Can We Trust AI-Powered Real-Time Embedded Systems?

Giorgio Buttazzo
Department of Excellence in Robotics & AI, Scuola Superiore Sant'Anna, Pisa, Italy

Abstract

The excellent performance of deep neural networks and machine learning algorithms is pushing the industry to adopt such a technology in several application domains, including safety-critical ones, as self-driving vehicles, autonomous robots, and diagnosis support systems for medical applications. However, most of the AI methodologies available today have not been designed to work in safety-critical environments and several issues need to be solved, at different architecture levels, to make them trustworthy. This paper presents some of the major problems existing today in AI-powered embedded systems, highlighting possible solutions and research directions to support them, increasing their security, safety, and time predictability.

2012 ACM Subject Classification Computer systems organization

Keywords and phrases Real-Time Systems, Heterogeneous architectures, Trustworthy AI, Hypervisors, Deep learning, Adversarial attacks, FPGA acceleration, Mixed criticality systems

Digital Object Identifier 10.4230/OASIcs.NG-RES.2022.1

Category Invited Paper

1 Introduction

Embedded computing platforms are becoming more complex every day to manage the increasing computational load generated by emerging applications, as autonomous vehicles (cars, trains, drones, aircrafts), advance robotic systems, intelligent appliances, and so on. Such systems are equipped with a variety of sensors that produce a large amount of data, hence demanding for real-time processing and high-performance computing. To provide the required computational power, computer architectures are evolving towards heterogeneous platforms that integrate on the same board, or even on the same chip, multicore processors of different types, field programmable gate arrays (FPGAs), general purpose graphics processing units (GPGPUs), and special co-processors optimized for executing operations on tensors, as tensor processing units (TPUs).

Although new tools and libraries are becoming available every year, developing safety-critical applications on top of such heterogeneous platforms, while providing the required guarantees, is quite difficult due to a number of non-trivial problems. The following list presents just a few of such problems, related to the use of artificial intelligence (AI) in safety-critical applications with real-time constraints.

- The interaction among the various components through the shared resources available on the computing platform (as buses, memories, and I/O devices) generates a significant amount of interference, introducing large and unpredictable delays on the computational activities. Such a large variability in responding to external events makes it is very difficult to provide timing guarantees on the application behavior. This is also a serious problem for the certification of safety-critical software components.
- Programming modern heterogeneous platforms requires a deep knowledge of low-level details of the architecture, which prolongs the developing and testing times.

- Distributing the computational activities in an optimal way between hardware accelerators and processors is not trivial, but it can make a huge difference in the overall system performance, as well as in satisfying the real-time application constraints.
- Deep neural networks (DNNs) are commonly developed and inferred by means of state-of-the-art frameworks (e.g., Tensorflow, Caffe, and PyTorch), which greatly simplify the implementation of new models. Unfortunately, however, none of the current frameworks is specifically optimized to be used in safety-critical environments, nor capable of providing bounded response times. This prevents their use in real-time applications like autonomous driving, where DNNs should have a highly predictable behavior, not only in the functional domain, but also in the time domain, responding within specific deadlines.
- The use of deep learning algorithms and the related frameworks increases the software attack surface, posing serious security issues for the overall system. This problem is exacerbated by the fact that such frameworks usually run on top of rich operating systems, as Linux, which are more vulnerable to cyber attacks.
- In spite of their excellent capabilities in perception tasks, deep neural networks have been shown to be prone to adversarial attacks, i.e., malicious inputs with imperceptible perturbations that force a neural network to produce a wrong output with a high confidence score.
- Similar threats derive from inputs that significantly differ from the distribution of the training set. Predicting the behavior of a neural network on such inputs is not easy and, in some cases, the network could also respond with a wrong output with a high confidence score.
- Finally, since the behavior of a neural network is not explicitly programmed, but encoded in a huge number of parameters, interpreting the output of a neural model and deciding whether its prediction can be trusted is a challenging task.

To address the issues described above, a lot of research is being devoted to support the development of AI-powered applications on top of heterogeneous platforms for safety-critical real-time systems.

The remainder of the paper is organized as follows: Section 2 discusses the problems and preliminary solutions related to architecture issues; Section 3 presents problems and promising solutions related to security issues; Section 4 describes the major threats caused by the use of AI algorithms and some solutions aimed at mitigating them; Section 5 proposes an architectural approach to address all the issues discussed above; and Section 6 states the conclusions and outlines some promising research lines.

2 Architecture issues

To be used in real time, the inference of modern DNN models requires hardware acceleration. This can be achieved by exploiting modern heterogeneous computing platforms equipped by GPUs or programmable hardware, as FPGA. This section discusses the main problems related to such computing platforms and presents existing solutions to them.

2.1 General platform issues

To cope with the different computational requirements of real-time applications, modern heterogeneous computing platforms integrate different processing elements, as multi-core processors of different types, general purpose GPGPUs, FPGAs, and tensor processing units.

The concurrent accesses to shared devices existing on such architectures, as buses, memory controllers, and high-level caches, create a significant interference on the computations, introducing long and variable delays in application tasks.

For instance, Cavicchioli et al. [12] observed significant and variable delays when using GPU acceleration on heterogeneous embedded platforms due to the contention occurring on shared memory, especially for memory-intensive GPU tasks. Restuccia et al. [29] identified some anomalous situations that can arise in an AXI bus arbiter in FPGA-based SoC and proposed a reservation mechanism to prevent this phenomenon and restore fairness during bus transactions.

