Vision Paper: Grand Challenges in Resilience: Autonomous System Resilience through Design and Runtime Measures

Saurabh Bagchi$^{\psi,\alpha}$, Vaneet Aggarwal$^{\alpha}$, Somali Chaterji$^{\alpha}$, Fred Douglis$^{\beta}$, Aly El Gamal$^{\alpha}$, Jiawei Han$^{\gamma}$, Brian J. Henz$^{\delta}$, Hank Hoffmann$^{\epsilon}$, Suman Jana$^{\zeta}$, Milind Kulkarni$^{\alpha}$, Felix Xiaozhu Lin$^{\alpha}$, Karen Marais$^{\alpha}$, Prateek Mittal$^{\eta}$, Shaoshuai Mou$^{\alpha}$, Xiaokang Qiu$^{\alpha}$, Gesualdo Scutari$^{\alpha}$

ψ: Corresponding author. All other authors are listed alphabetically.

α: Purdue University, β: Perspecta Labs, γ: University of Illinois at Urbana-Champaign, δ: Army Research Lab, ϵ: University of Chicago, ζ: Columbia University, η: Princeton University

1 Abstract

In this article, we put forward the substantial challenges in cyber resilience in the domain of autonomous systems and outline foundational solutions to address these challenges. These solutions fall into two broad themes: *resilience-by-design* and *resilience-by-reaction*. We use several application drivers from autonomous systems to motivate the challenges in cyber resilience and to demonstrate the benefit of the solutions. We focus on some autonomous systems in the near horizon (autonomous ground and aerial vehicles) and also a little more distant (autonomous rescue and relief).
For *resilience-by-design*, we focus on design methods in software that are needed for our cyber systems to be resilient. In contrast, for *resilience-by-reaction*, we discuss how to make systems resilient by responding, reconfiguring, or recovering at runtime when failures happen. We also discuss the notion of adaptive execution to improve resilience, execution transparently and adaptively among available execution platforms (mobile/embedded, edge, and cloud). For each of the two themes, we survey the current state, and the desired state and ways to get there. We conclude the paper by looking at the research challenges we will have to solve in the short and the mid-term to make the vision of resilient autonomous systems a reality. This article came out of discussions that started at the NSF-sponsored Grand Challenges in Resilience Workshop held at Purdue in 2019 with the co-authors contributing to going into the depth of the issues and then this article.

2 Introduction

We lay out our vision for resilience in autonomous systems and our view of the short-term and mid-term research challenges to realize the vision. Our view of resilience has two primary aspects.

1) *Resilience by design*: This is the aspect that designs and develops cyber systems so that they are resilient to a large set of quantifiable *perturbations*.
2) *Resilience by reaction*: This is the aspect that works at runtime when perturbations are incident on the cyber system and imbues the systems with the ability to "bounce back" quickly after a failure triggered by a perturbation.

Note, of course, that these two aspects of resilience are intertwined: systems can be *designed* so that they incorporate resilience by *reaction*.

We also make specific the notion of **perturbations** that we want to deal with. These take three forms: (i) *natural failures of hardware or software* (due to bugs, aging, misconfigurations, resource contentions in shared environments, downtime due to planned upgrades, etc.), (ii) *maliciously induced failures* or security attacks (from outside the system), and (iii) *unexpected inputs* (our target class of autonomous systems will have to deal with the physical environment and will interface with humans, which will produce unpredictable data to which the system will need to adapt). The outcome of a perturbation that is not handled can be a hard failure (crash or hang) or a soft failure (*i.e.*, missing a deadline for a latency-sensitive application).

We will first introduce as application drivers two autonomous application scenarios where perturbations need to be handled. We will then discuss the resilience by design aspect and then the resilience by reaction aspect. For each, we will lay out the vision for the end state in 10 years. Then we will talk of the short-term and mid-term research challenges, side-by-side with the promising approaches being investigated today.

Figure 1 shows a high-level schematic for the way we envision resilience in autonomous systems together with the universe of target perturbations.

3 Autonomous Systems as Application Drivers

We first look at the state of resilience in two exemplar classes of autonomous systems. By autonomous we do not mean completely autonomous, rather those at varying levels of autonomy which involve some human involvement. We speculate at a desired degree of resilience against perturbations and use these as examples as broad motivation for the solution directions that we lay out in the rest of the article.

The first class of autonomous systems is drones being used to deliver essential medical supplies in hard-to-reach areas. We include in our purview interactions among multiple drones and among drones and the non-(Computer Science)expert humans responsible for their resilient operation. The second class of autonomous systems is rescue and relief by say an international humanitarian agency in the face of a natural or a man-made disaster. This involves ground and aerial sensors, distributed inferencing from their inputs through processing at the edge as well as at the cloud.

Distributed Resilience in Multi-Agent Drones for Medical Deliveries

Problem and Current State The field of systems and control has recently been evolving from single monolithic system to teams of interconnected subsystems (or multi-agent networks). Because of the absence of centralized coordinator, algorithms for coordination in multi-agent networks (especially large-scale autonomous swarms) must be *distributed*, which achieve global objectives through only local coordination among nearby agents [17]. In order to guarantee all agents in the networks work as a cohesive whole, the concept of *consensus* naturally arises, which requires all agents in the network to reach an agreement regarding a certain quantity of interest [58], [77], [83]. A specific instantiation of this general idea of multi-agent systems is a **multiple drone system that is responsible for transporting essential supplies to a population affected by a natural disaster or medical supplies to a population where the surface transportation infrastructure is poor** [93], [29]. Some characteristics relevant to our discussion are that there are multiple cooperating agents involved, there is uncertainty in the physical as well as the cyber conditions (flying conditions may be variable and the network connectivity among multiple drones may be variable, as examples of the two kinds of uncertainty), and there is also human involvement, such as, to task the drones or to refill the supplies when the drone reaches a certain height above the ground at the home base.

Consensus is the basis of many distributed algorithms for computations [78], [120], [121], optimization [94], [22], [16], control [41], [36], [72] in multi-agent networks. The success of consensus-based algorithms relies on the assumption that all agents in the multi-agent swarm are cooperative, that is, each agent provides its own state value to its neighbor nodes and follows a common update protocol toward network objectives [83]. However, this particular autonomous system presents the salient challenge that it operates in an open and potentially adversarial environment, with exposure to a large and possibly unanticipated set of perturbations. On the one hand, distributed algorithms are inherently robust against individual node or link failures because of the absence of central controller; on the other hand, the strong dependence of distributed algorithms on local coordination raises a major concern of cyber attacks to the whole network through local attacks to one or more vulnerable agents [88]. Thus it is important to achieve *resilience* in order for autonomous multi-agent swarms to be used in critical applications like the current one.

There has been significant progress made in developing robust distributed algorithms by a combination of algorithmic and system-theoretic approaches in [15], [87], [116]. Further advancements will have to be made to handle hitherto unanticipated perturbations, to deal with the resource constraints of individual agents, to deal with time-varying characteristics such as link quality, and the possibility of multiple coordinated or uncoordinated perturbations. An entire new dimension arises due to the close interactions with humans—different patients may have different criticality requirements and these may change over short time periods, the level of cyber expertise of the human users both at the provider and at the consumer level will vary, and the time constants involved for some operations will be of human scale rather than cyber scale. Yet another dimension that needs significant research progress is scalability of these algorithms. While they have primarily been developed and evaluated under small world assumptions, they need to be re-designed or modified to operate at large scales, of the number of agents, of the distances (and latencies) involved and under the open-world assumption (new nodes can be added while the system is operational), etc.

Cooperative Autonomous Rescue with Active Adversary

Operationalizing artificial intelligence (AI) for military applications often brings to mind either offensive or defensive operations such as breaching defenses or defending assets by intercepting projectiles [111]. Upon closer consideration, many of the challenges faced when automating these operations, such as “dirty” data, i.e. data with low signal to noise ratios, and sparse data, i.e. small training data sets [53], are also faced when performing civilian rescue operations. These resiliency challenges will be illustrated below with a small military rescue vignette and correlated with current Army research efforts and gaps including sensor fusion, autonomous coordinated “swarms” [92], and resource constrained computing. [57]

Consider for a moment a complex future operating environment [82] where military operations take place in the dynamic cyber and physical environments of large urban areas. AI algorithms often require training from large amounts of data for maneuver to include a priori knowledge of infrastructure including roads, buildings, and subterranean passageways. During military operations urban environments can change rapidly as buildings are destroyed and barriers to movement are erected, leaving little existing

