A Safe, Secure, and Predictable Software Architecture for Deep Learning in Safety-Critical Systems

Alessandro Biondi, *Member, IEEE,* Federico Nesti, Giorgiomaria Cicero, Daniel Casini, *Student Member, IEEE,* and Giorgio Buttazzo, *Fellow, IEEE*
Scuola Superiore Sant'Anna, Pisa, Italy

***Abstract*—In the last decade, deep learning techniques reached human-level performance in several specific tasks, as image recognition, object detection, and adaptive control. For this reason, deep learning is being seriously considered by the industry to address difficult perceptual and control problems in several safety-critical applications (e.g., autonomous driving, robotics, and space missions). However, at the moment, deep learning software poses a number of issues related to safety, security, and predictability, which prevent its usage in safety-critical systems. This work proposes a visionary software architecture that allows embracing deep learning while guaranteeing safety, security, and predictability by design. To achieve this goal, the architecture integrates multiple and diverse technologies, as hypervisors, run-time monitoring, redundancy with diversity, predictive fault detection, fault recovery, and predictable resource management. Open challenges that stems from the proposed architecture are finally discussed.**

***Index Terms*—Deep learning, machine learning, deep neural networks, safety-critical systems, safety, security, fault-tolerance, predictability.**

I. Introduction

In recent years, artificial intelligence (AI) made enormous progresses thanks to the evolution of deep neural networks (DNNs) and deep learning methodologies, which reached human-level performance in several tasks, such as image classification, object detection, and control. For these reasons, several companies started considering the adoption of DNNs as key components for increasing the perceptual and control capabilities of autonomous systems, as advanced robots and self-driving cars.

The software running in this type of systems must satisfy several stringent requirements, especially when humans are involved in the loop. Among them, the following properties are particularly crucial:

- **Certifiability**. All safety-critical software components must be written according to strict coding standards (e.g., MISRA [1]) and certified by proper certification authorities.
- **Safety and fault tolerance**. The system must prevent catastrophic consequences on the user(s) and the environment by proper mechanisms aimed at tolerating faults and failures that could possibly occur in complex software routines.
- **Time predictability**. These systems must react to events in the environment within predefined time bounds, computed at design time based on a set of performance requirements. A control output delivered too late could be useless or even dangerous (as an example, think of a braking command in a self-driving car). This means that such systems must be analyzable and verifiable not only in the functional domain, but also in the time domain.
- **Security**. The software must be designed to protect the system from cyber attacks that could exploit vulnerable sections of the code to modify the software and take control of the system. For instance, in a self-driving vehicle, a cyber attack could alter the control flow of some critical code to take control of the steering subsystem, with potential catastrophic consequences.

Unfortunately, addressing such requirements is not straightforward when DNNs are used to process perceptual and control functions, mainly for the following reasons:

- DNNs are not 100% trustable. Although they proved to achieve human-level performance in several tasks, as image recognition and object detection, there can be several unknown corner cases in which a DNN could respond in a wrong way, especially when the input is quite different from the examples provided during training, due to particular circumstances, as occurred in the Tesla accident [2]. In addition, there are still anomalous behaviors that are not fully understood yet, as the case of adversarial images [3], which are properly generated from normal images by altering the values of some pixels to fool the network and cause a wrong desired output. This technique could be used, for example, to attack an autonomous vehicle by simply modifying a stop sign to cause the DNN to fail in recognizing it.
- DNN-based systems are commonly developed using frameworks (e.g., TensorFlow and Caffe) that are not compatible with the coding standards used for certifying safety-critical software. In addition, when used for object recognition and detection, DNNs process images produced by cameras that are typically acquired by a rich operating system (e.g., Linux or Android) that is far from being certified.
- Most of the inference engines used for executing DNNs are designed to push the average performance, but they introduce large and unpredictable delays in worst-case scenarios, especially when multiple DNNs need to be

executed concurrently on the same platform.

- DDNs and their related inference engines are complex and large software systems that increase the attack surface, making the overall system more vulnerable to cyber-attacks. Furthermore, as the software infrastructure that is typically required to infer DNNs is based on a rich operating system, which may even be connected to the Internet, the attack surface of a system can be even larger.