A timing analysis has also been proposed [28] to bound the execution of periodically-invoked hardware accelerators in nominal conditions. This analysis can be used to configure a latency-free hardware module named AXI Stall Monitor (ASM) to detect and safely solve possible stalls during AXI bus transactions. Efforts have also been devoted to analytically bound the delay experienced by AXI bus transactions issued by hardware accelerators on FPGA [30].

Hardware acceleration typically involves memory-intensive computations. Therefore, an accurate control of the memory traffic is crucial to achieve predictability in the execution of HW-tasks.

Pagani et al. [26] proposed a bandwidth reservation mechanism for AXI-based transactions on FPGAs able to control the bus traffic generated by hardware accelerators. The mechanism, named Memory Budget and Protection Unit (MBPU), aims at "shielding" hardware accelerators from excessive or unpredictable memory interference. MBPUs are installed between AXI master ports and the interconnect, enforcing a given budget of memory transactions within a periodic interval of time. Budgets are recharged in a periodic fashion and are configurable from the CPU via memory-mapped registers. MBPUs also protect the system from unrestricted accesses to memory by HW-tasks: this is accomplished by masking the accesses that fall outside a set of configurable memory address spaces.

2.2 GPU-related issues

Today, the most common way for accelerating DNNs is by executing them on a GPU-based platform. This solution has two main advantages: (i) the response time can be reduced by two orders of magnitude and (ii) the development is supported by standard frameworks. However, GPUs also have disadvantages. First, they are closed systems and multiple tasks are scheduled in a non preemptive fashion. This means that, if the system includes multiple neural networks with different complexity and periodicity requirements, those with shorter periods will be more likely to experience longer delays and higher response time variability. An example of non-preemptive schedule of three DNNs with different execution times and periods is illustrated in Figure 1.

Notice that, in this example, the total GPU utilization is less than one (U = 0.95), but DDN1 is forced to skip the second and the fifth execution instance, because it cannot preempt the execution of DNN3.

To solve this problem, Capodieci et al. [9], in collaboration with NVIDIA, proposed to modify the GPU internal scheduler with a preemptive scheduler based on Earliest Deadline First (EDF) [18], also providing bandwidth isolation by means of a Constant Bandwidth Server (CBS) [3]. Unfortunately, however, this solution is not yet available on commercial NVIDIA GPU platforms.

Other problems with GPU acceleration are due to the high power consumption and their significant weight and encumbrance, which prevent their usage in small embedded systems, as unmanned aerial vehicles (UAVs).

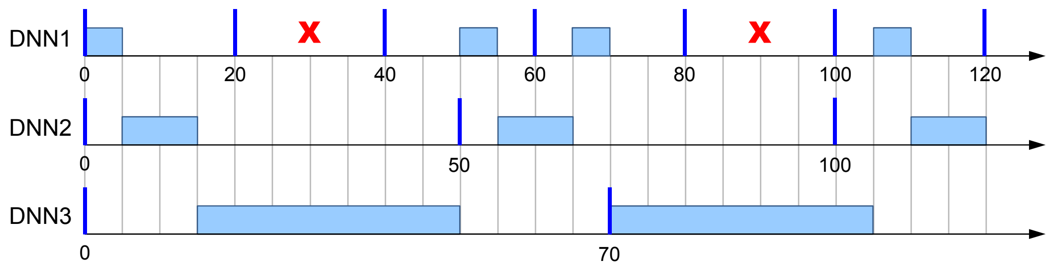

Figure 1 Example of non-preemptive schedule of three DNNs with different execution times and periods. As clear from the figure, DNN1 experiences longer and variable delays.

2.3 FPGA-related issues

An interesting alternative to GPUs for accelerating AI algorithms is provided by FPGAs. They are integrated circuits designed to be configured after manufacturing for implementing arbitrary logic functions in hardware. As such, they exhibit a highly predictable behavior in terms of execution times. In addition, they consume much less power with respect to GPUs and existing commercial platforms are characterized by lower weight, encumbrance, and cost. Hence, they represent an ideal solution for being used on battery-operated embedded systems with size, weight, power and cost (SWaP-C) constraints, as space robots, satellites, and UAVs.

Nevertheless, FPGAs have other problems when used as DNNs accelerators:

- No floating point unit (FPU) is available on the chip, unless it is explicitly programmed by the user, but consuming a significant fraction of the available fabric.
- Programming FPGAs is quite more difficult than programming CPUs or GPUs, and efficient coding requires a deep knowledge of low-level architecture details.
- The frameworks available today for developing AI applications on FPGA-based platforms are less rich and flexible than those available for GPUs, and the same is true for related libraries and tools.
- The overall FPGA area available in medium size SoCs could be insufficient to host more than one DNN, or even a single large DNN.

To overcome the problems outlined above, a lot of research has been carried out in the recent years.

The absence of an FPU is overcome by performing a preliminary parameter quantization to convert floating point numbers into integers with n-bit precision. Several quantization methods have been proposed in the literature [17], including symmetrical, asymmetrical, non-uniform, and statistical. An extreme quantization converts weights into binary numbers using the sign function. Courbariaux, Bengio, and David [14] have shown that a binarized DNN can achieve 98.8% accuracy in classifying the handwritten digits of the MNIST dataset. Other optimization steps (e.g., network pruning and layer fusion) can also be performed, both on GPUs and FPGAs, to reduce the computation time and the memory footprint of trained DNNs while minimizing the loss in accuracy.