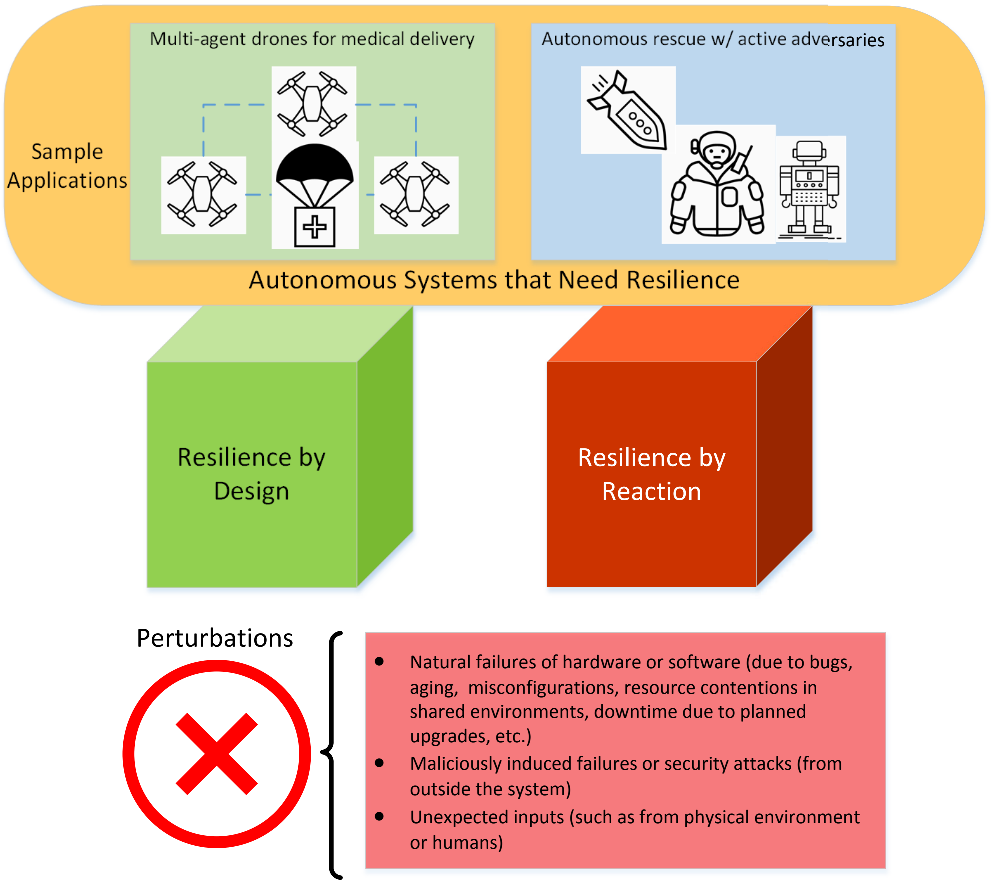
Multi-agent drones for medical delivery
Autonomous rescue w/ active adversaries
Sample Applications
Autonomous Systems that Need Resilience
Resilience by Design
Resilience by Reaction
Perturbations
Natural failures of hardware or software (due to bugs, aging, misconfigurations, resource contentions in shared environments, downtime due to planned upgrades, etc.)
Maliciously induced failures or security attacks (from outside the system)
Unexpected inputs (such as from physical environment or humans)

Fig. 1. High-level schematic of our vision for how resilience for autonomous systems can be structured. There are interplays between the two thrusts — resilience by design (offline techniques) and resilience by reaction (online techniques). We also show the target classes of perturbations that we consider in this article.

knowledge for aiding autonomous maneuver. In addition, instability of remaining structures is likely compromised so situational understanding (SU) must be gained on-the-fly, all while an adversary is actively employing anti-access and area denial (A2AD) capabilities. It is against this backdrop that future autonomous system will be called upon to locate, extract, and maneuver to safety either human teammates or other autonomous systems. Gaining SU in this environment will require fusing data from multiple sensing modalities over communications links that are unreliable at best, or denied at worst, requiring AI that achieves consensus on courses of action with incomplete information. Maneuvering through unstable structures requires understanding of the physical world including solid body mechanics, material strengths and failure modes, and physical systems. The rescue of people or equipment is an operation that embodies these challenges for AI and autonomous systems to perform.

There are early research efforts to begin addressing many of these challenging areas including developing new sensor modalities, sensor fusion, and robotic perception. In order to complete the task illustrated above, to perform a military rescue operation, in the complex dynamic urban environment will require significant effort in many areas including the integration of these efforts into complex systems of systems. For instance, open challenges exist when sensor fusion occurs with asynchronous data, unavailable data, or active deception. Open challenges also exist for autonomous maneuver in dynamic environments, e.g., off-road driving, and when interaction with the physical environment is required such as moving obstacles or sliding a chair to remove an obstacle. Finally, communication and coordination of cooperating sensors and autonomous systems when traditional modes of communication such as radio frequency (RF) links are unavailable or denied is a long standing challenge that is compounded for this scenario by the uncertainty of the physical environment. Advancements that address these challenges will support civilian rescue operations such as in natural disasters without endangering additional human lives of rescuers and the compression of timelines for rescue operations.

4 Resilience by Design

In this section, we discuss some design approaches to autonomous software systems that can make them resilient to the set of perturbations introduced earlier. For each we describe the problem context, some of the most promising techniques being researched today, and the desired end state and open challenges that we have to tackle to get there. This section includes discussion of attacks against building blocks of autonomous systems, resilient ML algorithms, immune-inspired resilient algorithms, program specification for resilience and keeping in mind human-in-the-loop.

We give in the side bar a distillation of the significant problems that we discuss in this section and the solution approaches. For the solution approaches, we categorize them as *Nascent* or *Developing*. The former means that there is little work so far but there is growing interest and some early promising results; the latter means there is a sizable body of work but it is growing and final answers are still in the future. By the nature of this article, none of the problems being discussed have fully mature solutions.

Problems:
1. Attacks against ML building blocks
2. Ease of generation and transferability of adversarial examples
3. Vulnerabilities of stochastic algorithms to corner cases
4. Synthesis of programs for resilience
5. Formal verification of correctness properties

Solution Approaches:
1. Whitebox testing and verification $\boxed{N}$
2. Optimization for resilience (in addition to accuracy) $\boxed{D}$
3. Evolutionary algorithms with resilience objectives $\boxed{N}$
4. Scalable verification techniques $\boxed{D}$
5. Verification with human-in-the-loop $\boxed{N}$

$\boxed{N}$: *Nascent*; $\boxed{D}$: *Developing*

4.1 Attacks against building blocks of autonomous systems

Problem Context.
Deep learning algorithms have been shown over the past decade to be very successful in various image and speech processing applications (see e.g., [47]), and more recently for wireless communication systems (see e.g., [123] and [129]). These success stories suggest the applicability of deep neural networks in a ubiquitous fashion in the near future. However, for this to happen, such algorithms have to be designed while taking into consideration potential exposure to adversarial attacks, especially with their recently discovered vulnerabilities (see e.g., [18]). Furthermore, these adversarial attacks are effective even when the attacker can only perturb the test data, and even there only a small part of each data point, as in evasion attacks (see e.g., [14]). One of the early and most efficient attacks that have been proposed in the literature is the Fast Gradient Sign attack (FGS) [48]. FGS highlights the vulnerability of neural networks, as the adversarial perturbation takes place in the direction of the gradient of the loss function:

$$\widetilde{x} = x + \eta * sign\left(\nabla_x J(w,x,y)\right). \tag{1}$$

In (1), x refers to the original input sample and $sign\left(\nabla_x J(w,x,y)\right)$ is the sign of the gradient of the employed cost function $J(w,x,y)$, which is a function of the input, x, the desired output, y, and the classifier weights, w. The parameter η is typically an l_p-bounded perturbation.

More stealthy attacks, that typically incur significantly higher computational cost, than the FGS attack have been proposed. Important examples are the evolutionary-algorithm-based attack introduced in [80], the feature-selection-based Jacobian Saliency Map attack introduced in [85], and the iterative Deep Fool and Carlini-Wagner (CW) attacks introduced in [76] and [32], respectively. In particular, iterative application of variants of the gradient sign concept were shown to present more effective attacks than the one step application of (1). This issue was closely analyzed in [61], where it was suggested that the one step FGS attacks leaks information about the true label. The classifier can then learn to perform very well on these adversarial examples by exploiting this leaked information. We are inspired by such analysis as it relies on characterizing the flow of label information to assess the effectiveness of an adversarial attack. More generally, we believe that a principled approach to quantifying the value of knowledge at both the attacker and defender, as well as characterizing the flow of important information regarding the output of the machine learning models and input sample distribution, is missing in the literature, and crucially needed to reach a deeper understanding of the resilience of the foundational blocks of autonomous systems.

Current State.
Recent work has also demonstrated the practical threat of adversarial examples in the context of real-world systems and constraints. For example, Papernot *et al.* [86] considered the scenario of a *black-box* threat model, in which the adversary does not have access to the details of the internal model structure or parameters, and also does not have access to the training data. Even in this challenging setting, the threat of adversarial examples persists. For example, it is possible for an adversary to locally train a model based on synthetically generated data, with associated labels obtained from interacting with the target model. The adversary can then use the locally trained model to generate adversarial examples, using standard techniques such as the FGS and CW attacks discussed above. This approach exploits the phenomenon of *transferability* of adversarial examples: with high likelihood, adversarial examples generated using the adversary's locally trained model successfully induce misclassification on the actual target model. Bhagoji *et al.* [13] explored an alternative attack approach for the black-box setting, which does not rely on transferability. Instead, the gradient term in (1) is replaced by an approximate estimate of the gradient which can be computed in a black-box manner by interacting with the target model.

Another thread of research has considered the question of designing physically-realizable adversarial examples for autonomous vehicles, whose effects persist in presence of real-world environmental constraints such as varying depth of perception, varying angle of perception, and varying brightness conditions. Standard attack techniques discussed previously produce adversarial examples that do not work well under such real-world conditions. Eykholt *et al.* [40] and Sitawarin *et al.* [105] modify the attack optimization problem to include such varying real-world conditions: their key insight is to incorporate a set of image transformations such as perspective transformations, image resizing, and brightness adjustment - as dataset augmentation techniques

- while designing adversarial examples and evaluating their efficacy.

Desired End State and Ways to Get There We note that machine learning driven autonomous systems need to be resilient not just to perturbed variants of the training or test data, but also to *unexpected* inputs, also known as *out-of-distribution examples*, that do not lie close to the training or test data. After all, applications such as autonomous vehicles operate in a dynamic environment and may naturally encounter objects that were not part of the training/test data. This observation has motivated a line of research on *open-world* machine learning, which augments conventional classifiers with another classifier for first deciding if the input is an in-distribution sample or an out-of-distribution sample [65], [19]. Approaches such as ODIN [65] rely on prediction confidence to make a determination about a sample being in-distribution or out-of-distribution. Detected out-of-distribution samples can simply be rejected. However, recent work by Sehwag *et al.* [99] has shown that existing open-world learning frameworks are not robust: an adversary can generate adversarial examples starting from out-of-distribution data to bypass detection. Here, instead of detecting whether the input sample is in-distribution or out-of-distribution, we detect whether each hidden layer, while processing the input sample, is in-distribution or out-of-distribution, as we have shown recently in [106].

