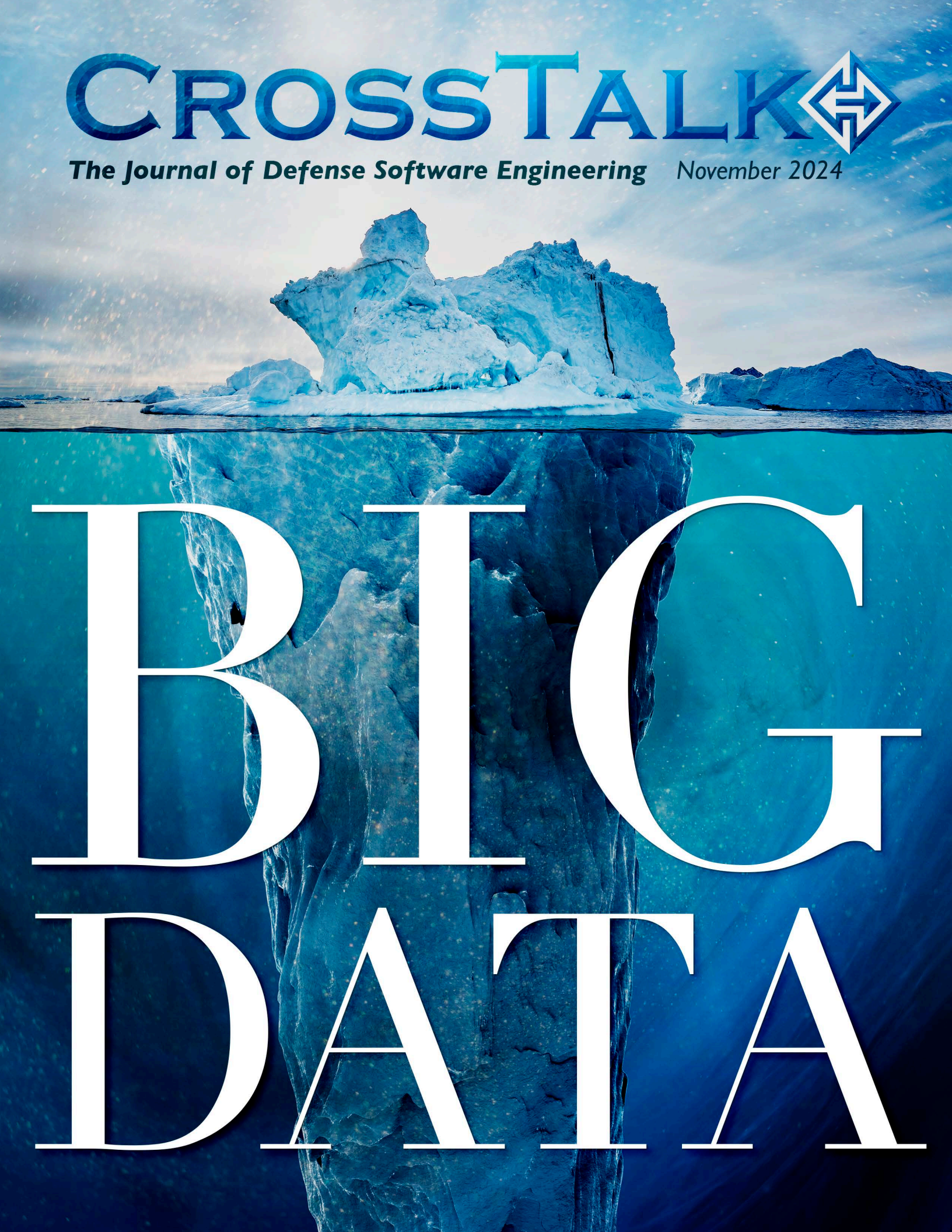
CrossTalk
The Journal of Defense Software Engineering November 2024



BIG DATA

CrossTalk: The Journal of Defense Software Engineering

CrossTalk: The Journal of Defense Software Engineering is sponsored by the Air Force Sustainment Center Software Directorate (AFSC/SW). It is also supported by other partners within the Department of Defense (DoD), other United States Air Force (USAF) systems, and the software engineering community. Established by Gen. Richardson, the Air Force Materiel Command Commander (AFMC/CC), the AFSC Software Directorate supports the AFMC Strategic Plan by facilitating and delivering an integrated software ecosystem across the Department of the Air Force (DAF).

AFSC/SW serves as the organic source of software engineering services for DAF weapon systems and equipment, and for all echelons of software development and sustainment, keeping pace with advancing technology, mission capability needs, and dynamics of the cyber environment.

The mission of CrossTalk is to encourage the engineering development and proper management of software to improve the reliability, sustainability, and responsiveness of our warfighting capability.

CrossTalk Online: Current and past issues are posted at the following locations: The Software Directorate website, All Partners Access Network (APAN), and Defense Technical Information Center (DTIC). The Software Directorate website houses the four most recent issues of Crosstalk while past issues can be found on APAN or DTIC.

https://afscsoftware.dso.mil/crosstalk/

https://community.apan.org/wg/crosstalk/

https://www.dodtechipedia.mil/dodwiki/x/HwDqFQ (Requires .mil domain for full support)

Subscriptions: Please send an email to the publisher to receive a notification when each new issue is published online. Readers can also sign up for notifications on APAN.

Article Submissions: We welcome articles of interest to the defense software community. Articles must be approved by the Technical Review Board (TRB) prior to publication. Please follow the Author Guidelines, available at any of the websites above. CrossTalk does not pay for submissions. Published articles remain the property of the authors and may be submitted to other publications. Security agency releases, clearances, and public affairs office approvals are the responsibility of the authors and their organizations. Potential articles can be emailed to: AFSC.SWSWDE.Crosstalk@us.af.mil

Reprints: Permission to reprint or post articles must be requested from the author or the copyright holder and coordinated with CrossTalk.

Trademarks and Endorsements: CrossTalk is an authorized publication for members of the DoD. Contents are not necessarily the official views of, or endorsed by, the U.S. government, the DoD, the sponsors, or co-sponsors. All product names referenced in this issue are trademarks of their respective companies.

Publishing Schedule and Back Issues: CrossTalk is currently being published quarterly. Please phone or email us to see if back issues are available, free of charge.

ISSN 2160-1577 (prior print versions); ISSN 2160-1593 (online)

CrossTalk Team
AFSC Software Directorate

Contact us

Phone
Lennis L. Burton, (801) 775-3262
Siria L. Snounou, (801) 777-4734
Destinie Comeau, (801) 775-3246

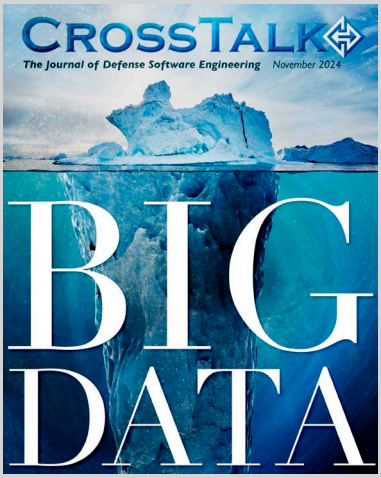
E-Mail
AFSC.SWSWDE.Crosstalk@us.af.mil

Connect with us on the AFSC Software Directorate LinkedIn page!

SOFTWARE DIRECTORATE

CrossTalk: The Journal of Defense Software Engineering

Directory



Cover Design by Kent Bingham

Big Data

The Data Atmosphere

Jay Crossler
Technical Fellow,
MITRE Corporation

As a new Air Force 2nd Lieutenant, data was like thin, high-altitude air—scarce and hard to grasp. We relied on basic tools like Visual Basic and Access, much like pilots depended on antique and limited instruments. Navigating through clear but empty skies, the lack of data made our missions uncertain.

As I became a Captain at the Pentagon working on critical programs, the metaphor shifted. Data was like scattered clouds—isolated pockets drifting without organization. These clouds held valuable insights but required immense effort to locate and interpret, often needing expensive database and analysts. To adapt, we had to climb to different altitudes, learn new skills like Python and web development, adjust our course, and venture into uncharted airspace to collect the data needed for true mission insight.

Today, data is as ubiquitous as the atmosphere itself, enveloping us like the dynamic clouds of a hurricane. It's not just about visible clouds but layers upon layers of swirling gases, storms, and jet streams moving in every direction. Modern analytic frameworks and Large Language Models (LLMs) are like advanced aviation systems that allow us to fly through these complex layers, effortlessly analyzing the data-rich air around us. Yet, amidst this abundance, the challenge lies in navigating ever-changing weather patterns to find those perfect storms that lead to valuable insights and mission success.

Synchronization: Aligning Our Flight Paths

In this roiling data atmosphere, data synchronization is like ensuring all aircraft in a formation operate with harmonized flight plans and real-time updates. Just as pilots rely on synchronized instruments to maintain formation integrity and prevent collisions, our Programs of Record find value in synchronizing data strategies, detailing what they will collect and store, and how it can work with others. Without

this alignment, we risk miscommunication, operational disarray, waste, or entering a dense mental fog where we believe something untrue. Synchronization promotes understanding—combining data from different models so our programs navigate with the same information, enabling coordinated and effective decisions.

Security: Shielding Against Data Turbulence

The skies have their perils. Turbulence, lightning storms, and unexpected weather fronts symbolize cyber threats that can disrupt missions. Pilots trust in the resilience of their aircraft and instruments to withstand these challenges. Similarly, we've learned to fortify our data systems against adversaries who seek to exploit vulnerabilities, much like hostile forces in the airspace. In this era of AI and state-sponsored cyber actors, we must protect access as well as ensuring data integrity to prevent poisoning our models or decision loops. Conversely, we must reduce the weight of our aircraft to be faster during storms and be smarter with how we use security approval checklists that take months—these can weigh us down while providing minimal improvement. Building a culture of security into data processes gives us the speed and agility to outrun coming storms.

Standardization: Speaking the Universal Language of Flight

Aviation relies on universal standards—common protocols for communication, navigation, and safety procedures. Without these, the skies would be chaotic and dangerous. In data, standardization allows systems and units to "speak" seamlessly—through Application Programming Interfaces (API), data dictionaries, machine-to-machine protocols, and unified data platforms. It ensures data from one source can be understood and utilized by others, much like pilots from allied countries coordinate through shared protocols. New AI tools help us find discrepancies or reduce costs for our data to fly in formation.

