2014, pp. 580-587. DOI: 10.1109/CVPR.2014.81.

19. Ren, S., He, K., Girshick, R., & Sun, J. Faster R-CNN: towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017, vol. 39, iss. 6, pp. 1137-1149. DOI: 10.1109/TPAMI.2016.2577031.

20. Ni, J., Shen, K., Chen, Y., Cao, W., & Yang, S. X. An improved deep network-based scene classification method for self-driving cars. *IEEE Transactions on Instrumentation and Measurement*, 2022, vol. 71, pp. 1-14. DOI: 10.1109/TIM.2022.3146923.

21. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. *An image is worth 16x16 words: transformers for image recognition at scale*. Available at: https://doi.org/10.48550/arXiv.2010.11929 (accessed 17 October 2025).

22. Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., & Zagoruyko, S. End-to-end object detection with transformers. *Proceedings of the 2020 European Conference on Computer Vision (ECCV). Lecture Notes in Computer Science*, vol. 12346, pp. 213-229. DOI: 10.1007/978-3-030-58452-8_13.

23. Rasti, B., Hong, D., Hang, R., Ghamisi, P., Kang, X., Chanussot, J., & Benediktsson, J. A. Feature extraction for hyperspectral imagery: the evolution from shallow to deep: overview and toolbox. *IEEE Geoscience and Remote Sensing Magazine*, 2020, vol. 8, iss. 4, pp. 60-88. DOI: 10.1109/MGRS.2020.2979764.

24. Hong, D., Gao, L., Yao, J., Zhang, B., Plaza, A., & Chanussot, J. Graph convolutional networks for hyperspectral image classification. *IEEE Geoscience and Remote Sensing Magazine*, 2021, vol. 59, iss. 7, pp. 5966-5978. DOI: 10.1109/TGRS.2020.3015157.

25. Shi, J., & Tomasi, C. Good features to track. *Proceedings of the 1994 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Piscataway, 1994, pp. 593-600. DOI: 10.1109/CVPR.1994.323794.

26. Bewley, A., Ge, Z., Ott, L., Ramos, F., & Upcroft, B. Simple online and realtime tracking. *Proceedings of the 2016 IEEE International Conference on Image Processing (ICIP)*. IEEE, Piscataway, 2016, pp. 3464-3468. DOI: 10.1109/ICIP.2016.7533003.

27. Wojke, N., Bewley, A., & Paulus, D. Simple online and realtime tracking with a deep association metric. *Proceedings of the 2017 IEEE International Conference on Image Processing (ICIP)*. IEEE, Piscataway, 2017, pp. 3645-3649. DOI: 10.1109/ICIP.2017.8296962.

28. Lazarevich, I., Grimaldi, M., Kumar, R., Mitra, S., Khan, S., & Sah, S. YOLOBench: benchmarking efficient object detectors on embedded systems. *Proceedings of the 2023 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*. IEEE, Piscataway, 2023, pp. 1161-1170. DOI: 10.1109/ICCVW60793.2023.00126.

29. Cantero, D., Esnaola-Gonzalez, I., Miguel-Alonso, J., & Jauregi, E. Benchmarking object detection deep learning models in embedded devices. *Sensors*, 2022, vol. 22, iss. 11. DOI: 10.3390/s22114205.

30. Stanley, K.O., Clune, J., Lehman, J., & Miikkulainen, R. Designing neural networks through neuro-evolution. *Nature Machine Intelligence*, 2019, vol. 1, pp. 24-35. DOI: 10.1038/s42256-018-0006-z.

31. Omelianenko, I., & Sinitsyn, I. Artificial swarm intelligence. *International Scientific Technical Journal "Problems of Control and Informatics"*, 2024, vol. 69, iss. 3, pp. 91-103. DOI: 10.34229/1028-0979-2024-3-7. (In Ukrainian).

32. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., & Ng, A. Y. ROS: an open-source robot operating system. *Proceedings of the 2009 ICRA workshop on open source software*. IEEE, Piscataway, 2009. P. 1-6. Available at: http://www.robotics.stanford.edu/~ang/papers/icraoss09-ROS.pdf (accessed 17 October 2025).

