

Oleksandr Skorlupin, Andrii Podorozhniak

National Technical University "Kharkiv Polytechnic Institute", Kharkiv, Ukraine

YOLO FAMILY ARCHITECTURES FOR EXPLOSIVE OBJECT DETECTION IN EMBEDDED SYSTEMS OF AUTONOMOUS UNMANNED VEHICLES: A COMPARATIVE ANALYSIS

Abstract. Relevance. The need to choose an effective YOLO architecture for explosive object detection in unmanned aerial vehicle embedded systems is driven by limited computational resources, real-time constraints, and energy efficiency requirements. **The object of research** is explosive object detection in embedded autonomous unmanned vehicle systems using YOLO family architectures. **The subject of the research** is the comparative efficiency of YOLO models in terms of detection accuracy, throughput, model size, and energy consumption on embedded hardware platforms. **The purpose of this paper** is to develop and apply an integral efficiency criterion based on neural network methods for selecting the most suitable YOLO architecture for explosive object detection and humanitarian demining tasks. **Research results.** The proposed criterion was applied to six YOLO models (v5n, v8n, v8s, v10n, v11n, v11s) deployed on Raspberry Pi 3 Model B+ and Raspberry Pi 4 Model B with 4 GB of RAM. Comparative evaluation under three scenarios showed substantial differences in detection quality, throughput, model size, and energy consumption. The ranking confirms the practical value of the methodology for model selection in embedded systems. **Conclusions.** The methodology provides a transparent and reproducible basis for comparing YOLO architectures in resource-constrained environments. Real-time operation remains challenging on low-power platforms, which justifies further optimization of deployment and inference.

Keywords: YOLO; explosive object detection; unmanned aerial vehicle; embedded systems; integral efficiency criterion; neural network; humanitarian demining; Raspberry Pi.

Introduction

Relevance. The task of automated detection of explosive objects using an embedded computer system onboard an autonomous unmanned aerial vehicle is an important research direction in the field of humanitarian demining [1–3]. The scale of contamination of Ukraine's territory by explosive remnants of war, which international organizations estimate to cover tens of thousands of square kilometers, combined with the productivity of manual demining at only 15–25 m² per deminer per day, creates a strong need for automated systems for preliminary terrain survey [4, 5].

Therefore, one of the most promising ways to address this problem is the deployment of a variety of robotic systems, including unmanned aerial vehicles (UAVs) and unmanned ground vehicles (UGVs) [6].

An overview of scientific works. In its most general form, the architecture of such a system includes image acquisition from an optical camera, resizing the image to the fixed input resolution of the neural network detector (typically 640×640 pixels), forward propagation through a YOLO-family convolutional neural network, and generation of georeferenced detection results in near real time [7].

In recent publications, methods are proposed for quantitatively assessing the effectiveness of systems of this type using classification quality metrics (Precision, Recall, F1), a spatial localization metric (IoU), and performance-oriented indicators (FPS, decomposition of the time budget for processing a single frame) [8, 9]. These methods assume that the target model has been predefined in advance and make it possible to evaluate its compliance with the functional requirements of the task.

At the system design stage, the opposite task arises: selecting, on a well-founded basis, the most appropriate architecture from several potentially suitable candidates

using the same criteria, but without conducting a full cycle of field tests for each of them.

The technical specificity of the task is complicated by several circumstances that must be taken into account in the selection methodology. The current YOLO family includes at least six architectural variants whose computational complexity is compatible with the resources of single-board computers such as the Raspberry Pi 3 Model B+ (quad-core Cortex-A53, 1 GB LPDDR2) and Raspberry Pi 4 Model B (quad-core Cortex-A72, 4 GB LPDDR4). This list includes the v5n, v8n, v8s, v10n, v11n, and v11s versions, ranging from the 2021 release to the 2025 architectures. Each of the named models can be run in several software inference formats – PyTorch, ONNX Runtime, TensorFlow Lite, and NCNN – each of which has its own trade-off between speed and deployment convenience on ARM processors [10]. Special attention should be given to the fact that the latest-generation models, especially YOLOv10, implement a training strategy based on dual label assignment; this eliminates the need for Non-Maximum Suppression (NMS) during inference and provides a deterministic latency gain, which is especially noticeable on platforms with limited computing power [11].

The combination of these factors makes model selection non-trivial in nature, and an arbitrary choice of a default architecture is methodologically unjustified. In the context of research focused on embedded detector deployment, the issue of selecting a well-founded architectural variant acquires independent scientific significance.

The aim of the article is to formalize and practically apply a methodology for the justified selection of a YOLO architecture for the task of detecting explosive objects by an embedded computer system onboard an unmanned aerial vehicle. The proposed methodology is developed as a logical continuation of the

criterion framework presented by the authors in, with the analytical apparatus extended to an integral efficiency criterion that accounts for the specific features of both target platforms and the operational conditions of the system in support of humanitarian demining.

Problem statement and characteristics of the onboard computing platforms

The task considered is the automated detection of explosive objects across six classes – anti-personnel and anti-tank mines, mortar and artillery shells, hand grenades, and small arms ammunition – in images received from an optical camera onboard an unmanned aerial vehicle, using the onboard computing system. The class nomenclature is inherited from earlier works [2, 6, 13] and corresponds to the most common types of explosive remnants of war in Ukraine [4, 14, 15].