Charting the Course Ahead

Our journey through the evolving data skies is about mastering the complexities of this dynamic atmosphere. We must harness technologies to synchronize data, ensuring our operating picture is up-to-date and trustworthy. Robust cybersecurity measures are needed to protect against threats—both in access and in trusting our data and algorithms' integrity—ensuring data streams remain clear and uncompromised. We must commit to standardization, adopting common data languages and protocols that enable seamless collaboration.

In this issue of CrossTalk, authors discuss many themes in regards to data. Dan Ruef explains how users must be able to analyze and manipulate data before the true benefits of data can be realized. Alex Stringer, Geoffrey Dolinger, Asad Vakil, Joseph Karch, and Timothy Sharp examine the current state of data within the DoD and offers solutions to better use data in the future. Raouf Dridi and Steve Reinhardt detail the considerable strengths and weaknesses of Artificial Intelligence for defense applications, touching on data in the cybersecurity realm. Alejandro Gomez discusses engineering security into Application Programing Interfaces to ensure they work correctly and the data inside the pipeline

is protected.

The Publishers' Choice articles amplify topics previously touched upon in other issues of CrossTalk but elevate the status of data within them. Elbert Dockery elucidates on the gap between software and hardware, including the human element that keeps this gap wide. Linda Wright describes the importance of cyber resiliency in operating environments.

Embracing the Infinite Horizon

The data atmosphere is vast and ever-expanding, like the boundless skies pilots have explored for generations. Each layer offers new opportunities and challenges. Artificial intelligence and machine learning becomes our co-pilot, helping us process and interpret the overwhelming influx of information. They won't make all the decisions but will enhance our roles if we learn where and when to trust them.

But technology alone isn't enough. It's the human element—the skill, intuition, and adaptability of our service members and contractors—that enables us to navigate this complex environment. By fostering a culture that values data as a strategic asset and encourages continuous learning and a security mindset, we empower our people to find those perfect storms that lead to groundbreaking insights, decisive action, and true understanding—even during dark skies and dangerous storms.

- Jay A. Crossler, Technical Fellow MITRE Corporation, MITRE

Call For Articles

If your experience or research has produced information that could be useful to others, Crosstalk can get the word out. We are specifically looking for articles on software-related topics to supplement upcoming theme issues. Below is the submittal schedule for the areas of emphasis we are looking for.

Keeping up with the Cloud

February 2025 Issue

Submission Deadline:
November 30, 2024

Strategizing with Agility

May 2025 Issue

Submission Deadline:
March 15, 2025

Future Technology: The Long Haul

August 2025 Issue

Submission Deadline:
June 15, 2025

Please follow the Author Guidelines for Crosstalk, available at the APAN or DTIC site.

We accept article submissions on software-related topics at any time, along with Letters to the Editor, Open Forum, and BackTalk. To learn more about the types of articles we're looking for, please visit the above sites or contact us by email or phone

Contact Us

By phone

Lennis L. Burton, (801) 775-3262
Siria L. Snounou, (801) 777-4734
Destinie Comeau, (801) 775-3246

By email

AFSC.SWSWDE.Crosstalk@us.af.mil

Tailoring Cybersecurity Big Data Systems to Gain Effectiveness and Efficiency

DANIEL J. RUEF
TECHNICAL MANAGER,
CARNEGIE MELLON UNIVERSITY - SOFTWARE ENGINEERING INSTITUTE

Introduction

Higher volumes of data are now able to be efficiently transferred and stored from a wider range of sources and connected to more significant computing power than could be conceived of when the term “big data” was first coined in the 1990s [1]. The meaning of “big data” has evolved alongside these technological advances, but there are many interpretations of the term which can cloud discussions. In this article, “big data” refers to a large volume of data (such as billions of individual records or terabytes of storage required per day) and/or multiple data sources that must be stored adjacently for correlation, but which cannot be standardized into a single schematic record definition.

Achieving and maintaining situational awareness of network activity is an essential and enduring problem for organizations to detect, mitigate, and respond to threats. Big data is needed for cybersecurity due to the ever-expanding set of data sources populated from different vantage points describing activity on enterprise networks and the evolution of threats to networks, hosts, and software supply chains. Collecting and analyzing data for security monitoring purposes requires significant financial, technological, and personnel investments. Effective enterprise monitoring requires synergy between network and host operations, security operations, systems integrators, and data analysts. Detections and workflows must have targeted purposes as an overwhelming majority of events and records are from benign and expected behaviors. Systems need to be architected to satisfy a broad group of stakeholders with the end goal of detecting and mitigating threats.

Big data offers many benefits, but users must be able to analyze and manipulate it efficiently in order to realize those benefits.

Breaking the "Single Pane of Glass" Misnomer

The primary goal of any big data solution is to allow its users to efficiently and effectively gain insights from all the data that has been collected. As a way to simplify things for stakeholders of a big data repository, many program architects often speak of the desire for a "single pane of glass" which is the idea of providing analysts a single platform to get what they need. A single platform for analysts is an excellent design feature, however the term "single pane of glass" can also be interpreted as a desire for the exact same interface and mechanics for any data source, and everything in between. This is different than a single view made up of targeted query results from multiple data sources placed into a single dashboard. The ambiguity of the meaning of the term can yield misaligned engineering assumptions, which can lead to inefficiencies, unmet analyst needs, and restrictive requirements.

The first problem is that there is no single pane of glass that will allow users to query whatever they want across many big data sources and efficiently return meaningful insights. Second, the term is an oversimplification that leaves stakeholders unclear about how the system will work and what the requirements are. There is an important difference between shielding users from engineering complexities and hiding crucial details of the data or system that can be leveraged for increased efficiency. A window on a space station is a single pane of glass for viewing the cosmos, but without the ability to aim, zoom, or interpret different types of data, little is gained from the easily accessible view.

Capabilities can be built where analysts use the same interface and computing platform for all data sources. This can be considered a single pane of glass from the users' perspective. However, to efficiently extract information from a variety of sources, users must focus a second pane of glass to get what they need. The interface through which data is made available makes for a second single pane of glass from the data's perspective. Users will be more effective if they can aim and focus their view to gain insights from vast quantities of data, similar to a telescope.

Telescopes collect and return data in targeted and focused ways. Upon receipt, that data is able to be analyzed in full-featured processing environments that meet analysts' computational needs. This multi-stage process provides analysts what they need: efficient and targeted retrieval paired with the freedom and power to investigate. The cyber data storage and analysis industry is mature, with many excellent commercial and open-source tools capable of handling a variety of record and data types and formats. When paired and configured correctly, analysts can have the right mix of efficiency and flexibility to perform their workflows successfully.

When executing a workflow or a threat hunt, analysts usually know which data source they need. Part of the single interface presented to them must include a data source selection where backend tooling handles the details of accessing the chosen data source. The data source selection should also allow analysts to specify parameters so that they can take advantage of efficiencies available in a particular data source based on how it is stored. For example, users of big data storage and analysis tools from the SEI's NetSA Security Suite, know that queries return much more quickly when they specify a time/date range, the sensor(s) needed, and the directionality of the flow records they are interested in [2]. Interfaces which hide these crucial details in an attempt to make all data sources look the same actually limit analysts' productivity. As analysts are being asked to perform evermore complex analyses, they can also handle data source specific mechanics.

To simplify analyst workflows, data system architects should think of building telescopes directly into data sources, even multiple telescopes, all returning their results via a common interface for powerful and flexible processing.

Deliberate and Extensible Ingest Pipelines

When high volumes of data from many different data sources all flow to one place, deliberate and extensible ingest pipelines are vital to keep data organized and usable. These pipelines provide well-structured opportunities for filtering, extract transform load (ETL) processes, data extraction, streaming analysis, and functionality reuse. They also lower the amount of time required to add a new data source. Anything that is done to the data "on the way in" saves load on the repository when you don't have to do it later using batch queries. An ingest process that is designed and instrumented well facilitates faster awareness of data outages and provides analysts up-to-date information about what data is being collected and from where.

Having specified stages of an ingest pipeline with clearly defined responsibilities prevents each data flow from being customized and limited to what is currently needed. Building known checkpoints and mechanisms to pass data through them provides a structure to which functionality can be added as needs arise.

Powerful ingest pipelines facilitate:

- Filtering and routing of data
- Analyst-specific data feeds
- Data format conversions
- The building of workflow driven caches or quick lookups
- Collection of baseline information during ingest for up-to-date situational awareness without the use of periodic batch queries
- A mechanism for real-time alerting and utilization of security indicators
- A standard toolset for data reduction and manipulation
- A path for applicable data to reach visualization tools

Tailored Big Data Solutions

Big data capabilities, especially at government scale, can be held back by an aversion to customized data and internally developed solutions. Solutions need to be tailored for their purpose to provide analysts what they need with sufficient context. This may mean a customized ETL process that prepares the data for long-term retention, or a solution providing targeted data feeds to advanced analytic processes.

Analyst workflows and targeted mitigations must drive the tailoring of big data systems. If analytic processes typically begin with an indicator, then the system can build caches for quick lookups to quickly discern whether that indicator is present in the repository. This saves analysts time waiting for broad-scoped queries to return what has a high chance of being an empty result. A fast-returning targeted data store that quickly gives the analyst the specific location (usually timestamps and sensors) of the data they seek, if present, is immensely valuable and not overly difficult to build.

A tailored ETL process can also be used to reduce the data down to a level that is truly needed for long-term retention, or to change the data format (e.g. JSON, CSV, IPFIX) to one requiring less space. Data reduction will either save money or extend retention time. For instance, if there are network connections such as incomplete TCP handshakes where the only useful information for the long term is the external IP address, then saving that in a cache to maintain a record of its presence may

be all that is needed, allowing for the full network flow to be dropped and not stored. To understand use cases such as this one, it is essential to have discussions with all stakeholders about why they need particular classes of data, and what they will do with them. Sometimes the mentality of "store every byte of data forever" is correct, but it can also bloat big data solutions and perpetuate the aversion to customized and tailored capabilities. Exploring the uses of various data allows systems to intelligently store that which must be retained while reducing the noise and carrying costs of storing non-critical data.

Determining generalities from detailed use cases and requirements can yield benefits for design and implementation of a system architecture. An example of this could be that targeted detections to gain insights of the current state of the network never go further than six months back, and that any reference to older data is just to determine the presence of an indicator. This insight informs the decision to reformat and re-organize data older than six months into something more efficient both in storage size and query response time. Informed generalities may support formal decisions on the retention time of different datasets, which directly affects cost and analyst utility.

The mature tooling and platforms available from the cyber data field provide building blocks for big data systems that are very flexible and powerful. However, they cannot provide the optimal data configurations and customizations to provide analysts what they need most effectively. Enterprise customization of extensible data pipelines and fully utilized configuration options for the building blocks allow the system to remain agile and evolve with new usage patterns and workflow needs. Providing mechanisms to tailor data at different locations is akin to utilizing a distributed system of valves to deliver resources rather than one long static pipe.

