electronics

MDPI

Article

Edge HPC Architectures for AI-Based Video Surveillance Applications

Federico Rossi * and Sergio Saponara

Department of Information Engineering, University of Pisa, 56122 Pisa, Italy; sergio.saponara@unipi.it
* Correspondence: federico.rossi@ing.unipi.it

Abstract: The introduction of artificial intelligence (AI) in video surveillance systems has significantly transformed security practices, allowing for autonomous monitoring and real-time detection of threats. However, the effectiveness and efficiency of AI-powered surveillance rely heavily on the hardware infrastructure, specifically high-performance computing (HPC) architectures. This article examines the impact of different platforms for HPC edge servers, including x86 and ARM CPU-based systems and Graphics Processing Units (GPUs), on the speed and accuracy of video processing tasks. By using advanced deep learning frameworks, a video surveillance system based on YOLO object detection and DeepSort tracking algorithms is developed and evaluated. This study thoroughly assesses the strengths, limitations, and suitability of different hardware architectures for various AI-based surveillance scenarios.

Keywords: HPC architectures; machine learning; image processing; edge computing; YOLO

Citation: Rossi, F.; Saponara, S. Edge HPC Architectures for AI-Based Video Surveillance Applications. *Electronics* **2024**, *13*, 1757. https://doi.org/10.3390/electronics13091757

Academic Editor: David Defour

Received: 6 April 2024
Revised: 26 April 2024
Accepted: 1 May 2024
Published: 2 May 2024

1. Introduction

Recent advancements in artificial intelligence (AI) have ushered in a transformative era for video surveillance, revolutionizing its capabilities across various domains. AI-driven video surveillance applications have emerged as powerful tools for enhancing security and surveillance in public spaces, critical infrastructures, and commercial premises [1,2]. Leveraging computer vision and machine learning algorithms, these applications autonomously analyze video feeds, detecting anomalies, identifying objects and individuals, and issuing real-time alerts to operators.

However, the efficacy and functionality of AI video surveillance systems are heavily contingent on the underlying hardware infrastructure [3,4]. The deployment of high-performance computing (HPC) architectures is pivotal for achieving efficient and accurate video processing capabilities [5–7]. With their immense computational prowess and parallel processing capabilities, HPC systems hold the potential to significantly enhance the speed and precision of video analysis tasks, facilitating real-time monitoring and swift responses to security threats [8].

In this paper, we embark on a comprehensive exploration of an AI-powered video surveillance application, employing a range of High-Performance Computing (HPC) architectures. Our primary goal is to meticulously examine the influence of various architectural configurations, encompassing traditional CPU-based systems and GPUs, on both the efficiency and speed of video processing operations. Through a rigorous comparative analysis of these architectural designs against a predefined set of evaluation metrics, we aim to elucidate their strengths, limitations, and suitability across a spectrum of surveillance scenarios. Our investigation delves into the intricate interplay between hardware architecture and the performance of AI-driven video surveillance systems. By systematically evaluating the efficacy of different HPC configurations, we aim to provide valuable insights into the optimal utilization of computational resources for enhancing surveillance capabilities.

To conduct our investigation, we harness cutting-edge deep learning frameworks and libraries. Specifically, we present a video surveillance application founded on the renowned

YOLO [9] object detection framework, considered in the overall system architecture shown in Figure 1, integrated with the DeepSort [10] tracking algorithm, implemented on the Pytorch framework. Employing a meticulous benchmarking methodology, we assess crucial performance parameters such as processing power, resource utilization, and energy efficiency. Moreover, we juxtapose findings from prominent computing systems such as x86 and ARM 64b CPUs, combined with GPU technology, within the realm of AI video processing. Figure 1 shows the system architecture envisioned in this work: several video surveillance appliances record and stream data (e.g., videos, images) to several EDGE and HPC platforms. These platforms independently perform AI-based video surveillance tasks, and their processed images are conveyed together to a control room for human evaluation.

The structure of this paper is as follows: Section 2 presents a brief review of related works in the field of AI and video-surveillance applications and architectures; Section 3 introduces the people-down application in the context of video surveillance, delineating its constituent elements and useful search and rescue in critical scenarios such as war, earthquake or other natural disasters; Section 4 expounds upon the benchmark setup; and Section 5 presents the benchmark results for the people-down application and draws some conclusions. All the work source code can be publicly found at https://github.com/federicorossifr/eupilot-cini-mandown (accessed on 2 May 2024).

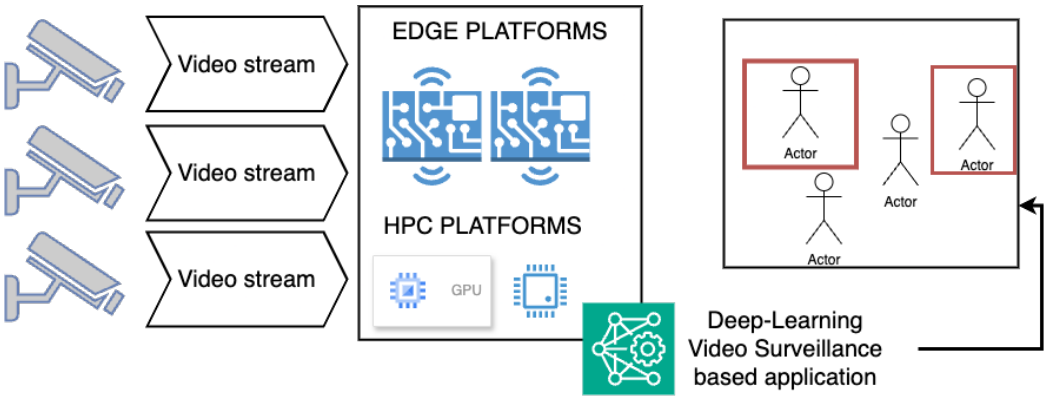


Figure 1. Overall system architecture, including camera surveillance system and deep-learning based edge-HPC system.