Detection should be performed directly onboard, without transmitting the video stream to a ground station, because analyses of edge-computing efficiency in highly mobile computing networks show that local processing enables decision-making that is on average 15% faster than a centralized architecture and preserves node operability under communication loss, which is of fundamental importance for post-war applications [16]. Raspberry Pi 3 Model B+ and Raspberry Pi 4 Model B, each in the 4 GB RAM configuration, were selected as the onboard computing platforms. The choice is motivated by three considerations: both platforms are widely available, have a well-established software ecosystem, and exhibit documented behavior under sustained computational load, which improves result reproducibility; prior experience with deploying computer vision systems on Raspberry Pi for license plate recognition is described in [17].

The computational-performance gap between the Pi 3B+ and Pi 4, documented at about 2.5 – 4 times [18], covers the typical range of onboard computers used in small UAVs, so the obtained estimates can be interpolated to intermediate configurations. The power consumption of both platforms, measured in watts, is compatible with the energy budget of a small multirotor aircraft: at a flight power of 200 – 400 W, onboard computing accounts for only a few percent.

Raspberry Pi 3 Model B+ is built around the Broadcom BCM2837B0 system-on-chip (SoC) with four ARM Cortex-A53 cores running at 1.4 GHz and 1 GB of LPDDR2 RAM. The Cortex-A53 microarchitecture is an energy-efficient, in-order design and provides about 2.24 DMIPS/MHz per core; the limited memory places an upper bound on the model size that can be loaded together with the operating system and inference software stack, while the platform's power use is about 1.9 – 2.1 W at idle and up to 3.6 W under full computational load.

Raspberry Pi 4 Model B uses the BCM2711 SoC with four Cortex-A72 cores clocked at 1.5 GHz and 4 GB of LPDDR4-3200 RAM. Cortex-A72 is a superscalar out-of-order microarchitecture that delivers about 4.72 DMIPS/MHz, roughly twice that of Cortex-A53 at the same frequency; combined with the faster memory subsystem, this provides a 2.5 – 4× increase in convolutional-network inference performance compared with the Pi 3B+. Power consumption is 3.4 – 4.0 W at idle and 6.0 – 7.6 W under load; during prolonged full-load operation, the platform requires a passive heatsink to avoid thermal throttling, which above 80°C reduces clock frequency and distorts speed measurements. The consolidated platform characteristics are presented in Table 1.

Table 1 – Characteristics of onboard computing platforms

Technical specification	Pi 3B+	Pi 4 (4 Gb)	Pi 5 (8 Gb)	Jetson Orin Nano Super
SoC	BCM2837B0	BCM2711	BCM2712	6×Cortex-A78AE + GPU 1024 CUDA
Processor	4×Cortex-A53	4×Cortex-A72	4×Cortex-A76	1,7
Clock frequency, GHz	1,4	1,5	2,4	8 ГГц LPDDR5
Memory	1 Гб LPDDR2	4 Гб LPDDR4	8 Гб LPDDR4X	7,0
Performance, DMIPS/MHz	2,24	4,72	6,4	7
Power (idle), W	1,9-2,1	3,4-4,0	2,7–3,0	15–25
Power (load), W	3,6	6,0-7,6	7,5–9,0	249
Approximate cost, USD	35	55	80	6×Cortex-A78AE + GPU 1024 CUDA

To complete Table 1 for a fuller comparison, next-generation platforms, Raspberry Pi 5 and NVIDIA Jetson Orin Nano Super, are also included, although they are not part of the study's comparative set. Raspberry Pi 5, based on the Cortex-A76 processor, delivers roughly four times higher performance than Raspberry Pi 4, but its power consumption under load rises to 7.5–9.0 W, while the total cost of the platform together with mandatory active cooling and a higher-power supply is about 1.5 times that of Raspberry Pi 4.

The NVIDIA Jetson Orin Nano Super platform, equipped with an Ampere-architecture GPU accelerator,

outperforms the considered single-board computers by an order of magnitude in computational performance; however, its power consumption in maximum-performance mode reaches 25 W, and its cost is about 4.5 times higher than that of Raspberry Pi 4.

For a small multirotor UAV, both factors are highly significant: higher power consumption shortens flight time for a fixed battery capacity, and the cost of the computing module becomes decisive in serial deployment, where loss of the aircraft in a demining zone is an expected operational risk. These platforms are therefore considered a direction for further expansion of

the study, provided that the energy and cost constraints are relaxed.

An essential part of the problem formulation is the choice of the software format for model execution. For ARM processors without hardware accelerators, several inference stacks are available: native PyTorch, ONNX Runtime, TensorFlow Lite, and the NCNN framework, which is optimized for mobile ARM platforms using vectorized NEON instructions. According to official measurements [18, 19], for the YOLOv8n model on Raspberry Pi 4, the NCNN format provides an inference time of 414.7 ms versus 560.0 ms in ONNX Runtime and 950.3 ms in TensorFlow Lite, corresponding to advantages of 1.35 and 2.29 times, respectively. In view of this, the subsequent analysis is carried out for the NCNN format as the primary one, with data for the other formats given for comparison in the comparative analysis.

