Federal Register Notice: 89 FR 78915, Federal Register :: Networking and Information Technology Research and Development Request for Information on Cyber-Physical Systems Resilience Research, September 26, 2024.

Request for Information on the National Cyber-Physical Systems Resilience Plan

J. Sukarno Mertoguno

Gregory Briskin

Jason H. Li

Kyung Kwak

Chapter 1
Physics, Cyber-systems & Machine Learning

J. Sukarno Mertoguno, Gregory Briskin, Jason H. Li and Kyung Kwak

Abstract Cyber-physical systems (CPS) are used in various safety-critical domains such as robotics, industrial manufacturing systems, and power systems. Faults and cyber attacks have been shown to cause safety violations, which can damage these systems and endanger human lives. The past decade has seen the proliferation of research efforts related to security and resilience in cyber physical systems, with an abundance of publications, workshops, and even media attention. More recently, artificial intelligence (AI) and machine learning (ML) have been reinvigorated and become the topic of paramount attention across the research community, mass media and society, highlighted by trademark successes in the fields of gaming, video, audio, and language translation. While it seems natural to apply AI/ML in CPS security and resilience, the authors would like to share some lessons learned and future directions as cautionary notes, which include: (1) the critical importance of physics and the physical world (P); (2) various means and effects that are introduced by cyber (C); (3) interactions and ramifications of P and C in a system (S); (4) system model and control in CPS; (5) enhanced robustness of control and autonomy in CPS by AI/ML; (6) pitfalls and appropriate positioning of AI/ML in CPS security and resilience; and (7) some challenges and opportunities for research and development.

J. Sukarno Mertoguno
School of Cybersecurity & Privacy, Georgia Institu[redacted]

Gregory Briskin, Jason H. Li, and Kyung Kwak
Trusted Science and Technology Inc.[redacted]

1.1 Introduction

Cyber physical systems (CPS) underlie many critical infrastructures and are prevalent across a wide range of areas including the electrical grid, factory production pipeline, machinery control, vehicular control, internet-of-things (IOT) devices, and commodity toy drones, just to name a few. By its nature, a CPS straddles the continuous-time physical domain and the discrete-time digital or cyber domain. Cyber components (e.g., communication and computing) couple with physical components (e.g., sensors and actuators) to carry out the intended functions of the CPS.

Cyber physical systems are required to satisfy safety constraints in various application domains such as robotics, unmanned vehicles (e.g., aerial or ground), industrial manufacturing systems, and power systems. However, the once isolated system of computer-controlled machinery is now more exposed to the external world than ever, which renders ample opportunities of remote system disruption via cyber threats, in addition to the tradition threat of a physical component failing. Both may result in safety violations.

Current emphasis on cyber security of CPS is on securing the operational technology (OT) network. For example, National Institute of Standards and Technology (NIST) devoted its guidance for securing CPS, SP 800-82 Rev.3 [1], solely to network security with network segmentation as the primary recommended solution. However, network or communication is only one facet of CPS. While network is an important CPS and critical infrastructure component, over-emphasis on networking security will not be sufficient for defending the underlying CPS and its infrastructure against motivated and well-resourced adversaries. A recent article indicates that the assumption of malicious events entering the system solely via the external network (hence the need for network segmentation) has been invalidated [2]. The exploits discussed in the article did enter the system through the local (internal) network and propagated within the internal bus, avoiding the security protection provided by network segmentation. A holistic view and approach for defending CPS is needed.

1.2 Physical Domain and Cyber Domain

In CPS, the ultimate goal is for the overall system to be stable and function as intended. A *resilient CPS* is expected to physically operate properly and in a predictable and controllable manner under ever-present external and environmental disturbance as well as adversarial cyber exploits. The objectives and emphasis for CPS resilience are *physical* stability and functionality. Cyber components and systems in CPS are means toward the end of achieving CPS resilience. The stability of cyber systems by itself is not the primary objective.

A cyber physical system contains cyber components that interact with and control the behavior of the physical system operating in a physical environment. Generally speaking, the cyber controller periodically samples the operation (or mission) objective (e.g., the expected set value of speed), measures the actual values of physical

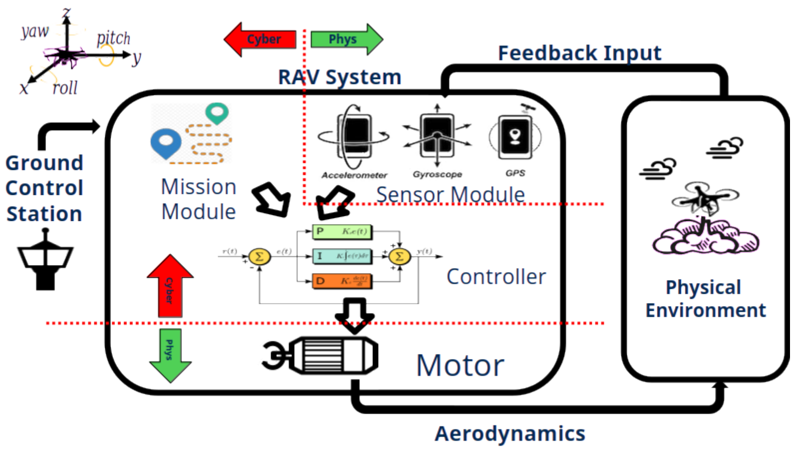

Fig. 1.1 Cyber Domain and Physical Domain in CPS, derived from MayDay [3]

variables via sensors (e.g., speed, altitude), contrasts measurements against the objective, and calculates the magnitude of control variables, which translates to direction and/or force to be asserted by the actuator onto the physical environment. Figure 1.1 shows an example of how the physical and cyber components interact in a particular cyber physical system – a robotic aerial vehicle (RAV) or drone.

It is worthwhile to point out the differences of physical and cyber components and (sub)systems, and the potential opportunities they may offer for building resilient cyber physical systems. The physical platform and the subsystems operate in a physical environment at physical speed (and time), governed by the laws of physics. The mass and dynamics of a physical implementation define its moment of *inertia*, which in turn influences the *response time* of the physical subsystems and the platform. Any physical subsystem of a CPS must obey the laws of physics, and the physical systems invariably have inertia.

CPS physical and cyber components differ significantly in the scale of their response time. Physical and mechanical components have relatively large time scales (low frequency), in the order of milliseconds and seconds. A heavier object has larger inertia and hence lower frequency (see subsection 1.2.2.6 for detailed description of inertia). For example, a large tanker vessel takes minutes to change its direction.

The cyber components operate at cyber speed, typically multiple orders of magnitude faster than that of physical components. The scan period of a CPS controller (1-500 Hz) is usually about one or two orders of magnitude smaller (faster) than that of the physical/mechanical machinery [4]. The clock speed of the controller CPU (GHz) is generally five to seven orders of magnitude faster than the scan cycle. The physical micro-mechanical sensing mass within a micro-electro-mechanical systems (MEMS) inertia measurement unit (IMU) has resonance frequency measured in KHz, in the 10-30KHz range [5], still one or two orders of magnitude faster than the controller's scan cycle of a drone.

Traditionally CPS researchers have been focusing on achieving *cyber stability*, which generally provides physical stability within the designed region of operation. This is definitely a prudent design methodology (see the first quadrant in Figure 1.2a with P(S) and C(S) denoting physical and cyber stability, respectively).

However, a cyber physical system may have to operate in a physical environment with disturbance so large (e.g., strong wind gust or other physical impact) that makes the controller algorithms or other cyber components struggle to work while out of the designed region of operation. Extended Kalman Filter (EKF) and robust control algorithms usually work effectively to absorb and tolerate relatively small disturbance, but the physical subsystems and overall platform may fail to maintain physical stability under large disturbance. This is quite interesting: cyber components work as designed but the physical systems are unstable, see the second quadrant in Figure 1.2a with P(U) and C(S) denoting physical instability and cyber stability, respectively, which indicates that the traditional focus for cyber stability is not always sufficient or effective. The community starts to notice this important realization, and leaders start to investigate alternatives to designing resilient cyber physical systems, such as the DARPA LINC program [6] and the DARPA FIRE program [7].

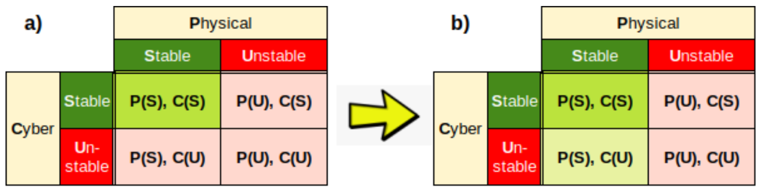

Fig. 1.2 Desired CPS operation space: a) Cyber-centric, b) Physical-centric

Particularly, since we argue that the goal of CPS resilience is physical stability and not necessarily cyber stability, cyber components controlling physical components only need to be stable at the scale of the frequency and response time of the physical components. Focusing on physical stability, therefore, opens up another design space for CPS resilience as the fourth quadrant in Figure 1.2b, with P(S) and C(U) denoting physical stability and cyber instability, respectively. It is important to emphasize again that C(U) means being unstable but also unnoticeable by physical components, not being unstable all the time for obvious reasons. Empowered by physical inertia and the differences in response time for physical and cyber components, the fourth quadrant represents a previously less-explored design space for achieving CPS security and resilience. Subsection1.2.2.6 explores cyber-attack resilient CPS design within the fourth quadrant.