To overcome the limitation of the FPGA area, Biondi et al. [6] proposed a programming framework, called FRED[1], to support the design, development, and execution of predictable software on FPGAs. FRED exploits dynamic partial reconfiguration and recurrent execution

[1] See details on `http://fred.santannapisa.it`.

to virtualize the FPGA area, thus enabling the user to allocate a larger number of hardware accelerators than those that could otherwise be fit into the physical fabric. FRED also integrates a tool for automated floorplanning [33] and a set of runtime mechanisms to enhance predictability by scheduling hardware resources and regulating bus/memory contentions [29].

An application targeted by FRED consists of a set of software tasks (SW-tasks) running on the CPU cores that can periodically invoke the execution of hardware accelerators (HW-tasks) to be dynamically programmed on the FPGA. The communication scheme adopted between SW-tasks and HW-tasks is illustrated in Figure 2.

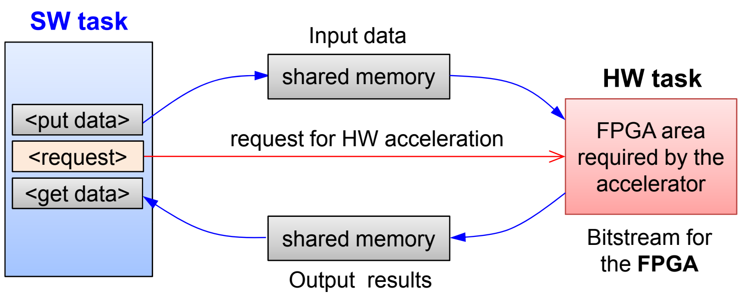

Figure 2 Communication scheme adopted between SW-tasks and HW-tasks in FRED.

The FPGA virtualization is achieved through a timesharing mechanism that replaces inactive accelerators (i.e., those that finished their computation and are waiting for the next activation) with active ones. In this way, the total number of HW-tasks that can run on the FPGA can be much higher than the number of HW-tasks that would statically fit in the physical area available on the fabric. Hence, this mechanism virtualizes the FPGA by creating a virtual area much larger than the physical one. The resulting approach is similar to multitasking, where tasks continuously change their context, or a virtual memory mechanism, where memory pages are swapped between hard disk and dynamic memory.

Thanks to a set of design choices and a proper scheduling infrastructure, resource contention delays experienced by tasks running under FRED are bounded and predictable, and hence they can be estimated to verify the system schedulability.

A full support for FRED has been developed under both FreeRTOS and Linux [25, 24]. The Linux support comes with a user-space daemon and a set of custom kernel drivers to handle the processor configuration port (PCAP) and the shared-memory communication buffers between CPUs and FPGA. A preemptable reconfiguration interface has also been developed by Rossi et al. [31] to achieve a finer control in scheduling the reconfiguration requests and a better control on the reconfiguration delays incurred by HW-tasks.

2.4 Framework issues

DNNs are commonly developed and inferred by means of state-of-the-art frameworks, as Tensorflow, Caffe, and PyTorch. Unfortunately, such frameworks are not optimized for being used in real-time applications and they are not supported by commercial real-time operating systems, as VxWorks and QNX. As a consequence, DNN tasks may be subject to a variable interference.

Casini et al. [10] addressed this problem by modifying the internal scheduler of the TensorFlow framework and adapting it for the SCHED_DEADLINE scheduling class of Linux. Extensive experiments demonstrated the effectiveness of the approach, showing a significant reduction of both average and longest-observed response times of TensorFlow tasks.

Recently, Restuccia and Biondi [27] proposed a set of techniques for accelerating DNNs on FPGA-based platforms with a highly predictable timing behavior under the Vitis AI frameworks by Xilinx. In Vitis AI, the execution of the DNN layers relies on the deep learning processing unit (DPU) core, a hardware accelerator optimized for the execution of convolutional DNNs. Based on an extensive profiling campaign conducted on the Xilinx Zynq Ultrascale+ platform, they proposed an execution model for the DPU employed to derive a response time analysis for guaranteeing real-time applications constraints.

3 Security issues

Safety-critical systems that make use of AI algorithms consist of several components with different complexity and requirements.

Consider for example a self-driving car. The functions responsible for steering, throttle modulation, braking, and engine control are highly critical and must satisfy stringent requirements in terms of safety, security, and real-time behavior. As such, they need to be managed by a real-time operating system, that must be certified to guarantee the required safety integrity levels.

On the other hand, high-level functions related to sensory perception, object tracking, and vehicle localization, which heavily rely on AI algorithms, need to be executed on a rich operating system (e.g., Linux) to exploit all the available device drivers, libraries, and development frameworks required for such complex computations. These components are far from being certified and offer a large software surface for cyber attacks.

In 2015, two hackers, Charlie Miller and Chris Valasek, discovered a vulnerability in the Jeep Cherokee, which they exploited to remotely access the vehicle and gain physical control, including steering, braking, turning on the wipers, blasting the radio, and finally, killing the engine to bring the vehicle to a complete stop [15]. They wrote a long paper [19] where they explain how they accessed the CAN bus through the infotainment system, detailing the full attack chain.