The system requirements are formulated as follows. The target video-processing rate is set at 25 frames per second, which corresponds to the standard PAL frame rate and typical onboard camera settings. The detector input resolution is 640×640 pixels, inherited from the training configuration of the YOLO family of models. The flight altitude in the calculation scenario is 20 m, which, at a typical camera field of view [20], yields a ground sampling distance on the order of centimeters per pixel and is sufficient for detecting anti-personnel mine-class objects. It is expected in advance that the target rate will not be achieved by any model on general-purpose processors without accelerators; quantifying this gap and identifying ways to reduce it are among the results of the work.

Overview of the compared YOLO family architectures

Six models from four YOLO generations are included in the comparison: YOLOv5n, YOLOv8n, YOLOv8s, YOLOv10n, YOLOv11n, and YOLOv11s. The selection criterion was the basic compatibility of model computational complexity with the resources of the target platforms: the smallest nano configurations of each generation were considered, along with two small-class configurations to assess whether moving to larger models on Raspberry Pi 4 is justified. Medium-class configurations and above were excluded because their computational complexity, exceeding 60 GFLOPs, leads to frame-processing times of several seconds even on Pi 4, which makes such a comparison practically meaningless. The 2025 architectures are included in the summary table of characteristics as reference points, but they are not used in the computational part for the reasons given below. Two-stage detectors such as Mask R-CNN, whose application to recognition tasks is discussed [21], are excluded from the comparison because their computational requirements are fundamentally higher and incompatible with the resources of the target platforms.

YOLOv5n (2021) is the smallest configuration of the fifth generation and contains 1.9 million parameters with a computational complexity of 4.5 GFLOPs. The

architecture uses a CSPDarknet backbone with C3 modules and an anchor-based detection scheme. The model achieves 28.0% mAP50-95 on the COCO validation set, which is the lowest among the compared models, but its minimal computational complexity makes its speed a benchmark for resource-constrained platforms.

YOLOv8n and YOLOv8s (2023) belong to the eighth generation, which introduced an anchor-free detection scheme with a decoupled head and C2f modules in the backbone. YOLOv8n has 3.2 million parameters at 8.7 GFLOPs and achieves 37.3% mAP50-95; YOLOv8s has 11.2 million parameters at 28.6 GFLOPs and 44.9%, respectively. The eighth generation is treated as the baseline for explosive-object detection because earlier work [2, 6] and the evaluation methodology were built on YOLOv8n, and the transfer of this generation to applied domains is documented, including in remote sensing tasks [22].

YOLOv10n (2024) uses a fundamentally different training scheme with a dual label assignment mechanism: during training, a one-to-many branch operates in parallel to provide a strong gradient signal, while a one-to-one branch produces a consistent output without duplicated detections. As a result, the non-maximum suppression stage is removed from inference [11]. For embedded platforms, this matters in practice because NMS is executed serially on the CPU, and on ARM architectures without a vectorized implementation its cost can reach tens of milliseconds per frame. YOLOv10n has 2.3 million parameters at 6.7 GFLOPs and achieves 38.5% mAP50-95, outperforming YOLOv8n in both accuracy and speed.

YOLOv11n and YOLOv11s (2024) represent the current generation, in which the C2f modules have been replaced by more compact C3k2 blocks, a C2PSA spatial attention module has been added, and the number of parameters has been reduced while preserving accuracy. With 2.6 million parameters (6.5 GFLOPs), YOLOv11n achieves 39.5% mAP50-95, the highest result among the nano-class models; YOLOv11s, with 9.4 million parameters (21.5 GFLOPs), reaches 47.0%, which is the maximum among all compared models. The consolidated characteristics are presented in Table 2.

Table 2 – Characteristics of the compared models (input 640×640, COCO val)

Model	Year	Param., million	GFLOPs	mAP50-95, %
YOLOv5n	2021	1,9	4,5	28,0
YOLOv8n	2023	3,2	8,7	37,3
YOLOv8s	2023	11,2	28,6	44,9
YOLOv10n	2024	2,3	6,7	38,5
YOLOv11n	2024	2,6	6,5	39,5
YOLOv11s	2024	9,4	21,5	47,0
YOLOv12n	2025	2,6	6,5	40,6
YOLOv26n	2025	2,4	5,4	40,1

Two newer architectures are included separately in Table 2, but they are excluded from the computational part of the study. YOLOv12n implements a spatial attention mechanism in place of part of the convolutional blocks and, according to formal metrics, outperforms YOLOv11n by 1.1 percentage points in mAP50-95 at the same computational complexity.

However, achieving the claimed speed requires optimized attention implementations designed for graphics accelerators; on general-purpose processors, attention operations are less efficiently vectorized than convolutions, so the actual inference time on ARM platforms is higher than that of YOLOv11n despite identical GFLOPs. An additional factor is lower stability during export: support for attention mechanisms in inference stacks for embedded platforms is incomplete, and conversion results vary across tool versions. YOLO26n is built on an anchor-free scheme without non-maximum suppression and, according to published measurements, outperforms YOLOv11n in both accuracy and speed. The limitation of this architecture lies in the requirements for the software environment: the model needs up-to-date versions of the training and export toolchain, support for which in stable builds of inference frameworks for ARM platforms was not available at the time of the study, and the full training cycle imposes higher computational demands than architectures of earlier generations. In view of this, both models are considered a direction for further research after the tool support has stabilized.

The comparison across generations reveals a clear trend: each successive nano-class generation delivers higher accuracy at lower or comparable computational complexity.