Application and Infrastructure Logs Are Not Specifically Designed for Security Purposes and Long-Term Retention

Traditional host and network-based sensors export data designed for threat hunting. Application and infrastructure logs, however, are built for troubleshooting, testing, event logging, and proof of service. While they can certainly be used for threat hunting, incident response, and situational awareness, their original form is likely not ideal for long-term retention. As a result, storing these logs in their original form for long periods of time can lead to inefficiencies for both resources and analysts.

As an example, application or infrastructure logs may supply a field for an account ID. This is usually a long string of characters in every record. This is valuable when troubleshooting, but not when retroactively analyzing the data for threats, as it is likely to be the same for every record. A value for the sensor itself, or the organization, is likely more valuable and requires less storage space.

It's essential to understand the purpose of storing these types of information and what analysts will do with them. Some logs may be used as primary sources to identify potential threats and incidents that require further inspection. Others are more suited to providing organizational context to alerts or confirmation of an incident or threat.

After enumerating the utility of retaining and analyzing a particular data source, the next step is identifying the form in which these records should be stored. Some sources will need to have all their data

stored in their original form. Some may need to have indicators extracted or lookups and conversions identified. Others may just need to have daily summaries generated for reference later. Whatever the usage, a tailored storage solution facilitating ease of access by analysts is paramount.

Purpose-driven utilization of application and infrastructure logs coupled with a strong ingest pipeline can reduce the need to retain the entirety of application logs and can allow for:

- Streaming analysis to look for changes, additions, or anomalies
- Creation of caches of essential information for automated enrichment
- ETL for long term retention and/or assimilation with traditionally used data sources

Conceptual Baselining

The objective of baselining is to understand and define what is normal or expected about a dataset. Statistical baselining provides the opportunity for automated anomaly detection with degrees of confidence based on mathematical models.

However, the connection between statistical baselines and human-level events can sometimes be opaque. For example, there is a different level of understanding and situational awareness required to convert “the range of users updating infrastructure is 4-6” to “the only people updating infrastructure are on the Infrastructure Admin Team.”

The first phrase is simple, collected data; the second phrase is an example of conceptual baselining, which is an understanding of how a network, host, or infrastructure should behave, and how that behavior translates to the data collected. Data from a sensor or log is a translation of an event initiated by a human or a system into a structured record. In order to respond with a mitigation, the results of the analysis of those records must then be translated back to a human- or system-level event.

Taking the time to get a feel for the data and maintain conceptual baselines pays dividends during incidents. It’s easier to respond to a statistical anomaly such as a seventh and eighth user updating infrastructure when it’s known that the first check should be to the Infrastructure Admin team to see if they added employees. Policies are typically written at the human and system level. The translation from data into human and system events is required for verification and enforcement of those policies.

Providing data and insights in a way that can turn “normal” into something described in words has immense value and provides significant context for analysts. It also leads more directly to mitigations: the end goal of many threat hunts.

Enrichments

It is essential to provide data enrichments to alerts and analytic outputs that allow threat hunters to quickly diagnose whether or not an event is suspicious. Enrichments must be data source and content-specific to be used effectively. When analysts are provided the appropriate context around alerts, they can internalize and more easily discuss what may have occurred. Upgrading “these five IP addresses…” to “two IT servers and three laptops in the finance department in the Pittsburgh office…” humanizes the events for analysts.

Automated enrichments can be used to answer standard initial questions that stem from an alert. Question such as: Have we seen this entity before? Which part of my organization is affected? What information is known about the entities involved?

Type	Description	Typical Sources
Events	Events or information extracted from related data sources	Raw telemetry such as netflow, host data, firewall logs
Entity	Information about machines, hosts, or user accounts	Geolocation, ASN Information, pas-sive DNS, identity manage-ment, or-ganizational information
Historical	Related information to the event in question: its frequency, other events related to entities involved, other events relating all entities	Profiles or baselines of historical activity, automated queries of repositories
Threat	Reputation scoring, attack tech-nique identified, correlation with other events and alerts	Third-party threat feeds, indicator sharing organization feeds, inter-nal alert tables

Figure 1. *Suggested Types of Data Enrichments [3].*

The faster analysts can triage alerts and report, the sooner mitigations can be initiated. Enrichments save time by answering common analyst questions before they're asked.

Conclusion

Big data environments must adapt and grow with evolving data sources and analyst needs. System architects should embrace the complexity and variation to provide efficient and effective mechanisms for analysts to elicit insights from vast quantities of data. They should design "telescopes" with the dexterity and power to make a wide variety of data available to analysts in a consumable way in a single location. Deliberately staged ingest pipelines will facilitate agile data collection and provide a well-organized opportunity for the automation of currently manual workflows, while reducing the amount of repository data queries. Full-featured and extensible ingest pipelines enable the tailoring of data to be security relevant and structured for both long-term storage and for data fusion for greater situational awareness. The deeper the analysts' understanding of their data sources, the quicker they will triage alerts and identify potential threats. The more questions analysts can pose from their under-standing, the more opportunities there are for effective workflow automation, data enrichment, and decision making, leading to mitigations, which should be an essential goal of every threat hunt.

References

[1] Lohr, Steve. "The Origins of 'Big Data': An Etymological Detective Story." The New York Times, The New York Times, 1 Feb. 2013, archive.nytimes.com/bits.blogs.nytimes.com/2013/02/01/the-ori-

gins-of-big-data-an-etymological-detective-story/.

[2] NetSA Security Suite. Software Engineering Institute. 2024. tools.netsa.cert.org.

[3] Hutchison, Sean. “Dealing with Noisy Behavioral Analytics in Detection Engineering.” Software Engineering Institute. October 30, 2023, insights.sei.cmu.edu/blog/dealing-with-noisy-behavioral-analytics-in-detection-engineering/

Acknowledgements

The following markings MUST be included in work product when attached to this form and when it is published.

For purposes of double anonymous peer review, markings may be temporarily omitted to ensure anonymity of the author(s).

Carnegie Mellon University 2024

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License. Requests for permission for non-licensed uses should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

CERT® and Carnegie Mellon® are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

DM24-1178

About the Author

Dan Ruef is the Technical Manager of the NetSA team in the CERT Division of the Carnegie Mellon Software Engineering Institute. He leads the research and engineering efforts to build capabilities, highlighted by SiLK, that provide network situational awareness by sensing, storing, and analyzing traffic for threats both on-premises and in the cloud. Dan has over 15 years of professional experience in cybersecurity and software engineering.

Dan Ruef

Network Situational Awareness Technical Manager

Software Engineering Institute

druef@cert.org

Introduction

The 21st century is an age of highly interconnected systems and broad data availability. As improvements in technology have led to increased processing power and low-cost data storage, Big Data has quickly become a topic of discussion in the national consciousness. Policymakers have keyed-in on the potential of Big Data when developing their vision for future weapons capabilities. This has led to the rise of a certain philosophy: “Data is all you need.” With some justification, this mentality has subconsciously permeated throughout the Air Force (AF) and wider Department of Defense (DoD) unchallenged, as reflected by several Big Data-related, strategic policy documents [1] [2] [3] [4]. These documents have provided the groundwork for transforming the structures of the DoD into a data-centric organization, and the first step in this process has been addressing the challenges of building physical, tangible means of sharing data within the organization using Big Data. This necessity for rapid change has become more pressing with the recognition that the DoD must adapt to an environment of Great Power Competition.

Recent technological advances made by China and Russia have produced growing anxiety among strategic policymakers. In response, DoD and AF leadership has refocused on developing capabilities that more efficiently and effectively utilize and distribute data among decision-makers and warfighters. The DoD has acquired vast quantities of operational data throughout its history. The hope among policymakers is that this data will be a strategic asset to affect positive outcomes on the battlefield. For example, shortening the sensor-to-shooter kill chain, adding Artificial Intelligence and Machine Learning (AI/ML) to Command and Control (C2) systems, and building predictive maintenance and repair systems all require storing, processing, filtering, and training large datasets. These kinds of strategic capabilities are challenging to develop from a networking architecture, software, and hardware perspective, and leadership wants expedited development of these models. That pressure has produced a desire to use generalizable methods for approximating systems based on patterns in data rather than relying on evaluations or derivations from experts. There is considerable value in this type of approach, as it can minimize the time and costs required to develop new models. However, it is putting a strain on the DoD’s current data infrastructure and there are, increasingly, scenarios being encountered in which little to no high-quality, relevant data is available. In addition, it has contributed

to the false assumption that data availability is somehow more important than system design. Several recent articles have echoed this point, arguing that both joint all-domain kill chains [5] and collaborative combat aircraft [6] require data processing and data management in order to develop actual capabilities. These challenges produce significant risk in the age of Big Data, and the stakes for the U.S. and its allies are high. This article presents an overview of common challenges encountered in Big Data systems and methods for mitigating those challenges.

Data Requirements in Modern Systems

Before introducing Big Data's limitations and solutions, it is important to provide an overview of common data concepts and terms. Within the DoD, data is primarily used in the fields of modeling, analytics, and AI/ML, so the discussion here will center on the role of data in those domains.

Data Quantity

The core requirement for large data systems is the availability of relevant data. The quantity of this data plays a massive role in the performance of these systems, particularly in systems using Deep Learning (DL) methods (models that use artificial neural networks) and Large Language Models (LLMs). Data availability is commonly described by its volume and velocity. In this context, the term volume refers to the amount of existing usable data for a given application, while velocity refers to the rate at which new data can be generated [7]. Data volume is essential for the initial creation of models and for conducting large-scale analytics to identify patterns and develop ML training functions. Velocity plays a central role in updating and evaluating existing models, as well as in certain ML techniques that make use of real-time data during their learning process to better understand more interactive environments. While these two aspects of data quantity pose a problem for collection and storage, it is ultimately increasing the amount of quality data that makes identifying and learning the complex trends in the data possible.

Data Cleaning

Data analytics and ML systems are very sensitive to the quality of the data they utilize. The quality of real-world datasets can be reduced by factors like bias, missing or corrupted data, and poor scaling. These can cause incorrect patterns to be identified in the data, thereby interfering with model development and training. The term veracity is used to describe the quality of available data. Data cleaning is an essential aspect of improving the veracity of a dataset. It refers to the removal or correction of data that contains missing or corrupted values and ensuring all the training data is representative of the problem the model is attempting to solve. The amount of cleaning that is required depends on the application but is generally tied to the quality of the training data. This stands in contrast to the concept of "just collect all the data," and requires intelligent and purposeful collection of data. While increasing the quantity of the data provides a benefit, it can quickly cause the cleaning process to become unmanageable if the quality of the data is poor.