2. Related Works

The integration of artificial intelligence (AI) into video surveillance systems has catalyzed a significant evolution in security practices, ushering in an era of autonomous monitoring and real-time threat detection. However, the efficacy and efficiency of AI-powered surveillance systems are intricately tied to the underlying hardware infrastructure, particularly high-performance computing (HPC) architectures [8,11,12]. As such, a growing body of research has emerged to investigate the impact of different HPC platforms on the speed and accuracy of video processing tasks with a focus on enhancing surveillance capabilities.

Several studies have explored the performance of various edge and HPC architectures in the context of AI-driven video surveillance and real-time object detection [4,13–15]. Their findings highlighted the potential of ARM and FPGA-based architectures in delivering competitive performance while offering improved energy efficiency compared to traditional x86 platforms. In [16], the authors reviewed the capabilities of three different units of HPC platforms: CPUs, GPUs, and Tensor Processing Units (TPUs), each of them serving unique functions within computing systems. The authors highlighted how the CPU primarily manages overall system performance, while the GPU is dedicated to rendering or processing graphics and video, possibly through AI. Acting as an additional hardware component alongside the CPU, the GPU enhances image and video processing capabilities.

On the other hand, the TPU finds its niche in fields like artificial intelligence, machine learning, and deep learning, offering specialized processing tailored to these domains [17].

Furthermore, researchers have explored novel architectural configurations to optimize the efficiency of AI-based surveillance systems. In [18], the authors proposed a hybrid architecture combining ARM-based CPUs, GPUs, and cameras to implement an AI-based surveillance system for a railway-crossing. Meanwhile, in [15,19–21], the authors presented potential applications of deep-learning techniques for crowd analysis and identification and for security assessment in architecture, engineering, and construction.

Object Detection

One of the core tasks in computer vision via deep learning is object detection [22], which includes recognizing and localizing entities in images or video streams. It is essential to many applications, including augmented reality, medical imaging, autonomous driving, and, of particular interest in this work, video surveillance.

Fundamentally, deep learning object detection is based on convolutional neural networks (CNNs), which are a subclass of deep learning models created especially for visual data analysis. A CNN is made up of several layers of linked neurons that process different parts of the input image and gradually learn to extract features that are important for detecting objects.

The region-based convolutional neural network (R-CNN) family of algorithms is one of the primary methods used in object detection with deep learning [23,24]. R-CNNs work by first producing a set of candidate bounding boxes, or region proposals, that might include objects in the image. After that, a CNN is fed these suggestions, and it extracts features from each region. Lastly, a classifier is used to identify and ascertain whether objects are present in each proposition.

By enhancing the detection process's accuracy and computing efficiency, later developments including Mask R-CNN, Fast R-CNN, and Faster R-CNN [25–27] have significantly enhanced object detection performance by building upon R-CNNs. These models efficiently create region proposals and extract multi-scale features from input pictures by using methods like region proposal networks (RPNs) [28] and feature pyramid networks (FPNs) [29].

Another notable approach in object detection with deep learning is the single-shot detection (SSD) framework [30–32], which streamlines the detection process by directly predicting object bounding boxes and class probabilities from feature maps at multiple scales. SSD models are renowned for their real-time performance and suitability for applications requiring fast inference speeds.

The object identification algorithm known as YOLO (You Only Look Once) [32] transformed computer vision by quickly and precisely identifying objects in images in real time. With just one neural network, YOLO can predict bounding boxes and the associated class probabilities for several objects at once in a single pass. Because of this method's remarkable speed, YOLO can be used in applications that need real-time object identification, like augmented reality, autonomous driving, and surveillance. Because of its effectiveness and efficiency, YOLO has become a mainstay in the object detection space. Iterations like YOLOv4 and YOLOv5 up to YOLOv8 (https://github.com/ultralytics/ultralytics, accessed on 2 May 2024) have all pushed the envelope in terms of speed and accuracy.

Moreover, transformer-based designs and the emergence of attention processes have propelled recent developments in object detection. Traditional anchor-based methods are no longer necessary thanks to models like DETR (DEtection TRansformer) [33], which use self-attention mechanisms to directly predict object locations and classes from full images in a single pass.

3. The People-Down Application

In this section, we introduce the AI-based video processing application that will be used as a benchmark in the next sections. This application automatically detects, tracks,

and counts people lying on the ground/floor in a video sequence. It comprises three main parts, which are executed for each of the video frames. Figures 2 and 3 shows an example output from the application.

Figure 2. Example of the output of a processed frame.

- An object detector: responsible for detecting objects of interest (OOIs) in a given frame and predicting bounding boxes around them
- A people-down classifier: responsible for filtering out boxes that are either not associated with people or associated with people not lying on the ground.
- A tracking algorithm: responsible for fingerprinting each detected lying person, following their movement across the frames. This is useful to avoid counting the same people multiple times in case the camera is moving (if mounted in a mobile drone) or the same people are observed by multiple cameras from different point of views.

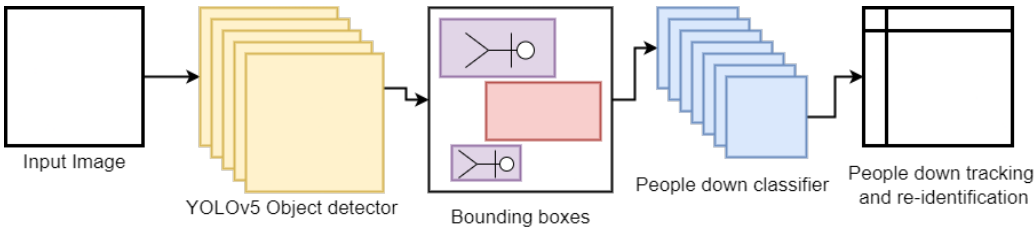


Figure 3. Data-flow diagram for the people-down application with all the processing steps.