A. This work

To address the problems presented above, this work proposes a visionary software architecture that allows embracing DNNs to control safety-critical systems, while coping with safety, security, and predictability issues, and enabling certification of the control software. To achieve this goal, the proposed architecture combines the following technologies:

- *Hypervisor technology*: a hypervisor is used to isolate components with different criticality and security levels, hence allowing to protect the safety-critical components from unexpected failures and cyber attacks by running them in separated execution domains. In addition, the hypervisor can be integrated with specific monitoring units aimed at detecting crashes and anomalous behaviors of the rich operating system running the DNNs, thus activating proper recovering procedures.
- *Redundancy and diversity*: redundancy and diversity is exploited to increase the robustness of deep learning components and cope with possible faulty outputs of the neural networks in corner-case situations.
- *Predictive fault detection*: digital-twin technology is employed to simulate a virtual replica of the system (digital twin) in order to analyze the consequences of control actions into the future, to detect possible faults in advance.
- *Fault recovery*: a recovery mechanism is included to exclude DNN-based controllers and switch to a simpler but safer controller when a faulty control action is detected or when the response of DNNs is judged to be non reliable.
- *Predictability*: a predictable DNN inference engine is provided to reduce and control the interference among multiple concurrent DNNs running on the same platform with different execution rates.

The rest of the paper is organized as follows: Section II briefly overviews the existing solutions; Section III presents the proposed architecture; and Section IV discusses the open challenges.

II. Existing solutions: a brief review

The literature on safe and secure software architectures is quite vast: hence, due to lack of space, it is not possible to report a detailed literature review in this paper. For this reason, this section concentrates on previous work focused on DNNs or autonomous systems.

In the past few years, several authors studied the robustness and the safety of DNNs by addressing both testing and formal verification issues. Testing mainly aims at finding corner cases in which faulty outputs are produced, while verification aims at determining whether a given property holds for the DNN, providing a mathematical proof if this is the case, or a counterexample if it is not. The interested reader can refer to the survey by Huang et al. [4] for a detailed review of the state-of-the-art of such techniques. Verification techniques [5] suffer from severe scalability issues, which make them not practically applicable to modern (complex) DNNs. Testing techniques have been demonstrated to be applicable to modern DNNs [4], [6]; however, they still require large-scale computations, and it is still not clear how to generate suitable test cases that can provide meaningful guarantees. In particular, it has been noted [7] that the classical notion of *coverage* in software engineering does not directly apply to DNNs: how to properly quantify the testing coverage of DNNs is still an open problem, notwithstanding very interesting recent findings [8].

This work adopts a very different approach with respect to those surveyed in [4]: *complex DNNs are assumed to be untrustworthy for being directly involved in the control loop of a safety-critical system*. Therefore, the authors of this paper argue that they should be coupled with a certifiable, safe controller in a (so called) *simplex* scheme [9] (also referred to as *safety executive* pattern by other authors [10]), and deployed by following redundancy paradigms with diversity. Note that this does not mean that testing techniques for DNNs are not useful in our approach, as they can help contain the faults of DNNs anyway. Verification techniques may also be used in our architecture to verify simpler neural components (see Section IV).

Concerning timing predictability of DNNs, some authors started addressing the problem by looking at the case of GPU-based accelerators. Zhou et al. [11] proposed a fine-grained pipelined scheduling of DNNs on GPUs with data fusion for video streaming applications. Similarly, Yang et al. [12] employed DNN decomposition and parallel pipelined execution to improve the throughput of multi-camera detection tasks for automated driving systems.

Limited attention has also been posed on system-level and architectural aspects to leverage DNNs in safety-critical systems while taking into account certification issues, applicability to standards, and technical requirements such as the compatibility of software stacks with critical components. For instance, Luo et al. [13] studied some of these issues in the context of autonomous driving, targeting the ISO 26262 safety standard, but not explicitly focusing on DNNs.

Overall, to the best of our records, no efforts have been spent in designing a comprehensive software architecture that allows embracing deep learning while ensuring safety, security, and predictability by design.

III. Proposed architecture

The software architecture proposed in this work is illustrated in Figure 1 and is composed of five major components, each described in one of the following sub-sections. It is conceived as a general design approach to develop next-generation control software that interacts with a physical plant by means of actuators, and both legacy and modern high-performance sensors (such as high-resolution 3D cameras).