Although cyber attacks to non critical components accessible by a wireless network cannot be avoided completely, the security of a system can greatly be enhanced by preventing such attacks from spreading to more critical components. This can be achieved by *isolating* software components with different level of criticality into different execution domains, through the use of a hypervisor.

3.1 Hypervisor-based architecture

A hypervisor is a software layer above the hardware platform able to create and manage multiple execution domains, each hosting a virtual machine with its own operating system.

An example of a hypervisor-based architecture for safety-critical embedded systems is the one proposed by the SPHERE project [7], which supports the creation of multiple virtual machines on the same computing platform, providing both time/memory isolation, security, real-time communication channels, and I/O virtualization to allow different virtual machines to shared the peripheral devices.

Figure 3 shows an example of a hypervisor managing two execution domains with different levels of criticality. One domain hosts a virtual machine running all safety-critical functions on a real-time operating system (RTOS), while the other domain hosts another virtual machine running AI-powered software on the Linux operating system. Such an architecture has been

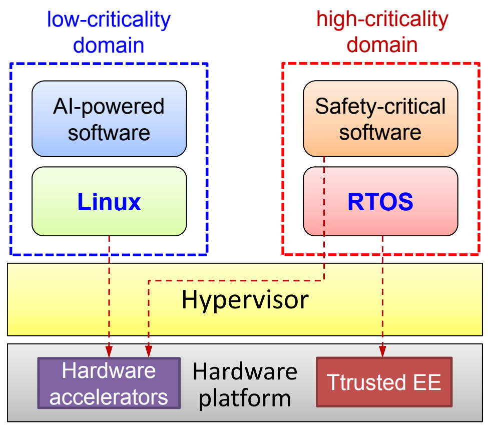

Figure 3 Example of a hypervisor managing two execution domains with different criticality.

successfully implemented and tested on a number of AI-powered control applications using CLARE [2], a novel hypervisor purposely designed to support mixed-criticality real-time systems exploiting AI hardware acceleration in heterogeneous embedded platforms.

3.2 CLARE hypervisor

The CLARE hypervisor is a novel bare-metal (type-1) hypervisor at the core of the CLARE software stack [2]. It integrates cutting-edge mechanisms to host safe, secure, and time-predictable virtual machines that can execute in isolation upon the same hardware platform. CLARE hypervisor follows a fully-static approach with off-line configurations and optimization to allocate the onboard resources to virtual machines. It has been designed to support modern heterogeneous platforms, such as GPGPU- and FPGA-based SoC, to better exploit and control their computational resources. In particular, it provides a number of real-time and security features that make it suitable for safety-critical systems, such as

- Improved key management and attack detection under control flow integrity by pointer authentication code [16].
- Hardware-based isolation exploiting the ARM TrustZone technology to perform key management and provide attack detection and recovery strategies.
- Virtualization of trusted execution environments leveraging ARM TrustZone [13].
- Protection mechanisms at the hypervisor level to control temporal and spatial interference among domains, also preventing side-channel attacks [20].
- I/O device virtualization and I/O related memory contention control, with related latency analysis [11].
- FPGA virtualization to allow multiple domains to exploit hardware accelerators in isolation.

4 AI related issues

Deep neural networks have shown an impressive performance in several recognition tasks, but their suitability for mission-critical applications has been questioned by Szegedy et al. [35] and many other authors [34], who showed that imperceptible perturbations added to an input sample can fool a neural network in perceiving objects that are not present in the input. Such perturbed inputs are called adversarial examples (AEs) and represent a serious

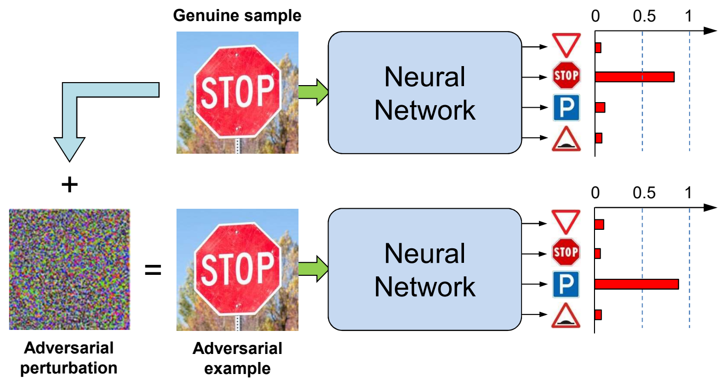

Figure 4 Example in which a neural network is able to correctly classify a genuine image of a stop sign (top). However, the same image can be modified by adding an adversarial perturbation, so that it is classified as a parking sign with a high confidence score (bottom).

threat for the security of AI-based systems. An example of adversarial image is shown in Figure 4, where the picture of a stop sign is perturbed in such a way that it is perceived by the network as a parking sign with a high confidence score.

Although a significant effort has been spent to develop defense methods against adversarial examples [5], the problem remains open and challenging, since these attacks violate the fundamental stationarity assumption of learning algorithms, i.e., that training and testing data are drawn from the same distribution.

The trustworthiness of DNNs is also threatened by genuine inputs characterized by a distribution that is quite different from that of the training samples. Such inputs are referred to as *out-of-distribution* (OoD) samples. Two examples of OoD images are shown in Figure 5.