3.1. The Object Detector

The object detection task is implemented using the YOLOv5 [32] One-shot Object Detector algorithm. This task detects all the OOIs in a frame and then discards the objects not classified as people. From a computing perspective, the algorithm completes three steps for each frame:

- Image pre-processing: each video frame is transformed to accommodate the correct size and format accepted by the YOLOv5 neural network model.
- Neural network inference: the proper step of OOI detection using the YOLOv5 neural network
- Result post-processing: YOLOv5 predictions are filtered and selected based on prediction confidence and class labels.

3.1.1. Image Pre-Processing

The object detector model requires each image to be in the correct format; depending on the original frame format, this results in at least the following steps:

- Image resizing: a 640×640 frame is required for the YOLOv5 input (smaller images are padded with black pixels)
- Channel transposing: each resized frame is transposed to match the channel layout CHW (channel, height, width) and RGB (red, green, blue) color formats.
- Data conversion and normalization: pixel data are converted from an 8-bit unsigned integer to a floating-point format on 16-bit or 32-bit (or even 8-bit, [34]). This also normalizes each pixel range from $[0, 255]$ to $[0, 1]$.
- Batch preparation: multiple images (if available) are batched together to increase parallel computation (when possible) of YOLOv5 inference.

3.1.2. Image Inference

Figure 4 details the process of the YOLOv5 inference pass (*bb* is the number of bounding boxes and *nc* is the number of class probabilities). From an architectural point of view, the YOLOv5 network is constructed as follows:

- Feature maps are extracted from the input image with a *backbone* of convolutional and pooling filters.
- Refinement of previous feature maps is performed at the *neck* of the architecture exploiting spatial fusion [35]. Spatial fusion techniques aim to merge these multi-scale feature maps to achieve more robust object detection. This fusion process helps the model capture fine-grained details of small objects and contextual information of larger objects in a single pass through the network. It also ensures that the detector can detect objects across different scales. There are various methods for spatial fusion, including concatenation, addition, or more sophisticated operations like spatial pyramid pooling or feature pyramid networks (FPNs, [29]). These techniques enable the model to leverage information from different scales and spatial resolutions.
- The feature maps output by the *neck* are used to finally predict and classify bounding boxes and classes for which the network was originally trained.

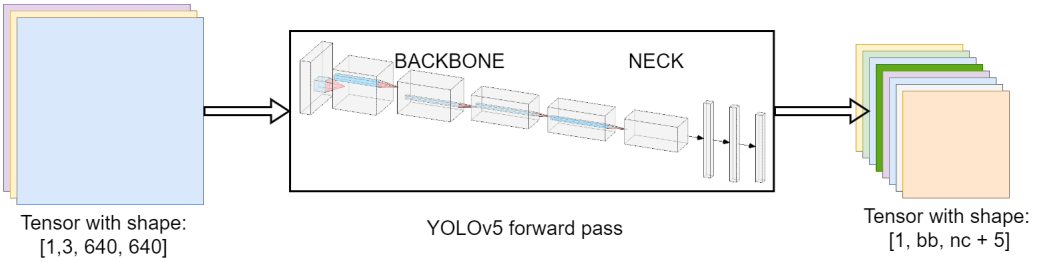


Figure 4. Workflow of YOLOv5 inference.

As shown in Figure 4, the output of the model is a three-dimensional tensor:

$$[\text{batch}, \text{bounding boxes}, \text{number of classes} + 5]$$

The first index refers to the original index of the frame in the batch if multiple images are provided. The second index refers to the bounding boxes that the network has detected. The $five$ additional items added to the 3rd dimension are the bounding box coordinates and score x, y, w, h, o_{score}. Indeed, for each of the bounding boxes, we obtain the pixel coordinates x, y and the dimension w, h, and the probability of each class label, $class_{score}$. Finally, the o_{score} measures the probability of having an OOI inside that bounding box. For example, if we use a YOLOv5 model trained on the COCO dataset [36], we obtain a tensor of roughly $[1, 25200, 80 + 5]$ for a single image, where there are 80 class probabilities and $five$ additional coordinates are added to the 3rd dimension of the tensor. Post-processing then must be used to reduce and select the actual OOIs between all the bounding boxes.

3.1.3. Image Post-Processing

The selection of the best bounding boxes is critical to the overall network accuracy. This process encompasses the following steps:

- Confidence-based filtering and scoring: any object with a o_{score} below a predefined threshold is discarded. Then, each predicted classed is given a *confidence score* $c_{score} = o_{score} \cdot \text{class}_{score}$ (e.g., for the COCO dataset, we will obtain a confidence score of 80 for each bounding box). Finally, only the highest confidence class is kept as a class for that bounding box.
- Class-based filtering: this step is specific to our use case; since we are only interested in people, we discard all the boxes that are not classified as person/human
- Non-maximum-suppression (NMS): all the bounding boxes are sorted by their confidence score computed before, and the box with the best score is appended to the list of selected boxes. All boxes with a high (i.e., above threshold) overlap, namely intersection over union (IoU), with one of the selected boxes are discarded. The threshold is typically set to 0.5.

At the end of this step, we obtain a list of bounding boxes with the following information: (i) pixel coordinates of the top left and bottom right corner of the bounding box, (ii) the confidence score for that bounding box, (iii) the predicted class for that bounding box.

3.2. People-Down Classifier and Tracking

After we obtained the list of bounding boxes containing people, we must process them to select only the ones that contain people that are lying on the ground or floor. Each bounding box is rescaled to the original frame dimension, and we retrieve the width and the height of the boxes. Then, we discard all the boxes that have an aspect ratio $\frac{\text{width}}{\text{height}}$ less than a predefined threshold a_t. This means that depending on the camera positioning, we are only keeping boxes that contain people lying down.