Recently, there has been rapid progress in discovering effective defense strategies. However, most of these defense techniques are based on custom tailoring to the employed machine learning model, and the reported robustness would not hold, not only if the model's architecture change, but even if its parameters change due to new training data. For instance, the currently proposed approaches for defending against the FGS attack as well as iterative variants like the CW attack rely on simulating the attack. Furthermore, these defense strategies assume knowledge of the attack strategy while designing the defense, and provide little or no guarantees if the attacker decides to make, even very slight, changes to its strategy (e.g., choosing another model than the target model). Finally, while there is a vast literature on combating adversarial *intentional* machine learning attacks, there is little available work on designing the models to be resilient to unexpected inputs, that could result for example from unexpected behavior of other system components.

We believe that there are three key aspects regarding the challenge of designing resilient machine learning algorithms: **1: Perceptibility of the attack.** In the case of images, this is easily defined through human perception. For other applications, straightforward detection mechanisms should be used to judge how easy is it to detect the presence of an attack. **2: Value of knowledge in adversarial and uncertain environments.** For example, how to best exploit a finite-length secret key to protect a machine learning model. **3: Computational cost associated with attacking a target model.** Modern-day cryptography approaches rely on high computational cost associated with breaking their defense. Ideally, if the attacker has full knowledge of the system and the defense strategy, as well as an unlimited computational power, then there is no hope for resilience beyond the fundamental limits of the learning model; see e.g., [12] for the design of classifiers that consider such attacks.

4.2 Resilient ML algorithms

Current State Over the past few years, significant advances in ML have led to widespread adoption and deployment of ML in security- and safety-critical systems such as self-driving cars, malware detection, and gradually in industrial control systems. However, ML systems, despite their impressive capabilities, often demonstrate unexpected/incorrect behaviors in corner cases for several reasons such as biased training data, overfitting, and underfitting of the models. In safety- and security-critical settings, an attacker can exploit such incorrect behaviors to cause disastrous effects like a fatal collision of a self-driving car. Even without an attacker trying to cause harm, a Tesla car in autopilot recently crashed into a trailer because the autopilot system failed to recognize the trailer due to its "white color against a brightly lit sky."

Existing ML testing approaches rely mostly on manually labeled real-world test data or unguided ad-hoc simulation to detect such corner-case errors. But these approaches do not scale well for real-world ML systems and only cover a tiny fraction of all possible corner cases (e.g., all possible road conditions for a self-driving car). A promising and active line of research is to build a novel set of testing and verification tools for systematically finding such cases and ensuring security and safety of ML systems.

Our key insight is that most limitations of existing ML testing techniques result from their blackbox nature, i.e., they do not leverage an ML system's internal behaviors (e.g., outputs of intermediate layers) to guide the test input generation process. We have been developing whitebox testing and verification tools for performing static/dynamic/symbolic analysis of ML systems [114], [119], [118], [89]. These types of analyses have been used successfully for testing and verification of traditional software. However, the existing tools are not suitable for testing/verifying ML systems for the following reasons. First, traditional software logic is written by the developer while the logic of ML systems is inferred automatically from training data. Next, unlike most traditional software, ML systems tend to be highly non-linear. Finally, for some of these autonomous systems, it is a challenge to specify what is the expected behavior, considering the large number of possible interactions among the cyber, physical, and human elements. Therefore, support will be useful for specifying envelopes of desirable (and/or undesirable) behavior from these systems, which should then be decomposed into desirable or undesirable output states from each constitutent algorithm.

Over the last three years, we have been building new testing and verification tools to bring more rigor to DL engineering [114], [119], [118], [89], [64]. Given the challenges in specifying a full functional spec of DNNs, we design our tools to check transformation-invariant properties such as "slight light condition change must not change the image class." We have explored different design tradeoffs between scalability, completeness, and soundness. Our tools have found thousands of corner-case errors in different DL systems including state-of-the-art image classifiers, object detectors, malware detectors, self-driving car software, and cloud computer vision systems built by Google, Amazon, IBM, and Microsoft. We have also been able to verify some of these DL systems on popular datasets. Our testing and verification tools often outperform other existing tools by orders of magnitude (5,000x on average). We are encouraged to see that the concepts and algorithms in our tools have already started to gain adoption by other research groups and the industry [81], [128], [30].

Desired End State and Ways to Get There Despite the promising initial results are, there are many difficult open challenges that are not yet addressed. For example, existing DNN verification tools focus on verifying properties on a limited set of test samples with the hope that the guarantees achieved on individual samples generalize to unseen samples. One way to minimize such assumptions is to try to adapt existing specific testing and verification techniques (e.g., interval analysis, mixed-integer programming) to reason about distributions of inputs instead of individual inputs. Another interesting direction is to support a richer set of safety properties, different types of neural networks (e.g., RNNs), and different activation functions such as Sigmoid and tanh.

4.3 Immune-inspired resilient algorithms

Immune-inspired algorithms fall in one of the following three sub-fields: *clonal selection*, *negative selection*, and *immune network algorithms*. These techniques are commonly used for clustering, pattern recognition, classification, optimization, and other similar ML domains. They are relevant to design of resilient systems because they are (under domain-specified assumptions) able to adapt to uncertain system conditions or unexpected inputs.

Often, these immune-inspired algorithms start with processes reminiscent of natural selection deploying the following steps: **initialization**, **selection**, **genetic operations** (encompassing crossover and mutation), and **termination**, which can occur when the algorithm has reached a maximum runtime or a set performance threshold. Each of these steps mimics a particular phase in the natural selection process. Further, in the selection phase of immune-inspired algorithms, there is a metric such as fitness function that measures how viable the solution is. Relevant to our discussion, the fitness function should incorporate metrics for resilience rather than just raw performance, *e.g.*, how does the transformation make the system more immune to new kinds of attacks. Alternately, reinforcement learning (RL) can enable the selection of the best fitness function and also confer resilience to the algorithms. This is especially helpful in the case of evolutionary algorithms such as genetic algorithms being used to solve difficult optimization problems where possible drawbacks are the long time to convergence and the possible convergence to a *set of fitness functions* on the Pareto frontier, rather than to a single optimal point.

Evolutionary algorithms: Evolutionary algorithms are useful because they can be used to search for resilient operating points of computing systems in a manner that does not incorporate strong assumptions about the behavior of the underlying system. There are essentially three very similar evolutionary algorithms: genetic algorithms (GA), particle swarm optimization (PSO), and differential evolution (DE), with GA more suitable for discrete optimization, while PSO and DE being natural fits for continuous optimization processes. In general, for evolutionary algorithms, after initialization, the population is evaluated and stopping criteria are checked. If none are met, a new population is generated, and the process is repeated till the criteria are met. For increasing the resilience of these algorithms, diversity-aware variants of these algorithms have been proposed [44]. In most current applications, the optimization process occurs offline, however, some recent systems have evolved to combine offline training with further online adaptations, such as in our recent work on optimization of configuration parameters of database systems in the face of dynamic real-world workloads [69], [70]. Our systems combine offline training of the neural network with *online adaptation using time-efficient genetic algorithms* to search for discrete optimized state spaces. Such a design makes the system more agile to real-world variability, such as intercepting dynamic, fast changing workloads querying a database.

Desired End State and Way to Get There For resilient system design, we would want static training and configuration of the system to be complemented with dynamic learning, even in the face of sparse data. Drawing inspiration from the immune system, one can think of the following characteristics for the algorithms that may confer real-world resilience: *stochasticity* (increasing the exploration space); *reinforcement learning*, which can contribute to emergent behavior without global coordination; *stigmergy* where the "agents" interact with the environment. Overall, these characteristics result in an emergent and probabilistic behavior, with redundancy and adaptivity in real-world settings, which are ideal for resilience.

4.4 Program Synthesis for Resilience

A resilient software system needs to be able to rapidly adapt itself for perturbations. This is essentially a complex and intellectually demanding programming task. Program synthesis, the process of automatically generating programs that meet the user's intent, holds promise to automate this programming task and significantly increase the level of autonomy. The last decade has seen tremendous progress in the efforts of program synthesis techniques, including the synthesis of SQL queries [28], cache coherence protocols [115] and network configurations [112], [39], or productivity software like Microsoft Excel [49], [50].

A major challenge for program synthesis is how to obtain a precise specification that reflects the programmer's goals. Toward addressing this challenge, programming-by-examples (PBE) has been an appealing technique [101], [102]. A PBE system is given a set of input-output examples and tasked to find a program whose behavior matches the given examples through iterative interactions with the user. Another promising technique is sketch-based synthesis [108], in which the programmer specifies a synthesis problem as a sketch or template, which is a program that contains some unknowns to be solved for and some assertions to constrain the choice of unknowns. While the combination of PBE and sketch-based synthesis has seen many successful applications [103], [109], [104], [25], [59], these techniques do not immediately allow the programmer to describe the resilience aspects of the target program, which are usually quantitative and optimization-oriented. For example, a critical assumption of PBE is that the user knows what the expected output is, at least for some sample inputs. In the resilience context, however, the programmer usually cannot quantitatively determine how resilient a program is. Similarly, providing a sketch of the desired resilient program is also challenging because the programmer may not know how a resilient program looks like. Therefore, automatic programming for resilient systems requires novel, user-friendly modalities for describing resilience-related objectives.

While there has been substantial effort in formally verifying hardware and software, these efforts have largely focused on functional correctness: ensuring that the system has the functionality you want. The problem is that verification is only as good as your specification: if your specification leaves details out (e.g., some functionality is unspecified), then formal guarantees do not address that aspect of your system at all. More importantly, *even if all the functionality of your system* is captured, specifications often do not consider *non-functional* aspects: timing, energy usage, interaction models, etc. These gaps in the specification leave open vulnerabilities. For example, underspecifying the timing information of the system may leave timing side channels open, allowing attackers to glean information about a system or even perturb its behavior. A specification that does not account for the interaction model of a system may not properly account for the ways that a human interacts with a system (e.g., not considering certain classes of inputs because the specification designer does not account for humans providing perverse inputs). In the context of this under-specification, no amount of verification can help provide resilience.