Generalizability

Recall that the goal of most Big Data applications is to develop approximate models for complex systems from patterns found in data. In essence, that process assumes that you can take a subset of example scenarios and create a model that works across a much larger range of scenarios. In the

fields of Big Data and ML, this concept is referred to as generalizability, which is used to describe how well a model works when it encounters scenarios that it has not directly seen before. The ability of a model to generalize well in a given application depends heavily on two key aspects of the data used to develop that model: value and variety. Here, value refers to how well a given set of data shows the trends underlying it, while variety refers to the diversity of relevant scenarios represented. The more key scenarios represented in a dataset, the easier it is for a model to identify the critical features in each of those scenarios, and the better a model trained on that data will be at generalizing.

Data Labeling

Arguably, one of the most challenging requirements for the use of large datasets in modern systems is the need to label the data. Many of the more popular uses of data, like ML, assume that large quantities of data can be used to generate an approximate model of a system. However, in order to develop that model, they must be trained. Models are trained by being provided example inputs, producing outputs, and then having their model parameters updated based on how good the outputs are. This necessitates the definition of the desired output, known as a label, for each input sample used for supervised learning. Labelling can range from dividing the training data up into different classes, such as organizing sensor data by the platform it was collected on, to adding additional information, such as exact location of a building on an aerial photograph. It is typically time-consuming and requires some knowledge of the target application space. Given the amount of stored data, labeling can be an intensive process but is ultimately critical in making data usable for DL systems.

Limitations: The Existence of High-Quality, Representative Data

In addition to volume, velocity, and variety, policymakers who are addressing the challenges of Big Data need to consider the quality and relevance of their datasets. AI/ML models excel at learning and identifying labeled patterns and features. However, when faced with unknown scenarios, their performance significantly decreases [8]. This characteristic is extremely relevant for military policymakers because, for many wartime applications, the scenarios encountered by the AI/ML system will be subject to influence by enemy actions. This could involve countermeasures, deception, or completely new and unknown scenarios. For example, if a model is trained to detect pattern of life activity in intelligence reports, but the intelligence reports are all biased towards reporting about enemy training exercises, then the model might be unable to accurately classify the pattern of life behavior during actual wartime operations. This issue was directly addressed by a 2024, five-volume study by the Rand Corporation [9][10]. The study, titled *Understanding the Limits of Artificial Intelligence for Warfighers*, dealt with cybersecurity, predictive maintenance, wargaming, and mission planning. The study's conclusions on two of the topics, cybersecurity and predictive maintenance, are particularly relevant to this issue.

In the cybersecurity domain, there has been a strong push to integrate AI/ML into defensive systems - Intrusion Detection Systems (IDS) - in order to prevent network attacks and malware. Integration would require models to be trained to detect cyberattacks, thereby necessitating large volumes of labeled network traffic training data to identify benign and malicious behavior. IDS containing AI algorithms can utilize statistical methods to categorize, cluster, and classify network traffic such that the algorithm can identify novel forms of malicious traffic. However, this ability is limited, and models must constantly be retrained to identify current threats, or otherwise fall victim to data drift [9]. Data drift occurs when the data a model encounters deviates from the data it was initially trained on, thereby

necessitating retraining.

The extent to which the training set differs from actual threats encountered in service is another major point of concern. The 2024 Rand study sought to directly investigate this matter to determine how much this data drift would degrade the performance of an AI system within an IDS. The study utilized two malicious network traffic datasets (created by the Canadian Institute for Cybersecurity) to train two separate neural networks to identify malicious network traffic. These neural networks were tested under two scenarios. The first scenario was a year-over-year comparison where each network was trained on 2017 data and then tested on 2018 data. The second scenario examined a day-over-day comparison, and it was designed to mimic the "historical training data" problem that IT professionals actually encounter. The network was trained on historical data from a varying number of days, and then it was tested against "live" traffic. For the first scenario, the models trained on 2017 data with an 80/20 training to test split were able to achieve 87% and 97% classification accuracy, respectively. When these models were tested on 2018 data, the accuracy for each model dropped to 73% and 72%. The second scenario saw decreasing performance as the number of days of historical training data increased, with approximately a 4% to 5% decrease in performance per additional day of historical training data [9][10]. These results show how training with historical data has limitations.

Conventional wisdom would assume that additional historical data would lead to improved performance, yet this was not the case. The report detailed that training on historical data caused the neural network to fixate on the outdated features of the data. As it was exposed to newer attacks, the misaligned training resulted in poorer performance. These results show that, as policymakers deploy limited resources, they must bear in mind that historical data, while potentially useful, requires additional system design considerations before naively being deployed to a training environment.

Predictive maintenance involves utilizing diagnostic tools that can pinpoint small reductions in efficiency that may suggest the need for maintenance. In the Air Force, spare parts are often packaged

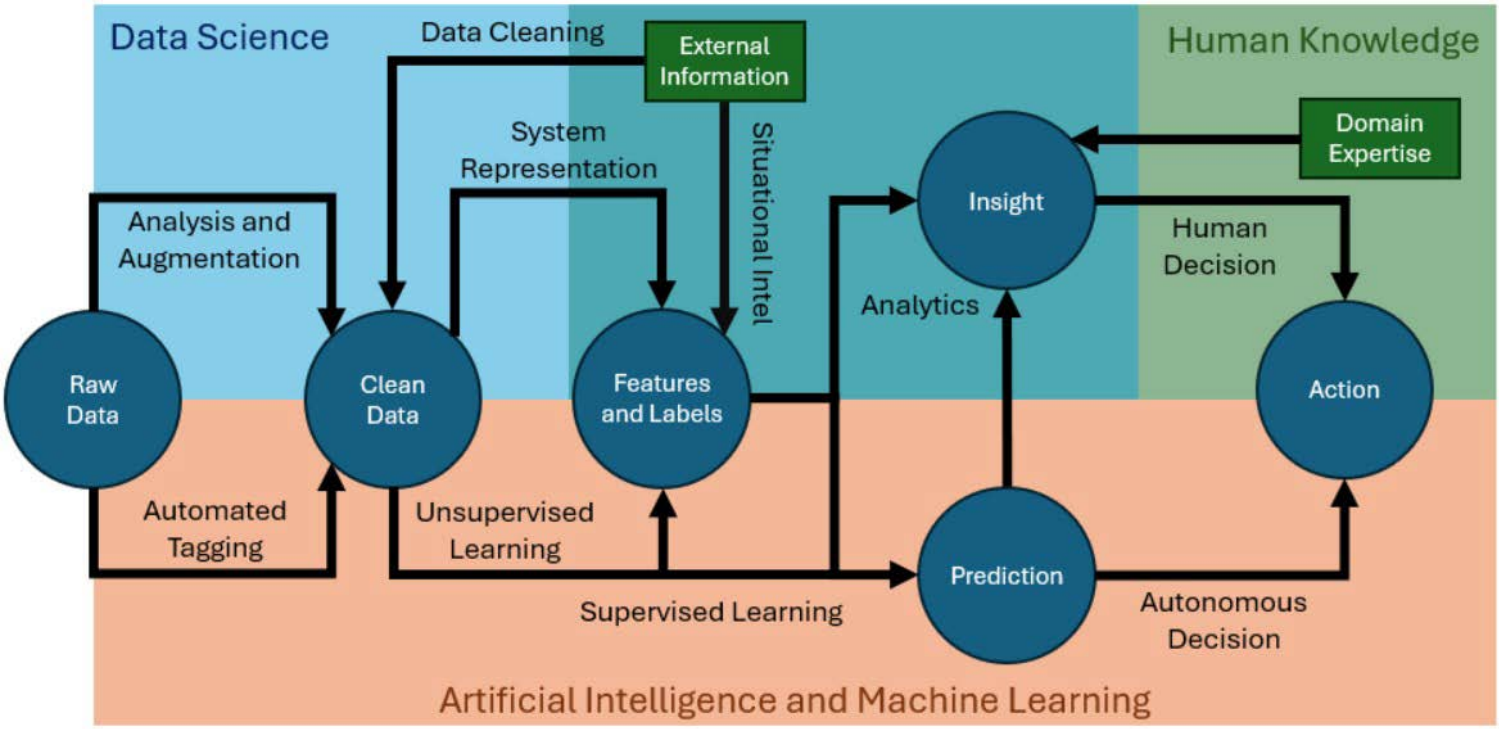


Figure 1. *A visual representation of the role of data science, human intervention, and AI/ML regarding Big Data.*

together into Readiness Spares Packages (RSPs) which contain enough spare parts for an aircraft to deploy for 30 days. The Rand Corporation study attempted to use A-10C aircraft data to test whether AI/ML methods, a Long Short-Term Memory (LSTM), would improve upon traditional, Poisson distribution-based estimation methods of part failure rates [10]. The study was able to access more than a decade of historical repair data and component/part failure rates for the A-10C. They were able to show that the LSTM did improve predictions on failure rates, but their final recommendations came with warnings.

First, the group noted that certain parts had failure rates which were insufficiently represented in the data, and the model was unable to make predictions as to the failure rate. Second, the report argued that the historical data alone might not be sufficient to provide the basis for estimating wartime part failure rates. The 2024 report referenced a 1995 study by investigating Air Force fighter repair and logistics rates during Desert Storm. This study found that logistical planning for spare parts, electronic warfare pods and munitions were frequently wrong when compared to peacetime planning estimates, and the authors emphasize that often the estimates were substantially wrong [11].

The demand for parts and munitions were specific to the theatre's conditions. Aircraft part break rates were linked to how aircraft were used in the different phases of the conflict and the different mission demands. This analysis reemphasizes the concern that an AI/ML model trained on peacetime data will be insufficient to predict the unknown scenarios that will occur should the U.S. Air Force enter a conflict with a near-peer adversary who has the capability to degrade communications systems and physical, logistics systems (like runways and warehouses) with cyber-attacks and long-range cruise and ballistic missiles [12] [13]. Additionally, aircraft will be flown differently during wartime. Longer sorties and flight hours that stress avionic systems and incur battle damage will change the demand for replacement parts [14]. To quote directly from one of the 2024 report's key findings:

> "AI cannot alleviate the scarcity of wartime data. It is unclear whether RSPs developed using peacetime data will be adequate for wartime operations. Moreover, one of the main limitations of AI for this application is its inability to estimate truly rare events, which might be more likely during wartime operations. As a result, different approaches to modeling AI could be required to deal with these changing circumstances. However, regular retraining and updating, which is possible with an AI model, can ensure the adaptability of these models during wartime" [10].