3.2.1. People Re-Identification

Since we want to track and count the number of people that are lying on the floor, we must employ a mechanism to keep track of the same person between different frames. This task can be labeled as an Object Re-Identification (re-ID) task [37]. One of the most effective techniques that solve this task is the Omni-Scale Feature Learning (OSFL, [29,38]) with a neural network model called OSNet. The idea behind the OSFL approach is similar to the YOLOv5 spatial fusion of multi-scale feature maps, being able to perceive both fine-grained details and global information. Figure 5 shows the base architecture of the neural network model used for the OSFL.

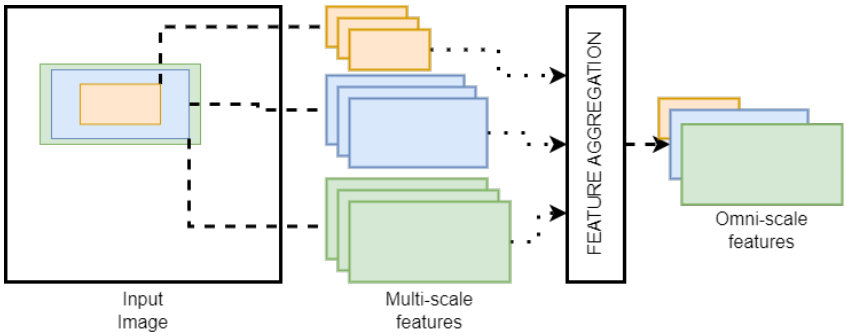


Figure 5. Omni-scale network for people re-identification architecture.

During the feature extraction process, a sequence of bottleneck blocks is used, each comprised of three convolutional layers. The first layer is a 1×1 convolutional layer that reduces the number of input channels, thereby simplifying the model. The second layer is a 3×3 depthwise convolutional layer that applies a single filter to each input channel

to extract spatial features. Finally, the third layer is another 1×1 convolutional layer that expands the number of output channels to include channel-wise information.

To facilitate the network's learning of deeper and more intricate features, the bottleneck blocks integrate a residual connection that adds the output of the final convolutional layer to the block's input. This also helps mitigate the risk of vanishing gradients during training.

After feature extraction, a global average pooling layer aggregates the features. This layer takes the output of the final bottleneck block and performs an average pooling operation across the spatial dimensions of the feature maps, resulting in a singular 512-dimensional vector of feature values that represent the fingerprint of that person in the bounding box.

3.2.2. People Tracking

Once we obtain the re-identification for the people in the bounding boxes, we want to track them across frames. A way to solve this problem is the Simple Online and Real-time Tracking (DeepSORT, [39]) algorithm. Globally, this approach is based on four components:

Detection

This task can be performed by the detector model shown in Section 3.1. The output of this step is exactly the set of bounding boxes coordinates alongside the classes and their confidence score.

Estimation

Estimation of movement of boxes across frames can be achieved by applying a Kalman Filter [40]. In particular, we define the filter state as $s = (x, y, r, h, \dot{x}, \dot{y}, \dot{a}\ )$, where x, y are the midpoint coordinates of the bounding box, h is the height of the box, r is the aspect ratio of the box, and the other quantities $\dot{x}, \dot{y}$, and $\dot{a}$ are the corresponding derivatives with respect to time. Since we can assume a constant aspect ratio due to the nature of the task, the state transition matrix F of the filter is the following with Δt being the inter-frame time.

$$F \in \mathrm{R}^{7\times 7} = \begin{pmatrix} 1 & 0 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & \Delta\, t \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

The observation matrix H is then the following:

$$H \in \mathrm{R}^{4\times 7} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Data Association and Track Management

Given a set of tracked boxes, data association consists of assigning each tracked box to one of the elements of the current frame detected boxes, using the fingerprint features computed before. Each tracked box is maintained for a given number of frames A_{max}. At each frame, a cost matrix C is computed where the element c_{ij} is the IoU between the i-th detection in the current frame and the j-th tracked bounding box. Then, a linear assignment problem is set and solved using the Hungarian algorithm [41], which allows us to find an assignment that maximizes the IoU between tracked and detected boxes. Algorithm 1 shows a pseudo-implementation of the association and tracking algorithm.

Algorithm 1 Matching algorithm

Require: Track indices $T = \{1, \ldots, N\}$, Detection indices $D = \{1, \ldots, M\}$, Maximum age A_{max}
Ensure: Matched detections M, Unmatched detections D
  Compute cost matrix $C \leftarrow c_{i,j}\ \forall i, j \in T, D$
  Initialize set of matches $M \leftarrow \varnothing$
  Initialize set of unmatched $U \leftarrow D$
  for $N \in 1, \ldots, A_{max}$ **do**
    Select tracks by age $T_n \leftarrow \{i \in T \mid age_i = n\}$
    Minimum cost matching $X = \{x_{i,j} | i \in T, j \in D\} = \text{hungarian}(C, T_n, U)$
    Update matches $M \leftarrow M \cup \{(i, j) | x_{i,j} > 0\}$
    Update unmatched $U \leftarrow U \setminus \{(j | \sum_i x_{i,j} > 0)\}$
  end for

4. Methodology and Benchmark Setup

In this section, we present the evaluation of our AI video surveillance application employing diverse HPC architectures. Our assessment focuses on key performance metrics including processing power, resource utilization, and energy efficiency. We begin by detailing the benchmark setup, which is followed by a comprehensive analysis of benchmark metrics for the people-down tracking application. The whole application described before was implemented in Python using the Pytorch framework for the neural network models and OpenCV for handling video sources. GPU metrics were collected using the `pynvml` python package, while CPU metrics were collected by using both hardware-related tools and the python `psutil` package when possible.

We conducted our evaluation using a variety of HPC architectures, encompassing both conventional CPU-based systems and GPUs. The hardware configurations included systems based on x86 and ARM 64b architectures, which were each equipped with different specifications in terms of CPU/GPU models, memory, and storage. The evaluation was performed using a dataset of 400 video frames extracted from a surveillance scenario with few people lying on the ground. Each platform was evaluated with the same software benchmark, analyzing the metric described in the previous section. Each frame is a JPEG-encoded image with a resolution of 1920×1080 pixels and an 8-bit `sRGB` color scheme.