33. Tan, M., Pang, R., & Le, Q. V. EfficientDet: scalable and efficient object detection. *Proceedings of the 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 10781-10790. DOI: 10.1109/CVPR42600.2020.01079.

34. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. SSD: Single Shot MultiBox Detector. *Proceedings of the 2016 European Conference on Computer Vision (ECCV). Lecture Notes in Computer Science*, 2016, vol. 9905, pp. 21-37. DOI: 10.1007/978-3-319-46448-0_2.

35. *Minikube*. Available at: https://minikube.sigs.k8s.io/docs (accessed 17 October 2025).

36. *Introduction to CVAT.ai: best image annotation tool explained in simple terms*. Available at: https://www.cvat.ai/resources/blog/introduction-to-cvat (accessed 17 October 2025).

37. *Ultralytics YOLO Docs. Performance Metrics Deep Dive*. Available at: https://docs.ultralytics.com/guides/yolo-performance-metrics (accessed 17 October 2025).

38. *ArduPilot Dev Team. SITL with Webots*. Available at: https://ardupilot.org/dev/docs/sitl-with-webots.html (accessed 17 October 2025).

39. *autopilot_bee_sim. Autopilot with Target Following for FPV Combat Drone (Simulator version)*. Available at: https://github.com/under0tech/autopilot_bee_sim.git (accessed 17 October 2025).

40. *DRMS platform for the development of applications and specialized software*. Available at: https://it-solutions.ua/it-infrastruktura/fizichna/platforma-drms (accessed 17 October 2025) (In Ukrainian).

41. Sinitsyn, I. P., Omelianenko, Ia. V., Tverdokhlib, Ye. M., Stepaniuk, M. Yu., Zhytkevych, O. B., & Verveyko, V.M. *Computer program "DRMS Platform". Certificate of registration of copyright for the work no. 122378 dated 12/26/2023*. Available at: https://iss.nas.gov.ua/storage/editor/files/private/Sinitsin/03-svidotstvo_DRMS.pdf (accessed 17 October 2025) (In Ukrainian).

Received 17.10.2025, Received in revised form 07.12.2025
Accepted date 15.01.2026, Published date 22.01.2026

ІНТЕЛЕКТУАЛЬНІ ТЕХНОЛОГІЇ ВИЯВЛЕННЯ ТА УРАЖЕННЯ ЦІЛЕЙ БЕЗПІЛОТНИМИ ЛІТАЛЬНИМИ АПАРАТАМИ НА ОСНОВІ ШТУЧНОГО ІНТЕЛЕКТУ ТА СИМУЛЯЦІЙНИХ ДАНИХ

І. П. Сініцин, А. Ю. Дорошенко, І. І. Кирилов, Я. В. Омельяненко, В. Є. Смірнов, О. А. Яценко

Предметом вивчення в статті є методологія створення інтегрованих інтелектуальних систем автоматизованого виявлення, класифікації, супроводу та ураження цілей безпілотними літальними апаратами з використанням технологій штучного інтелекту, комп'ютерного зору та симуляційного моделювання. **Метою** є розроблення архітектури та експериментального середовища для дослідження ефективності моделей глибокого навчання, зокрема сімейства YOLO, у режимі реального часу на різних апаратних платформах із урахуванням обмежень ресурсів і енергоспоживання, а також кількісна оцінка покращення ключових характеристик системи. **Завданнями** дослідження є створення мікросервісної архітектури системи на базі Kubernetes і ROS2; формування спеціалізованих наборів даних для тренування моделей; інтеграція симуляторів AirSim, ArduPilot SITL і Mission Planner у єдине середовище випробувань; а також порівняльне оцінювання продуктивності моделей YOLO різних версій на платформах від low-power ARM/NPU до high-end GPU. Використовуваними **методами** є симуляційне моделювання, автоматизоване навчання нейронних мереж, квантизація моделей, оптимізація інференсу за допомогою TensorRT, а також статистичний аналіз отриманих метрик. Отримані такі **результати**. Розроблено уніфіковану методологію експериментів і побудовано програмно-апаратну платформу, що забезпечує повний цикл дослідження — від генерації та розмітки даних до аналізу продуктивності моделей. Проведено експериментальні тести, які показали, що YOLOv8n та YOLOv11n забезпечують найкраще співвідношення точності та швидкодії на низькоресурсних платформах. Найбільш ефективними для бортового використання визначено Google Coral Dev Board Micro та NVIDIA Jetson Orin Nano. Розгорнуто симуляційний стенд, що успішно моделює польоти БПЛА, процеси наведення та ураження цілей. **Висновки.** Результати підтвердили доцільність використання симуляційних даних і мікросервісного підходу для створення автономних інтелектуальних систем керування БПЛА, продемонстрували помітні покращення продуктивності, обчислювальної ефективності та споживання енергії, а також надали практичні орієнтири для вибору архітектури «модель–платформа». Наукова новизна отриманих результатів полягає у створенні комплексного середовища дослідження, яке об'єднує сучасні інструменти штучного інтелекту, симуляційне моделювання та апаратну оптимізацію в єдину відтворювану структуру, що дозволяє об'єктивно оцінювати ефективність алгоритмів виявлення та ураження цілей у реальному часі.