YOLOv11n outperforms YOLOv5n by 11.5 percentage points in mAP50-95 with only a 44% increase in complexity, while YOLOv10n achieves accuracy comparable to YOLOv11n with no NMS stage at all. At the same time, moving from the nano class to the small class within the same generation increases computational complexity by 3.3 times, whereas the accuracy gain is only 7.5–7.6 percentage points; the quantitative impact of this asymmetry for resource-constrained platforms is assessed below.

Comparison criteria and integral efficiency indicator

Model comparison is performed using four partial indicators: detection quality, effective speed, model size, and the energy consumed to process a single frame. These partial indicators are reduced to a single integral criterion, which formalizes model selection under a given requirement profile and eliminates the subjectivity of comparing models by only one characteristic.

Detection quality is characterized by the mAP50-95 metric on the COCO validation set, according to the official data provided by the developers [11, 19, 23]. Although the target domain of the task – images of explosive objects captured from flight altitude – differs from the COCO distribution, the relative ranking of models by accuracy after fine-tuning on a specialized dataset, as shown by the practice of transferring YOLO-

family models to applied domains [17, 21, 22], is preserved, which makes this indicator suitable for comparative analysis.

Speed is evaluated through a frame-processing time model. Inference time is assumed to be proportional to the model's computational complexity and inversely proportional to the platform's effective performance:

$$t_{\text{inf}} \approx C / P_{\text{eff}}, \quad (1)$$

where C – model's computational complexity, GFLOPs; P_{eff} – platform's effective performance for convolutional inference tasks, GFLOPS. The value of P_{eff} is determined by calibration against the measured inference time of a reference model: according to [19, 20], for YOLOv8n ($C=8.7$ GFLOPs) on Raspberry Pi 4 in NCNN format, the inference time is 414.7 ms, from which $P_{\text{eff}} \approx 21.0$ GFLOPS. For models for which direct measurements are unavailable, inference time is obtained by extrapolation using equation (1), calibrated with the reference YOLOv8n model for the nano class and YOLOv8s for the small class.

The total frame processing time t_{full} consists of preprocessing, inference, and postprocessing:

$$t_{\text{full}} = t_{\text{pre}} + t_{\text{inf}} + t_{\text{nms}}, \quad (2)$$

where t_{pre} – time spent on resizing the image resolution and normalization; t_{nms} – non-maximum suppression time, which for models with a classical detection head on ARM processors is estimated at 20 ms for Pi 4 and 30 ms for Pi 3B+, while for YOLOv10n it is zero by architectural design [11]. The effective system throughput FPS_{eff} is defined as:

$$FPS_{\text{eff}} = 1 / t_{\text{full}}. \quad (3)$$

The energy characteristic is expressed as the energy consumed to process a single frame:

$$E_{\text{fr}} = P_{\text{load}} \cdot t_{\text{full}}, \quad (4)$$

where P_{load} – the platform's average power consumption under computational load, in watts. Based on measurements [20], 6.0 W is assumed for Raspberry Pi 4 and 3.6 W for Raspberry Pi 3B+. Equation (4) reflects the dual effect of slow inference on a weaker platform: despite lower absolute power consumption, the energy per frame for Pi 3B+ may exceed that of Pi 4 because of the longer processing time, which, for a fixed onboard battery capacity, directly reduces the number of frames processed during a flight. The model size in single-precision format is estimated by the number of parameters:

$$M = 4 \cdot N_{\text{par}}, \quad (5)$$

where N_{par} – the number of model parameters; the coefficient 4 corresponds to the size of a single-precision number in bytes. For the Pi 3B+ platform with 1 GB of RAM, this metric limits the set of feasible models once the memory used by the operating system, the inference stack, and intermediate network activations is taken into account.

Because the partial metrics have different units and opposite optimization directions, they are normalized to the range before aggregation. For metrics to be maximized (mAP, FPS_{eff}), direct min-max normalization is used:

$$\tilde{X}_i = (X_i - X_{\min}) / (X_{\max} - X_{\min}), \quad (6)$$

for metrics to be minimized (λ), inverse normalization is used:

$$\tilde{X}_i = (X_{\max} - X_i) / (X_{\max} - X_{\min}), \quad (7)$$

where X_i – the value of the indicator for the i -th model; $X_{\min}$ and $X_{\max}$ are the minimum and maximum values of the indicator among the compared models, respectively.

The integral efficiency criterion of the i -th model E_i is defined as a weighted sum of the normalized partial indicators:

$$E_i = (w_1 \cdot mAP_i + w_2 \cdot FPS_i + w_3 \cdot M_i + w_4 \cdot E_i), \quad (8)$$

where $w_1 \dots w_4$ – the weighting coefficients reflect task priorities and satisfy the normalization condition that their sum equals one.

The weights are justified on substantive grounds. For the task of detecting explosive objects, missing an object has the highest cost, so the weight of the detection-quality indicator cannot be lower than the weights of the resource-cost indicators. Three scenarios are considered. The balanced scenario ($w = 0.40; 0.30; 0.15; 0.15$) corresponds to general-purpose requirements without a pronounced priority. The detection-completeness-priority scenario ($w = 0.55; 0.20; 0.125; 0.125$) models a careful survey of a limited area, where flight speed is reduced and the cost of a miss is maximal.