For each platform, we ran the same detection and re-identification algorithm shown in the previous sections 100 times over the 400 frames of the dataset. For each processed frame, we collected the metrics mentioned above. At the end of the evaluation, we computed each metric's mean, variance, and distribution for the comparison.

We selected the following metrics and measured their instantaneous value for each of the processed frames in every evaluation run.

- Processing Power: We quantified the processing power of each architecture by measuring the average frames per second (FPS) achieved during video processing tasks. Higher FPS values indicate greater computational efficiency and faster real-time monitoring capabilities.
- Resource Utilization: Resource utilization metrics, including CPU and GPU utilization rates, were monitored throughout the evaluation. By analyzing resource usage patterns, we gained insights into the effectiveness of hardware acceleration and parallel processing capabilities offered by different architectures.
- Energy Efficiency: Energy consumption was assessed to evaluate the energy efficiency of each HPC architecture. Power consumption measurements were recorded during video processing tasks, allowing us to compare the energy efficiency of CPU and GPU-based systems under varying workloads when possible.

We considered the following HPC computing architectures available at the GreenDataCenter of the University of Pisa:

- NVIDIA Grace Hopper super chip (GH200) with a 72-core ARM CPU (Neoverse V2 64b architecture with vectorized instruction-set extension), NVIDIA H100 GPU, and 480 GB unified memory.
- NVIDIA A100 GPU with an Intel Xeon Gold 6238R 28-core CPU.
- Ampera Altra ARM Neoverse 64b N1 CPU with 80-cores ARM64 enabled with the ARM NEON vector ISA.
- NVIDIA T4 GPU with an Intel Xeon Cascadelake 8-core CPU, 23 GB main memory, and 15 GB GPU memory.
- NVIDIA Jetson Orin AGX with 12 ARM 64b Cortex-A78 cores, Ampere-based GPU, and 32 GB unified memory.

We may allocate the processing steps shown in Section 3 to the GPU or the CPU depending on the considered architecture. Table 1 shows how we allocated the different steps for the benchmarks above.

Table 1. Device allocation for each computing step and architecture considered.

	GH200		**A100 + INTEL**		**ARM N1**		**JETSON**		**T4 + INTEL**	
	CPU	**GPU**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**
Image pre/post	✓		✓		✓		✓		✓	
YOLOv5		✓		✓	✓			✓		✓
People classifier		✓		✓	✓			✓		✓
DeepSORT		✓		✓	✓			✓		✓

5. Results and Discussion

In this section, we present the findings of our comprehensive evaluation of various HPC architectures in the context of an AI-based video surveillance application. Building upon the groundwork laid out in the preceding sections, where we introduced the transformative potential of AI-driven video surveillance and outlined our methodology for benchmarking different architectures, the focus now shifts to the outcomes of our analysis. From the achieved results in Figures 6 and 7, in terms of FPS processed in real time, the use of a GPU makes an essential contribution. Indeed, Ampera Altra ARM Neoverse N1 performs worse than all the others. Particularly, the ARM Neoverse core has higher performance than the Cortex-A78 CPU in the Jetson Orin but the latter, thanks to the GPU part, has an FPS rate higher than 2. The performance results are roughly independent of the specific task performed; see Figure 7a,b. The best for performance is mixing a CPU with a vectorized instruction set (ARM Neoverse V2) plus a GPU. As we can see from Figure 8, in terms of energy efficiency, mixing Cortex-A architecture with Nvidia GPU in the Orin AGX leads to a figure of 1 W per 1 FPS, while in the Grace Hopper GH200, the efficiency drops to 3 W per 1 FPS. Roughly 1 W per 1 FPS is achieved also by the NVIDIA T4 GPU with an Intel Xeon Cascadelake 8-core CPU.

Our results highlight how important GPUs are to improving edge servers' real-time computing power for video surveillance. The significant boost in frame-per-second (FPS) rates that GPU acceleration provides emphasizes how important parallel computing is for meeting the computational demands of sophisticated AI algorithms. But this performance boost comes at the cost of increased power consumption; therefore, energy efficiency indicators must be carefully taken into account in addition to processing power.

Although GPU-accelerated architectures are widely used, our research also emphasizes the promise of ARM-based systems, especially when combined with GPU co-processors. Despite its relatively inferior performance when compared to its x86 counterparts, the Ampera Altra ARM Neoverse N1 showed encouraging results in terms of performance-per-watt when ARM cores were integrated with GPUs, as demonstrated by platforms such as the Jetson Orin. This emphasizes how important it is to investigate heterogeneous designs to maximize energy efficiency without compromising too much processing power.