1.2.1 System Model and Control in CPS

Modeling is essential to every scientific and engineering enterprise. For both scientists and engineers, the “thing being modeled” (referred to as *target*) is typically an object, process, or system in the physical world. But it could also be another model as manifested in model refinement for formal verification. A “model” of a target is any description of the target that is not Kant’s *thing-in-itself*. For example, mechanical engineers use Newton’s laws as models for how a system will react to forces. Computer engineers model digital circuits as instruction set architectures (ISAs), programs as executions in an ISA, and applications as networks of program fragments [8]. Each of these models rests on a modeling paradigm. For example, a source code is a model of what a machine should do when it executes the program, but the source code is not what is actually run on a machine. The Java programming language, for example, is just such a modeling paradigm. Models abstract away details, and layers of models may be built on top of another. A CPS system consists of such layered models from hardware all the way up to applications it runs.

The *fidelity* of a model is the degree to which it emulates the target. When the target is a physical object, process, or system, model fidelity is never perfect. But as stated in reference [9], “essentially, all models are wrong, but some are useful”. As highlighted in reference [8], in science the value of a model lies in how well its properties match those of the target, whereas in engineering the value of the target lies in how well its properties match those of the model. A scientist constructs models to help understand the target. An engineer constructs targets to emulate the properties of a model, since for an engineer a model represents a design and the target is the implementation. These two uses of models are complementary.

For CPS modeling and control, therefore, it is critical to always keep a clear mind in terms of the *thing*, the *model*, the *purpose* of the model, and the *interactions* between the thing and the model in either a science or engineering context. For example, simplicity and clarity of target semantics may dominate over accuracy and detail, and optimizing over a model does not necessarily bring about desired effects or benefits to the target.

Moreover, it is important to note that models (and analyses and controls over these models) have their inherent *region of operation*, a concept commonly known in each individual disciplines but unfortunately often ignored in real-world practices. This is particularly true in CPS where multi-layer models exist, and their interactions lack sufficient attention. The re-invigoration of AI/ML makes this awareness even more relevant, in terms of where and when AI/ML could help analyze and even take over some control of the cyber physical system without adversely affecting the physical or cyber operation stability. Special care must be taken to understand the boundary of each model, interactions among models, appropriate positioning of AI/ML models and algorithms, and anticipated and measurable effects in the physical world.

1.2.2 CPS-specific Cyber Security Challenges and Solutions

Traditional cyber defense for CPS has mainly focused on the level of human-machine interface (HMI) and security information and event management system (SIEM). This is largely due to the similarity to established cyber protections for hosts and networks, and the information technology (IT) mindset possessed by practitioners. However, this leaves the lower-levels of the cyber physical system vulnerable to attacks not common in a traditional IT environment.

Protection of low-level components and subsystems includes protecting the interconnect and computation or logic of the controllers. In general, cryptographic protections provide a way to disrupt potentially rogue modules from snooping at the bus. Although this is effective for protection, it might be considered unsuitable since the bus data is mainly useful only in real time when interpreted in context of the control model and physical situation. The overhead is simply too high for each involving module on the bus to conduct encryption and decryption constantly. In addition, compromises at the controller level, e.g., rogue control signals issued by the compromised controller, render encryption irrelevant (encrypting the rogue data does not help security or resilience), or even harmful since the attack traffic/attacker communication is protected by encryption.

Protecting the controllers themselves includes (and is not limited to) fault avoidance, fault tolerance, and model- or reference-based CPS security. Formal methods which attempt to reason over certain properties of an *implementation model* against a *specification model* is the dominant technique for fault avoidance. Considering the typical tractability and practicality of creating both models, using formal methods is an excellent approach for achieving CPS security and resilience via fault avoidance. But this would imply the complete redevelopment of the system (or at least the subsystem subject to formal methods) from scratch. This is very expensive for legacy systems which are prevalent in industry and military applications.

Fault tolerance is a complementary approach to fault avoidance. This method assumes that vulnerabilities exist in the controller code and strives to mitigate the effect of exploits by ensuring proper operations of the physical system part of CPS, even under successful cyber attacks, thus rendering CPS resilience. Fault tolerance methods often involve detection and recovery, including stateful component, subsystem or system level recovery.

Reference/model-based CPS security relies on the fact that a CPS, unlike general IT systems, is generally well constrained within its operation space and intended behaviors. The operations are periodic and predictable, and reference models for algorithms and the operating environment can be developed and used to detect discrepancy between the observed operation and models. Discrepancy beyond some tolerance threshold may indicate flaw, damage, disruption, or exploits.

1.2.2.1 Cyber Attacks against CPS and Critical Infrastructure

Our goal for CPS resilience is to have the physical systems behave properly regardless of fault or disruption (cyber or otherwise). In keeping with reality, we make no assumption that a system is devoid of bugs or vulnerabilities. Rather, we seek to enable a CPS to tolerate and live with existing bugs and vulnerabilities it may have.

We assume an Advanced Persistent Threat (APT)-like adversary, whose goal is to create maximum disruption, major damage, and difficult and lengthy recovery time. To defeat system protection & fault tolerance and to achieve maximum disruption & major damage, an adversary generally needs to subvert and affect many individual controllers (various systems components) simultaneously and in coordination. Uncoordinated one or two subversion and denial of service attacks are unlikely to cause major disruption or damage.

There are generally two methods to subvert or negatively-effect the behavior of a controller: (a) manipulate or inject malicious input to cause improper control output, or (b) hijack and own the controller via either rogue reprogramming command (from console) or malicious input that corrupt program execution, hijack the program control and own the controller. Note that cyber attacks that leak (confidential) information can be used to gather intelligence and help plan for an attack, but by itself cannot subvert the operational behavior of a cyber physical system.

To significantly influence a set of controllers of diverse functionalities and types, an adversary will need to inject many different inputs/signals in a coordinated manner, which is difficult to achieve in practice and often requires the adversary to own many controllers to perform coordinated, multiple signal injections. The most dangerous exploit is when devices/controllers were stealthily taken over one by one, and then upon triggering event(s), simultaneously act (in coordination) and disrupt the systems. Stealthily owning controllers are the prerequisite for APT-like coordinated attacks. An adversary can stealthily own a controller in several ways. One of them involves reprogramming (re-flashing) the controller itself, e.g., in the case of Stuxnet. To do this, the adversary will generally have to own either the maintenance laptop or the human machine interface (HMI) console, and issue malicious updates from the corrupted laptop or console. This risk can be reduced by requiring multi-factor authentication for firmware update/re-flashing. Another method for owning a controller would be to exploit a (software) vulnerability, and send/inject malicious inputs that will corrupt and take over the controller.

Byzantine fault tolerant++ (BFT++) is a family of cyber resilience methods that rely on the periodicity of CPS and the physical inertia to tolerate cyber attacks. The BFT++ family of CPS resilience prevents this particular class of methods for hijacking and owning the controller. Additionally, BFT++ is generic and agnostic to the particulars of malware or malicious inputs. Refer to subsection 1.2.2.6 for the detailed description of BFT++.

1.2.2.2 Anatomy of CPS/Controller Owning Cyber Exploits

Fault tolerance systems, such as byzantine fault tolerant (BFT) and quad redundant control (QRC), have been proven effective for safety critical systems. They rely on redundancy to detect and recover from faults, and essentially provide fault tolerance against natural disruption and random faults.

Cyber attacks present a new type of challenges. They can force faults in many components and subsystems simultaneously, which leads to a "common-mode failure" that traditional fault tolerance cannot effectively deal with. Worse, if the adversary is successful in compromising a component, there is no obvious fault signal to detect, and the controllers continue to actuate the system while compromised and under the control of an adversary. Attempts to deal with common-mode failures have been made through diversification, but the type of diversification must be appropriate to the class of causes of common-mode failures that the CPS owner wants to mitigate, and special care must be taken with respect to what, when and how much diversification is deployed depending on CPS and mission requirements.

The process of a cyber exploit involves two virtual stages: first, exploiting a flaw/vulnerability in the program's code to alter its intended execution path, and second, taking control of the system to execute the attacker's commands. This is analogous to a fumble in football, where an opposing team must not just cause a fault, but recover the ball to gain possession, as shown in Figure 1.3. A successful exploit will succeed in both stages, leading to compromised systems under attacker control. A condition when the first stage is successful but the second stage fails will generally manifest into a crash, due to corrupted (as opposed to compromised) cyber components and subsystems.

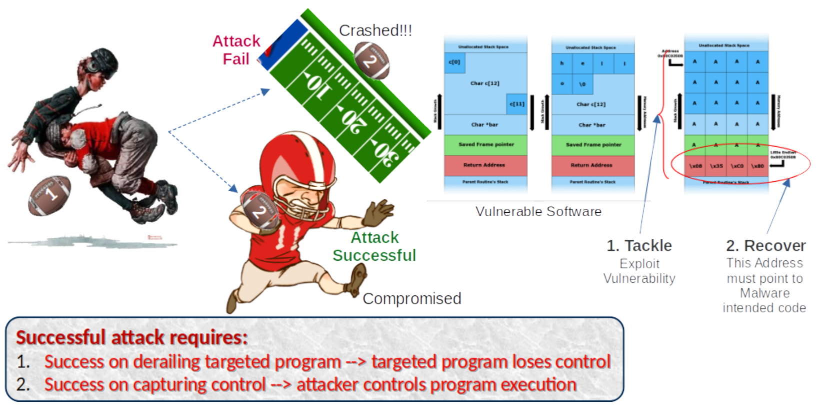
Attack Fail
Crashed!!!
Attack Successful
Compromised
Vulnerable Software
1. Tackle
Exploit Vulnerability
2. Recover
This Address must point to Malware intended code
Successful attack requires:
1. Success on derailing targeted program --> targeted program loses control
2. Success on capturing control --> attacker controls program execution

Fig. 1.3 Two Virtual Stages of Cyber Exploit

1.2.2.3 Defense: White Listing and Operation Segmentation

Many CPS operational environments can be analyzed and segmented into several different modes of operation. Within each of these modes, the set of valid (allowed) operations can be whitelisted. Operation segmentation is analogous but orthogonal to network segmentation. While network segmentation limits exploits effects and propagation by limiting allowable communications, operation segmentation prevents CPS disruption by limiting incompatible and hazardous co-occurrence of operational commands/events.