Of course, as specifications get more complex to account for all of the possible vulnerabilities and interactions, the *scalability* of formal techniques comes under pressure, and formal verification becomes considerably harder. Indeed, this means that in the presence of difficult verification tasks such as modeling systems that involve human interaction, practitioners often use highly simplified models of the system under inspection: easing the verification task by reducing the fidelity of the verification. While this "solves" the scalability problem, it leaves open the question of *how* to do this model simplification: a model must still capture enough behavior of the system for the formal guarantees to be meaningful. Deciding how to model systems, therefore, becomes a serious bottleneck for verification, and there have correspondingly been only a handful of studies of formal verification for human-in-the-loop systems.

Nevertheless, we identify verification of human-in-the-loop systems as an important challenge for the design of future systems, especially as more and more systems represent a collaboration between automation and humans. Consider, for example, airline auto-pilot systems that expect certain input behavior from humans but fail when the system enters an unexpected regime that causes humans to apply unexpected control inputs.

Desired End State and Way to Get There

We propose a couple of promising directions to pursue in this space. First, can models be automatically developed for human-in-the-loop systems? Is it possible to automatically simplify a complex model (that might be automatically generated from observations of a system) to target particular desired properties such that humans can then intervene only to ensure that the model is faithful? As an example, consider observing the throttle inputs to a vehicle along with observing the motion of that vehicle to automatically infer a model relating the controls to the state of the vehicle. Second, can we tackle the scalability problem by *detecting simpler properties?* Rather than pursuing full formal specification and verification, it may be sufficient to make human-in-the-loop systems more robust by automatically identifying and flagging unexpected behavior (e.g., a conflict between the current state of an airplane and the types of control inputs being applied by a pilot), relying on the *presence* of a human in the loop to perform more fine-grained corrective action?

Constructing resilient systems with human in the loop further raises new challenges. The system designers should formally specify the *envelope* behaviors of humans and expected by humans. For instance, the design of a driver-assistance system may specify the maximum tolerable latency in recognizing objects to be 10 ms; an interactive image retrieval system may specify the minimum time that the user is given to annotate an image; an AI-based auto-completion code composer may specify the maximum human input rate is 50 program tokens per minute. With such explicit specifications of human behaviors, compiler and runtime may reason about system resilience by taking into human factors into account. For instance, they may assert if the human users are given enough think time to react to the detected anomaly, or if the human users are overwhelmed by the amount of training samples they have to annotate.

As the demands of resilience, and formal design principles for resilience, grow, we believe a key problem that should be tackled is scalability. How can larger systems be verified? How can more complex specifications be verified? Are there modular approaches to specification such that the verification task can be tuned to the particular property that we desire to address?

5 Resilience by Reaction

Here we talk about the online measures to deal with perturbations to ensure that as much of the system functionality as possible is maintained. We structure our discussion in terms of the execution platforms (mobile, edge, cloud, or HPC) and the algorithms that are executing. That is, we consider what changes can be made to either the execution platform or the algorithms to ensure that the cyber system continues to operate in a resilient manner despite the occurrence of perturbations at runtime.

Like in the design section, we give in the side bar a distillation of the significant problems that we discuss in this section and the high-level solution approaches being developed. For the solution approaches, we categorize them as *Nascent* or *Developing*.

Problems:
1. Live reconfiguration or adaptation of distributed applications
2. Detecting anomalies in streaming textual data
3. Approximate computing while meeting resilience guarantees
4. Handling stragglers in distributed computation
5. Robust policies for adapting autonomous systems

Solution Approaches:
1. Live reconfiguration of distributed applications $\boxed{N}$
2. Monitoring and optimizing network middle boxes $\boxed{D}$
3. Stream data mining with soft real-time guarantees $\boxed{N}$
4. Flexible approximation and execution on a variety of platforms $\boxed{D}$
5. Straggler mitigation techniques being data-centric, heterogeneous, and proactive $\boxed{D}$
6. Specification of high-level resilience goals for autonomous systems $\boxed{N}$
7. ML + Control Theory to achieve these goals $\boxed{N}$

$\boxed{N}$: *Nascent;* $\boxed{D}$: *Developing*

5.1 Tuning Configurations of Distributed Applications for Resilience

Given the rise in data being generated by different sectors, especially in IoT and automation, scalable data engines, low-latency in-memory engines (e.g., Redis), and stream analytics engines, are on the rise. In the context of scalable data processing engines, we can consider the changing, unpredictable workload patterns as perturbations to the system, in the face of which the system will need to react. Without such reaction, the rate of servicing the requests will drop, and in some pathological cases, requests for services will be silently dropped.

Most static database configuration tuners, such as [69], [117], [37], tend to be "reactive" because they use optimization techniques for changing the configuration parameters *when* there is a change in the application characteristic, e.g., change in read-write ratios for database workloads. However, for global-scale, multi-tenant execution pipelines and data repositories, such as the metagenomics repository MG-RAST [24], [122], the workloads may be more dynamic and unpredictable, making it harder to "react" to workload changes on the fly. The goal for the reconfiguration is to maximize the system's performance, using metrics such as the database's throughput and tail-latency. For such cases, recently, predictive configuration tuners that can work with dynamic workloads have been designed, such as the NoSQL database configuration tuner SOPHIA [70] and *Optimus Cloud* [68]. SOPHIA incorporates a workload predictor *and* a cost-benefit analyzer (CBA) in the optimization protocol that takes into account the cost of a reconfiguration (transient dip in throughput, possible transient unavailability of data) and the benefit (improved performance due to more optimized configuration). Subsequently, the system's configuration parameters are changed only when the CBA determines that the benefit outweighs the cost of reconfiguration. Then the system implements a graceful, decentralized scheme for the reconfiguration, so that data never becomes unavailable or is availability-aware, in sync with the organization's service-level agreement (SLA). In the context of analytics workloads running on cloud platforms, some current works, Selecta [60] and Cherrypick [3] can optimize the parameters of the cloud computing environment by predicting what will be optimal for the just-arrived application. Naturally, all this is predicated on accurate enough prediction of changing workload patterns.

End State and How to Get There

The continuing challenge remains to perform live upgrade of systems due to various events (changes in workload characteristics, data availability or consistency requirements, spatial migration of computing equipment), without degrading the data availability. An additional dimension to this problem is introduced by the use of cloud-hosted software, including some hybrid solutions, where part of the execution is on-premises and part on a remote cloud platform. The update could be to the configuration parameters, or more intrusively, to the software itself.

We can get there by learning from the long line of work on live upgrades of software systems, primarily focused on single-node software [110], [54]. We will have to bring in new distributed protocols that can progressively upgrade a distributed application, while keeping data continuously available and while respecting any end-user consistency requirement (SLA). It will be important to bring in a predictive component to such work, so that the reconfiguration can be initiated prior to the event, but in anticipation of it. Such proactive action can ensure high availability as well as decision as to whether the reconfiguration is beneficial at all. With respect to cloud deployments, cloud providers have mechanisms for data migration and VM migration but they have costs in terms of performance or simply \$ costs. Therefore, the prediction can determine whether the benefit normalized by the \$ cost or the transient performance impact is tolerable to the application.

5.2 Distributed Enclave Defense Using Configurable Edges

Problem and Current State In a geographically distributed system, the state of the network can change rapidly due to any of a number of factors. As mentioned in the introduction, perturbations can result from failures, overt attacks, or simply competition for resources. In most environments, there is no central arbiter with global knowledge of the state of the network. Instead, endpoints at the edges must infer network characteristics and adapt to changes to those characteristics. This adaptation can take many forms, including rerouting traffic, transforming content, and others.

There is a long history of adaptation on a case-by-case basis. For instance, Fox et al. described a mechanism for dynamic transcoding of web images into low-resolution forms [42]. Split TCP [8], [9] separates a TCP connection into one connection between a client and a proxy, and another connection between the proxy and a server; this allows the proxy to treat each part of the connection appropriately for its characteristics, such as high delay or loss. Middleboxes [100] are a generalization of this approach, interposing for various optimizations including Split TCP [63].

A holistic view that deals with dynamic changes to the network topology and workloads is more challenging. The DEDUCE[1] system from Perspecta Labs adds a "bump in the wire" between edges of a network and the WAN. The DEDUCE box is a middlebox that monitors system behavior and performs a number of optimizations. It splits connections for TCP and UDP , allowing it to transparently reroute via other DEDUCE edge nodes, change characteristics such as TCP congestion optimizations (e.g., from Cubic [52] to BBR [20], forward error correction, transcoding, and others. To do this, it needs a strong view of the state of the network [38], as well as the ability to evaluate ongoing network utility. It continually updates its plans [26] for how best to achieve its goals, which may be implicit (competing best-effort flows) or explicit (information about specific flows with deadlines).

Note that some aspects of its optimizations, such as rerouting, are similar to the functionality of the underlying IP network: IP can dynamically detect network outages and find new routes. The difference is that DEDUCE can apply a more holistic view, for instance rerouting one flow, consistently, along a particular path and a different flow along an alternate path. IP, by comparison, would intermix the packets along each path, leading to packet reordering and other performance implications.

End State and How to Get There DEDUCE is an example of a set of cooperating middleboxes that operate in isolation from the rest of the network. That is, each DEDUCE endpoint can work with other DEDUCE endpoints to optimize traffic, but any traffic to non-participating edge networks is untreated. For the Internet to be truly resilient, techniques such as these should be more broadly adopted. This means for example that any communicating parties would be able to change their TCP congestion treatment dynamically (e.g., learning the best algorithm for a given situation [107]), add or remove forward error correction or other content-level treatments [66], adapt content to reduce bandwidth requirements [21], etc.