The DoD faces many challenges in making effective use of its data over the coming years. There is a lack of available data for many applications, and much of the data that is available is not relevant or currently in a usable state. Major investment will be required to correct for these issues. There are also challenges associated with creating infrastructure for collecting and automatically cleaning and labeling data. Handling large quantities of data also puts a strain on computing systems and networks. Techniques that reduce data requirements and improve the overall value of available data will play an important role in addressing these challenges. **Figure 1** provides a visual flow of these challenges and the coordinated roles that data science, human expertise, and AI/ML must balance to utilize Big Data for effective capability.

In this paper, we present three commonly used approaches to addressing inherent risks in the philosophy of "data is all you need." The first will address methods for using and preprocessing real data to improve model performance. The second will address the general topic of data synthesis. The last will address using smart system design as a method to offset or reduce data requirements. Individually, these approaches serve as effective methods to improve model performance, but they are not mutually exclusive and can be used together for compounding benefits in many situations. While DL models are capable of learning a lot without data augmentation, synthetic data, or smart design, the

application of one or more of these methods can drastically improve model efficiency and reduce the amount of necessary training data [15]. Determining which of these method(s), if any, is optimal to an application is dependent on the application's data and objectives. Applications in which the domain the data is from is well understood, such as wireless communication, smart design, and preprocessing, may be more desirable[16][17]. In situations where data is either scarce or sensitive, synthetic data may be the better approach [18][19].

Solutions

1: Data Augmentation and Preparation Techniques

Data Augmentation

One of the primary ways that large data requirements for DL can be mitigated is through data augmentation. This term refers broadly to methods for artificially generating data from an existing dataset and is relatively straightforward. First, assume that a model must be developed such that it learns to accurately model patterns in data and generalizes well. Then, given some quantity of data that is assumed to contain relevant patterns the system needs to learn, identify modifications that can be made to that data to produce a larger dataset.

For a computer vision application, consider a dataset of images. If the image set is small, the model may not learn the desired input-output relationship or worse, overfit to the training data. A simple augmentation method for images might take the form of duplicating and then randomly rotating each image in the dataset. This results in an artificially expanded dataset, with the new entries still containing relevant features and patterns but with different spatial configurations.

Data augmentation plays a critical role in improving the performance and robustness of DL models. It can take a variety of different forms depending on the type of data being augmented and the target application. Selecting the right method is a critical part of the data preparation process and should be considered early in development. There are several categories into which data augmentation methods can fall: data alteration, interpolation/extrapolation, feature manipulation, and model-based.

Data alteration involves taking existing data and generating new entries by adding noise, transforming data entries, or eliminating portions of them. These methods can help to increase the volume of data available and to make models more robust during training. However, it does little to improve the diversity of a dataset, can reduce the overall fidelity of a model trained on it, and doesn't improve the variety of the training data set.

Interpolation and extrapolation methods, commonly used on sequential or time-series data, involve approximating new data entries by fitting a mathematical function to the entries. That function can then be used to pad entries within a sequence (in the case of interpolation) or add entries to the beginning/end of the sequence (in the case of extrapolation). Like data alteration, these augmentation methods can increase the volume of available data but don't add significant variety.

Feature manipulation methods focus on altering feature level values during the model training process, rather than altering the input data itself. This requires direct integration into the model, systematically injecting noise or making alterations to feature vectors. These methods can be applied regardless of data type or application space but are especially beneficial when each data entry is large (as is the case with images or videos). This is because the dimensionality of a feature vector is typically small, making feature manipulation more efficient in these cases than data entry manipulation. Con-

versely, feature sets learned from large datasets are typically not as well understood, so there is inherent risk to modifying them as a means to augmenting the training process, potentially reducing model fidelity.

Model-based data augmentation is an emergent form of augmentation that has garnered significant attention from the research community over the last decade. It uses generative DL models (such as Variational Autoencoders or Generative Adversarial Networks) to create entirely synthetic data entries based on the features contained within an existing dataset. Generative models are typically designed with two primary components: the encoder and decoder. The encoder learns important patterns in data and maps them to a latent feature space. The decoder then samples a feature vector from that space and uses it to generate an output. These models have been adapted for data augmentation by training the decoder to produce outputs of the same form as the desired data entries. The assumption being that the decoder can identify the main dataset features and can use the encoder's latent vectors to produce realistic generated data as its output.

Data Preparation

Some of the more costly tasks required to start working with large datasets are data labeling and data cleaning. Both labeling and cleaning can be completed by a human, but this becomes expensive and time consuming as the size of the data grows. It is increasingly important to improve the efficiency of the labeling and cleaning process as the volume increases, especially if the velocity of new data is high. If all of the collected data is required for training, there exists a tipping point where the time it takes to clean and label the data is slower than the rate at which new data is generated. In these cases, it is important to have more efficient cleaning and labeling techniques. Automated techniques (like classical heuristics, expertly designed algorithms, ML models, and customized reward functions) can often speed up the process of creating labeled, high quality data sets.

Some data cleaning can be automated straightforwardly by just setting up heuristic requirements for the quality of the data. Any data that does not pass this quality heuristic can then either be removed or tagged to be fixed by another method. This reduces some of the burden of sifting through the data, but still requires intervention in the event the corrupted data needs to be repaired. Depending on the domain of the data, existing ML models may be available that can help correct corrupted data. There are inherent issues with using ML to correct data (that you are using to train other ML models), but, if done carefully, it's possible to clean the data quickly and efficiently.

Automatic Data labeling is more complicated than cleaning and fundamentally can't be fully automated by DL. However, semi-supervised and unsupervised learning methods can be used to rapidly automate most of the labeling. ML can also be used in tandem with human input to actively learn whenever the labels it generates are ambiguous. If a dataset's domain is well understood, an expert may be able to define algorithms that automate portions of the labeling. Algorithmic labeling will work to label large parts of the data, but if this algorithm were perfect, there obviously wouldn't be a need for training a ML model. Both the ML and traditional algorithmic approaches can be used to reduce the amount of data that needs to be labeled but can't completely provide the labeled data that supervised learning needs [20][21].

Machine learning contains different methods of training models. Supervised learning requires labeled data and achieves high accuracy rates after training on this data, but there are other methods which can forgo the need of labeled data. Methods such as unsupervised learning and Reinforcement Learning (RL) can be trained without the need for labeled data. Unsupervised techniques can do this automatically with some success but have limited applicability in real systems. Reinforcement learning, on the other hand, has shown considerable promise but requires additional steps to be effective. RL is a machine learning domain that has models learn based off the impact of their actions on a rep-

resentational or live environment. As a result, RL does not use data directly but relies on either analysis of data and/or insights due to other data science and ML methods applied to data (see **Figure 1**). This environment is dynamic and changes as a result of actions taken by a model.

The second vital component for RL based systems is a viable reward function. The reward function needs to be purposefully designed, as this is what will guide the learning algorithm to find the model that best fits the challenge-based measures of success. Once a good reward function is derived and a model has learned robust and viable behavior from the representative environment, then the model can be trained continuously while interacting with the real world [22][23]. This is a massive benefit to the usage of Big Data, as some of this data gathered can be used instantly. Additionally, RL methods can be used with human input to be fine-tuned with domain expertise.

2: The Role of Analysis and Synthesis in Reducing Data Dependency

Sometimes existing data is not enough, even after attempting to augment it. Perhaps owing to the data having too much noise or being inconsistently varied such that the ML model has difficulties in learning the trends or maybe there just isn't enough data to sufficiently train a model. In these cases, more effort must be placed on understanding the environment the training data was gathered from. This additional study of the problem space leads to a better understanding of the environment containing the problem, which can be used to conduct a first principle approach to training ML systems. Deeper understanding of the environment the data is from can be combined with physical and mathematical rules to create fully synthetic data, allowing the ML model to learn on perfect, noiseless data. After training on synthetic data, the models can then be tested on a smaller set of real data, verifying the model's performance.

Analysis

The first step in generating synthetic data is to understand where and how the data was generated. This usually requires an expert in the field to perform some analysis of the problem space. As most problems are grounded in reality, there are usually some basic principles upon which the complex trends in the data are based, even if the data is highly complex. There are aspects of the analysis process that must be done carefully, such as identifying and removing biases. Also, sufficient analysis must be done to reach a basic enough understanding to pull out the first-principles rules of the data. If these first principles can be derived through extended analyses of existing data, they can be used to generate clean and versatile data.

Data Synthesis

Once these first principles are derived, they can be used to synthesize data. This process, depicted in **Figure 2**, involves the generation of data from a simulated environment designed using a combination of statistical analysis of real data and expert knowledge of the application domain. When simulating these first principles, it is important to focus on the different distributions or categories of data that are important to the model. It is imperative to make sure that each scenario that the model should cover is included in the simulation, otherwise the training data will be blind to those categories. This process is also used for the representative environment required for successful RL model development.