For example, let us consider the cyber physical systems that control the operation of a ship, and for the purpose of illustration assume that the engineers decided to segment the ship operation into three modes: steaming mode, in-port mode, and maintenance mode. Maintenance mode is akin to the superuser mode in modern operating systems where (almost) all operations are allowed. For simplicity, let us consider three different operations: dropping anchors, brisk-steaming (above 5 knots), and re-flashing the controller. One can see that dropping-anchors and brisk-steaming are mutually exclusive. It will not be prudent to drop an anchor while briskly steaming, hence dropping anchors is not within the steaming mode whitelist, and brisk-steaming will not be in the in-port mode whitelist. Similarly, re-flashing a controller should only be performed in the maintenance mode.

Operation segmentation improves the operational safety of a cyber physical system. Separating the maintenance mode from other operation modes also enhances the system's cyber security posture by whitelisting out disallowed behaviors and requiring additional privilege for critical activities, such as re-flashing a controller. While it cannot completely prevent cyber attacks, operation segmentation erects barriers against various malicious activities that may otherwise readily perform once a foothold is obtained in a component or subsystem.

Operation segmentation focuses all the working aspects of the cyber physical systems onto the operators, which are responsible for the CPS operations, including approving and initiating CPS maintenance. This is a judicious method compatible with the principles of separation of duties and least privilege for building computer systems [10]. Current trends in modern cars, which are systems of cyber physical systems, however, are diverging from this philosophy. In the case of modern cars, it is the manufacturers who often initiate the system update, with or without the awareness of the operator (owner). There are both pro and con arguments that can be made for this context.

1.2.2.4 Defense: Reference Model Based CPS Security

Since cyber physical systems extensively communicate with their physical environment, system security relies not only on cyber security but also on securing the physical part of the system. This means the cyber layer, as well as the platform (including the physical) layer and their inter-dependency, must be considered together. For example, the platform layer covers the whole run-time environment containing artifacts like operating system and middleware, as well as the physical part of the system such as sensors and actuators, etc. Hence, as an entity that senses and interacts with the real world, a CPS could be exploited by an adversary and cause harmful impacts. Depending on the level of the attacker's access and capabilities, either or both sensing components and control software can be subjects of a compromise.

One of the most common security approaches for detection of attacks against control software is a comparison of true and faulty signals, thus necessitating trusted redundancies. For example, if an extra electronic control unit (ECU) hardware is retrofitted to the robotic vehicle with no access channel from the outside, it is shielded from the attacker, and hence can be trusted. Such CPS can still operate as intended with its original control software, while the control signal can also be used to enable comparison against the retrofitted ECU for attack detection and response. Instead of changing the original control system, an external piece of hardware can be used to monitor the given ECU with minimum modification to the original system. Independently implementing the CPS control and sensing logic software on the external hardware enables high-accuracy error detection.

Such combination of the software and hardware redundancy has been proven to successfully detect a variety of attacks on the sensor, controller, vehicle dynamics, actuator, and controller operating systems [4]. The attack detection must combine control algorithms such as state-estimation, fault detection and diagnosis, fault tolerant control parameter and controller estimations to detect CPS dynamic changes. Specifically, it detects changes in the original system by comparing instantaneous outputs in real-time, while it is shielded from attackers. The entire diagnosis process can cover both the cyber and physical domains. A smooth variable structure filter (SVSF) was used to estimate system states and identify system parameters, which is proven to be robust to model uncertainty and noise. The system diagram of such an approach is shown in Figure 1.4.

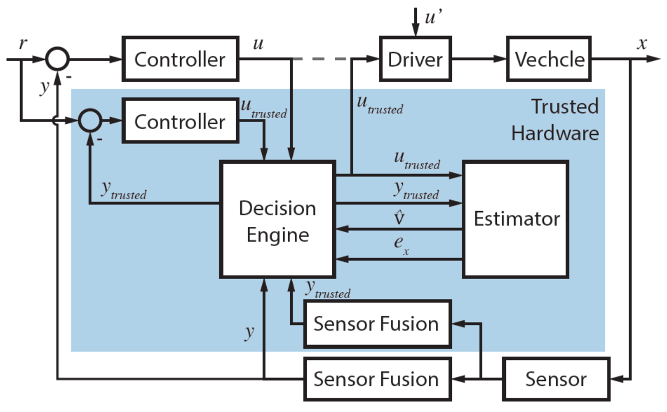

Fig. 1.4 System diagram of the FDI approach using hardware redundancy (reprinted from [4])

Raw sensor data are extracted and used to compare with the feedback from the original system to determine if the sensor fusion result or code in the original system has been modified. The decision engine will integrate the error between the two measurements within a fixed time window and identify sensor attacks using thresholds. The sensor fusion results are also fed to its internal controller and SVSF-based estimator for further security diagnosis and attacks detection.

The sensory system is also critical for CPS safety. Recent advances in adversarial studies demonstrated successful sensing fault generation by targeting the physical vulnerabilities of the sensors. A complete CPS sensor safety design must contain both an fault detection and isolation (FDI) unit and a fault recovery (FR) function to tolerate the detected flaws. When faults are identified and isolated by the FDI, the recovery logic should then be able to maintain the correct state with as much stability as possible using the remaining incomplete sensory systems.

As CPS sensors, the actuation system is also vulnerable and can be easily compromised via similar cyber and physical domain strategies. Actuator failures not only affect the normal operations, due to the implicit dynamics from the actual system, they also introduce the need of FDI design to distinguish the exact sensing and actuation faults. A flawed sensor may induce multiple state anomalies simultaneously and CPS may yield a similar abnormal action to two completely different types of failures (e.g., sensor or actuator failure which complicates pinpointing the failure source). Hence, to achieve proper FDI capability, the inherent coupling of the CPS dynamics must be considered. Further, when multiple CPS states malfunction simultaneously, it is hard to identify the exact failure sensor. The cascading effect may also have to be considered. For example, the high-level sensor abnormalities can affect low-level sensing in cascade for UAVs (e.g., attitude twitching can be subject to the frequent loss of position feedback).

One major circumstance to help sensor recovery is the fact that in most cases, excluding the most catastrophic, all the onboard sensors cannot be rendered defective at the same time. It is hard to compromise multiple sensors simultaneously because they typically measure different physical terms and possess distinct working principles, communication methods, and signal bandwidth.

To recover the sensor readings, installation of a backup sensory system is the most widely used approach. Through a simple comparison and replacement, this hardware redundancy is effective against the traditional software-based sensor faults and attacks, such as numeric error, trojans and data spoofing. However, this approach is not sufficient when the CPS encounters some well-designed attacks that concern both cyber and physical properties of the targeted sensors. This is because the redundant sensors exhibit the same physical vulnerabilities as the original ones. For example, a redundant attitude sensor would be incapable of nullifying the effects of resonating the inertial sensors via external excitation. It is highly likely both the original and redundant sensors would fail.

As an alternative to a redundant hardware approach, the redundancy-free methods for CPS sensor FDI and fault recovery (FR) as reported in [11] make the most sense. Figure 1.5 shows the system diagram, which consists of a fine-grained sensor FDI architecture and a sensor complementary FR in parallel. For fine-grained FDI, a basic

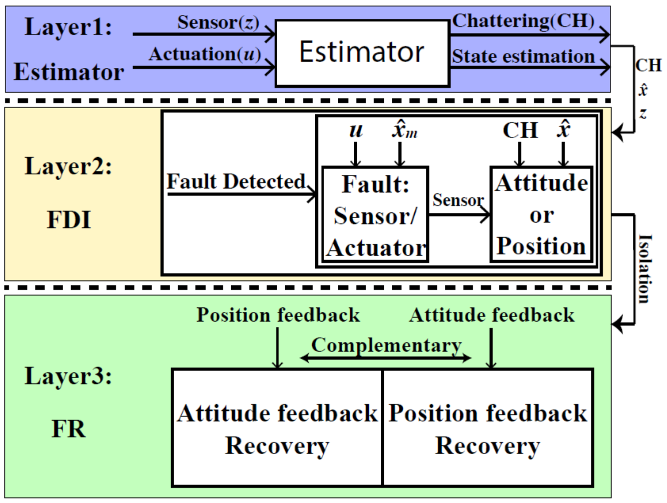

Fig. 1.5 FDI/FR without hardware redundancy (reprinted from [11])

state estimator for a rough early warning of faults combined with the un-measurable actuator state and modeling uncertainties are utilized. Instead of adding auxiliary sensors, the method uses the original sensor arrays and leverages complementary sensor estimations for FR implementation.