The ability to adapt traffic is only half the solution, however. The bigger challenge is in deciding *how* to adapt, in the presence of incomplete, distributed information. Network tomography [38] is still in relatively early stages, but the ability to gather and process dynamically changing state is crucial to network resilience. A crucial aspect of this processing is being able to distinguish between failures due to resource contention and those due to hardware outages. For instance, if losses are due to congestion, adding extra redundancy via FEC simply contributes to the congestion; if losses are due to a faulty router, redundancy may provide appropriate resilience. Similarly, the reaction to a denial of service attack may be different from the reaction to normal high traffic.

1. DEDUCE stands for *Distributed Enclave Defense Using Configurable Edges*.

5.3 Detecting Anomalies in Real Time through Text Mining

In many scenarios, anomalies can also be uncovered through mining textual information (e.g., intruders may respond to system requests with naïve or threatening languages, a system may generate warning messages when detecting unusual situations, or people who observe something abnormal may signal alarms). It is thus critical to detect anomalies through text mining, in real-time. Preprocessing should be conducted beforehand by learning the text embedding space in typical situations with the distributions of phrases, topics, sentences, language features, and sentiments computed and stored, by using advanced text embedding methods developed recently, such as Word2Vec [73], Elmo [91], BERT [35], and JoSE [71].

End State and How to Get There. There is a need to develop stream data mining methods that can operate in real-time, e.g., they can calculate in an online manner, distribution of text elements and monitor the distribution closely. The language features (e.g., phrases, aspects, sentences, paragraphs, or topics) that substantially deviate from the typical ones can be considered as semantic outliers and should be detected and analyzed promptly by integration of text embedding and outlier detection analysis methods [130]. Alternatively, one may also use classification methods to train the system beforehand by collecting text messages in previously happened abnormal situations and go against the text happening in usual situations and such trained models can be used to signal anomalies on the fly. One can also integrate text classification and text outlier detection mechanisms to further enhance the quality of online anomaly detection.

5.4 Approximate Computation with Resilience Guarantees

Problem and Current State Many computations are inherently approximate—they trade off quality of results for lower execution time or lower energy. Approximate computing has recently emerged as an area that exposes additional sources of approximation at the computer system level, e.g., in programming languages, compilers, runtime systems, operating systems, and hardware architectures, thereby enabling us to re-define how we think about programs that implement novel solutions to an important class of problems. This has important implications for resilience because many demanding applications (such as, streaming video analytics)cannot run on resource-constrained devices (such as, IoT devices) because they exhaust the limited resources (memory, memory bandwidth, compute, IO, etc.). Therefore approximation has emerged as a potential technology, thus broadeing the domain of possible execution platforms for a wide variety of applications. One challenge of the area of approximate computing has been that the accuracy and performance of applying approximate system-level techniques to a specific application and input sets are hard to predict and control. Today this leads to too conservative choices for approximation [62], unacceptable quality outputs [97], [7], and even incorrect executions [96]. While the current approximate computing approaches show that the techniques have a lot of promise, making robust predictions about accuracy and performance is a key challenge to successful adoption of approximate computing in real-world applications.

The relevant current works in this space (approximation with resilience guarantees) answer the following three broad questions, in one or more of the domains of mobile applications, streaming video analytics, image processing, visualization of scientific computation, etc.

1) *When to approximate*. It searches for the period of the application's execution that is most productive to approximate. This is driven by early evidence that depending on when a specific technique is applied, there may be wide variations in time to convergence of the application or the quality of the output [75]. The granularity of the decision will be application specific and also subject to execution time constraints. Finer-grained monitoring and control are likely to lead to better performance-quality tradeoff, but with a law of diminishing returns. However, such monitoring and control come with their own overhead as well.
2) *How to approximate*. Any approximation technique typically accommodates one or more configuration settings, which captures how aggressively the approximation is done. For example, with a Neural Network-based video analytics query processing, we have to determine what is the optimal number of layers or the level of downsampling of the video frame. As another example, for a demanding computational genomics application, we have to decide how to segment the data and process in parallel for creating an SVM model in a distributed manner [46]. Consider that many applications in our target domains comprise pipeline of multiple software components or methods, each of which can benefit from one of several approximation techniques, and each technique comes with its configuration setting.
3) *How to approximate in input-aware manner*. It appears from some early evidence [125], [62] that in some important cases, the above two decisions have to be made in an input-aware manner. For example, if one is approximating a video stream and the stream consists of relatively static scenes, more aggressive approximation can be applied than if it is a sports scene with fast movements. Particularly, since we want to bound the accuracy loss with approximation, it is important to take the input dependence into account. For our target domains, the characteristic of the input may change within the stream, requiring that the *when* and *how* decisions be revisited.

End State and How to Get There There is the need to provide sound and practical techniques to approximate computation, under varied and unseen input data, while bounding the loss in accuracy and providing robust estimates for reduction in energy consumption and other resources. This will enable the grander vision off applications that can "flit" effortlessly between multiple execution platforms depending on the three axes of what is the capability of the platform, what is the resource demand of the computation (including approximation), and what is the cost of moving computation or data and orchestrating possibly distributed execution among the platforms.

There is the need to develop core algorithms to predict the impact of approximate computing on the accuracy and output quality. Further, we should create the models such that they take into account the input dataset and the state of the execution in deciding on the appropriate approximation configuration. The current approximate computing techniques are often inflexible and may miss profitable approximation opportunities or may mispredict the error rate. They are also fragile in the sense that their performance can fluctuate unacceptably under different input datasets. Fine-grained input-aware approximation algorithms that the community is developing has the potential to overcome these key challenges of approximate computing. Moreover, there is the need to show how to combine system-level and application-specific approximation techniques in the various target domains.

5.5 Resilience to Stochastic Task Sizes in Distributed Computation

Large scale computing jobs require multi-stage computation, where computation per stage is performed in parallel over a large number of servers. The execution time of a task on a machine has stochastic variations due to many contributing factors such as co-hosting, virtualization, hardware and network variations [27], [124]. A slow server can delay the onset of next stage computation, and we call it a *straggling* server. One of the key challenges in cloud computing is the problem of straggling servers, which can significantly increase the job completion time [45], [84], [51]. Resilience to straggling servers is essential to counter the possibility of missing deadlines in job execution. This is relevant in autonomous systems of the kinds introduced earlier because there are (soft) timing requirements and many workloads execute on cloud computing platforms.

Current State: Some of the key approaches to mitigate the effect of stragglers are to have speculative execution which acts after the tasks have already slowed down [34] or proactive approaches that launch redundant copies of a task in the hope that at least one of them will finish in a timely manner [4], [5]. When redundant tasks are launched on different servers, one approach is to perform an erasure-coding that provides significantly more flexibility as compared to replication. By having coding-theory based redundancy approaches, the user waits for any k out of the n servers to finish, while each server runs a smaller fragment of the task [2], [6]. Coding-theoretic techniques have been proposed to mitigate the effect of stragglers in gradient computation [113], [127], [98]. In [95], an approximate variant of the gradient coding problem is introduced, in which approximate gradient computation is done instead of the exact computation. A stochastic block code and an efficient decoding method for approximate gradient recovery are provided in [23].

Desired End State and Ways to Get There: Even though different approaches for straggler mitigation have been provided, efficient approaches require a holistic framework to understand the different design tradeoffs, including the completion time of the jobs, and the additional server costs spent for the jobs that will eventually not be completed. In order to come up with such a holistic framework, it would be essential to develop data-centric proactive approaches that leverage coding-theoretic and queuing-theoretic techniques. We note that deep reinforcement learning based approaches have been considered for scheduling jobs on the servers [90], [31], [10], [67], [1]. Reinforcement learning approaches with speculative execution to mitigate stragglers have been considered in [79]. However, such approaches do not consider multiple jobs, heterogeneous servers, coding-theoretic flexibilities, and approximate computing. Further, the allocation among different users must satisfy joint objectives, e.g., fairness, thereby needing decentralized, scalable solutions rather than centralized approaches [1]. Accounting for all these degrees of freedom significantly enlarges the design space and efficient approaches that explore the design space is an interesting problem.

5.6 Adaptability with Resilience Guarantees

Current State

Computing systems must function effectively in dynamic environments where application workloads, available resources, and user requirements can all fluctuate in unpredictable ways. To handle these dynamics, system developers create *mechanisms* that enable the system to detect changes and then react to those changes. Unfortunately, the *policies* that govern how these mechanisms are applied are often ad hoc and heuristic based. These heuristics are developed by experts and tend to work very well on the system for which they were designed, but they are not robust to changes.

As an example, Samsung's scheduler for the Galaxy S9 smartphone has many heuristics governing when to change clockspeed and when to migrate a process between its fast, high-power cores and its slower, energy-efficient cores. The S9+ was anticipated to be an upgrade with higher performance and longer battery life due to improved processor design. However, product reviewers found that in practice, the S9+ exhibited lower performance and shorter life as scheduling heuristics tuned for the S9 produced poor results on the S9+ [43]. The S9/S9+ is one example demonstrating how fragile heuristic-based resource management can be, on complex, modern processor designs, and it demonstrates the need for more principled resource management.

Desired End State

Our goal is the design and development of a set of robust policies that govern how autonomous systems should react to unforeseen circumstances. These policies should be based on well-founded principles and come with clearly stated assumptions about the conditions under which they would be expected to work and the mechanisms available for the policies to be enforced. Furthermore, we advocate for autonomous systems that do not just react, but react to accomplish some high-level, user-defined goal. For example, such goals might be meeting a certain latency constraint with minimal energy or finding the most accurate model on an energy budget.