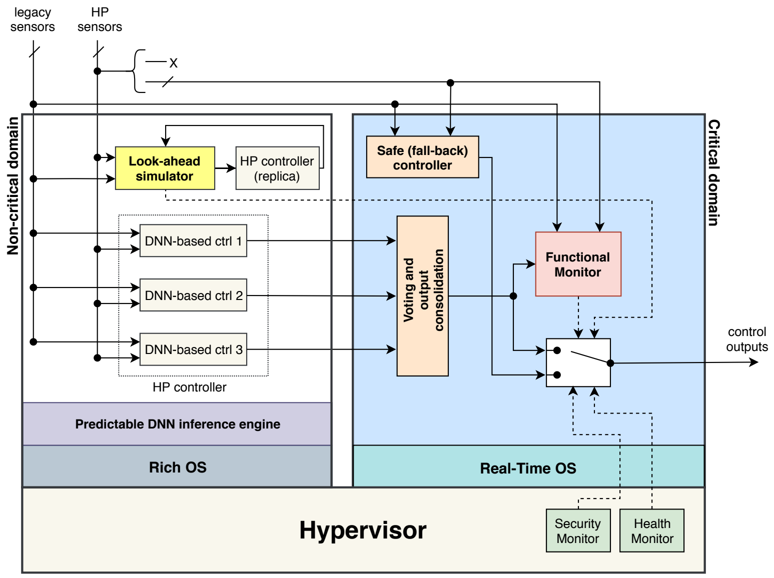

Fig. 1: Illustration of the proposed architecture.

A. *Hypervisor-centric multi-OS infrastructure*

The proposed architecture is built upon a bare-metal hypervisor that serves the execution of two execution domains: (i) a *non-critical domain*, which is managed by a rich operating system such as Linux; and (ii) a *critical domain*, which is managed by a real-time operating system (RTOS). The hypervisor and the critical domain represent the trusted computing base of the architecture and are the only software components that are assumed to be *certified*. The entire system must be fail-safe or fail-operational as long as these two components are not compromised, whereas the non-critical domain can crash or be subject to cyber-attacks.

Most of the high-performance communication buses and network interfaces, and in turn the corresponding high-performance sensors connected via them, are only exposed to the non-critical domain. The same holds for complex hardware accelerators offered by the underlying computing platform (e.g., GPUs). This design choice is driven by practical matters: in most of the cases, the device drivers and the software stacks required to manage these components are only available for rich operating systems. Conversely, legacy sensors are exposed to both the execution domains, as they are typically connected via simpler peripherals (such as analog-to-digital converters) that can be managed without requiring complex software.

Finally, the actuators that enable physical actions on the controlled plant are only exposed to the critical domain, as they may have a direct impact on the safety properties of the entire system. Inter-domain communication channels are offered by the hypervisor to implement safe, secure, and predictable control loops.

B. *Simplex architecture with redundancy*

Two controllers are employed: a *high-performance (HP) controller* based on DNNs, which executes in the non-critical domain, and a simpler *safe controller* based on rigorous engineering, which executes in the critical domain. To increase its robustness and deal with corner-case scenarios that are difficult to detect during testing, the HP controller is composed of diverse replicas of DNN-based controllers. Each of such DNN-based controllers may employ different DNNs built with different models and/or trained with different data sets, provided that they take the same inputs and produce the same kind of outputs. Note that a DNN-based controller can either produce outputs directly with a DNN or rely on DNNs to perform a specific task in the control logic (e.g., perception). The outputs of the HP controller are transmitted to the critical domain by means of communication channels offered by the hypervisor, where they are subject to voting and consolidation to produce the actual commands for the actuators. The safe controller is developed with well-established engineering techniques, such as model-based design with hardware-in-the-loop or simulation-based testing, and shall be able to keep the physical plant in fail-safe or fail-operational conditions for any input produced by the sensors.

A *switching logic* is employed in the critical domain to provide the physical plant with either the control outputs produced by the HP controller or those produced by the safe controller. This design is conceived in such a way that the HP controller is used in regular operating conditions (hopefully, most of the time), while the system must switch to the safe controller whenever the outputs produced by the HP controller are judged not reliable or are not present (e.g., in the presence of a crash or a denial-of-service attack in the non-critical domain). Note that, in this way, DNN-based controllers can be totally excluded from the certification process.