Ключові слова: автоматизоване виявлення та ураження цілей; безпілотні літальні апарати; глибоке навчання; комп'ютерний зір; нейронні мережі; симуляційне моделювання; штучний інтелект.

Сініцин Ігор Петрович – д-р техн. наук, проф., член-кореспондент НАН України, директор, Інститут програмних систем Національної академії наук України, Київ, Україна.

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

Дорошенко Анатолій Юхимович – д-р фіз.-мат. наук, проф., проф. кафедри інформаційних систем та технологій, пров. наук. співроб. відділу теорії комп'ютерних обчислень, Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського» та Інститут програмних систем Національної академії наук України, Київ, Україна.

Кирилов Іван Ігорович – асп., Інститут програмних систем Національної академії наук України, Київ, Україна.

Омельяненко Ярослав Вікторович – асп., мол. наук. співроб., Інститут програмних систем Національної академії наук України, Київ, Україна.

Смірнов Валентин Євгенович – асп., Інститут програмних систем Національної академії наук України, Київ, Україна.

Яценко Олена Анатоліївна – канд. фіз.-мат. наук, старш. наук. співроб. відділу комп'ютерних обчислень, Інститут програмних систем Національної академії наук України, Київ, Україна.

Igor Sinitsyn – Doctor of Technical Sciences, Professor, Corresponding Member of the NAS of Ukraine, Director, Institute of Software Systems of the National Academy of Sciences of Ukraine, Kyiv, Ukraine.
e-mail: ips@nas.gov.ua, ORCID: 0000-0002-4120-0784, Scopus Author ID: 10046546600.

Anatoliy Doroshenko – Doctor of Physical and Mathematical Sciences, Professor, Professor of the Department of Information Systems and Technologies, Head of the Computing Theory Department, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute" and the Institute of Software Systems of the National Academy of Sciences of Ukraine, Kyiv, Ukraine.
e-mail: doroshenkoanatoliy2@gmail.com, ORCID: 0000-0002-8435-1451, Scopus Author ID: 26662433900.

Ivan Kyrylov – Ph.D. Student, Institute of Software Systems of the National Academy of Sciences of Ukraine, Kyiv, Ukraine.
e-mail: ivan.kyrylov.science@gmail.com, ORCID: 0009-0006-9756-5391.

Iaroslav Omelianenko – Ph.D. Student, Junior Researcher, Institute of Software Systems of the National Academy of Sciences of Ukraine, Kyiv, Ukraine.
e-mail: yaric@newground.com.ua, ORCID: 0000-0002-2190-5664.

Valentyn Smirnov – Ph.D. Student, Institute of Software Systems of the National Academy of Sciences of Ukraine, Kyiv, Ukraine.
e-mail: vesbox@gmail.com, ORCID: 0009-0006-4022-5951.

Olena Yatsenko – Ph.D. in Physics and Mathematics, Senior Researcher at the Computing Theory Department, Institute of Software Systems of the National Academy of Sciences of Ukraine, Kyiv, Ukraine.
e-mail: oayat@ukr.net, ORCID: 0000-0002-4700-6704, Scopus Author ID: 28667561500.