Ways to Get There A first step to achieve the vision of adapting to high-level goals is to make those goals explicit in the program. In other words, there should be a (possibly domain-specific) language for describing a program's quantifiable behavior and the desired range of behavior that represents successful deployment. Furthermore, this language should also specify which behavior is a constraint—which must be respected for correct operation—and which is an objective—to be minimized or maximized subject to the constraints. In addition, this language should describe what system components can be changed to affect the goals. The idea of specifying high-level goals and mutable program components was a key part of the Self-aware computing project [56], language support for making this specification a first-class object appeared in the later Proteus project [11], and VStore [126], a video data store that respects encoding/decoding throughput constraint while maximizing storage efficiency.

Clearly defining goals is key to resilience by reaction, as it is only through the definition of goals that the system can observe they are not being met and react to restore correct operation. Of course, there is still the question of how to react, or how to map measurements of goals into appropriate settings for the mutable system components. We advocate a mixture of machine learning and control theory to achieve this mapping. Machine learning models are well-suited to capturing the complex tradeoff spaces that can arise in computing systems with competing goals and many mutable components. Control theory is well-suited to ensuring that constraints are met. Recent work demonstrates how the two can be combined to achieve the best of both approaches [74].

While recent research demonstrates that it is possible to build adaptive systems to meet goals, one major challenge is unaddressed: How can multiple, independent adaptive systems collaborate effectively, if developed independently by different stakeholders? For example, a mobile application might have goals in terms of responsiveness and image quality, while a mobile operating system might have goals in responsiveness and energy efficiency. If those systems are developed independently, they could easily make conflicting decisions, negating their potential benefits [55]. Thus, a common interface is likely necessary for specifying adaptive components and the goals they effect; a high-level negotiation mechanism are needed for coordinating their adaptations amongst different such components.

6 The Road Ahead

Here we look at the road ahead with a summary of the short-term and mid-term research and transition challenges.

1) *Creating resilient systems out of individually vulnerable components.* There will be increasing needs to build resilient systems out of components that are *not* individually resilient to the perturbations that the system will have to face. Potentially there will be a large number of such components composing the system. These components will be vulnerable due to innate design and implementation vulnerabilities, or due to unpredictable interactions with the external environment, either cyber or physical.
2) *Speeding up the cycle of design and generation of attacks and defenses against ML algorithms.* There will be a two-pronged need in this space of resilient ML—designing algorithms that are resilient by design to a well-quantified set of perturbations and speeding up

the discovery of vulnerabilities in realized implementations of the ML algorithms. The speeding up will imply a partially automated process for discovery and patching of vulnerabilities in the ML applications, as was envisioned by the DARPA Cyber Grand Challenge competition [33].

3) *Synthesis of resilient programs by automated means.* There is a growing body of work on automatic synthesis of programs from specifications. We have to consider that the synthesized program meets well-quantified resilience guarantees. The automatic synthesis can take the form of full program synthesis or, what is more likely, augmentation of an existing program for the purpose of increasing its resilience. In this approach, we can take inspiration from non-traditional sources, such as, immune systems in biological organisms.
4) *Automated configuration of increasingly complex systems, for performance as well as for resilience.* This includes efforts to automatically navigate the large space of configuration parameters and determine the close-to-optimal settings within a reasonable time bound, perhaps even online. While there is a growing body of work on automated configuration for performance, it will become important to perform such reconfiguration while meeting resilience goals, such as, server uptime and data availability.
5) *Use of compute power close to the client devices to increase the resilience of large-scale multi-tier systems.* This involves the use of edge computing resources for redundant execution, in addition to its traditional use for reducing latency of short-running queries. We will move to some autonomous systems whose algorithms can execute in parts in each of the three tiers of execution—client devices, edge computing devices, and cloud computing devices. The partitioning can happen flexibly, even at runtime, and can be done not just for performance but also for resilience. Different degrees of redundancy will be employed for different applications and at different tiers of the hierarchy.
6) *Expanding the scope of approximate computation and distributed computation to include resilience as a first-order principle.* Approximate computation will become an increasingly powerful means to execute demanding applications on resource-constrained devices, or where energy resource is at a premium. We need methods to approximate computation while still being able to provide resilience guarantees, likely probabilistic. The same applies to distributed computation, which will become increasingly relevant to scale up demanding ML applications. In such cases also, the loss in accuracy relative to the centralized computation must be bounded and quantified.

Acknowledgments

This work is supported in part through NSF Award for the Grand Challenges in Resilience Workshop at Purdue March 19–21, 2019 through award number CNS-1845192. DEDUCE was sponsored by DARPA under the EdgeCT program HR0011-15-C-0098. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

References

[1] Mridul Agarwal and Vaneet Aggarwal. A reinforcement learning based approach for joint multi-agent decision making. *arXiv preprint arXiv:1909.02940*, 2019.

[2] Mehmet Fatih Aktas, Pei Peng, and Emina Soljanin. Straggler mitigation by delayed relaunch of tasks. 45(2):224–231, 2018.

[3] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 469–482, 2017.

[4] Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, and Ion Stoica. Effective straggler mitigation: Attack of the clones. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 185–198, Lombard, IL, 2013. USENIX.

[5] Ajay Badita, Parimal Parag, and Vaneet Aggarwal. Optimal server selection for straggler mitigation. *arXiv preprint arXiv:1911.05918*, 2019.

[6] Ajay Badita, Parimal Parag, and Vaneet Aggarwal. Sequential addition of coded tasks for straggler mitigation. In *IEEE Infocom*, 2020.

[7] Woongki Baek and Trishul M Chilimbi. Green: a framework for supporting energy-conscious programming using controlled approximation. In *ACM Sigplan Notices*, volume 45, pages 198–209. ACM, 2010.

[8] Ajay Bakre and BR Badrinath. I-tcp: Indirect tcp for mobile hosts. In *Proceedings of 15th International Conference on Distributed Computing Systems*, pages 136–143. IEEE, 1995.

[9] Hari Balakrishnan, Venkata N Padmanabhan, Srinivasan Seshan, and Randy H Katz. A comparison of mechanisms for improving tcp performance over wireless links. *IEEE/ACM transactions on networking*, 5(6):756–769, 1997.

[10] Husamelddin AM Balla, Chen Guang Sheng, and Jing Weipeng. Reliability enhancement in cloud computing via optimized job scheduling implementing reinforcement learning algorithm and queuing theory. In *2018 1st International Conference on Data Intelligence and Security (ICDIS)*, pages 127–130. IEEE, 2018.

[11] S. Barati, F. A. Bartha, S. Biswas, R. Cartwright, A. Duracz, D. Fussell, H. Hoffmann, C. Imes, J. Miller, N. Mishra, Arvind, D. Nguyen, K. V. Palem, Y. Pei, K. Pingali, R. Sai, A. Wright, Y. Yang, and S. Zhang. Proteus: Language and runtime support for self-adaptive software development. *IEEE Software*, 36(2):73–82, March 2019.

[12] A. N. Bhagoji, D. Cullina, and P. Mittal. Lower bounds on adversarial robustness from optimal transport. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

[13] Arjun Nitin Bhagoji, Warren He, Bo Li, and Dawn Song. Practical black-box attacks on deep neural networks using efficient query mechanisms. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XII*, pages 158–174, 2018.

[14] B. Biggio, I. Corona, D. Maiorcal, B. Nelso, N. Srndic, P. Laskov, G. Giacinto, and F. Rolil. Evasion attacks against machine learning at test time. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 2013.

[15] Z. Bouzid, M. G. Potop-Butucaru, and S. Tixeuil. Optimal byzantine resilient convergence in uni-dimensional robot networks. *Theoretical Computer Science*, 411(34):3154–3168, 2010.

[16] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

[17] F. Bullo, J. Cortes, and S. Martinez. *Distributed Control of Robotic Networks*. Princeton University Press, 2009.

[18] Szegedy C., W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations (ICLR)*, 2014.

[19] Elisa Cabana, Rosa E. Lillo, and Henry Laniado. Multivariate outlier detection based on a robust mahalanobis distance with shrinkage estimators, 2019.

[20] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. Bbr: Congestion-based congestion control. *Commun. ACM*, 60(2):58–66, January 2017.

[21] Surendar Chandra, Carla Schlatter Ellis, and Amin Vahdat. Application-level differentiated multimedia web services using quality aware transcoding. *IEEE Journal on Selected Areas in Communications*, 18(12):2544–2565, 2000.

[22] T. Chang, A. Nedic, and A. Scaglione. Distributed constrained optimization by consensus-based primal-dual perturbation method. *IEEE Transactions on Automatic Control*, 59(6):1524–1538, 2014.

[23] Zachary Charles and Dimitris Papailiopoulos. Gradient coding via the stochastic block model. *arXiv preprint arXiv:1805.10378*, 2018.

[24] Somali Chaterji, Jinkyu Koo, Ninghui Li, Folker Meyer, Ananth Grama, and Saurabh Bagchi. Federation in genomics pipelines: techniques and challenges. *Briefings in bioinformatics*, 20(1):235–244, 2017.

[25] Swarat Chaudhuri, Martin Clochard, and Armando Solar-Lezama. Bridging boolean and quantitative synthesis using smoothed proof search. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '14, pages 207–220, New York, NY, USA, 2014. ACM.

[26] Jingkai Chen, Cheng Fang, Christian Muise, Howard Shrobe, Brian C Williams, and Peng Yu. Radmax: Risk and deadline aware planning for maximum utility. In *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[27] Dazhao Cheng, Jia Rao, Yanfei Guo, and Xiaobo Zhou. Improving mapreduce performance in heterogeneous environments with adaptive task tuning. pages 97–108, Bordeaux, France, 2014. ACM.

[28] Alvin Cheung, Armando Solar-Lezama, and Samuel Madden. Optimizing database-backed applications with query synthesis. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '13, pages 3–14, New York, NY, USA, 2013. ACM.

[29] CNBC. Zipline, which delivers lifesaving medical supplies by drone, now valued at $1.2 billion, May 2019.

[30] Jeremy M Cohen, Elan Rosenfeld, and J Zico Kolter. Certified adversarial robustness via randomized smoothing. *arXiv preprint arXiv:1902.02918*, 2019.