The FDI design is based on smooth variable structure filter (SVSF), which is a sliding-mode-based state estimator with a prediction-correction workflow. In one iteration, a model-based prediction function generates *a priori* state estimation first, then a discrete corrective action is taken by adding a corrective gain. The corrective gain is not only used to guarantee the stability of the estimator but also rectify the bounded estimation error robustly. Subsequently, the updated *posteriori* estimation and state measurement carry out the next iteration. The fault is detected by examining sensor FDI procedure-residual check (i.e., the discrepancy between estimation and actual sensor reading). If the residual rises beyond a certain threshold, a fault is supposed to be present and will be reported to begin the recovery.

The sensor fault recovery without hardware redundancy is CPS domain specific. For example, for UAVs, due to the geometric correlation of the vehicle dynamics, position and attitude feedback can be used to compensate each other. During the recovery process, with the fine-grained FDI, the compromised sensor reading is rejected and compensation from other trusted sensors will be utilized. For example, in case of an inertia sensor failure, position information can be used to derive an alternative attitude for flight control. When the UAV loses its position feedback, the inertia measurement can be utilized to compensate position drift.

In summary, reference models provide feasible and robust means for state estimation, behavior prediction, discrepancy checking, decoupling of sensor and actuator faults, and diagnosing multiple faults and accurately isolating the source of faulty elements, thus offering a well-grounded base for building security and resilience in cyber physical systems.

1.2.2.5 Defense: Vulnerability Prevention

To prevent against the first stage of a cyber exploit (see Section 1.2.2.2), CPS software needs to be devoid of any exploitable vulnerability. Fault or vulnerability avoidance generally falls within the first quadrant (P(S), C(S)) of Figure 1.2. CPS software can be analyzed against exploitable vulnerability, and the location where a vulnerability is identified will be *hardened* with security checks or assertions. Software vulnerability analysis generally uses both static (e.g., symbolic execution) and dynamic analysis (e.g., fuzzing) tools for finding exploitable vulnerabilities. *Formal verification* is another approach for assuring that the software is devoid of flaws or vulnerabilities. We will describe hardening and formal methods in what follows.

Security hardening

Vulnerability analysis and hardening is usually performed in several steps: (i) static software program analysis, (ii) instrumentation, (iii) symbolic execution, and (iv) fuzzing with dynamic tracing/feedback-guided fuzzing with sanitizing.

Software program analysis includes combination of static analysis (e.g., dependency analysis, program slicing, etc.) and a symbolic exploration of the program's state space (e.g., "can we execute it until we find an overflow?" or "let's execute only program slices that lead to a memory write to find an overflow."). Symbolic execution also takes advantage of instrumentation for precise results.

Fuzzing is a form of software testing where an application is run with random (potentially malformed) inputs while monitoring the runtime for unexpected behaviors, e.g., crashes, memory exhaustion, or infinite loops. There are generally two types of fuzzing: (i) *Blackbox fuzzing* (i.e., fuzzing with no knowledge about the target application) may not be effective in many cases as most inputs are likely to explore very shallow code paths. This severely limits the fuzzing ability to uncover bugs in deep parts of the code. (ii) *Coverage-guided fuzzing* tackles this problem by using program traces generated by the inputs as a feedback mechanism to tailor future inputs to the fuzzing target.

In essence, fuzzing depends on program crashes to detect and report bugs. Consequently, bugs that do not trigger crashes are not caught through fuzzing. Therefore, for effective fuzzing, software must be instrumented with sanitizers (e.g., a memory checking code such as memory leaks and initialization, heap and stack overflows, illegal accesses, etc.) either at compile time or at the binary level.

Program analysis and instrumentation can be performed for newly developed software during CPS software code compilation, or, on the available binary code for legacy software. For the former case (i.e., compiler-time analysis and hardening), the mainstream software build tools (e.g., GCC and LLVM) provide extensive interface and framework for analysis and code optimization in its Intermediate Representation (IR) form during code compilation. For the latter case (i.e., binary analysis and rewriting), analysis in the form of symbolic execution with various static analyses on binaries is performed as three distinct steps: (i) loading a binary into the analysis program, (ii) translating a binary into an IR, and (iii) performing the actual analysis.

The rewriting part for instrumentation and hardening presents a few difficult challenges. Specifically, dis-ambiguating reference and scalar constants, so that a program can be "re-flowed" (i.e., having its code and data pointers adjusted according to the inserted instrumentation and data section changes) is a major challenge. During assembly, labels are translated into relative offsets or relocation entries. A static binary rewriter must recover all these offsets correctly. There are three fundamental techniques to rewrite binaries: (i) lifting the code to an intermediate representation, (ii) trampolines, which rely on indirection to insert new code segments without changing the size of basic blocks, and (iii) reassemblable assembly, which creates an assembly file equivalent to what a compiler would emit (i.e., with relocation symbols for the linker to resolve).

Lifting code to IR for recompilation requires correctly recovering type information from binaries, which remains an open problem. Trampolines may significantly increase code size and do not scale very well. Consequently, we believe that *reassemblable assembly* is the most promising approach, which creates assembly files that appear to be compiler-generated (i.e., do not contain hard-coded values but assembly labels). Symbolizing the assembly allows security-oriented rewriters to directly modify binaries, which is similar to editing compiler-generated assembly files. Once modified, the symbolized assembly files can be assembled using any off-the-shelf assembler to generate an instrumented binary.

There are a number of security mechanisms and sanitizers to harden programs written in unsafe languages, each of which mitigating a specific type of memory error. It includes various memory checking techniques, undefined behavior monitors, control flow integrity trackers, temporal safety enforcers, etc. The major problem is that the execution slowdown caused by various security mechanisms is often non-linearly accumulated, making the combined protection prohibitively expensive.

One of the most viable approaches to mitigate this problem is to use *security diversification* consisting of N variants that are both functionally identical in normal situations and behaviorally different when under attacks. Hence, although each program version may be vulnerable to certain types of attacks, the security of the whole system relies on the notion that an attacker has to simultaneously succeed in attacking all variants in order to compromise the whole system. In addition, different and even conflicting security mechanisms can be combined to secure a program while reducing the execution slowdown by automatically distributing runtime security checks in multiple program variants [12]. This can be achieved by making sure that conflicts between security checks are inherently eliminated and execution slowdown is minimized with parallel execution. The N-version execution engine synchronizes these variants so that all distributed security checks work together to guarantee the security of a target program. The notional workflow diagram is shown in Figure 1.6.

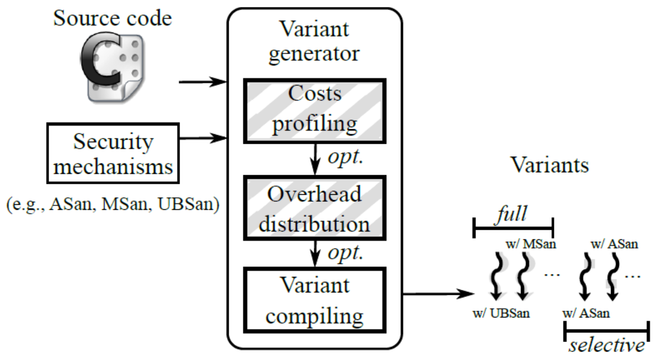

Fig. 1.6 Variant Generation Workflow (reprint from [12])

Formal Methods

While program analysis tools are indispensable to find vulnerabilities, they essentially explore part of the state and execution space, and hence can never provide complete guarantees. Formal methods provide complementary means for vulnerability prevention. Generally speaking, formal methods are system design and analysis techniques that use rigorously specified mathematical models to build software and hardware systems. In other words, formal methods use mathematical proofs as a complement to system testing (e.g., fuzzing) in order to ensure correct behaviors.

Using the terms mentioned in Section 1.2.1, consider a binary executable as the *thing*. In general, the *Concrete Model* or M_c is comprised of (but not limited to) an instrumented C or C-like program (source). The concrete model represents the actual executable but is more generic, meaning some properties are present in the model that are not reflected in the binary. An *Abstract Model* or M_A (for reasoning) is comprised of formal constructs such as Hoare's logic (or its variants such as separation logic, higher order logic, etc.). It can be derived from the binary or M_c and captures all properties in binary or M_c but is more generic and might include properties not in binary or M_c. In an ideal world, no error in lifting or abstraction is made and the three concepts (executable, M_c, and M_A) are all identical. In reality, however, this is not the case and much research has been conducted to shorten the gaps between them. One notable methodology is counter-example guided abstract refinement (CEGAR) where M_A is iteratively checked against a given property and refined if the check fails [13]. Most of the formal verification efforts adopt a similar model refinement approach.

Over the last decade, the understanding of formal methods and development of tools have improved to the point where formal verification of real-world software has started to become feasible. Examples include functional correctness proofs of microkernels and cryptography libraries. Formal methods have also been used to identify deep vulnerabilities in software, revitalizing the field of program analysis. With respect to vulnerability prevention, seL4 is the first formally verified microkernel with a functional correctness proof of the abstracted source code against the specification, effectively asserting the absence of typical programming errors such as null pointer dereferences, buffer overflows, and arithmetic exceptions [14]. The development of seL4 was supported by the DARPA High-Assurance Cyber Military Systems (HACMS) program, which aimed to create technology for the construction of high-assurance cyber-physical systems, where high assurance is defined to mean functionally correct and satisfying appropriate safety and security properties. Since it's original proof over a decade ago, seL4 has seen reasonable successes in both continual development and early adoption. For example, due to the offered higher-level of confidence for assurance, seL4 was selected by the AFRL Agile and Resilient Embedded Systems (ARES) program to serve as the separation kernel providing memory allocation and isolation based on the hardware memory management support.