The main component that controls the switching logic is the *functional monitor*. It takes in input the sensors exposed to the critical domain and the outputs produced by the HP controller to decide whether the latter may be dangerous for the physical plant. It also implements a watchdog mechanism to detect tardy (i.e., not delivered within control-dependent deadlines) or missing outputs from the HP controller, to which it reacts by switching to the safe controller. Finally, the functional monitor must also be capable of deciding when it is possible to switch back to the HP controller, paying attention at implementing hysteresis mechanisms to avoid frequent switches and/or make the system unstable [14]. The design of the functional monitor is one of the major challenges in realizing the proposed architecture — further details are provided in Section IV. The switch to the safe controller can also be driven by a look-ahead simulator and hypervisor-level monitors, which are discussed in Sections III-C and III-D, respectively. All these components that can trigger the switching logic are conceived to be uncorrelated with each other, i.e., each of them is in charge of independently detecting a certain class of faults with some component-specific technique.

C. Look-ahead simulator

Digital-twin technology is employed as an additional component of the architecture. It consists of a simulation of the physical plant that is controlled by the same control logic used for the real plant. The HP controller is replicated to be stimulated by the inputs produced by the simulation. Indeed, a different instance of the HP controller is required because its internal state during simulations will be different from the internal state maintained during the actual control of the real plant. A replica of the voting and output consolidation logic is also integrated within the simulator to process the outputs produced by the replica of the HP controller. The objective of this simulation environment is to *predict* the effect of the outputs produced by the HP controller on the physical plant, with the purpose of detecting possible dangerous control actions in advance. To do so, the simulator may also need to test multiple possible evolutions of the system state (corresponding to different future inputs). For these reasons, this component is denoted as *look-ahead simulator*. When a potentially-dangerous control action is detected in simulation, the simulator triggers the switch to the safe controller to exclude the HP controller, hence preventing a potential fault. Note that a late fault detection by the functional monitor may allow the safe controller to bring the plant into a fail-safe state only. Conversely, if a fault is detected in advance by the look-ahead simulator, it is possible to enable earlier switches to the safe controller (before the intervention of the functional monitor) that may allow the safe controller to keep the plant in a fail-operational state. This clearly improves the system performance in the presence of faults and simplifies fault recovery.

The look-ahead simulator is implemented in the non-critical domain because it relies on a replica of the HP controller. Note that it may generate a considerable increase of the computing workload due to the replication of the HP controller, which in turn increases the required amount of computing resources and hence the cost and the energy consumption of the system. Furthermore, disposing of an accurate physical model of the plant may not be possible for some application scenarios. For these reasons, the look-ahead simulator is considered as an *optional* component.

D. Hypervisor-level monitoring

Two run-time monitoring mechanisms are employed at the hypervisor level to ensure safety and security guarantees, namely the *security monitor* and the *health monitor*. The security monitor is in charge of detecting cyber-attacks and unauthorized intrusions in the system, with the purpose of triggering recovery actions whenever one of such critical events is identified. For instance, if control-flow integrity [15] is enforced in the non-critical domain, the security monitor can detect control-flow violations (i.e., illegal execution flows) and react by forcing the switch to the safe controller in the critical domain, independently of the functional monitor. The detected attacks may also be logged by the security monitor by relying on a secure storage. The health monitor supervises the execution of the software domains and the operating conditions of the underlying computing platform. Similarly to the security monitor, it triggers recovery actions whenever failures are detected. The typical tasks performed by the health monitor include (i) the detection of software crashes in the non-critical domain, to which it reacts by handing over the control of the system to the critical domain; and (ii) the diagnostic test of peripheral devices and memories.

E. Predictable DNN Inference Engine

As pointed out in the introduction, existing DNN frameworks introduce computation delays that are difficult to predict, and leave room for several pathological scenarios that lead to large delays compared to those exhibited in the average case. When supporting the execution of multiple, concurrent DNNs, possibly running at different rates, the timing performance of DNNs can be even worse, especially when the DNNs contend for hardware accelerators that do not support fine-grained preemptions of ongoing computations or a user-controllable sharing of the computing resources. These issues are further exacerbated in the context of the proposed architecture, when the computing workload related to DNNs is increased due to redundancy and replication for the look-ahead simulator.