[31] Delong Cui, Zhiping Peng, Weiwei Lin, et al. A reinforcement learning-based mixed job scheduler scheme for grid or iaas cloud. *IEEE Transactions on Cloud Computing*, 2017.

[32] Wagner D. and Carlini N. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy*, 2017.

[33] DARPA. Cyber Grand Challenge (CGC), 2016.

[34] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[35] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proc. of 2019 Conf. of North American Chapter of Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019*, pages 4171–4186, 2019.

[36] F. Dorfler, M. Chertkov, and F. Bullo. Synchronization in complex oscillator networks and smart grids. *Proceedings of the National Academy of Sciences*, 110(6):2005–2010, 2013.

[37] Songyun Duan, Vamsidhar Thummala, and Shivnath Babu. Tuning database configuration parameters with ituned. *Proceedings of the VLDB Endowment*, 2(1):1246–1257, 2009.

[38] Nick Duffield. Simple network performance tomography. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pages 210–215. ACM, 2003.

[39] Ahmed El-Hassany, Petar Tsankov, Laurent Vanbever, and Martin Vechev. Netcomplete: Practical network-wide configuration synthesis with autocompletion. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 579–594, Renton, WA, 2018. USENIX Association.

[40] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning visual classification. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 1625–1634, 2018.

[41] J. A. Fax and R.M. Murray. Information flow and cooperative control of vehicle formations. *IEEE Transactions on Automatic Control*, 49(9):1465–1476, 2004.

[42] Armando Fox, Steven D Gribble, Eric A Brewer, and Elan Amir. Adapting to network and client variability via on-demand dynamic distillation. *ACM SIGOPS Operating Systems Review*, 30(5):160–170, 1996.

[43] Andrei Frumusanu. Improving the Exynos 9810 Galaxy S9: Part 2 - Catching Up with the Snapdragon, April 2018. Anandtech (www.anandtech.com).

[44] Thomas Gabor, Lenz Belzner, Thomy Phan, and Kyrill Schmid. Preparing for the unexpected: Diversity improves planning resilience in evolutionary algorithms. In *2018 IEEE International Conference on Autonomic Computing (ICAC)*, pages 131–140. IEEE, 2018.

[45] P. Garraghan, X. Ouyang, R. Yang, D. McKee, and J. Xu. Straggler root-cause and impact analysis for massive-scale virtualized cloud datacenters. *IEEE Transactions on Services Computing*, 12(1):91–104, Jan 2019.

[46] Asish Ghoshal, Ananth Grama, Saurabh Bagchi, and Somali Chaterji. An ensemble svm model for the accurate prediction of non-canonical microrna targets. In *Proceedings of the 6th ACM Conference on Bioinformatics, Computational Biology and Health Informatics (BCB)*, pages 403–412, 2015.

[47] I. J. Goodfellow, A. Courville, and Y. Bengio. *Deep Learning*. MIT Press, 2016.

[48] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[49] Sumit Gulwani. Automating string processing in spreadsheets using input-output examples. In *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '11, pages 317–330, New York, NY, USA, 2011. ACM.

[50] Sumit Gulwani, William R. Harris, and Rishabh Singh. Spreadsheet data manipulation using examples. *Commun. ACM*, 55(8):97–105, August 2012.

[51] Y. Guo, J. Rao, C. Jiang, and X. Zhou. Moving hadoop into the cloud with flexible slot management and speculative execution. 28(3):798–812, March 2017.

[52] Sangtae Ha, Injong Rhee, and Lisong Xu. Cubic: a new tcp-friendly high-speed tcp variant. *ACM SIGOPS operating systems review*, 42(5):64–74, 2008.

[53] Brian Henz and Tien Pham. Army Research Laboratory essential research area: AI and ML (Conference Presentation). In Thomas George, Achyut K. Dutta, and M. Saif Islam, editors, *Micro- and Nanotechnology Sensors, Systems, and Applications X*, volume 10639. International Society for Optics and Photonics, SPIE, 2018.

[54] Michael Hicks and Scott Nettles. Dynamic software updating. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 27(6):1049–1096, 2005.

[55] Henry Hoffmann. Jouleguard: Energy guarantees for approximate applications. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP '15, pages 198–214, New York, NY, USA, 2015. ACM.

[56] Henry Hoffmann, Jim Holt, George Kurian, Eric Lau, Martina Maggio, Jason E. Miller, Sabrina M. Neuman, Mahmut Sinangil, Yildiz Sinangil, Anant Agarwal, Anantha P. Chandrakasan, and Srinivas Devadas. Self-aware computing in the angstrom processor. In *Proceedings of the 49th Annual Design Automation Conference*, DAC '12, pages 259–264, New York, NY, USA, 2012. ACM.

[57] Mee Seong Im, Venkat R. Dasari, Lubjana Beshaj, and Dale Shires. Optimization problems with low swap tactical computing, 2019.

[58] A. Jadbabaie, J. Lin, and A. S. Morse. Coordination of groups of mobile autonomous agents using nearest neighbor rules. *IEEE Transactions on Automatic Control*, 48(6):988–1001, 2003.

[59] Jinseong Jeon, Xiaokang Qiu, Jonathan Fetter-Degges, Jeffrey S. Foster, and Armando Solar-Lezama. Synthesizing framework models for symbolic execution. In *ICSE'16*, pages 156–167. ACM, 2016.

[60] Ana Klimovic, Heiner Litz, and Christos Kozyrakis. Selecta: heterogeneous cloud storage configuration for data analytics. In *USENIX Annual Technical Conference (USENIX ATC)*, pages 759–773, 2018.

[61] A. Kurakin, I. J. Goodfellow, and S. Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.

[62] Michael A Laurenzano, Parker Hill, Mehrzad Samadi, Scott Mahlke, Jason Mars, and Lingjia Tang. Input responsiveness: using canary inputs to dynamically steer approximation. In *ACM SIGPLAN Notices*, volume 51, pages 161–176. ACM, 2016.

[63] Franck Le, Erich Nahum, Vasilis Pappas, Maroun Touma, and Dinesh Verma. Experiences deploying a transparent split tcp middlebox and the implications for nfv. In *Proceedings of the 2015 ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization*, pages 31–36. ACM, 2015.

[64] Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. Certified robustness to adversarial examples with differential privacy. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 656–672, 2019.

[65] Shiyu Liang, Yixuan Li, and R. Srikant. Enhancing the reliability of out-of-distribution image detection in neural networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.

[66] Benyuan Liu, Dennis L Goeckel, and Don Towsley. Tcp-cognizant adaptive forward error correction in wireless networks. In *Global Telecommunications Conference, 2002. GLOBECOM'02. IEEE*, volume 3, pages 2128–2132. IEEE, 2002.

[67] Jinwei Liu and Haiying Shen. Dependency-aware and resource-efficient scheduling for heterogeneous jobs in clouds. In *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 110–117. IEEE, 2016.

[68] Ashraf Mahgoub, Alexander Michaelson Medoff, Rakesh Kumar, Subrata Mitra, Ana Klimovic, Somali Chaterji, and Saurabh Bagchi. OPTIMUS CLOUD: Heterogeneous configuration optimization for distributed databases in the cloud. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 1–15, 2020.

[69] Ashraf Mahgoub, Paul Wood, Sachandhan Ganesh, Subrata Mitra, Wolfgang Gerlach, Travis Harrison, Folker Meyer, Ananth Grama, Saurabh Bagchi, and Somali Chaterji. Rafiki: A middleware for parameter tuning of nosql datastores for dynamic metagenomics workloads. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*, pages 28–40. ACM, 2017.

[70] Ashraf Mahgoub, Paul Wood, Alexander Medoff, Subrata Mitra, Folker Meyer, Somali Chaterji, and Saurabh Bagchi. SOPHIA: Online reconfiguration of clustered nosql databases for time-varying workloads. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 223–240, 2019.

[71] Yu Meng, Jiaxin Huang, Guangyuan Wang, Chao Zhang, Honglei Zhuang, Lance Kaplan, and Jiawei Han. Spherical text embedding. In *Advances in neural information processing systems (NeurIPS)*, 2019.

[72] M. Mesbahi and M. Egerstedt. *Graph Theoretic Methods in Multi-Agent Networks*. Princeton University Press, 2010.

[73] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 3111–3119, 2013.

[74] Nikita Mishra, Connor Imes, John D. Lafferty, and Henry Hoffmann. Caloree: Learning control for predictable latency and low energy. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '18, pages 184–198, New York, NY, USA, 2018. ACM.

[75] Subrata Mitra, Manish K Gupta, Sasa Misailovic, and Saurabh Bagchi. Phase-aware optimization in approximate computing. In *2017 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pages 185–196. IEEE, 2017.

[76] S. M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. Deepfool: A simple and accurate method to fool deep neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR),*, 2016.

[77] L. Moreau. Stability of multi-agent systems with time-dependent communication links. *IEEE Transactions on Automatic Control*, 50(2):169–182, 2005.

[78] S. Mou, J. Liu, and A. S. Morse. A distributed algorithm for solving a linear algebraic equation. *IEEE Transactions on Automatic Control*, 60(11):2863–2878, 2015.

[79] Nenavath Srinivas Naik, Atul Negi, and VN Sastry. Improving straggler task performance in a heterogeneous mapreduce framework using reinforcement learning. *International Journal of Big Data Intelligence*, 5(4):201–215, 2018.

[80] A. M. Nguyen, J. Yosinski, and J. Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[81] Augustus Odena and Ian Goodfellow. Tensorfuzz: Debugging neural networks with coverage-guided fuzzing. *arXiv preprint arXiv:1807.10875*, 2018.

[82] Department of the Army. The operational environment and the changing character of warfare. Technical Report TRADOC Pamphlet 525-92, Department of the Army, 10 2019.

[83] R. Olfati-Saber, J. L. Fax, and R. M. Murray. Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, 95:215–233, 2007.