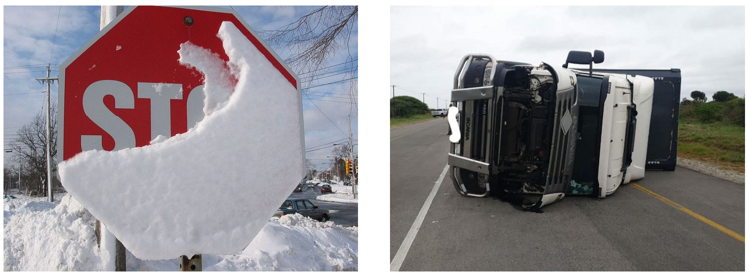

Figure 5 Two examples of OoD images that could cause a deep neural network to produce a wrong output.

Considering that the prediction score of a DNN can be high in the presence on both AEs and OoD samples, the output score of the best classified class cannot be considered as an indication of the prediction confidence of the model.

Several methods have been proposed in the literature to detect AEs and OoD samples. Two of these methods are presented below with more details.

4.1 Detection by input transformations

One method for detecting AEs relies on the fact that DNN models are usually robust to certain types of input transformations (e.g., translation, rotation, scaling, blurring, noise addition, etc.). This means that, if a genuine image is correctly recognized by a DNN, the prediction score reduces only slightly when the same image is translated, rotated, or modified with one the mentioned transformations. However, the same is not true for most AEs and it has been observed that they result to be more sensitive to input transformations, which cause a much higher degradation in the prediction score.

This property of AEs has been exploited by some authors [37] to detect whether an input x is adversarial or genuine. If $y = f(x)$ is the top class score produced by the DNN on input x and $y_T = f(T(x))$ is the score produced on the transformed input $T(x)$, a simple detection method is to consider x to be adversarial if the difference $y - y_T$ is higher than a given threshold τ.

Unfortunately, it is possible to generate adversarial examples that are robust to input transformations. To cope with this case, Nesti et al. [22] proposed a new method, called *defense perturbation*, capable of detecting AEs that are robust to input transformations. The defense perturbation is generated by a proper optimization process capable of making robust AEs sensitive again to input transformations. Furthermore, the paper introduces multi-network AEs that can fool multiple DNN models simultaneously, presenting a solution for detecting them.

4.2 Detection by coverage analysis

A different approach for detecting AEs is based on a deeper analysis of the neuron activation values in the different DNN layers. In fact, in order to force a DNN model to classify an input with a desired wrong class, AEs usually cause an overactivation of some neurons in different network layers. To identify such neurons, Rossolini et al. [32] presented a new coverage analysis methodology capable of detecting both adversarial and out-of-distribution inputs.

The approach works in two distinct phases: in a preliminary (off-line) phase, a trusted dataset is presented to the DNN and the neuron outputs, in each layer and for each class, are analyzed and aggregated into a set of covered states, which all together represent a sort of *signature* describing how the model responds to the trusted samples for each given class. Then, at runtime, each new input is subject to an evaluation phase, in which the activation state produced by the input in each layer is compared with the corresponding signature for the class predicted by the network. The higher the number of activation values outside the range observed during the presentation of the trusted dataset, the higher the probability that the current input is not trustworthy. The approach is schematically illustrated in Figure 6.

The nice thing about this approach is that the comparison against the signature allows computing a confidence value c distinct from the prediction score, indicating how much the current prediction can be trusted.

4.3 Interpretability issues

Another problem of complex machine learning models is that they are hardly interpretable by humans. In fact, they encode their input-output function in millions of parameters and, therefore, it is not trivial to understand why a given input produces a certain output. Many people demand for transparency and precise mechanisms as a prerequisite for trust. Especially for safety-critical applications, developers cannot trust a critical decision system if it is not possible to explain the reasons that brought to that decision.

To address this issue, a new branch of research, referred to as *Explainable AI* (XAI) [36], started around 2014 with the goal of reconstructing and representing in a comprehensible fashion the features that caused an AI model to produce a given output.

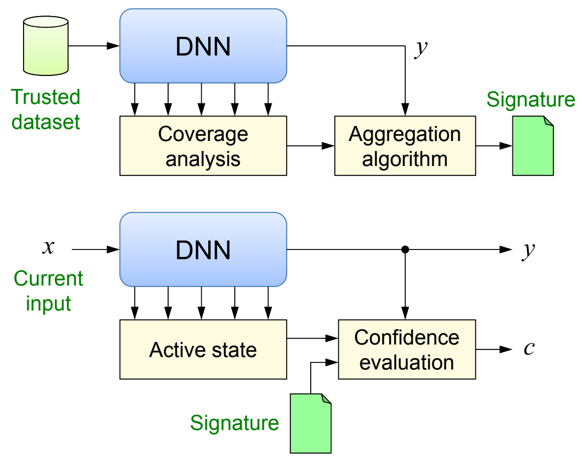

Figure 6 Overview of the coverage-based method proposed in [32]: (top) off-line phase for producing the signature for each layer and each class; (bottom) online detection phase based on comparing the current activation states with the stored signature. It produces a confidence value c indicating how much the current prediction can be trusted.

Thanks to these methods, it has been found that, in some cases, a DNN learned to classify images using features that have nothing to do with the object of interest, but appear frequently in most training samples of a specific class (e.g., water for the class *ship*, snow for the class *wolf*, or even copyright tags present in most of the images of a given class). Such biases in the training data limit the generalization capabilities of a neural model and can cause wrong predictions that would be quite harmful in applications like self-driving cars or medical diagnoses.