For these reasons, a predictable DNN inference engine is employed in the non-secure domain. It adopts (i) principled scheduling schemes for the computing resources, and (ii) optimization-based mapping of the DNN layers to the computing resources, with the purpose of guaranteeing worst-case response-time bounds or provable long-tail latencies for inference tasks. The timing guarantees provided by this engine are used to configure the watchdog mechanism of the functional monitor (see Sec. III-B).

IV. Open challenges

Realizing the architecture proposed in the previous section requires overcoming several challenges, ranging from the investigation of unexplored (or limitedly-explored) research topics to the selection, composition, and configuration of existing solutions. The authors of this paper believe that the proposed architecture can serve as a research platform upon which several problem-specific contributions can be proposed, both in terms of theoretical/analytical results and system-level mechanisms.

The design of the functional monitor is probably the most challenging task. Existing solutions addressed the problem by using pure control-theoretic approaches, such as the identification of a safe stability region of the plant [16], reachability analysis for hybrid systems [14], [17], or just reactive approaches based on fault detection [18]. However, these solutions may be either not applicable to complex plants, or lead to poor performance (i.e., a too conservative behavior of the functional monitor) such that DNN-based controllers become ineffective. Therefore, the authors still consider this research topic widely open. In particular, an interesting direction may consist in investigating the design of functional monitors with neural networks. Note that the task performed by the functional monitor is intrinsically different from the one performed by the DNN-based controllers, as its main objective is to detect (or infer on the presence of) faulty outputs produced by DNN-based controllers. The logic required to *detect* a faulty control output may be far simpler than the one required to *compute* a correct output. Therefore, simpler DNNs that are fully-verifiable with formal techniques, or either testable with a very high degree of coverage in a reasonable amount of time may be devised to implement the functional monitor. Furthermore, note that the inputs of DNN-based controllers are just additional data that may or may not be used by the functional monitor depending on the specific application (see the connections at the top of Figure 1). For instance, when the HP controller includes DNNs used for image perception, the functional monitor could not take such images as inputs, but just the control outputs produced by the HP controller and other information coming from non-camera sensors.

Concerning the realization of the look-ahead simulator, the key challenge consists in achieving accurate simulations of the plant. Indeed, for complex application environments, it may be extremely difficult to dispose of accurate models. Research efforts should be spent to assess whether the adoption of less accurate, but tractable, models can lead to reasonable performance for specific application scenarios. The most relevant performance metrics for the look-ahead simulator are (i) the ratio of false positives, which should clearly be as low as possible, and (ii) the prediction time with which it is capable of detecting faults in advance, which should be shorter than the one of the functional monitor. The look-ahead simulator also originates challenges related to the efficiency of its implementation. Indeed, it may necessitate to manage large amounts of data (think of stereo high-resolution images and point clouds) and multiple simulations (to cope with different future inputs) in a limited amount of time. For these reasons, it is either applicable in systems with highly powerful computing platforms (e.g., when cost constraints are not particularly relevant) or in resource-constrained systems with loose timing constraints, such as industrial machineries for manufacturing automation.

Despite relevant achievements have been reached in optimizing the inference of DNNs, very limited efforts have been devoted to improving their execution predictability. Due to the large availability of software stacks, e.g., as those offered by Nvidia, most proposals focused on the case of hardware acceleration with GPUs. They however are known to suffer from low predictability due to both the adopted scheduling policies and the scarce information publicly available on their internal structure. Future research should also assess whether other technologies are more suited to predictably infer DNNs. In particular, the authors believe that the use of field-programmable gate arrays (FPGAs), especially those that support dynamic partial reconfiguration [19], represent a very flexible solution to enable the predictable acceleration of DNNs. Indeed, they allow deploying efficient and customized accelerators that exhibit a very regular (and hence predictable) execution behavior, while also offering the possibility of adopting a fine-grained control of the memory traffic, which in the case of DNNs tend to significantly affect the timing performance of the accelerators due to the large amount of data involved in modern DNN models.

Future research should also not ignore state-of-the-art optimization techniques for inferring DNNs, such as the use of integer weights and layer fusion, when designing predictable inference engines. For instance, in the case of GPUs, efficient tools such as TensorRT [20] should not be ignored.