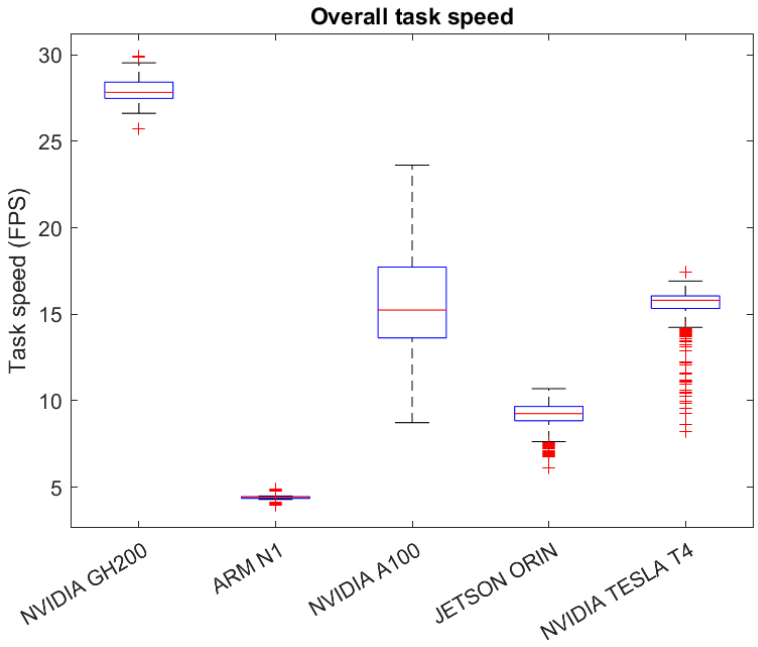
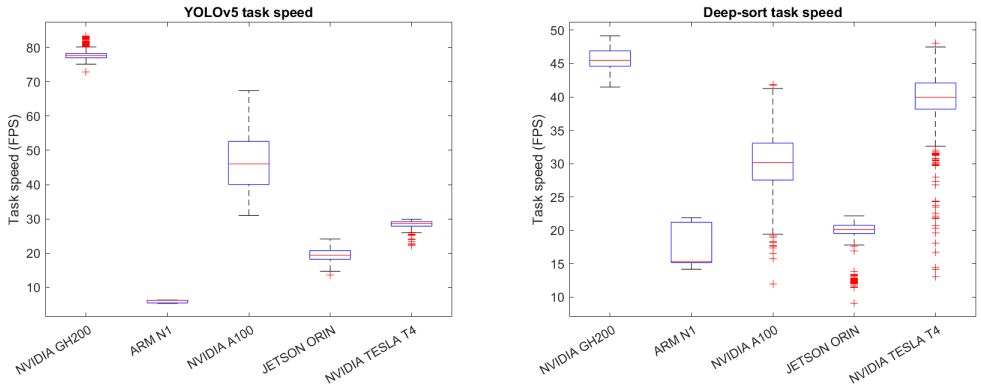
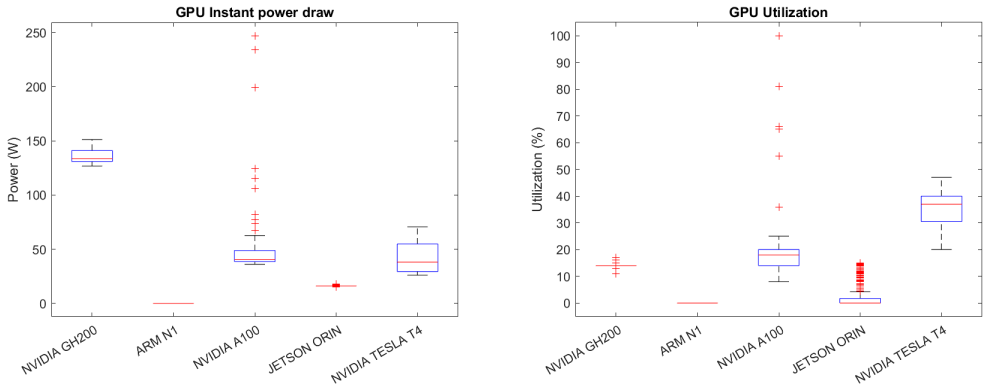


Figure 6. Overall FPS performance.



(**a**) YOLOv5 component FPS performance (**b**) DeepSort component FPS performance

Figure 7. Mean, variance, and distribution of FPS performance of people-down application modules for the different architectures.



(**a**) GPU power draw distribution (**b**) GPU utilization distribution

Figure 8. Mean, variance, and distribution of GPU performance metrics for the overall people-down application for the different architectures.

Additionally, the Intel Xeon Cascadelake combination with a Tesla T4 GPU was found to be another attractive option with significant performance-per-watt benefits by our evaluation team. This emphasizes how crucial it is to take into account the synergistic impacts of components within a heterogeneous architecture in addition to their individual performance.

6. Conclusions

In this work, we examined the impact of different platforms for HPC edge servers, including x86 and ARM CPU-based systems and GPUs, on the speed and accuracy of video processing tasks. By using advanced deep learning frameworks, a video surveillance application based on YOLO object detection and DeepSort tracking algorithms was developed and evaluated. We then assessed the strengths, limitations, and suitability of heterogeneous architectures to run such applications. For each system, we measured several metrics, in particular, GPU power consumption and FPS speed. The results highlight the critical role that GPUs play in enhancing edge servers' real-time processing capacity for AI video surveillance as well as the critical role that parallel computing plays in addressing complex algorithms' computational demands in the context of edge and high-performance computing platforms for AI video surveillance applications. Energy-efficiency indicators must be carefully evaluated in addition to computing power, even while GPU acceleration improves FPS rates dramatically at the expense of power consumption. Moreover, the research underscores the promise of ARM-based systems, particularly in conjunction with GPU co-processors like the Ampera Altra ARM Neoverse N1, which demonstrates encouraging performance-per-watt results. Moreover, investigating heterogeneous designs—such as combining Tesla GPUs with Intel Xeon processors—offers significant gains in performance per watt, emphasizing the importance of synergistic effects in edge HPC platforms for AI video surveillance. Future works will expand the analysis to other algorithms for AI video surveillance and other EDGE/HPC platforms and architectures.

Author Contributions: Conceptualization, F.R. and S.S.; methodology, F.R. and S.S.; software, F.R.; validation, F.R. and S.S.; formal analysis, F.R. and S.S.; investigation, F.R. and S.S.; resources, F.R. and S.S.; data curation, F.R. and S.S.; writing—original draft preparation, F.R.; writing—review and editing, S.S.; visualization, F.R.; supervision, S.S.; project administration, S.S.; funding acquisition, S.S. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by EU Horizon 2020 Research and Innovation projects The European Pilot under Grant 101034126, TextaRossa under Grant 956831, and in part by the Italian Ministry of University and Research (MUR) in the framework of the Crosslab and FoReLab projects (Departments of Excellence).

Data Availability Statement: All the work source code can be publicly found at https://github.com/federicorossifr/eupilot-cini-mandown (accessed on 2 May 2024).