Identifying such biases in the training set is the main objective of XAI, which in the last years proposed several methods and tools for making AI decisions more interpretable to humans [21]. For instance, Pacini et al. [23] presented X-BaD, a flexible tool for detecting biases in training sets while providing user-interpretable explanations for DNN outputs. The tool can also be used to compare the performance of different XAI methods.

5 Towards trustworthy AI-based systems

From the considerations presented in the previous sections, it should be clear that, although AI algorithms exhibit a great performance in several perception and control tasks, they have intrinsic weaknesses in terms of safety, security, timing predictability, and certifiability. Does it mean that complex cyber-physical systems cannot take advantage of such an amazing software technology? Fortunately, there is a promising way to exploit the power of modern deep learning algorithms in safety-critical systems.

While we cannot prevent AI algorithms from being attacked or producing unsafe results, we can take a number of countermeasures to prevent them from harming the whole system.

For instance, the level of temporal predictability, as well as the level of security of the whole system can be increased by using a suitable real-time hypervisor capable of isolating the safety-critical components from the AI-powered functions in two separated virtual machines, as described in Section 3. In this way, an attack to the AI domain cannot propagate to the high-criticality domain, which can be protected by exploiting the hardware security features available in modern computer architectures.

To cope with adversarial attacks and OoD samples, the defense methods described in Section 4 are essential to detect both malicious as well as unsafe inputs that would cause a DNN to produce a wrong output. In these cases, the system must react by excluding the attacked AI component from the decision pipeline and switching to a simpler, but safer, backup control module that can bring the system into a safe state. For instance, in a self-driving car, the backup module could take control of the vehicle to stop it at the side of the road.

The approach described above has been undertaken by Biondi et al. [8] for exploiting deep learning models in safety-critical systems. The architecture includes two execution domains illustrated in Figure 7: a high-performance domain, running under Linux, and a safety-critical domain, running under the Erika Enterprise real-time kernel [1], both managed by the CLARE hypervisor [2].

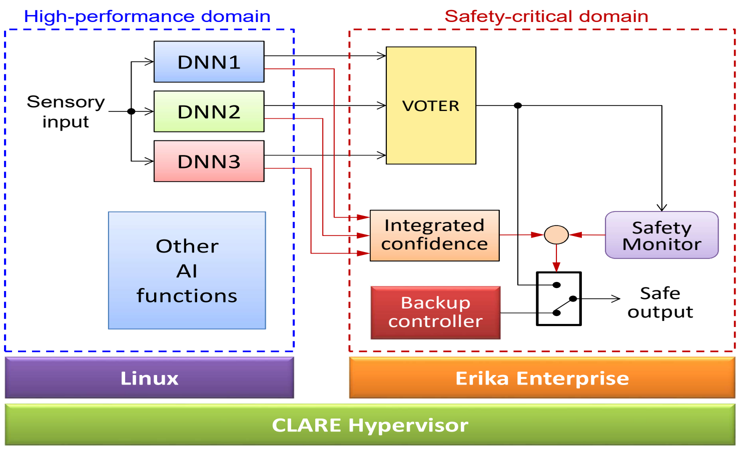

Figure 7 Architecture scheme proposed in [8], including a high-performance domain (running all the AI functions) and a safety-critical domain (running all critical control functions), managed by the CLARE hypervisor.

The high-performance domain is in charge of executing all the AI algorithms and tasks that must run under Linux, whereas the safety-critical domain runs all the vital system functions under the Erika Enterprise real-time kernel [1].

To increase the level of robustness against AEs and OoD inputs, each perceptual function is replicated by three different DNN models trained on different datasets. Each DNN also includes a coverage analyzer to provide a confidence signal (represented by the red arrow coming out from each DNN box). The three confidence signals are then integrated into an overall confidence signal, which is used to decide whether to switch to the safe controller. The three redundant DNN outputs go instead to a majority voter to resolve possible disagreements in the predictions.

An extra safety feature is represented by the presence of a *Safety Monitor*, which is in charge of detecting possible DNN outputs that could have a negative consequence on the controlled system. In fact, even in the presence of a non-malicious input, a DNN could generate an output that is not detected as unsafe by the voter and the confidence integrator, but could cause the controlled system to fail or misbehave. If such a condition is detected by the Safety Monitor, the system control is switched from the high-performance AI controller to the backup controller.

A first version of the architecture has been successfully implemented and tested on a control system for an inverted pendulum. In this case, the Safety Monitor uses a Lyapunov approach to detect when the system state exits a safe region of the state space and enters a margin region in which the stability can still be recovered by the safe controller. When this happens, the control is given to the backup controller. The AI controller is re-enabled when the system state goes back to the safe zone.

Figure 8 illustrates a simplified and qualitative representation of the state space for an inverted pendulum, where the black dot represents the current system state, the safe region is visualized in green, the recoverable region in yellow, and the unstable region in gray.