The selection and the configuration of hypervisor-level monitoring mechanisms, and their integration with the switching logic, is another system-level problem to be solved. The support for restart-based fault-tolerance mechanisms, such as the one proposed by Abdi et al. [21], is another interesting direction to be investigated.

Finally, interesting research can be carried out by applying the proposed architecture to specific application scenarios to demonstrate its effectiveness in practical settings.

References

[1] "MISRA, https://www.misra.org.uk/."

[2] National Transportation Safety Board, "Collision between a car operating with automated vehicle control systems and a tractor-semitrailer truck near williston, Florida May 7, 2016," in *Highway Accident Report NTSB/HAR-17/02*, 2017.

[3] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *arXiv:1312.6199v4 [cs.CV], https://arxiv.org/abs/1312.6199*, 2014.

[4] X. Huang, D. Kroening, M. Kwiatkowska, W. Ruan, Y. Sun, E. Thamo, M. Wu, and X. Yi, "Safety and trustworthiness of deep neural networks: A survey," in *arXiv:1812.08342v2 [cs.LG], https://arxiv.org/abs/1812.08342v2*, 2018.

[5] S. W. Xiaowei Huang, Marta Kwiatkowska and M. Wu, "Safety verification ofdeep neural networks," in *in 29th International Conference on Computer Aided Verification (CAV)*, ser. CAV '17, 2017.

[6] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the 40th International Conference on Software Engineering*, 2018.

[7] K. Pei, Y. Cao, J. Yang, and S. Jana, "Deepxplore: Automated whitebox testing of deep learning systems," in *Proceedings of the 26th Symposium on Operating Systems Principles*, 2017.

[8] Y. Sun, X. Huang, D. Kroening, J. Sharp, M. Hill, and R. Ashmore, "Testing deep neural networks," in *arXiv:1803.04792 [cs.LG], https://arxiv.org/abs/1803.04792*, 2019.

[9] Lui Sha, "Using simplicity to control complexity," *IEEE Software*, vol. 18, no. 4, July 2001.

[10] B. P. Douglass, *Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems*. Addison-Wesley Professional, 2002.

[11] H. Zhou, S. Bateni, and C. Liu, "S3DNN: Supervised streaming and scheduling for GPU-accelerated real-time DNN workloads," in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2018.

[12] S. W. M. Yang, J. Bakita, T. Vu, F. D. Smith, J. H. Anderson, and J. Frahm, “Re-thinking cnn frameworks for time-sensitive autonomous-driving applications: Addressing an industrial challenge,” in *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April.

[13] Y. Luo, A. K. Saberi, T. Bijlsma, J. J. Lukkien, and M. van den Brand, “An architecture pattern for safety critical automated driving applications: Design and analysis,” in *2017 Annual IEEE International Systems Conference (SysCon)*, April 2017.

[14] S. Bak *et al.*, “Real-time reachability for verified simplex design,” in *2014 IEEE Real-Time Systems Symposium*, 2014.

[15] N. Burow, S. A. Carr, J. Nash, P. Larsen, M. Franz, S. Brunthaler, and M. Payer, “Control-flow integrity: Precision, security, and performance,” *ACM Comput. Surv.*, vol. 50, no. 1, Apr. 2017.

[16] D. Seto, B. Krogh, L. Sha, and A. Chutinan, “The simplex architecture for safe online control system upgrades,” in *Proceedings of the 1998 American Control Conference*, June 1998.

[17] S. Bak *et al.*, “Sandboxing controllers for cyber-physical systems,” in *2011 IEEE/ACM Second International Conference on Cyber-Physical Systems*, 2011.

[18] K. Vivekanandan *et al.*, “A simplex architecture for intelligent and safe unmanned aerial vehicles,” in *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2016.

[19] A. Biondi, A. Balsini, M. Pagani, E. Rossi, M. Marinoni, and G. Buttazzo, “A framework for supporting real-time applications on dynamic reconfigurable fpgas,” in *Proc. of the IEEE Real-Time Systems Symposium (RTSS 2016)*, December 2016, pp. 1–12.

[20] “Nvidia tensortRT, https://developer.nvidia.com/tensorrt.”

[21] F. Abdi, C.-Y. Chen, M. Hasan, S. Liu, S. Mohan, and M. Caccamo, “Guaranteed physical security with restart-based design for cyber-physical systems,” in *Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems*, 2018.