However, while formal methods provide a rigorous way for vulnerability prevention, it is important to point out that the proofs are usually carried out between *models* (e.g., abstract and concrete models), as opposed to against the *thing* or binary executable in this case. For example, just because seL4 is formally verified does not necessarily mean the binary executable runs exactly as expected in the specification on the target computer architecture. Additional binary level assertion is still needed if the purpose is to provide execution assurance directly on the computer hardware. Again, this is a reminding and cautionary tale related to the *thing* and its *models*, as discussed in Section 1.2.1.

1.2.2.6 Defense: Vulnerability Tolerance

Most safety-critical systems utilize some type of redundant architecture to deal with faults. Examples include hot backups; dual, triple, or quad-redundant architecture; or Byzantine fault tolerance where assumptions about the fault conditions are random, and faulty replicas may behave arbitrarily. Fault tolerance provides a means to automatically deal with faults and recover from them. Cyber attacks, however, will drive fault tolerant system into *common mode failures* (see Section 1.2.2.2). The challenge is how to retrofit existing fault tolerance architecture to rectify faults caused by cyber attacks.

A typical cyber physical system offers certain properties and advantages one would not find in a general IT system. This is because the physical aspect allows for a certain degree of predictability in the behaviors of the system.

- **Periodicity:** The cyber subsystem that directly interacts with the physical plant runs in continuous cycles. For example, throughout its execution the controller reads values from sensors, calculates the error correction signal, and writes out actuator values. For the commonly used industrial controllers, Programmable Logic Controllers (PLCs), this is called the *scan cycle*.
- **Inertia:** Any physical subsystem of a CPS must obey the laws of physics and physical systems inherently have inertia. The scan cycle of a controller is typically engineered to be fast enough such that an issue in a small number of cycles will

be dampened out by the existing inertia. The cycle frequency is set depending on the system but common values vary anywhere between 1 Hz and 1 kHz.

Due to this predictability offered by inertia and periodicity, anomaly detection approaches can be naturally used to detect anomalies and threats in the system. A resilience strategy can also be developed to detect attacks by monitoring actions such as subverting control flow, reprogramming controllers, or overriding sensors that are out of the normal operation ranges.

Cyber vulnerability (and attack) tolerance does not rely on the need for software to be devoid of vulnerabilities. Instead, it assumes that unknown vulnerability exists within the software and strives to maintain the safety and normalcy of system operation regardless. Vulnerability tolerance methods focus on the second stage of cyber exploits (see section 1.2.2.2) and will generally have to perform timely recovery within the limited time afforded by the physical systems' inertia, as the first stage of cyber exploits may have already occurred and the cyber systems may have been corrupted. In what follows we will describe some example techniques, tools and frameworks for providing vulnerability and attack tolerance, empowered by inertia and predictability unique in cyber physical systems. These include Software Brittleness, Byzantine Fault Tolerance++ (BFT++), You Only Live Once (YOLO), and CPS Cyber Resilience Architecture (CRA).

Software Brittleness

For some certain types of critical cyber physical systems, avoiding operating in degraded or compromised state is of paramount importance, and fast program exit and re-start (called *software brittleness*) is required when a cyber attack succeeds and the program control is lost. Examples include Industrial Automation and Control Systems(IACS), Supervisory Control and Data Acquisition (SCADA) control systems and devices, Programmable automation controllers (PAC), remote terminal units (RTU), Master terminal units (MTU), intelligent electronic devices (IED), etc. Software brittleness is a novel concept enabled by the new design space, i.e., the fourth quadrant with P(S) and C(U) as shown in Figure 1.2b. Essentially, the inherent physical inertia allows enough room for cyber components to reconstitute themselves via fast crash-and-recovery.

Code randomization/diversification for software brittleness can be implemented at either pre-distribution or post-distribution stages. Both types of diversification (i) provide the level of code diversification sufficient to guarantee that an attack that succeeds in the original program will fail in the variants, and (ii) assure prompt attack discovery through self-monitoring capabilities of the diversified code. Using N-voting system with simultaneously running multiple generated variants will assure prompt discovery and recovery. There is an integrated set of diversification techniques available at both the source and binary code levels against most known attacks (e.g., memory corruption, code injection and re-use, control flow hijacking, information leaks, etc.). The conceptual approach toward software brittleness, called Binary code Randomization for Attack Sensitive Software (BRASS), is shown in Figure

1.7. It has been demonstrated that this approach provides prompt attack discovery and program abort & recovery with low performance and size overhead [15]. In the CPS context, software brittleness can be included in controllers, for example, and managed through some vulnerability/fault tolerance framework, which comes next.

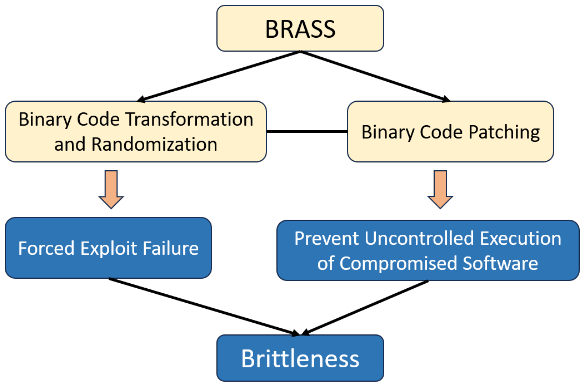
BRASS
Binary Code Transformation and Randomization
Binary Code Patching
Forced Exploit Failure
Prevent Uncontrolled Execution of Compromised Software
Brittleness

Fig. 1.7 The conceptual approach toward software brittleness

Byzantine Fault Tolerance++ (BFT++)

BFT++ is a family of cyber attack resilience methods that rely on the periodicity of CPS and the physical inertia to tolerate cyber attacks [16]. The initial concept of BFT++ was developed by the Office of Naval Research (ONR). It operates in the fourth quadrant (P(S), C(U)) in Figure 1.2b. MITRE Corp. maintains a reference design for BFT++ for the NAVY and DoD in general. BFT++ has been demonstrated to withstand US-NAVY sponsored ”Hack The Machine” hackathon.

Figure 1.8 illustrates the main components of the BFT++ design. It is built over the classical BFT systems. *Artificial diversity* in the form of diversified software compilation or diversified processors (ISAs) is used to break common-mode failure (C0 & C1), and delayed input sharing (delay queue) is inserted into the input of the backup unit/controller for stateful (warm) recovery. For a CPS system that does not demand stateful recovery, the delay queue and backup unit (C2) can be omitted.

BFT++ uses artificial software diversification applied to existing code. As an alternative, two (or more) distinct processors of different instruction set architectures can also be used to provide diversity. When used in combination with traditional fault tolerance architecture, this is also effective at absorbing (and tolerating) cyber faults. Note that attackers only have one opportunity per scan cycle to provide corrupting inputs. Diverse replicas will have different code layouts, making it almost impossible for attackers to inject malicious code that works across all replicas simultaneously. Due to the real-time nature of the periodic control loops, synchronization across replicas is built-in, and an attack can be detected if the program results vary across replicas or if timely responses are not provided.

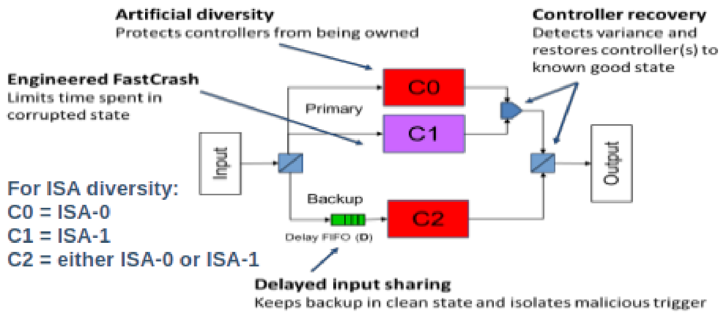
Artificial diversity
Protects controllers from being owned
Controller recovery
Detects variance and restores controller(s) to known good state
Engineered FastCrash
Limits time spent in corrupted state
C0
C1
C2
Primary
Backup
Input
Output
Delay FIFO (D)
For ISA diversity:
C0 = ISA-0
C1 = ISA-1
C2 = either ISA-0 or ISA-1
Delayed input sharing
Keeps backup in clean state and isolates malicious trigger

Fig. 1.8 General Design of BFT++ Cyber Attack Resilience Methods for CPS

The next crucial step is enabling the system to recover. Diversity within the system can make it more fragile, so fast-acting and automated recovery must be employed to counterbalance this. Without recovery, the attacker could maintain control of one (compromised) replica and leave the others crashed – a clearly unacceptable state.

Crash detection is the first part of the recovery process. Ideally, we want to detect a potential compromise via a crash of one of the diversified replicas as soon as possible. In our case, a replica failing to produce timely output by the end of the epoch is considered to have crashed. This serves as a canary that there has been a compromise.

Next, a small message queue is employed in front of one of the replicas (henceforth referred to as the ”protected replica”). This is key, because when a potential compromise is detected (via the crash detection), the message(s) triggering said crash are trapped in the queue before reaching the protected replica. Upon crash detection, this queue can be flushed removing the offending messages. While this introduces a small delay to the protected controller, the physical inertia of the system allows BFT++ to absorb this without impact to the real-time operation.

Finally, recovery begins, and the state of the replicas are restored. Restoring from a checkpoint is possible but requires much resources to handle the overhead of saving checkpoints as well as a way to deal with the staleness of state upon a restore. Instead, the strategy advocates designating one or more replicas as backups and time-delaying them, so they process inputs one or more cycles behind the primaries.