The speed-priority scenario ($w = 0.20; 0.55; 0.125; 0.125$) corresponds to a first-pass survey of large areas, where detections are subject to further verification and the decisive factor is the survey rate.

Table 4 – Performance and energy per frame rating (NCNN, 640×640)

Model	t, ms (Pi4)	FPS (Pi4)	E, J (Pi4)	t, ms (3B+)	FPS (3B+)	E, J (3B+)
YOLOv5n	235	4,26	1,41	674	1,48	2,43
YOLOv8n	435	2,30	2,61	1275	0,78	4,59
YOLOv10n	320	3,13	1,92	959	1,04	3,45
YOLOv11n	330	3,03	1,98	960	1,04	3,46
YOLOv8s	1062	0,94	6,37	3156	0,32	11,36
YOLOv11s	803	1,24	4,82	2380	0,42	8,57

Analysis of the data in Table 4 allows us to state the following. On the Raspberry Pi 4, nano-class models provide an effective speed of 2.3 - 4.3 frames per second; the fastest is YOLOv5n, the most accurate among the class is YOLOv11n at a speed of 3.0 frames per second. YOLOv10n, due to the absence of the NMS stage, reaches 3.1 frames per second - the second fastest result with an accuracy inferior to YOLOv11n by only one percentage point. The small class models on the Pi 4 operate at the limit of one frame per second, which is only suitable for frame-by-frame analysis of still scenes. On the Raspberry Pi 3B+, no model exceeds 1.5 frames per second, and small class

Comparative analysis based on benchmark data

The comparison is based on official performance measurements on the Raspberry Pi 4 platform and model characteristics from the developers' documentation [18, 19]. Table 3 presents the measured inference time of the YOLOv8n model on the Raspberry Pi 4 across the main software execution formats.

Table 3 – YOLOv8n inference time on Raspberry Pi 4 by execution format (640×640, FP32)

Format	Time, ms	Relative to NCNN
NCNN	414,7	1,00
OpenVINO	534,9	1,29
ONNX Runtime	560,0	1,35
TF SavedModel	816,5	1,97
TensorFlow Lite	950,3	2,29
PyTorch	1068,4	2,58

Data in Table 3 confirm NCNN as the primary execution format: its advantage over ONNX Runtime is 35%, and over TensorFlow Lite it is 129%. This is due to the optimization of NCNN's computational kernels for ARM NEON vector instructions and the more efficient distribution of workload across four cores without the overhead of general-purpose runtimes.

Table 4 summarizes the estimated total frame processing time, effective throughput, and energy per frame for six models on both platforms, obtained using formulas (1) – (4) calibrated against the measured values for YOLOv8n and YOLOv8s. For Raspberry Pi 3B+, slowdown coefficient of 3.0 relative to Pi 4 was applied, corresponding to the midpoint of the documented range 2.5 – 4.0 [18]; postprocessing time was increased in proportion to processor performance.

models require 2.4-3.2 s per frame; additionally, the deployment of YOLOv8s and YOLOv11s on a platform with 1 GB of memory is complicated by competition for memory with the operating system and inference stack. Energy estimates show that despite the lower power consumption, the Pi 3B+ consumes 4.59 J of 4.59 J on frame processing with the YOLOv8n model versus 2.61 J of the Pi 4, i.e. 1.76 times more. With a fixed energy budget of the onboard battery, this directly reduces the number of frames that can be processed in a single flight and makes the Pi 4 platform more energy efficient despite the higher absolute consumption.

The issue of external hardware inference accelerators, which are potentially capable of bringing performance closer to the target values, requires separate consideration. For Raspberry Pi platforms, two main solutions are available - Google Coral USB Accelerator with 4 TOPS performance in integer format and Hailo-8L modules with 13 TOPS performance. The use of both solutions as part of the on-board system based on the platforms in question was evaluated negatively for the following reasons.

The Hailo-8L module requires a PCIe interface, which is physically absent on the Raspberry Pi 3B+ and Pi 4, and is compatible only with the Raspberry Pi 5 platform through a specialized adapter, that is, it is not suitable for the target platforms of the study. Coral USB Accelerator is compatible with the USB interface, but is limited to the execution of only fully quantized integer models in TensorFlow Lite format with compilation for Edge TPU; the conversion of modern YOLO models to this format is accompanied by a degradation of accuracy in operations that are not supported by the accelerator and are rolled back to the CPU, and software support for the product by the developer is actually terminated.

Additional factors are an increase in the power consumption of the system by 2.0-2.5 W, complication of the power circuit due to peak currents of more than 900 mA, additional payload mass and the appearance of an additional failure point in the flight configuration. Given the totality of the above limitations, further analysis was performed for the platforms' own computing resources.

Calculation example of the application of the integral criterion

The application of criterion (8) is demonstrated for the calculation scenario adopted in Section 1: flight altitude 20 m, target video stream frequency 25 frames per second, Raspberry Pi 4 main platform. Normalization of partial indicators is performed according to formulas (6) and (7) on a set of six models; the initial data are the values of mAP50-95 from Table. 2, effective speed and energy per frame from Table. 4, size of models in ONNX format according to the documentation [18]. Normalized values of partial indicators and final values of the integral criterion for three scenarios of weighting factors are presented in Table. 5.