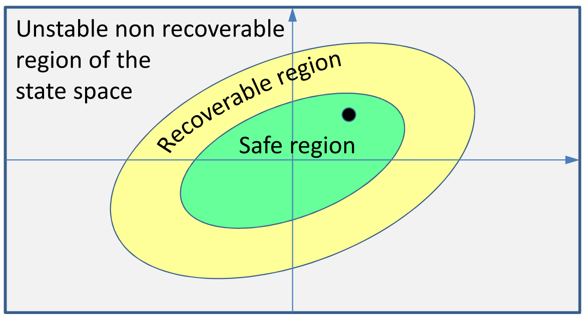

Figure 8 Qualitative representation of the state space for an inverted pendulum. The black dot represents the current system state.

On the same line, Belluardo et al. [4] presented a safe and secure multi-domain software architecture tailored for autonomous driving.

6 Conclusions

This paper presented a set of problems that today prevent the use of deep learning algorithms in mission-critical systems, as self-driving vehicles, autonomous robots, and medical applications. Among them, the most relevant issues are the low predictability of modern heterogeneous architectures and the high vulnerability of AI software to cyber attacks.

Fortunately, the research community is readily reacting to address such problems and some solutions to overcome such limitations have already been proposed at different architecture levels. However, the problems are many and complex, so several issues remain unsolved and require some joint effort from the AI and the real-time research communities.

References

1 Erika Enterprise RTOS. URL: `https://www.erika-enterprise.com/`.

2 The CLARE Software Stack. URL: `https://accelerat.eu/clare`.

3 Luca Abeni and Giorgio Buttazzo. Resource Reservation in Dynamic Real-Time Systems. *Real-Time Systems*, 27(2):123–167, July 2004.

4 L. Belluardo, A. Stevanato, D. Casini, G. Cicero, A. Biondi, and G. Buttazzo. A Multi-domain Software Architecture for Safe and Secure Autonomous Driving. In *Proc. of the 27th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2021)*, Online event, August 18–20, 2021.

5 Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84:317–331, December 2018.

6 Alessandro Biondi, Alessio Balsini, Marco Pagani, Enrico Rossi, Mauro Marinoni, and Giorgio Buttazzo. A Framework for Supporting Real-Time Applications on Dynamic Reconfigurable FPGAs. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS 2016)*, Porto, Portugal, November 29 – December 2, 2016.

7 Alessandro Biondi, Daniel Casini, Giorgiomaria Cicero, Niccolò Borgioli, and Giorgio Buttazzo et al. SPHERE: A Multi-SoC Architecture for Next-generation Cyber-Physical Systems Based on Heterogeneous Platforms. *IEEE Access*, 9:75446–75459, May 2021.

8 Alessandro Biondi, Federico Nesti, Giorgiomaria Cicero, Daniel Casini, and Giorgio Buttazzo. A Safe, Secure, and Predictable Software Architecture for Deep Learning in Safety-Critical Systems. *IEEE Embedded Systems Letters*, 12(3):78–82, September 2020.

9 N. Capodieci, R. Cavicchioli, M. Bertogna, and A. Paramakuru. Deadline-Based Scheduling for GPU with Preemption Support. In *Proc. of the 39th IEEE Real-Time Systems Symposium (RTSS 2018)*, Nashville, Tennessee, USA, December 11–14, 2018. URL: `http://arxiv.org/abs/1312.6199`.

10 Daniel Casini, Alessandro Biondi, and Giorgio Buttazzo. Timing Isolation and Improved Scheduling of Deep Neural Networks for Real-Time Systems. *Software: Practice and Experience*, 50(9):1760–1777, September 2020.

11 Daniel Casini, Alessandro Biondi, Giorgiomaria Cicero, and Giorgio Buttazzo. Latency Analysis of I/O Virtualization Techniques in Hypervisor-Based Real-Time Systems. In *Proceedings of the 27th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2021)*, Online event, May 18–21, 2021.

12 R. Cavicchioli, N. Capodieci, and M. Bertogna. Memory Interference Characterization Between CPU Cores and Integrated GPUs in Mixed-Criticality Platforms. In *Proc. of the 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2017)*, Limassol, Cyprus, September 12–15, 2017.

13 Giorgiomaria Cicero, Alessandro Biondi, Giorgio Buttazzo, and Anup Patel. Reconciling Security with Virtualization: A Dual-Hypervisor Design for ARM TrustZone. In *Proceedings of the 18th IEEE International Conference on Industrial Technology (ICIT 2018)*, Lyon, France, February 20–22, 2018.

14 Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. BinaryConnect: Training Deep Neural Networks with binary weights during propagations. In *Proc. of the 29th Conference on Neural Information Processing Systems (NIPS 2015)*, Montreal, Canada, December 7–10, 2015.

15 Blane Erwin. The Groundbreaking 2015 Jeep Hack Changed Automotive Cybersecurity, 2021. URL: `https://fractionalciso.com/the-groundbreaking-2015-jeep-hack-changed-automotive-cybersecurity/`.

16 Giulia Ferri, Giorgiomaria Cicero, Alessandro Biondi, and Giorgio Buttazzo. Towards the Hypervision of Hardware-based Control-Flow Integrity for Arm Platforms. In *Proceedings of the Italian Conference on CyberSecurity (ITASEC 2019)*, Pisa, Italy, February 12–15, 2019.

17 Yunhui Guo. A Survey on Methods and Theories of Quantized Neural Networks. *ArXiv*, abs/1808.04752, 2018.

18 C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):40–61, January 1973.

19 Charlie Miller and Chris Valasek. Remote Exploitation of an Unaltered Passenger Vehicle, August 10, 2015. URL: `http://illmatics.com/Remote%20Car%20Hacking.pdf`.