Acknowledgments: We thank the personnel of the Green DataCenter of the University of Pisa (https://start.unipi.it/en/computingunipi) (accessed on 2 May 2024). In particular, we thank M. Davini and F. Pratelli for having provided us with the computational resources that have been used in the experimental section.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Sharma, V.; Gupta, M.; Kumar, A.; Mishra, D. Video Processing Using Deep Learning Techniques: A Systematic Literature Review. *IEEE Access* **2021**, *9*, 139489–139507. [CrossRef]
2. Rossi, F.; Mugnaini, G.; Saponara, S.; Cavazzoni, C.; Sciarappa, A. Evaluation of AI and Video Computing Applications on Multiple Heterogeneous Architectures. In *Applications in Electronics Pervading Industry, Environment and Society*; Bellotti, F., Grammatikakis, M.D., Mansour, A., Ruo Roch, M., Seepold, R., Solanas, A., Berta, R., Eds.; Lecture Notes in Electrical Engineering; Springer: Cham, Switzerland, 2024; pp. 130–136. [CrossRef]
3. Wang, X. Intelligent multi-camera video surveillance: A review. *Pattern Recognit. Lett.* **2013**, *34*, 3–19. [CrossRef]

4. Dilshad, N.; Hwang, J.; Song, J.; Sung, N. Applications and Challenges in Video Surveillance via Drone: A Brief Survey. In Proceedings of the 2020 International Conference on Information and Communication Technology Convergence (ICTC), Jeju, Republic of Korea, 21–23 October 2020; pp. 728–732. [CrossRef]
5. Webb, J. High performance computing in image processing and computer vision. In Proceedings of the 12th IAPR International Conference on Pattern Recognition, Vol. 2—Conference B: Computer Vision and Image Processing. (Cat. No.94CH3440-5), Jerusalem, Israel, 9–13 October 1994; Volume 3, pp. 218–222. [CrossRef]
6. Cococcioni, M.; Rossi, F.; Ruffaldi, E.; Saponara, S. Fast deep neural networks for image processing using posits and ARM scalable vector extension. *J. Real-Time Image Process.* **2020**, *17*, 759–771. [CrossRef]
7. Cococcioni, M.; Rossi, F.; Ruffaldi, E.; Saponara, S. Vectorizing posit operations on RISC-V for faster deep neural networks: Experiments and comparison with ARM SVE. *Neural Comput. Appl.* **2021**, *33*, 10575–10585. [CrossRef]
8. Yi, S.; Jing, X.; Zhu, J.; Zhu, J.; Cheng, H. The Model of Face Recognition in Video Surveillance Based on Cloud Computing. In *Advances in Computer Science and Information Engineering*; Jin, D., Lin, S., Eds.; Springer: Berlin/Heidelberg, Germany, 2012; pp. 105–111.
9. Jocher, G.; Chaurasia, A.; Stoken, A.; Borovec, J.; Kwon, Y.; Michael, K.; Fang, J.; Yifu, Z.; Wong, C.; Montes, D.; et al. *ultralytics/yolov5: V7.0—YOLOv5 SOTA Realtime Instance Segmentation*; Version v7.0; Zenodo: Meyrin, Switzerland, 2022. [CrossRef]
10. Wojke, N.; Bewley, A. Deep Cosine Metric Learning for Person Re-identification. In Proceedings of the 2018 IEEE Winter Conference on Applications of Computer Vision (WACV), Lake Tahoe, NV, USA, 12–15 March 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 748–756. [CrossRef]
11. Sacchi, C.; Regazzoni, C.S.; Dambra, C. Remote cable-based video surveillance applications: The AVS-RIO project. In Proceedings of the 10th International Conference on Image Analysis and Processing, Venice, Italy, 27–29 September 1999; IEEE: Piscataway, NJ, USA, 1999; pp. 1214–1215.
12. Jabłoński, M.; Przybyło, J. Evaluation of MoG Video Segmentation on GPU-based HPC System. *Comput. Inform.* **2016**, *35*, 1141.
13. Baobaid, A.; Meribout, M.; Tiwari, V.K.; Pena, J.P. Hardware Accelerators for Real-Time Face Recognition: A Survey. *IEEE Access* **2022**, *10*, 83723–83739. [CrossRef]
14. Shemonaev, D.; Gal, B.L.; Jego, C.; Besseau, A. Implementation of an Assignment Algorithm for Object Tracking on a FPGA MPSoC. In Proceedings of the 2023 26th Euromicro Conference on Digital System Design (DSD), Golem, Albania, 6–8 September 2023; pp. 373–380. [CrossRef]
15. Wang, Q.; Gao, J.; Lin, W.; Li, X. NWPU-Crowd: A Large-Scale Benchmark for Crowd Counting and Localization. *IEEE Trans. Pattern Anal. Mach. Intell.* **2021**, *43*, 2141–2149. [CrossRef]
16. Nikolić, G.S.; Dimitrijević, B.R.; Nikolić, T.R.; Stojcev, M.K. A Survey of Three Types of Processing Units: CPU, GPU and TPU. In Proceedings of the 2022 57th International Scientific Conference on Information, Communication and Energy Systems and Technologies (ICEST), Ohrid, North Macedonia, 16–18 June 2022; pp. 1–6. [CrossRef]
17. Jouppi, N.P.; Young, C.; Patil, N.; Patterson, D.; Agrawal, G.; Bajwa, R.; Bates, S.; Bhatia, S.; Boden, N.; Borchers, A.; et al. In-Datacenter Performance Analysis of a Tensor Processing Unit. In Proceedings of the 44th Annual International Symposium on Computer Architecture, Toronto, ON, Canada, 24–28 June 2017.
18. Sikora, P.; Malina, L.; Kiac, M.; Martinasek, Z.; Riha, K.; Prinosil, J.; Jirik, L.; Srivastava, G. Artificial Intelligence-Based Surveillance System for Railway Crossing Traffic. *IEEE Sens. J.* **2021**, *21*, 15515–15526. [CrossRef]
19. Sreenu, G.; Durai, S. Intelligent video surveillance: A review through deep learning techniques for crowd analysis. *J. Big Data* **2019**, *6*, 1–27. [CrossRef]
20. Rane, N.; Choudhary, S.; Rane, J. Artificial Intelligence (AI) and Internet of Things (IoT)-Based Sensors for Monitoring and Controlling in Architecture, Engineering, and Construction: Applications, Challenges, and Opportunities. 2023. Available online: https://ssrn.com/abstract=4642197 (accessed on 5 April 2024).
21. Qian, Y.; Zhang, L.; Hong, X.; Donovan, C.; Arandjelovic, O. Segmentation Assisted U-shaped Multi-scale Transformer for Crowd Counting. In Proceedings of the 33rd British Machine Vision Conference 2022, BMVC, London, UK, 21–24 November 2022; BMVA Press: Durham, UK, 2022.
22. Szegedy, C.; Toshev, A.; Erhan, D. Deep neural networks for object detection. *Adv. Neural Inf. Process. Syst.* **2013**, *26*, 1–9.
23. Chen, C.; Liu, M.Y.; Tuzel, O.; Xiao, J. R-CNN for small object detection. In Proceedings of the Computer Vision—ACCV 2016: 13th Asian Conference on Computer Vision, Taipei, Taiwan, 20–24 November 2016; Revised Selected Papers, Part V 13; Springer: Berlin/Heidelberg, Germany, 2017; pp. 214–230.
24. Hmidani, O.; Alaoui, E.I. A comprehensive survey of the R-CNN family for object detection. In Proceedings of the 2022 5th International Conference on Advanced Communication Technologies and Networking (CommNet), Marrakech, Morocco, 12–14 December 2022; IEEE: Piscataway, NJ, USA, 2022; pp. 1–6.
25. Liu, B.; Zhao, W.; Sun, Q. Study of object detection based on Faster R-CNN. In Proceedings of the 2017 Chinese Automation Congress (CAC), Jinan, China, 20–22 October 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 6233–6236.
26. He, K.; Gkioxari, G.; Dollár, P.; Girshick, R. Mask r-cnn. In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 22–29 October 2017; pp. 2961–2969.
27. Girshick, R. Fast r-cnn. In Proceedings of the IEEE International Conference on Computer Vision, Santiago, Chile, 7–13 December 2015; pp. 1440–1448.