Table 5 – Normalized indicators and integral criterion according to three scenarios (Raspberry Pi 4)

Model	mAP	FPS	M	E	E (Balance)	E (Completeness)	E (Speed)
YOLOv5n	0,000	1,000	1,000	1,000	0,600	0,450	0,800
YOLOv8n	0,489	0,409	0,869	0,758	0,563	0,554	0,526
YOLOv10n	0,553	0,659	0,963	0,897	0,698	0,668	0,705
YOLOv11n	0,605	0,629	0,926	0,885	0,702	0,685	0,693
YOLOv8s	0,889	0,000	0,000	0,000	0,356	0,489	0,178
YOLOv11s	1,000	0,091	0,185	0,313	0,502	0,630	0,312

In a balanced scenario, the first two positions are occupied by YOLOv11n with an integral score of 0.702 and YOLOv10n with a score of 0.698; the difference of 0.004 is within the margin of error of the initial data, so the models are recognized as almost equivalent. In the scenario of the priority of completeness of detection, the leadership of YOLOv11n becomes more pronounced - 0.685 versus 0.668 for YOLOv10n - due to the highest accuracy among the nano class; the third position is occupied by YOLOv11s with a score of 0.630, for which the maximum accuracy partially compensates for the low performance. In the performance priority scenario, the first place is occupied by YOLOv5n with a score of 0.800, the minimum computational complexity of which outweighs the lowest accuracy; YOLOv10n with a score of 0.705 is ahead of YOLOv11n due to the absence of a non-maximal suppression stage. Small class models in all three scenarios are located at the bottom of the ranking: on the platforms in question, the increase in accuracy does not compensate for the loss of performance and energy efficiency.

The results obtained allow us to formulate practical recommendations. For further experimental research, the YOLOv11n model was chosen as the main one, which ranks first in two of the three scenarios, in particular, in the critical scenario for the problem of the priority of

completeness of detection. The YOLOv10n model is identified as an equivalent alternative, the feasibility of which increases with more stringent latency requirements; comparison of both models on a specialized dataset of images of explosive objects is the subject of the next stage of research. The YOLOv5n model retains a niche for the Raspberry Pi 3B+ platform, where its speed of 1.5 frames per second is the most achievable among the configurations considered.

A comparison of the achieved indicators with the target frequency of 25 frames per second reveals a gap of 5.8 times for the fastest configuration (YOLOv5n on the Pi 4) and 8.3 times for the recommended configuration (YOLOv11n on the Pi 4). Bridging the gap is possible in three ways, which are not mutually exclusive. Reducing the input resolution from 640×640 to 320×320 reduces the computational complexity by about four times, which is estimated (1) to increase the performance of the YOLOv11n to 10-12 frames per second at the cost of degrading detection of small objects. Thinning of the frame flow with the processing of each k-th frame allows you to match the detection rate with the flight speed: at a speed of 5 m/s and a viewing bandwidth of more than 15 m, the processing of three frames per second provides multiple overlap of the terrain without gaps. An additional reserve is the quantization of the model to an

integer format and other optimization methods that can provide an increase in performance by 1.5-2 times; the quantification of this reserve on target platforms is the subject of a separate work.

Conclusions

The problem of a reasonable choice of the architecture of the neural network detector of the YOLO family for the built-in system for detecting explosive objects on the computing platforms Raspberry Pi 3 Model B+ and Raspberry Pi 4 Model B is formulated.

A comparative analysis of six models of four generations of the YOLO family was performed according to official benchmarks with extrapolation to the research platforms. It was found that on the Raspberry Pi 4, nano-class models provide an effective speed of 2.3-4.3 frames per second in NCNN format, while small class models do not exceed 1.25 frames per second, and on the Raspberry Pi 3B+, only nano-class configurations retain practical suitability. The inexpediency of using external hardware accelerators for target platforms is substantiated: the Hailo-8L module is instructively incompatible due to the lack of a PCIe interface, and the Coral USB Accelerator is limited by an outdated software stack and creates additional energy and mass costs in the flight configuration.

Based on the results of the calculation of the integral criterion, the YOLOv11n model is recommended as the main one for further experimental research, since it ranks first in the balanced scenario and the completeness of detection priority scenario, and the YOLOv10n model as an almost equivalent alternative with the architectural advantage of the absence of the non-maximal suppression stage. The gap between the achievable speed and the target frequency of 25 frames per second by 5.8-

8.3 times is stated and the ways to overcome it are determined: reducing the resolution of the detector input, thinning the frame flow taking into account the flight speed and quantizing the model. Quantitative study of methods for optimizing models on target platforms is the direction of further work.

Author Contributions

Conceptualization, A.P. and O.S.; methodology, A.P.; validation, A.P. and O.S.; formal analysis, A.P.; investigation, A.P.; resources, O.S.; data curation, O.S.; writing-review and editing, A.P. and O.S. All authors have read and agreed to the published version of the manuscript.

Funding

This research received no external funding.

Conflicts of interest

The authors declare that they have no conflicts of interest in relation to the current study, including financial, personal, authorship, or any other, that could affect the study, as well as the results reported in this paper.

Data Availability Statement

No new data were created or analyzed in this study. Data sharing is not applicable to this article.

Use of artificial intelligence

During the preparation of this manuscript, the authors used ChatGPT (GPT-5) for the purposes of grammar checking and language refinement. The authors have reviewed and edited the output and take full responsibility for the content of this publication.