This method for cyber resilience has allowed older control systems to identify cyber attacks during their normal operation, automatically triggering a quick and efficient recovery process. However, there is still a concern that attackers may exploit this system behavior to launch an availability attack. While we have prevented any exploit from affecting the system’s integrity, it is possible for a known vulnerability or bug to trigger the recovery process and cause the system’s availability to be compromised. To address this issue, we designed a mitigation strategy known as ”Shims” that filters out any malicious inputs that cause the recovery architecture to send a crash signal. By implementing shims at the input point for the controllers, replaying an exploit after it has already been used against the system will be prevented.

For a particular BFT++ implementation [16], the architecture has three redundant diversified controllers operating in a traditional fault-tolerant architecture. The artificial diversity makes it difficult for a cyber attacker to compromise all controllers with the same malicious input. Although an exploit may be successful against one replica, it will cause the diversified replicas to crash. Next, it incorporates delayed input sharing (e.g., FIFO message queue) to trap bad messages before reaching a ”protected controller”. This introduces a delay to the protected controller, but ensures the system to continue operation and to be reconstituted after the cyber exploit. The recovery timing of the system is governed by several timing parameters, such as Tcrash, Tsc, D, Td, and Tr. Tsc, Td, and Tr are system parameters, and D needs to be appropriately set for automated recovery to be possible. The two critical points that determine the system's recovery timing are the *brittleness* of the controllers and how quickly the system can restore a controller to the normal state. The physical subsystems with higher inertia are generally more tolerant of losing control signals for a short time. In general, the following relationship between these parameters must hold for BFT++ to be applicable to a legacy cyber physical system [16]:

$$Tcrash \leq D * Tsc \leq Td - Tr$$

Parameters	Definitions
Tcrash	Time between attack and crash
Tsc	The scan cycle period
D	FIFO queue length (number of slots)
Td	Maximum control loss tolerable by physical systems
Tr	Recovery latency

You Only Live Once (YOLO)

YOLO is another CPS cyber resilience method that relies on the physical inertia to withstand cyber attacks. YOLO and its variants use periodic restart and does not require any redundant controller [17]. It also operates in the fourth quadrant (P(S), C(U)) in Figure 1.2b. YOLO was developed at Columbia University under the sponsorship of the Office of Naval Research.

Figure 1.9 depicts the design philosophy for YOLO. YOLO implements periodic restart to limit the duration of potential compromise. An adversary who managed to compromise the system only has control over the system for a maximum duration of the restart period. The YOLO restart period can be designed to be short enough to prevent an adversary to achieve persistence within the CPS system, while still within the tolerable region provided by the physical systems' inertia. During each restart, the controller is reset to its ‘clean’ state by loading its software from a read-only module and clearing out all the volatile memory. YOLO also implements software diversity

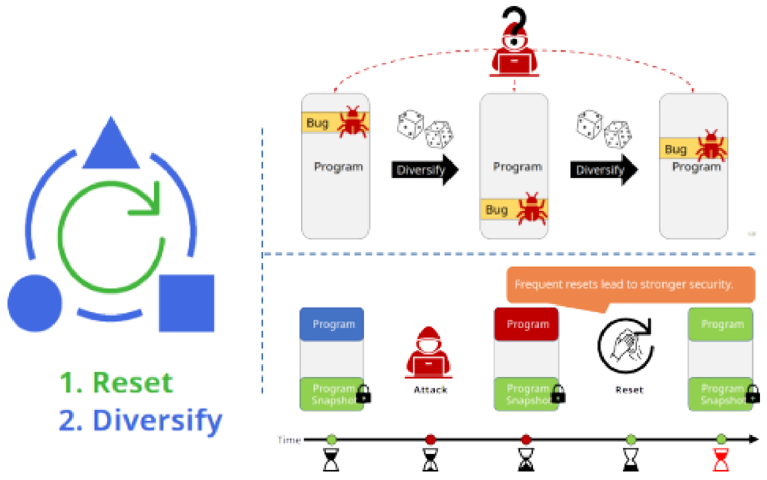

Fig. 1.9 General Design of YOLO Cyber Attack Resilience Methods for CPS [17]

after each restart to ensure that the attacker cannot exploit the same vulnerabilities. YOLO has been demonstrated to be practical for automotive engine management unit, drone controller and a missile launcher.

In YOLO, the restating latency and state recovery time need to be within the range that the inertia of the physical systems can tolerate. Proper engineering analysis and design is required to accelerate the restarting process and to avoid lengthy reboot and initialization latency of the cyber system. YOLO does not require replication, and hence it is cheaper to implement than BFT++. However, its protection is not as deterministic as that of BFT++, and current version of YOLO does not support stateful (warm) recovery.

CPS Cyber Resilience Architecture (CRA)

Existing CPS cyber resilience architectures, including BFT++ and YOLO, have been analyzed and summarized into a timing-based formulation framework [18]. Within this framework, safety analysis and computation of control policies and design parameters can be performed for each pair of CRA method and CPS application.

The framework relies on the insight that the cyber subsystem operates in one of a finite number of modes. It defines a hybrid system model that captures a CPS adopting any of these architectures (CRAs). Analysis within this framework uses the transition model of the hybrid system to derive architecture-agnostic sufficient conditions for control policy and timing parameters that ensure safety of the CPS. The analysis will then formulate the problem of joint computation of control policies and associated timing parameters for the CPS to satisfy a given safety constraint and derive sufficient conditions for the solution. Utilizing the derived conditions, they provide an algorithm to compute control policies and timing parameters relevant to the employed architecture. The framework efficacy has been demonstrated in a case study involving automotive adaptive cruise control. The study was performed for each of the CRA methods in their framework, and proved that the algorithm converges to a feasible solution under certain conditions.

Figure 1.10 visualizes the operation of a drone employing YOLO, under continuous cyber attacks. It shows three operation zones: desired operating zone, zone of tolerance, and danger zone. The drone is expected to operate in the desired safe zone, and danger zone can only be safely entered when the vehicle is in normal mode, otherwise catastrophic crash may occur. The vehicle (drone) can be safely restarted and recovered within the tolerated zone. Safety cannot be guaranteed if the drone enters tolerated zone in a corrupted state.

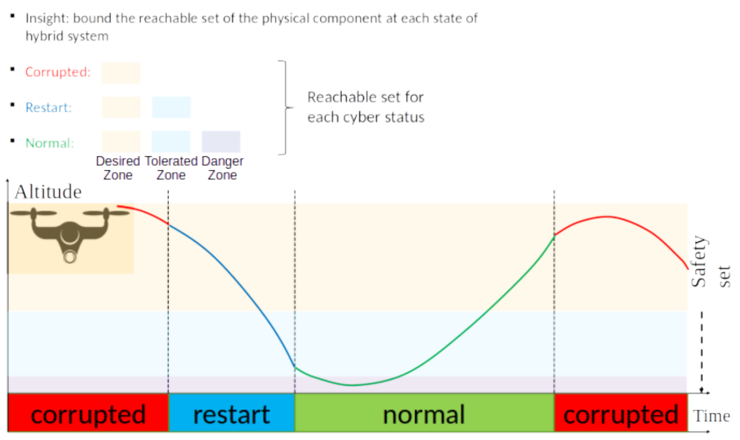
Insight: bound the reachable set of the physical component at each state of hybrid system
Corrupted:
Restart:
Normal:
Reachable set for each cyber status
Desired Zone
Tolerated Zone
Danger Zone
Altitude
Safety set
corrupted
restart
normal
corrupted
Time

Fig. 1.10 An illustration of CPS Safety Analysis within the Framework [18]

1.3 Machine Learning and CPS

In this chapter, the terms *machine learning* and *neural networks* will be used interchangeably. At the most general level of interpretation, machine learning is a super-set of statistical machine learning (i.e., neural networks). The term *machine learning* in general also includes learning heuristics and other logical and formal form of learning mechanisms. Contemporary use of the term *machine learning* generally refers to various forms of neural networks, which are often considered as surrogates for formal/logical control algorithms in the CPS context.

Robotic devices and vehicles can perform some of its functions using machine learning and especially reinforcement learning (RL), e.g., RL for drone fault recovery [19]. Training neural networks may be performed by guiding the robotic devices to function within its physical environment. Training may also be conducted in the virtual simulation environment [19], where the sensory and control input as well as the actuators and their dynamics are simulated using physics models of the actuators, sensors and the physical environment. The use of physics models for training a CPS system is generally safe as the laws of physics are universal, relatively complete, consistent and context insensitive.

Machine learning may also be used for correlating various monitored and logged events in cyber physical systems. It helps correlate cyber events such as network events, activation of computing events, sensed and computed parameters' values, etc., with observed physical and environmental events. Trained this way, machine learning models the operational behavior of the cyber physical systems and can be used to highlight unexpected behavior and anomalies. The use of machine learning in this case is inherently incomplete. While well trained machine learning algorithm/model is expected to generalize and cover the CPS operation space, there is no practical, assured way to ascertain that it covers all of the possible cases of the application/CPS-operation, e.g., corner and unexpected cases. Such machine learning model will produce false positives and false negatives. The quality of machine learning output (prediction) is significantly dependent on the methods, the quality of data, and models used for training. However, with proper operator due diligence and supervision, the deployment of machine learning can significantly improve the safety and security of CPS operation, as a complementary means to the traditional model-based mechanisms.

1.3.1 Enhancing CPS robustness with Machine Learning

It has been well established that the traditional cyber techniques in software and firmware can no longer sufficiently protect the system and ensure safe operation of the cyber physical systems when attacks are launched against the physical components of the CPS, such as signal spoofing or using sound wave to resonate the IMU sensors. As a result, undesirable performance or even loss of control would occur. Given that attacks/faults cannot be fully prevented, fault/vulnerability tolerance and CPS resilience and recovery strategies are required.