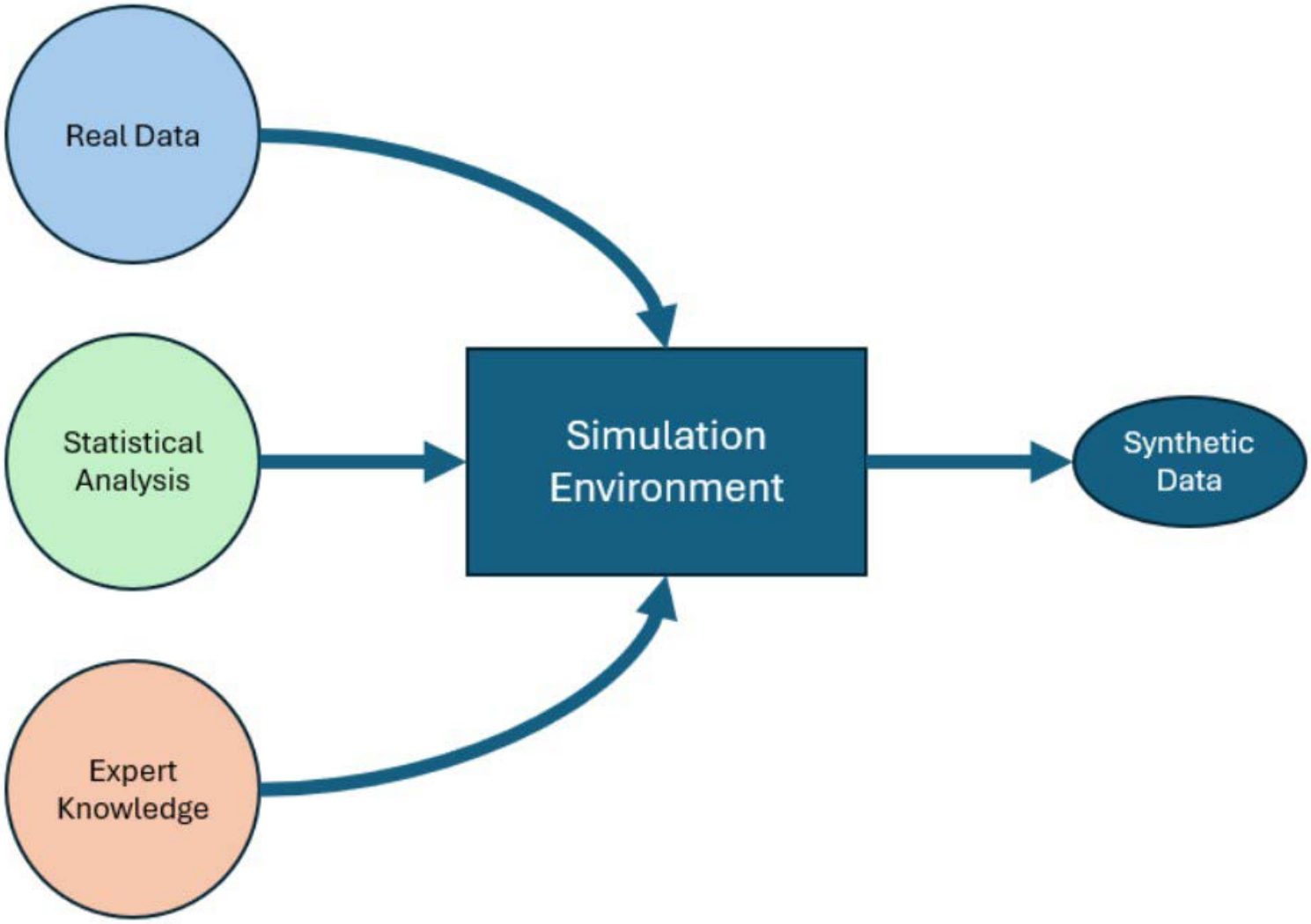


Figure 2. *A simple representation of data synthesis.*

There are, of course, risks and challenges in utilizing synthetic datasets for AI training. These can include inheriting data distribution biases, the manifestation of errors owing to inaccurate base data, insufficient noise levels, over-smoothing, neglecting temporal and dynamic aspects, and inconsistency. With that in mind, a key advantage of using synthetic data is that it also allows for the control of bias. Bias occurs when there is some unintended pattern in the data due to collection methods or analysis. Real data isn't necessarily immune to bias, as seen in instances where AI inherits racial or gender biases from its training data [24][25]. The higher complexity and more numerous variables of real data can make noticing such biases difficult. While synthetic data is still susceptible to bias, if the expert generating the data understands the relevant type of bias, it is trivial to control and remove bias when setting up the data synthesizing process. Additionally, synthetic data also allows the generation of noiseless data. Even when ML training on noisy data is necessary, it is simple to add specific noise to the training data during generation. All of these methods add flexibility to synthesized data. But data generation helps alleviate another problem: data drift. Since data drift is a change in the input or output data over time, it is infeasible for collected data to catch these trends. Not only can synthesized data preemptively ready models for this data drift, they can also be used to quickly retrain models as soon as any data drift is detected in the real data. While all these methods require an expert to perform in-depth analysis, the benefits in model accuracy and resilience are well worth the effort.

Testing

Once synthetic data is used to train a model, it is important to ensure the performance of the model is not degraded when applied to real data. Ideally, the synthetic data is very similar to the real data that it is based upon. However, this process is to ensure that the trained model is robust and generalizable to any real data it encounters. The process of using synthetic data starts with analyzing existing real data, synthesizing high quality data to train on, and then testing on the original real data. This comes with the benefit of requiring less real data (for testing) than what would be needed for training using real data. It also means that when modeling rare events, such as in predictive maintenance, the model can massively benefit by reducing required data while still confirming that the training data applies to the real data. This holds true for training data that covers a wide variety of categories, but other techniques can aid in the data requirement in that case.

3: The Impact of System Design on Data Requirements

Another method in reducing data requirements is smart design. In general, training data will contain certain trends which are used by a model to make predictions. The more complicated the trends, the more difficult it is for the model to learn. As a consequence, this leads to larger models which naturally require an increased amount of data. This has resulted in impressive accomplishments, however it has also increased data and processing requirements, as well as made models more difficult to understand or troubleshoot. In most cases, complex problems can be broken down to a sequence of smaller problems that build upon each other. In these situations, using effective system design to break the problem down into smaller parts can improve performance while reducing model complexity. This allows the expert to use smaller models which come with several benefits, including significantly reduced data requirements.

Preprocessing

The first system design method that can help to reduce data requirements during training is preprocessing. Many applications of DL integrate traditional algorithms for preprocessing. Using these algorithms, it is possible to preprocess the training data further before feeding it into the DL algorithm. A preprocessing step allows the model to skip over some initial computation, allowing the model to be both smaller and train quickly on less data. Not only does this make the process more efficient, but it can also allow deep learning to be applied to areas where it could not before. For example, Stringer, et al. demonstrated a targeted preprocessing of radar data that made the critical features for identifying interference more consistently observable, allowing the application of a simple DL model to the traditionally challenging field of radar detection. This preprocessing also helped to allow for the use of smaller, more specialized models [26].

Using smaller models provides many benefits, starting with a more manageable training process. While smaller models fail in complicated problem spaces, they require less time to learn simple trends, thereby inherently requiring less data, reducing the overall cost of building and training the model. When choosing the model size, another consideration is the interpretability of smaller models. Interpretability signifies how easy it is for a human to understand what and why a decision is made. Having interpretable models is important to build user trust and diagnose problems when a system does not perform as intended [27][28]. Interpretability is important to have for these reasons, but it also brings about benefits in data usage. Having well-defined, understandable models to process parts of the training data allows the expert to choose a subset of the training data that specifically targets that model. This means that, on top of needing less data, it is easier to see what data is needed for the training process. All of the benefits of small models culminate in the ability to build cohesive structures with multiple models working together to solve the larger overarching problem.

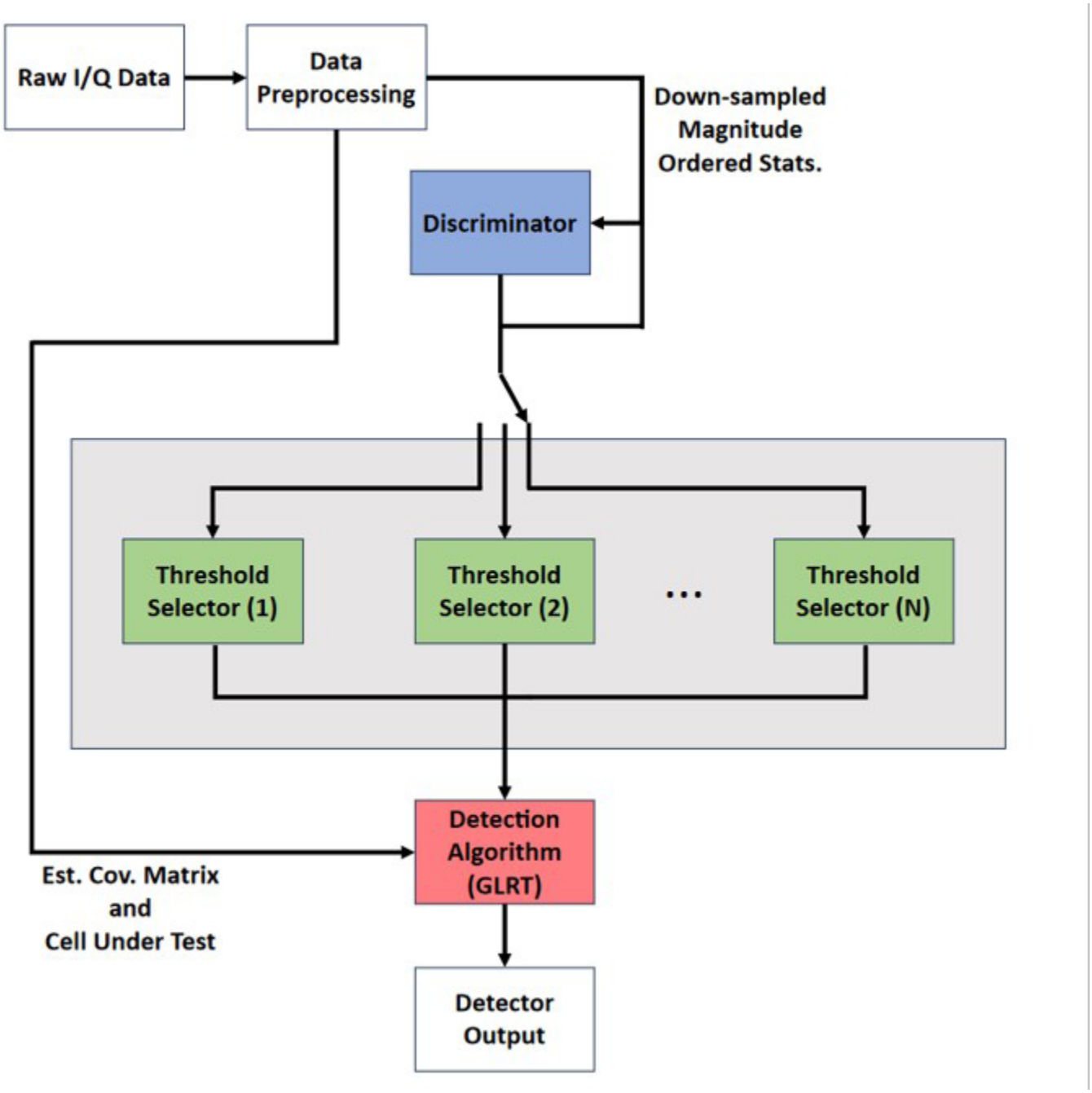


Figure 3. *Shows components of the metacognative radar detector. The Blue and Green boxes are ML models. This architecture benefits from data pre-processing and the separation of tasks to expert subsystems.*

Integrating multiple ML models into systems is another design method that can be effectively utilized to mitigate data risks. This approach requires more initial expert design and analysis than training monolithic models. However, it also provides the reduction in data requirements gained from using small models while also scaling well to larger problem spaces. Like preprocessing, it is possible to distribute some of the calculations done on the training data, and then recombine it. This grants the benefits of preprocessing the data beforehand and combining the inherent explainability of multiple smaller and simpler ML models. **Figure 3** shows an example of an AI application to radar detection as described in Stringer, et al [26]. In this structure, data preprocessing, post-processing, and ML models are used. The structure allows for the system to split the data to be processed by specialized expert models. These experts are trained on specific subsets of the data, resulting in less data that

is needed to prepare the system. Further, using structured systems like this increases robustness by allowing parts of the system to be switched out without impacting the rest of the system. Additionally, when encountering operational obstacles such as data drift, an expert can analyze which components the data drift effects and replace those parts. Because only small parts of the system are updated, this can be done quicker and with less effort than replacing the whole system.