REFERENCE

1. Crowther, E., & Shepel, N. (2024), "Phase II Testing of UAV/Sensor Systems for Humanitarian Mine Action in Ukraine", *The Journal of Conventional Weapons Destruction*, 28(3), Article 8. Available at: <https://commons.lib.jmu.edu/cisr-journal/vol28/iss3/8/>.
2. Podorozhniak, A., Liubchenko, N., Skorpupin, O., Korolenko, S., & Stas, A. (2025), "Mobile explosive object detection system for humanitarian demining needs", *2025 IEEE 6th KhPI Week on Advanced Technology (KhPIWeek 2025)*, Kharkiv, Ukraine, pp. 1–6. doi: <https://doi.org/10.1109/KHPIWEEK61436.2025.11288620>
3. Levchenko, D., Podorozhniak, A., & Liubchenko, N. (2025), "Tools and methods for explosive objects detection using artificial intelligence and computer vision", *Control, Navigation and Communication Systems*, 3(81), pp. 117–121. doi: <https://doi.org/10.26906/SUNZ.2025.3.117>
4. United Nations Mine Action Service (2026), *Ukraine*. Available at: <https://unmas.org/en/where-we-work/ukraine>
5. Humanity & Inclusion (2026), *Landmines: use, contamination and civilian harm in Ukraine*. Available at: https://www.hi.org/sn_uploads/document/HI-Factsheet-Landmine-use-in-Ukraine-2026-final-ENG-V2_1.pdf
6. Podorozhniak, A., & Skorpupin, O. (2026), "Landmine detection using robotic systems based on multispectral video imaging", *Control, Navigation and Communication Systems*, 2026, 2(84), pp. 160–164, doi: <https://doi.org/10.26906/SUNZ.2026.2.160>.
7. Vivoli, E., Bertini, M., & Capineri, L. (2024), "Deep Learning-Based Real-Time Detection of Surface Landmines Using Optical Imaging", *Remote Sensing*, 16(4), 677. doi: <https://doi.org/10.3390/rs16040677>
8. Al-Slemani, A.S.A., & Abubakr, G. (2024), "Adaptive Landmine Detection and Recognition in Complex Environments using YOLOv8 Architectures", *Journal of Smart Systems Research*, 5(2), pp. 110–120. doi: <https://doi.org/10.58769/joinssr.1542886>
9. Dodić, D., Vujović, V., Jovković, S., Milutinović, N., & Trpkoski, M. (2025), "SAHI-Tuned YOLOv5 for UAV Detection of TM-62 Anti-Tank Landmines: Small-Object, Occlusion-Robust, Real-Time Pipeline", *Computers*, 14(10), 448. doi: <https://doi.org/10.3390/computers14100448>
10. Sapkota, R., & Karkee, M. (2025), "Ultralytics YOLO Evolution: An Overview of YOLO26, YOLO11, YOLOv8, and YOLOv5 Object Detectors for Computer Vision and Pattern Recognition", *arXiv:2510.09653 [cs.CV]*. doi: <https://doi.org/10.48550/arXiv.2510.09653>
11. Wang, A., Chen, H., Liu, L., Chen, K., Lin, Z., Han, J., & Ding, G. (2024), "YOLOv10: Real-Time End-to-End Object Detection", *arXiv:2405.14458 [cs.CV]*. doi: <https://doi.org/10.48550/arXiv.2405.14458>
12. Podorozhniak, A., Liubchenko, N., Skorpupin, O., & Oleksiienko, D. (2026), "Unmanned mine detection system using an optimized YOLO26 model for edge computing on STM32", *Proceedings of 16-th International Scientific and Technical*