Traditionally, there are two types of fault-tolerant control: passive fault-tolerant control (PFTC) and active fault-tolerant control (AFTC). AFTC has a fault detection and diagnostics (FDD) component to identify the source of the fault, reconfigure a controller, and compensate for such fault. The FDD component is usually an observer and can generate residual signals to indicate the fault. Both sensor and actuator attacks or failures can be detected with system models. Meanwhile, PFTC does not have an FDD mechanism, but aims to improve the controller's robustness and tolerate the fault condition or attack. AFTC can pinpoint the fault and act accordingly, but if the FDD is not designed with care, the implementation could lead to delay in detection or false positives and greatly affect the performance. While PFTC cannot isolate faults, they could potentially achieve more robust performance.

Machine learning (ML) and reinforcement learning (RL) have been explored in developing FTC strategies. However, most of the ML/RL methods were only evaluated in simulation, and their real-world performance is unknown. Deploying reinforcement learning policies onto real systems in this case is extremely challenging since training has to be performed in simulation before trained models being transferred to real cyber physical systems to recover from sensor and actuator faults.

Reference [19] demonstrates that RL-based policy trained in simulation can indeed be transferred to real unmanned aerial vehicles to recover from sensor and actuator faults. Unlike traditional FTC, this policy does not require fault detection and diagnosis (FDD) nor tailoring the controller for specific attack scenarios. Instead, the policy runs simultaneously alongside the controller without the need for fault detection and activation. When the CPS is operating normally, the policy generates no or minimum control command adjustment and does not interfere with the operation. When the fault condition arises, or the CPS is under attack, the policy takes the state inputs and generates appropriate actuator command adjustment with little or no delay to compensate for the fault/attack condition.

For simulation, identical sensor fusion and control algorithms must be implemented so the closed-loop dynamics of the vehicle in the simulation can approximate the real vehicle. The approach allows for simulation validation statically through open-loop and closed-loop tests. The fault resilient policy optimization is formulated with a standard reinforcement learning problem, where the agent is the quadcopter and the environment is the simulated world. The attacks were implemented by replacing the actuator signal or the sensor value with a random number.

The training takes place and policy is implemented during dynamic simulation on existing legacy system through minimally intrusive software retrofitting. The control algorithm is used in a dynamic simulation during which a fault-tolerant policy is optimized using reinforcement learning to maintain CPS control under various simulated attacks. The RL-assisted fault-tolerant control workflow is shown in Figure 1.11.

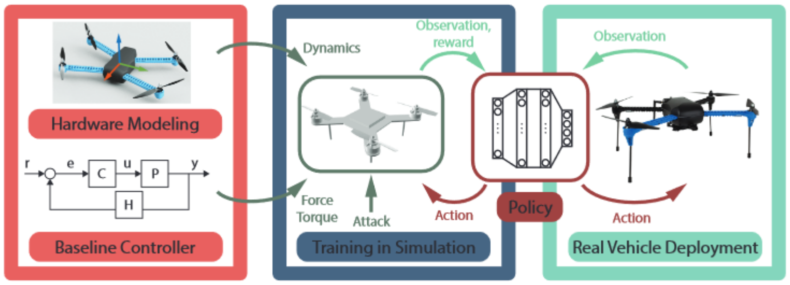

Fig. 1.11 RL-assisted fault-tolerant control workflow, derived from [19]

Since the system dynamics is largely deterministic, reference [19] uses the actor-critic deep deterministic policy gradient (DDPG) algorithm for training. Fully connected multi-layer perceptions (MLPs) serve as the function approximator for policy representation. MLP splits between a nonlinear control module and a linear control module. Intuitively, nonlinear control performs forward-looking and global control, while linear control stabilizes the local dynamics around the residuals of global control. This improves training sample efficiency, final episodic reward, and generalization of learned policy, while requiring a smaller network and being generally applicable to different training methods. Only nonlinear fault-tolerant policy needs to be learned. This approach views the policy as an optimized FDD and FTC control approximated by a neural network. It can be leveraged (with the proper simulation setup) for a variety of cyber physical systems with a learned policy designed as an add-on to other closed-loop mechanisms.

Another use of ML is to function as a surrogate ("Digital Twins") and to be used as a reference to detect anomalies and cyber disruption. In this role, the ML model will correlate cyber and physical events and flag any inconsistencies, plausible faults or anomalies. Such approaches extend machine learning (ML) methods for analyzing system logs of CPS and identifying the key CPS entities to reconstruct the critical steps of a plausible attack. Forensic analysts collect diverse system logs from multiple CPS components. The massive volumes of logs are often analyzed offline or monitored in real-time to debug system failures and identify sophisticated threats and vulnerabilities. There are several techniques being developed to extract features/sequences from logs to automate intrusion and failure detection and to discover associations among disparate log events through event correlation.

Working with the text logs requires integration of natural language processing (NLP) and deep learning techniques into data provenance analysis to identify attack and non-attack sequences. The typical steps include (i) processing system logs; (ii) building optimized causal dependency graph, from which the semantically augmented event sequences are constructed; and (iii) learning a sequence-based model that represents the attack semantics to recover key attack entities describing the attack story at inference time. The key challenges for such solutions are (i) additional overhead on a running system, (ii) integration of diverse logs, (iii) scalability of the large and complex causal graphs, (iv) accuracy of constructed sequences models, and (v) efficient automation.

One of the promising approaches is based on the assumption that the crucial steps of different attacks in a causal dependency graph may share similar patterns. This allows for identification of key attack steps through an attack symptom event, based on those sequences that share semantically similar attack patterns to the ones it had pre-learned. Such knowledge helps to substantially save time when investigating large causal graphs and helps in constructing the attack story from a limited number of attack symptoms.

1.3.2 Roles and Pitfalls of AI in CPS

As the complexity of automation increases, the roles machine learning may play in CPS are also expected to grow. The use of machine learning often requires an extensive set of labelled data for training, and the curation of this large, labelled data set is often problematic. While data can be scraped from the Internet, labelling them requires tedious manual effort, and is often outsourced to third-world country or Mechanical Turk (Amazon). It can be a very expensive proposition.

Fortunately, many CPS operations are governed by physics, with formulas and models that have been developed and proven over decades. Due to the computation limitation of many CPS devices, and the potential complexity of the interaction among physical phenomena, it is often not practical to deploy detailed physics models as reference to the operation of the CPS infrastructure. A surrogate – a (computationally) lighter weight machine learning algorithm/model – can be trained with labelled data generated by these complex physics models and practically deployed as the reference model. This surrogate model trades off precision and determinism/correctness with computation cost.

However, machine learning for cyber components at the level of software execution is quite challenging. Unlike natural language, image and video processing, there is no public, large-scale, comprehensive, and well-labelled data set that researchers can use for evaluating the efficacy of machine learning for cyber security and resilience. Research works in this area are often forced to develop their own data for training and evaluation. This effort is both expensive and non-comprehensive, limiting the quality and generality of the research effort.

The periodicity and predictability of CPS operation help reduce the overall challenge, as they potentially provide "structures" and "constraints" for the learning problem at hand. In machine learning, *knowledge* about the problem domain and relevant features extracted from the domain knowledge still play a critical role. Properly observing/incorporating physics-based models in the CPS software and machine learning process will help focus the training process, constrain the search space, and enhance the performance of the resulting machine learning model.

Transfer learning offers an appealing way to help reduce the size of training data needed to achieve reasonable performance. Employing transfer learning, one can adopt a suitable pre-trained ML model whose size and structure can accommodate the target problem space. The pre-trained ML model is expected to have its internal weight well configured and distributed, especially for the application it was trained for. It serves as the initial condition and foundation for training the target application. This pre-trained model will then be trained again with labelled data for the new (target) application.

Employing pre-trained model, the required training data is not as large as that of training the machine learning model from scratch. The trade-off is that the configuration and many of the hyper-parameters of the neural networks are not tunable and can incur the computing cost of employing larger than optimum (for the target application) neural networks. Large Language models, e.g., GPT-4, BERT, etc. are powerful examples of pre-trained models for natural language. It is harder to find a suitable pre-trained graph-based machine learning model, as the data encoding for graph neural networks is generally very specific to the application. Fortunately, the training data requirement for various graph neural networks tends to be modest. For cyber components and software execution that are typically represented in graphs, it is still unclear how and how well transfer learning may help in model training with reduced data set.

Generative Adversarial Network (GAN) offers another attractive method in dealing with training data. A GAN consist of two neural networks, the generator and the discriminator. In a GAN setup, the two neural networks contest with each other in the form of a zero-sum game, where one agent's gain is another agent's loss. The generator strives to generate samples that fool the discriminator, and the discriminator strives to accurately detect or classify the generated samples. Both neural networks are trained together and co-evolve against each other. After the initial setup, the generator and discriminator will challenge and train each other in an unsupervised manner. GAN requires minimal if any training data, making it very attractive for domains lacking large-scale, labelled data.

Unwise use of ML, such as employing generative adversarial network (GAN) without properly constraining it with physical models/rules, will likely violate the laws of physics and make it inappropriate or even dangerous to deploy. This is because unconstrained GAN will operate and explore solutions in an (un-grounded) virtual world with a much larger space than that of the physically constrained environment the CPS operates within. As GAN is a very attractive method, it is important to understand the problem space before deploying it. To illustrate, consider two slightly different applications. One is a valid application of GAN, and the other is not.