Work Smarter, Not Harder, With Data

Big Data is at the center of the DoD's strategy in strengthening our deterrence against near peer adversaries. It enables technologies that can improve battlefield efficiency, reduce decision-making timelines, and automate a variety of systems. However, the availability of relevant data (especially modern wartime data [6]), processing requirements, and costs required for working with large datasets also create major vulnerabilities. To mitigate these vulnerabilities, it is essential that the DoD identify methods of reducing data requirements and more effectively utilizing the data it has in a sustainable manner [29]. The role of Big Data will continue to grow in the coming years, but a greater emphasis on the augmentation, labeling, and quality of the data will be necessary to create robust and generalizable models as well as maintaining a technological advantage in the field. Indiscriminate use of real data in a DL model is not a strategy that is appropriate for the Air Force, as it inherently lacks means of combating data drift and bias. For this reason, the adoption of smarter data practices, the incorporation of domain knowledge, model transparency, as well as an understanding of DL models is necessary to the development of reliable and robust AI tools. Implementing these practices is necessary for addressing and adapting to the ever-changing threats that the Air Force and these AI tools will face.

References

[1] D. of Defense, "Digital modernization strategy 2019," https://media.defense.gov/2019/jul/12/2002156622/-1/-1/1/dod-digital-modernization-strategy-2019.pdf

[2] "Dod data strategy," https://media.defense.gov/2020/Oct/08/2002514180/-1/-1/0/DOD-DATA-STRATEGY.PDF

[3] "Creating data advantage," https://media.defense.gov/2021/May/10/2002638551/-1/-1/0/DEPUTY-SECRETARY-OF-DEFENSE-MEMORANDUM.PDF

[4] "2023 data, analytics, and artificial intelligence adoption strategy," https://media.defense.gov/2023/Nov/02/2003333300/-1/-1/1/DOD_DATA_ANALYTICS_AI_ADOPTION_STRATEGY.PDF

[5] M. Dunn. "Goldilocks kill chains and the just right data," Military Review, vol. May-June, no. 1, pp. 1–9, 2024.

[6] J. Breau, K. Erhardt, and J. Reddis. "Collaborative combat aircraft need data to train for combat," The Mitchell Forum, vol. 52, no. 1, pp. 1–7, 2023.

[7] P. Gao, Z. Han, and F. Wan. "Big data processing and application research," in 2020 2nd International Conference on Artificial Intelligence and Advanced Manufacture (AIAM), 2020, pp. 125–128.

[8] Y. Rotalinti, et al. "Detecting drift in healthcare ai models based on data availability," in Machine Learning and Principles and Practice of Knowledge Discovery in Databases, I. Koprinska, P. Mignone, R. Guidotti, S. Jaroszewicz, H. Fröning, F. Gullo, P. M. Ferreira, D. Roqueiro, G. Ceddia, S. Nowaczyk, J. Gama, R. Ribeiro, R. Gavaldà, E. Masciari, Z. Ras, E. Ritacco, F. Naretto, A. Theissler,

P. Biecek, W. Verbeke, G. Schiele, F. Pernkopf, M. Blott, I. Bordino, I. L. Danesi, G. Ponti, L. Severini, A. Appice, G. Andresini, I. Medeiros, G. Graça, L. Cooper, N. Ghazaleh, J. Richiardi, D. Saldana, K. Sechidis, A. Canakoglu, S. Pido, P. Pinoli, A. Bifet, and S. Pashami, Eds. Cham: Springer Nature Switzerland, 2023, pp. 243–258.

[9] J. Steier, et al. “Understanding the Limits of Artificial Intelligence for Warfighters: Volume 2, Distributional Shift in Cybersecurity Datasets.” Santa Monica, CA: RAND Corporation, 2024.

[10] L. A. Zhang, Y. Ashpari, and A. Jacques. “Understanding the Limits of Artificial Intelligence for Warfighters: Volume 3, Predictive Maintenance.” Santa Monica, CA: RAND Corporation, 2024.

[11] R. A. Pyles and H. L. Shulman. “United States Air Force Fighter Support in Operation Desert Storm.” Santa Monica, CA: RAND Corporation, 1995.

[12] D. Snyder, et al. “Command and Control of U.S. Air Force Combat Support in a High-End Fight.” Santa Monica, CA: RAND Corporation, 2021.

[13] J. A. Leftwich, et al. “Advancing Combat Support to Sustain Agile Combat Employment Concepts: Integrating Global, Theater, and Unit Capabilities to Improve Support to a High-End Fight.” Santa Monica, CA: RAND Corporation, 2023.

[14] J. Fleming, et al. “Naval Logistics in Contested Environments: Examination of Stockpiles and Industrial Base Issues.” Santa Monica, CA: RAND Corporation, 2024.

[15] S. C. Slota, et al. “Good systems, bad data?: Interpretations of <scp>ai</scp> hype and failures,” Proceedings of the Association for Information Science and Technology, vol. 57, no. 1, Oct. 2020. [Online]. Available: http://dx.doi.org/10.1002/pra2.275

[16] S. K. Das. “Smart Design and Its Applications: Challenges and Techniques.” Singapore: Springer Singapore, 2021, pp. 1–6. [Online]. Available: https://doi.org/10.1007/978-981-33-6195-9_1

[17] M. Frye, J. Mohren, and R. H. Schmitt. “Benchmarking of data preprocessing methods for machine learning-applications in production,” Procedia CIRP, vol. 104, pp. 50–55, 2021, 54th CIRP CMS 2021 - Towards Digitalized Manufacturing 4.0. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2212827121009070

[18] S. James, et al. “Synthetic data use: exploring use cases to optimize data utility,” Discover Artificial Intelligence, vol. 1, no. 1, Dec. 2021. [Online]. Available: http://dx.doi.org/10.1007/s44163-021-00016-y

[19] S. A. Assefa, et al. “Generating synthetic data in finance: opportunities, challenges and pitfalls,” in Proceedings of the First ACM International Conference on AI in Finance, ser. ICAIF ’20. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3383455.3422554

[20] A. Gadetsky and M. Brbic. “The pursuit of human labeling: A new perspective on unsupervised learning,” in Advances in Neural Information Processing Systems, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36. Curran Associates, Inc., 2023, pp. 60 527–60 546. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/file/be38c-74290c251820e396680a82ce12d-Paper-Conference.pdf

[21] C. Shi, et al. “Auto-dialabel: Labeling dialogue data with unsupervised learning,” in Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, E. Riloff, D. Chiang, J. Hockenmaier, and J. Tsujii, Eds. Brussels, Belgium: Association for Computational Linguistics, Oct.-Nov. 2018, pp. 684–689. [Online]. Available: https://aclanthology.org/D18-1072

[22] A. Shrivastava, et al. “Learning from simulated and unsupervised images through adversarial training,” in 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 2242–2251.

[23] A. Stringer, et al. “Application of generative machine learning for adaptive detection with limited sample support,” in 2024 IEEE Radar Conference (RadarConf24), 2024, pp. 1–6.

[24] J. E. Fountain. “The moon, the ghetto and artificial intelligence: Reducing systemic racism in computational algorithms,” Government Information Quarterly, vol. 39, no. 2, p. 101645, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0740624X21000812

[25] E. Albaroudi, T. Mansouri, and A. Alameer. “A comprehensive review of ai techniques for addressing algorithmic bias in job hiring,” AI, vol. 5, no. 1, pp. 383–404, 2024. [Online]. Available: https://www.mdpi.com/2673-2688/5/1/19

[26] A. Stringer, et al. “A metacognitive approach to adaptive radar detection,” IEEE Transactions on Aerospace and Electronic Systems, vol. 60, no. 1, pp. 168–185, 2024.

[27] D. Minh, et al. “Explainable artificial intelligence: a comprehensive review,” Artificial Intelligence Review, vol. 55, no. 5, p. 3503–3568, Nov. 2021. [Online]. Available: http://dx.doi.org/10.1007/s10462-021-10088-y

[28] A. Vakil, et al. “Finding explanations in ai fusion of electro-optical/passive radio-frequency data,” Sensors, vol. 23, no. 3, 2023. [Online]. Available: https://www.mdpi.com/1424-8220/23/3/1489

[29] P. Villalobos, et al. “Will we run out of data? limits of llm scaling based on human-generated data,” arXiv preprint arXiv:2211.04325, 2024, dOI: https://doi.org/10.48550/arXiv.2211.04325

About the Authors

Alexander Stringer received a B.S. degree in electrical engineering from the New Mexico Institute of Mining and Technology. He received M.S. and Ph.D. degrees in electrical and computer engineering from the University of Oklahoma, with a focus on machine learning and radar signal processing. In June 2020, he helped found the 76 SWEG office of research, where he currently serves as an applied research lead. His current research interests include first principles-based machine learning, cognitive radar, metacognition, and machine learning explainability.

Alex Stringer

Research Lead

AFSC/SW/76 SWEG

alexander.stringer@us.af.mil

Geoffrey Dolinger received the B.S. degree in electrical engineering from Oklahoma Christian University. He received the M.S. degree in electrical and computer engineering from Oklahoma State University with a focus on controls and neural networks. In July 2021, he joined the 76 SWEG office of research, where he currently serves as an applied research lead. He is also a doctoral candidate in the electrical and computer engineering department at University of Oklahoma. His current research interests include cognitive radar, metacognition, reinforcement learning, and multiagent game theory.

Geoffrey Dolinger

Research Lead

AFSC/SW/76 SWEG

geoffrey.dolinger@us.af.mil

Asad Vakil graduated from the Ohio State University in 2018 with a Bachelor's degree in Electrical and Computer Engineering, a Masters in Electrical and Computer Engineering in 2022 at Oakland University, and is currently in the process of completing a PhD in Electrical and Computer Engineering at Oakland University. The primary focus of his research was on heterogenous sensor fusion between passive radio frequency and electro optical modalities for the purposes of achieving explainable target detection. He currently works as a civilian engineer at the 76th SWEG, 559th Software Engineering Squadron.