- Conference "Current directions in the development of information and communication technologies and control tools", 2026, vol. 5, pp. 50–52, available at: <https://repository.kpi.kharkov.ua/handle/KhPI-Press/103564>.
13. Skorlupin, O., & Podorozhniak, A. (2024), "Optical methods for detecting explosive objects using autonomous unmanned systems", *Proceedings of 12-th International Scientific and Technical Conference "Problems of informatization"*, 2024, vol. 2, p. 132, available at: <https://repository.kpi.kharkov.ua/handle/KhPI-Press/87663>.
 14. Popov, M.O. (2022), "Remote mine detection technology based on the analysis of survey materials from unmanned aerial vehicles: status and prospects", *Visnyk of the National Academy of Sciences of Ukraine*, no. 5, pp. 25–36. doi: <https://doi.org/10.15407/vsn2022.05.056>
 15. Mishchuk, V., & Podorozhniak, A. (2024), "Analysis of Trade-Offs Between Accuracy and Speed of Real-Time Object Detectors for the Tasks of Explosive Ordnance Detection", *2024 IEEE 5th KhPI Week on Advanced Technology, KhPIWeek 2024*, Kharkiv, Ukraine, pp. 1–5. doi: <https://doi.org/10.1109/KHPIWEEK61434.2024.10878035>
 16. Kuchuk, H. and Malokhvii, E. (2024), "Integration of IoT with cloud, fog, and edge computing: a review", *Advanced Information Systems*, 8(2), pp. 65–78. doi: <https://doi.org/10.20998/2522-9052.2024.2.08>
 17. Podorozhniak, A., Liubchenko, N., Petrukovich, D., & Onishchenko, D. (2023), "Implementation of automatic license plate recognition system with Raspberry Pi", *Control, Navigation and Communication Systems*, 1(71), pp. 134–137. doi: <https://doi.org/10.26906/SUNZ.2023.1.134>
 18. Ultralytics Inc. (2024), "Raspberry Pi with Ultralytics YOLO: Benchmarks and Setup Guide", available at: <https://docs.ultralytics.com/guides/raspberry-pi/>
 19. Ultralytics Inc. (2026), "Ultralytics YOLO Documentation", available at: <https://docs.ultralytics.com/>
 20. Hashimov, E., Pashayev, A., & Khaligov, G. (2025), "Camera control algorithm and image quality assessment method to obtain a quality image", *Advanced Information Systems*, 9(3), pp. 50–56, 2025, doi: <https://doi.org/10.20998/2522-9052.2025.3.06>
 21. Podorozhniak, A., Liubchenko, N., Sobol, M., & Onishchenko, D. (2023), "Usage of Mask R-CNN for automatic license plate recognition", *Advanced Information Systems*, 7(1), pp. 54–58. doi: <https://doi.org/10.20998/2522-9052.2023.1.09>
 22. Bajić, M., & Potočnik, B. (2023), "UAV Thermal Imaging for Unexploded Ordnance Detection by Using Deep Learning", *Remote Sensing*, 15(4), 967. doi: <https://doi.org/10.3390/rs15040967>
 23. Khanam, R., & Hussain, M. (2024), "YOLOv11: An Overview of the Key Architectural Enhancements", *arXiv:2410.17725 [cs.CV]*. doi: <https://doi.org/10.48550/arXiv.2410.17725>

Received (Надійшла) 22.06.2026

Accepted for publication (Прийнята до друку) 09.09.2026

Publication date (Дата публікації) 25.09.2026

ВІДОМОСТІ ПРО АВТОРІВ / ABOUT THE AUTHORS

Скорлупін Олександр Володимирович – аспірант кафедри комп'ютерної інженерії та програмування, Національний технічний університет "Харківський політехнічний інститут", Харків, Україна;

Oleksandr Skorlupin – postgraduate student, Computer Engineering and Programming Department, National Technical University "Kharkiv Polytechnic Institute", Kharkiv, Ukraine;

e-mail: oleksandr.skorlupin@cit.khpi.edu.ua; ORCID Author ID: <https://orcid.org/0009-0001-4295-4115>;

Scopus ID: : <https://www.scopus.com/authid/detail.uri?authorId=60435005700>.

Подорожняк Андрій Олексійович – кандидат технічних наук, доцент, професор кафедри комп'ютерної інженерії та програмування, Національний технічний університет "Харківський політехнічний інститут", Харків, Україна;

Andrii Podorozhniak – Candidate of Technical Sciences, Associate Professor, Professor of Computer Engineering and Programming Department, National Technical University "Kharkiv Polytechnic Institute", Kharkiv, Ukraine;

e-mail: andrii.podorozhniak@khpi.edu.ua; ORCID Author ID: <https://orcid.org/0000-0002-6688-8407>;

Scopus ID: <https://www.scopus.com/authid/detail.uri?authorId=57202229410>.

Архітектури сімейства YOLO для виявлення вибухонебезпечних об'єктів у вбудованих системах автономних безпілотних засобів: порівняльний аналіз

О. В. Скорлупін, А. О. Подорожняк

Анотація. Актуальність. Необхідність вибору ефективної архітектури YOLO для виявлення вибухонебезпечних об'єктів у вбудованих системах безпілотних літальних апаратів зумовлена обмеженими обчислювальними ресурсами, вимогами до роботи в реальному часі та енергоефективності. **Об'єктом дослідження** є виявлення вибухонебезпечних об'єктів у вбудованих автономних безпілотних системах із використанням архітектур сімейства YOLO. **Предметом дослідження** є порівняльна ефективність моделей YOLO за точністю виявлення, пропускну здатністю, розміром моделі та енергоспоживанням на вбудованих апаратних платформах. **Метою цієї статті** є розроблення та застосування інтегрального критерію ефективності, заснованого на неймережевих методах, для вибору найпридатнішої архітектури YOLO для виявлення вибухонебезпечних об'єктів і завдань гуманітарного розмінування. **Результати дослідження.** Запропонований критерій було застосовано до шести моделей YOLO (v5n, v8n, v8s, v10n, v11n, v11s), розгорнутих на Raspberry Pi 3 Model B+ та Raspberry Pi 4 Model B з 4 ГБ оперативної пам'яті. Порівняльне оцінювання за трьома сценаріями показало суттєві відмінності в якості виявлення, пропускій здатності, розмірі моделі та енергоспоживанні. Отримане ранжування підтверджує практичну цінність методики для вибору моделі у вбудованих системах. **Висновки.** Методика забезпечує прозору й відтворювану основу для порівняння архітектур YOLO в ресурсно обмежених середовищах. Робота в реальному часі залишається складним завданням на малопотужних платформах, що зумовлює подальшу оптимізацію розгортання та інференсу.

Ключові слова: YOLO; виявлення вибухонебезпечних об'єктів; безпілотний літальний апарат; вбудовані системи; інтегральний критерій ефективності; нейронна мережа; гуманітарне розмінування; Raspberry Pi.