In the first application, a neural network is being developed for detecting malware (a discriminator). To anticipate 0-day malware, it is trained in a GAN environment – a malware generator neural network is developed and coupled with the malware detector in a GAN configuration, and then let loose (they play against each other). This is an appropriate and efficient way for inoculating the detector (discriminator) against 0-days.

In the second application, a dark-hat is mining for 0-day malware that is guaranteed effective against a target that is defended by VirusTotal. The dark-hat decided to deploy GAN, using a similar set up as the first application above. This is an ineffective solution, and the dark-hat will have false-confidence that his mined 0-day will be effective, for the following reason: his discriminator is not grounded to and does not represent VirusTotal. Developing a discriminator that can become a surrogate to VirusTotal will be very difficult if not impossible. VirusTotal and its evolution is influenced by factors that are not under the dark-hat control. The Dark-hat's discriminator will respond and evolve to the generator challenges in a manner that is independent of VirusTotal, and provide feedback to the generator that does not reflect VirusTotal behaviors. One can speculate that given enough resource and time, one can train a super discriminator that is better at detecting malware than ViriusTotal. However, unless one can prove or have well founded confidence that the superior discriminator is a complete superset of VirusTotal capability (no Malware that VirusTotal can detect the discriminator cannot detect), it still cannot provide the assurance that the synthesized malware will pass detection by VirusTotal.

1.3.3 Future Direction for AI in CPS

As discussed in previous sections, statistical models embodied as neural networks (machine learning) are effective in CPS related automation, including surrogate for control policy, automated fault recovery [19], surrogate as digital twin, anomaly detection, etc.

However, care must be taken in deploying neural networks as they are after all statistical machinery and hence cannot completely capture causality and are prone to make mistakes. Unless the utility property of the application itself is statistical, an error detection and exception handler will be required to detect and mitigate the effect of incorrect neural network results. An application is said to have a *statistical utility* if occasional mistakes are expected and tolerated, as long as their frequency is not too large (below a certain threshold) and the overall performance of the algorithm is still above the acceptable performance level. That is, in an application with statistical utility, only average matters and individual error does not. An application does not have a statistical utility if an individual error/mistakes matters.

CPS is a field where the inertia of the physical systems can tolerate a limited duration of errors. However, an individual CPS is susceptible to prolonged errors. A system of CPS devices provides additional resilience, as long as the effects of prolonged errors in a subset of the system components are generally observable within the systems, and the overall systems adapt to the anomalies, or an operator can be alerted for and rectify the operational anomalies.

Various forms of neural networks and various configurations of systems of neural networks have been deployed in CPS infrastructures. Machine learning will also excel in approximating the modeling and controlling of the behavior of a complex system whose behavior is not easily describable with logic or sets of logic. The role of neural networks and systems of neural networks is expected to grow in CPS and process control & automation in general.

Machine learning excels when the utility of the application itself is statistical, and when the application logic is extremely complex to be completely captured using logic or other formal methods. For this reason, an understanding of the problem's space, property and the environment surrounding the problem is the key for successful application of neural networks and the selection of the particular neural network algorithms. A good understanding of problem space will also help avoid pitfalls described in the previous section.

During our journey of studying the property of a problem or task and their potential solutions, the authors observed that *it is easier to solve problems of statistical nature with statistics*, and vice versa, *it is simpler to solve problems of logical nature using logical process*. This dichotomy is analogous to the dichotomy of frequency domain and time domain in signal processing. There are classes of problems that are simpler to solve in frequency domain, and there are other classes that lend themselves to have natural solutions in time domain. In general however, while less efficient or precise, statistical process can be used to approximate a logical one, and logical process (such as logic in modern digital computer) can emulate/simulate statistics.

A system of machine learning algorithms can be arranged in logical manners or simply feed each other for large scale automation. Properly designed systems of machine learning algorithms may mask or alleviate individual algorithm weakness and provide much more accurate and capable ultimate results. There may also be the case where it is prudent to include logical reasoning algorithms into the systems of neural networks, creating hybrid symbolic and statistical (neural networks) systems.

It can be argued that arranging multiple neural networks in a logical pipeline has already shown the promise of hybrid logical-statistical system design. For example, ONR developed the Learn2Reason concept [20][21], advocating the development of a hybrid system of neural networks algorithms and logic-based reasoning. The development of Learn2Reason was inspired by Daniel Kahneman's system-1 and system-2 concept with respect to thinking fast and slow [22]. Initial description of Learn2Reason [20] suggests a blackboard like implementation where the logical and probabilistic/statistical process interact, however, most of the research mentioned in [21] employed the pipelines structure. Recent news [23] indicates that Google's large language model employs logic, in term of generated program, to solve particular tasks where logical processes clearly surpass statistics, e.g., counting, performing arithmetic calculation, reverse spelling a word, etc. The article also stated that Google was following Kahneman's system-1 and system-2 concept [22] in this work.

A hybrid logical and statistical (neural network) based machine learning is the future. It allows for both statistical process and logical process to do what it can do best, and together they provide a superior performance than that of each individual type (logical or statistical). This hybrid learning system will find its place in CPS and CPS-based critical infrastructure of the future.

References

1. Stouffer K et al (2022) Guide to Operational Technology (OT) Security. *NIST ITL Computer Security Resource Center* https://doi.org/10.6028/NIST.SP.800-82r3.ipd
2. Higgins KJ (2023) OT Network Security Myths Busted in a Pair of Hacks. Available via DARKReading: ICS/OT Security https://www.darkreading.com/ics-ot/ot-network-security-myths-busted-in-a-pair-of-hacks Accessed 6 Sept 2023
3. Kim T et al (2020) From control model to program: investigating robotic aerial vehicle accidents with MAYDAY. In: 29th USENIX Security Symposium., Boston, August 2020
4. Fei F et al (2018) Cross-Layer Retrofitting of UAVs Against Cyber-Physical Attacks. In: IEEE International Conference on Robotics and Automation (ICRA), Brisbane, 21-25 May 2018
5. Son Y et al (2015) Rocking Drones with Intentional Sound Noise on Gyroscopic Sensors. In: 24th USENIX Security Symposium., Washington DC, 12-14 August 2015
6. Learning Introspective Control (LINC). https://www.darpa.mil/program/learning-introspective-control Accessed 6 Sept 2023
7. Faithful Integrated Reverse-Engineering and Exploitation (FIRE) https://defencescienceinstitute.com/funding-opportunity/darpa-fire/ Accessed 6 Sept 2023
8. Lee EA (2016) Fundamental Limits of Cyber-Physical Systems Modeling. ACM Trans. on Cyber-Physical Systems:1–26. https://dl.acm.org/doi/10.1145/2912149
9. Box GEP, Draper NR (1987) Empirical Model-Building and Response Surfaces, Wiley, New York
10. Saltzer JH, Kaashoek MF (2009) Principles of Computer System Design, An Introduction, Morgan Kaufmann, Burlington
11. Tu Z et al (2018) Redundancy-Free UAV Sensor Fault Isolation And Recovery. Preprint at https://arxiv.org/pdf/1812.00063v1.pdf
12. Xu M et al (2017) Compositing Security Mechanisms through Diversification. In: USENIX Annual Technical Conference (ATC'17), Santa Clara, 12-14 June 2017
13. Clarke E et al (2000) Counterexample-guided abstraction refinement. In: International Conference on Computer Aided Verification, Chicago, 15-19 July 2000
14. Klein G et al. (2014) Comprehensive formal verification of an OS microkernel. ACM Trans. on Computer Systems (TOCS) 32.1:1-70.
15. Briskin G, Li JH (2022) Binary code Randomization for Attack Sensitive Software (BRASS), Final Report to the Office of Naval Research, Available to US performers upon request and approval.
16. Mertoguno JS et al (2019) A physics-based strategy for cyber resilience of CPS. In: Proc. SPIE 11009, Autonomous Systems: Sensors, Processing, and Security for Vehicles and Infrastructure, Baltimore, 2 May 2019
17. Arroyo MA et al (2019) YOLO: frequently resetting cyber-physical systems for security. In: Proc. SPIE 11009, Autonomous Systems: Sensors, Processing, and Security for Vehicles and Infrastructure, Baltimore, 2 May 2019
18. Al Maruf A et al (2023) A Timing-Based Framework for Designing Resilient Cyber-Physical Systems under Safety Constraint. ACM Trans. Cyber-Phys. Syst. 7.3:1-25
19. Fei F et al (2020) Learn-to-Recover: Retrofitting UAVs with Reinforcement Learning-Assisted Flight Control Under Cyber-Physical Attacks. In: Proc. IEEE International Conference on Robotics and Automation (ICRA), Virtual, June 2019
20. Mertoguno JS (2014) Human Decision Making Model for Autonomic Cyber Systems. Journal on Artificial Intelligence Tools. 23.6:1-6 https://doi.org/10.1142/S0218213014600239
21. Mertoguno JS (2019) Toward Autonomy: Symbiotic Formal and Statistical Machine Reasoning. In: Proc. 1st IEEE International Conference on Cognitive Machine Intelligence. Los Angeles, 12-14 December 2019
22. Kahneman D (2013) Thinking Fast and Slow. Farrar, Straus and Giroux, New York
23. Amadeo R (2023) Google's Bard AI can now write and execute code to answer a question., ars TECHNICA (2023) https://arstechnica.com/google/2023/06/googles-bard-ai-can-now-write-and-execute-code-to-answer-a-question/ Accessed 6 Sept 2023