28. Tang, P.; Wang, X.; Wang, A.; Yan, Y.; Liu, W.; Huang, J.; Yuille, A. Weakly supervised region proposal network and object detection. In Proceedings of the European Conference on Computer Vision (ECCV), Munich, Germany, 8–14 September 2018; pp. 352–368.
29. Lin, T.Y.; Dollár, P.; Girshick, R.; He, K.; Hariharan, B.; Belongie, S. Feature Pyramid Networks for Object Detection. *arXiv* **2017**, arXiv:1612.03144.
30. Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S.; Fu, C.Y.; Berg, A.C. Ssd: Single shot multibox detector. In Proceedings of the Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, 11–14 October 2016; Proceedings, Part I 14; Springer: Berlin/Heidelberg, Germany, 2016; pp. 21–37.
31. Zhang, Z.; Qiao, S.; Xie, C.; Shen, W.; Wang, B.; Yuille, A.L. Single-shot object detection with enriched semantics. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–22 June 2018; pp. 5813–5821.
32. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection. *arXiv* **2016**, arXiv:1506.02640.
33. Carion, N.; Massa, F.; Synnaeve, G.; Usunier, N.; Kirillov, A.; Zagoruyko, S. End-to-end object detection with transformers. In Proceedings of the European Conference on Computer Vision, Glasgow, UK, 23–28 August 2020; Springer: Berlin/Heidelberg, Germany, 2020; pp. 213–229.
34. Cococcioni, M.; Rossi, F.; Ruffaldi, E.; Saponara, S. Small Reals Representations for Deep Learning at the Edge: A Comparison. In Proceedings of the Next Generation Arithmetic: Third International Conference, CoNGA 2022, Singapore, 1–3 March 2022; Revised Selected Papers; Springer: Berlin/Heidelberg, Germany, 2022; pp. 117–133. [CrossRef]
35. Liu, S.; Huang, D.; Wang, Y. Learning Spatial Fusion for Single-Shot Object Detection. *arXiv* **2019**, arXiv:1911.09516.
36. Lin, T.Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; Zitnick, C.L. Microsoft COCO: Common Objects in Context. In Proceedings of the Computer Vision—ECCV 2014, Zurich, Switzerland, 6–12 September 2014; Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T., Eds.; Springer: Cham, Switzerland, 2014; pp. 740–755.
37. Leng, Q.; Ye, M.; Tian, Q. A Survey of Open-World Person Re-Identification. *IEEE Trans. Circuits Syst. Video Technol.* **2020**, *30*, 1092–1108. [CrossRef]
38. Zhou, K.; Yang, Y.; Cavallaro, A.; Xiang, T. Omni-Scale Feature Learning for Person Re-Identification. In Proceedings of the 2019 IEEE/CVF International Conference on Computer Vision (ICCV), Seoul, Republic of Korea, 27 October–2 November 2019; pp. 3701–3711. [CrossRef]
39. Wojke, N.; Bewley, A.; Paulus, D. Simple online and realtime tracking with a deep association metric. In Proceedings of the 2017 IEEE International Conference on Image Processing (ICIP), Beijing, China, 17–20 September 2017; pp. 3645–3649. [CrossRef]
40. Li, Q.; Li, R.; Ji, K.; Dai, W. Kalman Filter and Its Application. In Proceedings of the 2015 8th International Conference on Intelligent Networks and Intelligent Systems (ICINIS), Tianjin, China, 1–3 November 2015; pp. 74–77. [CrossRef]
41. Kuhn, H.W. The Hungarian method for the assignment problem. *Nav. Res. Logist. Q.* **1955**, *2*, 83–97. [CrossRef]