20 Paolo Modica, Alessandro Biondi, Giorgio Buttazzo, and Anup Patel. Supporting Temporal and Spatial Isolation in a Hypervisor for ARM Multicore Platforms. In *Proceedings of the 18th IEEE International Conference on Industrial Technology (ICIT 2018)*, Lyon, France, February 20–22, 2018.

21 Christoph Molnar. Interpretable Machine Learning: A Guide for Making Black Box Models Explainable, 2021. URL: `https://christophm.github.io/interpretable-ml-book/`.

22 Federico Nesti, Alessandro Biondi, and Giorgio Buttazzo. Detecting Adversarial Examples by Input Transformations, Defense Perturbations, and Voting. *IEEE Transactions on Neural Networks and Learning Systems*, August 2021.

23 Marco Pacini, Federico Nesti, Alessandro Biondi, and Giorgio Buttazzo. X-BaD: A Flexible Tool for Explanation-Based Bias Detection. In *Proc. of the IEEE International Conference on Cyber Security and Resilience*, Online event, July 26–28, 2021.

24 M. Pagani, A. Biondi, M. Marinoni, L. Molinari, G. Lipari, and G. Buttazzo. A Linux-Based Support for Developing Real-Time Applications on Heterogeneous Platforms with Dynamic FPGA Reconfiguration. *Future Generation Computer Systems*, To appear.

25 Marco Pagani, Alessio Balsini, Alessandro Biondi, Mauro Marinoni, and Giorgio Buttazzo. A Linux-based Support for Developing Real-Time Applications on Heterogeneous Platforms with Dynamic FPGA Reconfiguration. In *Proceedings of the 30th IEEE International System-on-Chip Conference (SOCC 2017)*, Munich, Germany, September 5–8, 2017.

26 Marco Pagani, Enrico Rossi, Alessandro Biondi, Mauro Marinoni, Giuseppe Lipari, and Giorgio Buttazzo. A Bandwidth Reservation Mechanism for AXI-based Hardware Accelerators on FPGAs. In *Proc. of the Euromicro Conference on Real-Time Systems (ECRTS 2019)*, Stuttgart, Germany, July 9–12, 2019.

27 Francesco Restuccia and Alessandro Biondi. Time-Predictable Acceleration of Deep Neural Networks on FPGA SoC Platforms. In *Proc. of the 42nd IEEE Real-Time Systems Symposium (RTSS 2021)*, Online event, December 7–10, 2021.

28 Francesco Restuccia, Alessandro Biondi, Mauro Marinoni, and Giorgio Buttazzo. Safely preventing unbounded delays during bus transactions in FPGA-based SoC. In *Proceedings of the 28th Annual Int. Symposium on Field-Programmable Custom Computing Machines (FCCM 2020)*, Fayetteville, Arkansas, USA, May 3–6, 2020.

29 Francesco Restuccia, Marco Pagani, Alessandro Biondi, Mauro Marinoni, and Giorgio Buttazzo. Is Your Bus Arbiter Really Fair? Restoring Fairness in AXI Interconnects for FPGA SoCs. *ACM Transactions on Embedded Computing Systems*, 18(5-51):1–22, October 2019.

30 Francesco Restuccia, Marco Pagani, Alessandro Biondi, Mauro Marinoni, and Giorgio Buttazzo. Modeling and analysis of bus contention for hardware accelerators in FPGA SoCs. In *Proceedings of the 32nd Euromicro Conference on Real-Time Systems (ECRTS 2020)*, Online event, July 7–10, 2020.

31 Enrico Rossi, Marvin Damschen, Lars Bauer, Giorgio Buttazzo, and Jörg Henkel. Preemption of the Partial Reconfiguration Process to Enable Real-Time Computing with FPGAs. *ACM Transactions on Reconfigurable Technology and Systems*, 11(2):10:1–10:24, November 2018.

32 Giulio Rossolini, Alessandro Biondi, and Giorgio Buttazzo. Increasing the Confidence of Deep Neural Networks by Coverage Analysis. In *arXiv:2101.12100 [cs.LG]*, January 2021. `arXiv:2101.12100`.

33 Biruk Seyoum, Alessandro Biondi, and Giorgio Buttazzo. FLORA: FLoorplan Optimizer for Reconfigurable Areas in FPGAs. *ACM Transactions on Embedded Computing Systems*, 18(5-73):1–20, October 2019.

34 Tom Simonite. AI Has a Hallucination Problem That's Proving Tough to Fix, March 12, 2018. URL: `https://www.wired.com/story/ai-has-a-hallucination-problem-thats-proving-tough-to-fix/`.

35 Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *Proc. of the 2nd International Conference on Learning Representations (ICLR 2014)*, Banff, AB, Canada, April 14–16, 2014. URL: `http://arxiv.org/abs/1312.6199`.

36 The Royal Society. Explainable AI: the basics – policy briefing, 2019. URL: `https://royalsociety.org/-/media/policy/projects/explainable-ai/AI-and-interpretability-policy-briefing.pdf`.

37 Shixin Tian, Guolei Yang, and Ying Cai. Detecting Adversarial Examples through Image Transformation. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI-18)*, New Orleans, Louisiana, USA, February 2—7, 2018.