[84] Xue Ouyang, Peter Garraghan, Changjian Wang, Paul Townend, and Jie Xu. An approach for modeling and ranking node-level stragglers in cloud datacenters. pages 673–680, San Francisco, USA, 2016.

[85] N. Papernot, P. Mcdaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami. The limitations of deep learning in adversarial settings. In *IEEE European Symposium on Security and Privacy*, 2016.

[86] Nicolas Papernot, Patrick D. McDaniel, Ian J. Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2017, Abu Dhabi, United Arab Emirates, April 2-6, 2017*, pages 506–519, 2017.

[87] F. Pasqualetti, A. Bicchi, and F. Bullo. Consensus computation in unreliable networks: a system theoretic approach. *IEEE Transactions on Automatic Control*, 57(1):90–104, 2012.

[88] F. Pasqualetti, F. Dorfler, and F. Bullo. Attack detection and identification in cyber physical systems. *IEEE Transactions on Automatic Control*, 58(11):2715–2719, 2013.

[89] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. DeepXplore: Automated whitebox testing of deep learning systems. In *Proceedings of the 26th ACM Symposium on Operating Systems Principles*, 2017.

[90] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, and Chuanxiong Guo. Optimus: an efficient dynamic resource scheduler for deep learning clusters. In *Proceedings of the Thirteenth EuroSys Conference*, page 3. ACM, 2018.

[91] Matthew E. Peters, Mark Neumann, Luke Zettlemoyer, and Wen-tau Yih. Dissecting contextual word embeddings: Architecture and representation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 1499–1509, 2018.

[92] B. Piekarski, B. Sadler, S. Young, W. Nothwang, and R. Rao. Research and vision for intelligent systems for 2025 and beyond, 2016.

[93] National Public Radio. Medical Cargo Could Be The Gateway For Routine Drone Deliveries, March 2018.

[94] S. Rahili and W. Ren. Distributed continuous-time convex optimization with time-varying cost functions. *IEEE Transactions on Automatic Control*, 62(4):1590–1605, April 2017.

[95] Netanel Raviv, Rashish Tandon, Alex Dimakis, and Itzhak Tamo. Gradient coding from cyclic mds codes and expander graphs. In *International Conference on Machine Learning*, pages 4302–4310, 2018.

[96] Michael Ringenburg, Adrian Sampson, Isaac Ackerman, Luis Ceze, and Dan Grossman. Monitoring and debugging the quality of results in approximate programs. In *ACM SIGPLAN Notices*, volume 50, pages 399–411. ACM, 2015.

[97] Mehrzad Samadi, Janghaeng Lee, D Anoushe Jamshidi, Amir Hormati, and Scott Mahlke. Sage: Self-tuning approximation for graphics engines. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 13–24. ACM, 2013.

[98] Shanuja Sasi, V Lalitha, Vaneet Aggarwal, and B Sundar Rajan. Straggler mitigation with tiered gradient codes. *arXiv preprint arXiv:1909.02516*, 2019.

[99] Vikash Sehwag, Arjun Bhagoji, Liwei Song, Chawin Sitawarin, Daniel Cullina, Mung Chiang, and Prateek Mittal. Analyzing

the robustness of open-world machine learning. In *12th ACM Workshop on Artificial Intelligence and Security*, 2019.

[100] Vyas Sekar, Sylvia Ratnasamy, Michael K Reiter, Norbert Egi, and Guangyu Shi. The middlebox manifesto: enabling innovation in middlebox deployment. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, page 21. AcM, 2011.

[101] Rishabh Singh. Blinkfill: Semi-supervised programming by example for syntactic string transformations. *Proceedings of the VLDB Endowment*, 9(10):816–827, 2016.

[102] Rishabh Singh and Sumit Gulwani. Predicting a correct program in programming by example. In *International Conference on Computer Aided Verification*, pages 398–414. Springer, 2015.

[103] Rishabh Singh, Sumit Gulwani, and Armando Solar-Lezama. Automated Feedback Generation for Introductory Programming Assignments. In *PLDI*, pages 15–26, 2013.

[104] Rishabh Singh and Armando Solar-Lezama. Synthesizing data structure manipulations from storyboards. In *ESEC/FSE'11*, pages 289–299. ACM, 2011.

[105] Chawin Sitawarin, Arjun Nitin Bhagoji, Arsalan Mosenia, Mung Chiang, and Prateek Mittal. DARTS: deceiving autonomous cars with toxic signs. *CoRR*, abs/1802.06430, 2018.

[106] Kirthi Shankar Sivamani, Rajeev Sahay, and Aly El Gamal. Non-intrusive detection of adversarial deep learning attacks via observer networks. *IEEE Letters of the Computer Society*, 2020.

[107] Anirudh Sivaraman, Keith Winstein, Pratiksha Thaker, and Hari Balakrishnan. An experimental study of the learnability of congestion control. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 479–490. ACM, 2014.

[108] Armando Solar-Lezama. Program sketching. *International Journal on Software Tools for Technology Transfer*, 15(5):475–495, Oct 2013.

[109] Armando Solar-Lezama, Liviu Tancau, Rastislav Bodik, Sanjit Seshia, and Vijay Saraswat. Combinatorial sketching for finite programs. In *ASPLOS'06*, pages 404–415. ACM, 2006.

[110] Craig AN Soules, Jonathan Appavoo, Kevin Hui, Robert W Wisniewski, Dilma Da Silva, Gregory R Ganger, Orran Krieger, Michael Stumm, Marc A Auslander, Michal Ostrowski, et al. System support for online reconfiguration. In *USENIX Annual Technical Conference, General Track*, pages 141–154, 2003.

[111] David K. Spencer, Stephen Duncan, and Adam Taliaferro. Operationalizing artificial intelligence for multi-domain operations: a first look. In Tien Pham, editor, *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, volume 11006, pages 1 – 10. International Society for Optics and Photonics, SPIE, 2019.

[112] Kausik Subramanian, Loris D'Antoni, and Aditya Akella. Genesis: Synthesizing forwarding tables in multi-tenant networks. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL 2017, pages 572–585, New York, NY, USA, 2017. ACM.

[113] Rashish Tandon, Qi Lei, Alexandros G Dimakis, and Nikos Karampatziakis. Gradient coding: Avoiding stragglers in distributed learning. In *International Conference on Machine Learning*, pages 3368–3376, 2017.

[114] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th international conference on software engineering*, pages 303–314. ACM, 2018.

[115] Abhishek Udupa, Arun Raghavan, Jyotirmoy V. Deshmukh, Sela Mador-Haim, Milo M.K. Martin, and Rajeev Alur. TRANSIT:

[119] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Formal security analysis of neural networks us-

Specifying Protocols with Concolic Snippets. In *PLDI*, pages 287–296, 2013.

[116] N. H. Vaidya, L. Tseng, and G. Liang. Iterative approximate byzantine consensus in arbitrary directed graphs. In *Proceedings of ACM Symposium on Principles of Distributed Computing*, pages 365–374, 2012.

[117] Dana Van Aken, Andrew Pavlo, Geoffrey J Gordon, and Bohan Zhang. Automatic database management system tuning through large-scale machine learning. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1009–1024. ACM, 2017.

[118] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Efficient formal safety analysis of neural networks. In *Advances in Neural Information Processing Systems*, pages 6367–6377, 2018.

ing symbolic intervals. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 1599–1614, 2018.

[120] X. Wang, S. Mou, and D. Sun. Improvement of a distributed algorithm for solving linear equations. *IEEE Transactions on Industrial Electronics*, 64(4):3113–3117, 2017.

[121] X. Wang, J. Zhou, S. Mou, and M. J. Corless. A distributed algorithm for least square solutions. *IEEE Transactions on Automatic Control*, 2019. DOI: 10.1109/TAC.2019.2894588.

[122] Andreas Wilke, Jared Bischof, Wolfgang Gerlach, Elizabeth Glass, Travis Harrison, Kevin P Keegan, Tobias Paczian, William L Trimble, Saurabh Bagchi, Ananth Grama, et al. The mg-rast metagenomics database and portal in 2015. *Nucleic acids research*, 44(D1):D590–D594, 2015.

[123] Liu X., Yang D., and A. El Gamal. Deep neural network architectures for modulation classification. In *Asilomar Conference on Signals, Systems, and Computers*, 2017.

[124] Yu Xiang, Tian Lan, Vaneet Aggarwal, and Yih-Farn R Chen. Joint latency and cost optimization for erasure-coded data center storage. *IEEE/ACM Transactions on Networking (TON)*, 24(4):2443–2457, 2016.

[125] Ran Xu, Jinkyu Koo, Rakesh Kumar, Peter Bai, Subrata Mitra, Sasa Misailovic, and Saurabh Bagchi. Videochef: efficient approximation for streaming video processing pipelines. In *USENIX Annual Technical Conference (USENIX ATC)*, pages 43–56, 2018.

[126] Tiantu Xu, Luis Materon Botelho, and Felix Xiaozhu Lin. Vstore: A data store for analytics on large videos. In *Proceedings of the Fourteenth EuroSys Conference 2019*, page 16. ACM, 2019.

[127] Min Ye and Emmanuel Abbe. Communication-computation efficient gradient coding. In *International Conference on Machine Learning*, pages 5606–5615, 2018.

[128] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. Deeproad: Gan-based metamorphic testing and input validation framework for autonomous driving systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pages 132–142. ACM, 2018.

[129] X. Zhang, Seyfi T., S. Ju, S. Ramjee, A. El Gamal, and Y. C. Eldar. Deep learning for interference identification: Band, training SNR, and sample selection. In *IEEE International Workshop on Signal Processing Advances in Wireless Communications*, 2019.

[130] Honglei Zhuang, Chi Wang, Fangbo Tao, Lance M. Kaplan, and Jiawei Han. Identifying semantically deviating outlier documents. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, pages 2748–2757, 2017.