Asad Vakil

Research Engineer

AFSC/SW/76 SWEG

asad.vakil@us.af.mil

Joseph Karch is a Research Engineer with the United States Air Force. He graduated with a B.S. in Computer Engineering and a M.S. in Business Administration from Oklahoma Christian University. Joseph has worked for the 76 Software Engineering Group (SWEG) for over two years, and has worked on the applied research team for over a year and a half. He has contributed to publications related to artificial intelligence and machine learning combined with the fields of radar, navigation, and computer vision. He has been a co-author on four IEEE publications. This work has been presented at international conferences in Barcelona, Spain and Sydney, Australia.

Joseph Karch

Research Engineer

AFSC/SW/76 SWEG

joseph.karch.5@us.af.mil

Timothy Sharp is a Research Engineer with the United State Air Force. He received a B.S. in Computer Engineering, Physics, and Mathematics from The Virginia Polytechnic and State University. In January of 2023 he joined the 76 SWEG office of research where Timothy focuses on conducting research in machine learning (ML) applied to the fields of Radar, Navigation, and Computer Vision. Timothy has contributed to the field through several publications. In his role, Timothy has been pivotal in advancing the application of ML in various domains, including the development of innovative solutions for radar and navigation systems. He has collaborated with interdisciplinary teams to drive research that bridges theoretical ML concepts with practical implementations in defense technology.

Timothy Sharp

Research Engineer

AFSC/SW/76 SWEG

timothy.sharp.7@us.af.mil

Harnessing AI for Military Use Cases: A Practical Guide to Capabilities and Limitations

RAOUF DRIDI
CHIEF TECHNICAL OFFICER
TRANSFORM COMPUTING, INC

STEVE REINHARDT
CEO
TRANSFORM COMPUTING, INC

Introduction

Almost everyone engaged in modern society is aware of the emergence of Artificial Intelligence (AI) in the form of Large Language Models (LLMs), stemming from OpenAI's announcement of the release of ChatGPT 3.0 in November of 2022. Numerous news items have described the astonishing advances of this form of AI, from fooling some users with its almost eerie humanity, writing excellent short stories and creating compelling images, to translating natural language and providing very good information via chat interfaces. Fewer news items have chronicled the shortcomings of AI, such as its sometimes-perplexing hallucinations (blatant errors), its propagation of human biases, and the unsustainable growth in the cost, both economic and electrical, of training the next new (usually much bigger) model. Our approach in this article reflects that of Jesse Ehrenfeld, recent president of the American Medical Association, who said "It is clear to me that AI will never replace physicians — but physicians who use AI will replace those who don't"[1]. That is, AI has useful abilities, and should be used where practical to solve problems in the real world, being carefully cognizant of its strengths and weaknesses.

AI has attracted colossal amounts of investment, development focus, and hype, so progress has been almost unbelievably rapid, even for tech-savvy observers. Using AI breaks the mold of conventional software engineering, where software engineers work with proven, stable methods. It forces software engineers into a different regime, using a method that is highly capable but unproven and rapidly changing, in its major capabilities and its Application Programming Interfaces (APIs), and this reality will not be suitable for all military contexts. Readers should check for developments since this article was finalized (November 2024).

This article will first describe some of current AI's strengths and weaknesses. It will then detail the work to overcome one of those weaknesses, the lack of explainability, which often causes AI's human overseers not to trust AI's results enough to use them for critical use cases. To make Explainable AI's use and value concrete, it is applied to an important domain, the resilient and efficient operation of dynamic energy microgrids (i.e., military bases) primarily based on renewable energy generation. The article addresses possible strategies for avoiding other AI weaknesses and then summarizes with

approaches that can help organizations exploit AI's considerable strengths while avoiding the pitfalls of its considerable weaknesses.

Current AI Strengths

Prediction of Natural-Language Responses

The core capability that enables LLMs to provide value is their ability to predict the next word (token) in a sentence (string of tokens) based on the massive training the LLM has undergone. (Tokens can also be (e.g.) pixels in the context of images or notes in the context of music.) This may seem like magic, but it's really just the repeated measurement of what words are most likely to follow other words in the training material. Current LLMs have huge numbers of parameters (70 billion for Llama 3.1 70B released in July 2024, with unconfirmed reports of up to ~2 trillion parameters for OpenAI GPT-4, Anthropic Claude 3, and Google Gemini variants), meaning they can plausibly track a huge number of potential concepts, and this power is brought to bear daily for the many regular users of these LLMs.

Translation

This same predict-the-next-token capability enables LLMs to perform some difficult aspects of natural-language translation compellingly, choosing appropriate synonyms and phrasing based on the detailed context, which delivers translations that seem more accurate and nuanced to the human ear than previous machine translations. Translating natural language via an LLM is still subject to some of LLMs' weaknesses and so should be validated by a human, but many observers believe that LLMs' performance on language translation will improve quickly and become the de facto approach.

Current AI Weaknesses

Unsustainable Computational Intensity

As AI and LLMs have become more capable, their expected high economic value has created a high sense of urgency to make them better quickly. Because the training phase of LLM development is the most time-consuming and is extremely parallel, using more processors (typically graphics processors (GPUs)) can reduce the runtime proportionally, with the expense of more money and electricity. This has led to astonishing growth in the amount of computing used for training LLMs over the last 10+ years. One of the first efforts to highlight this, in 2018, was by OpenAI, which showed that from 2012 to 2018, the computing/electricity used for training doubled every 3.4 months, which translates to increasing about 100 times every 2 years, as shown in the figure below [2]. Even for those used to the rapid growth of compute power over the last few decades, usually measured as doubling every 2 years, it's hard to grasp a growth rate of 100 times every 2 years, which extrapolates to 10 billion times (10^{10}) every decade. To be clear, the growth rate in recent computing power was for a fixed budget, which AI training is not; AI-training budgets have grown drastically, to the point where Elon Musk recently announced a new system installed by xAI that has 100,000 NVIDIA H100 GPUs, which nominally would consume 150MW of power and cost $3-4 billion. And xAI plans to double the size of the system in short order [3].

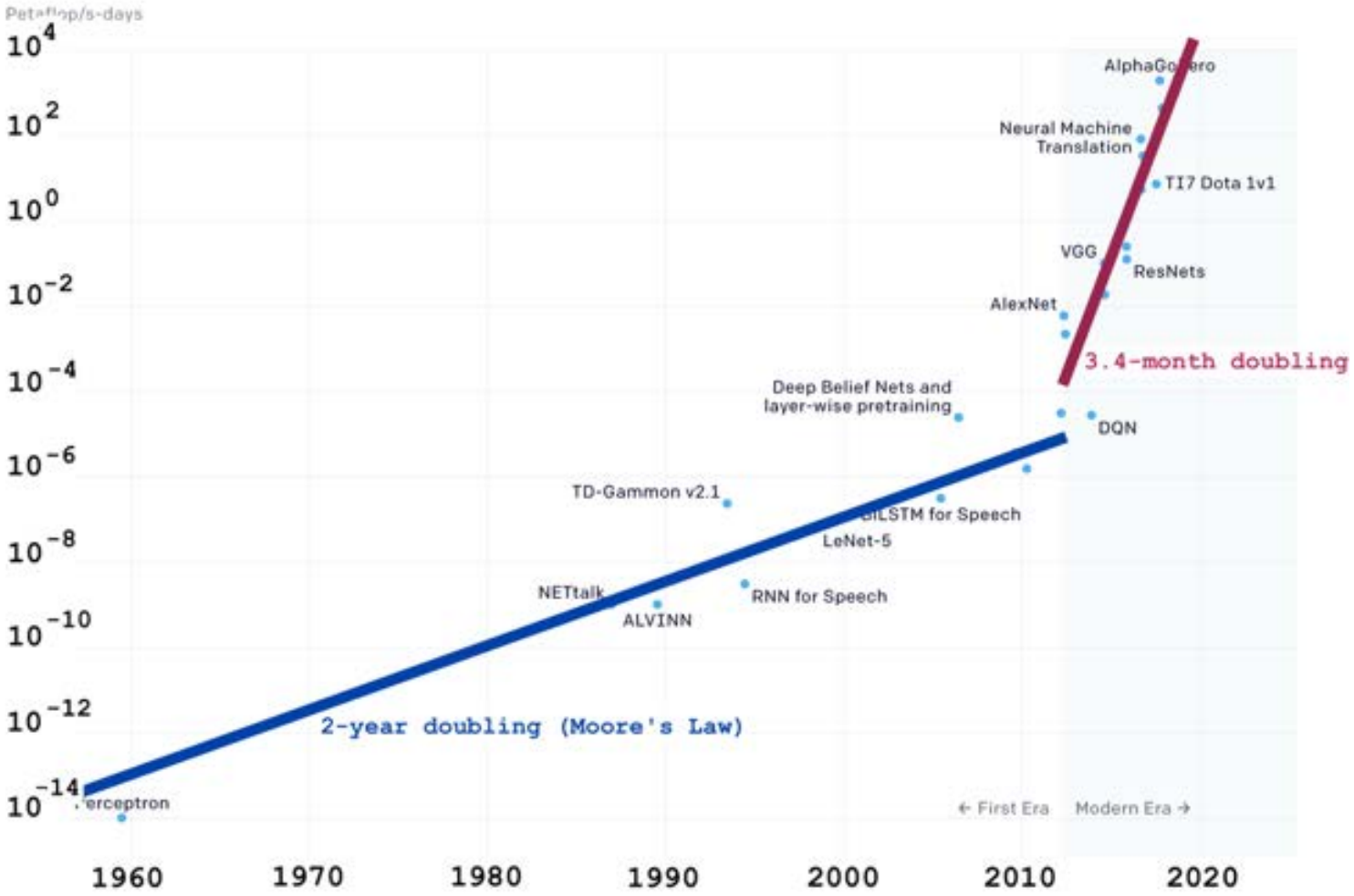


Figure 1. *Growth in the amount of computing used for AI, 1960-2022, from OpenAI [2].*

It can be expected that growth will not continue at this rate, as it would consume an untenable fraction of generated electricity. For example, one analysis by WellsFargo predicts that AI would grow from 0.2% (8 terawatt-hours (TWh)) of U.S. electricity demand in 2024 to 652 TWh in 2030. If U.S. electricity generation doesn't also grow, which is unlikely, AI would consume 16% of U.S. electricity in 2030 [4]. This tension between supply and demand could be resolved in a number of ways; as an AI user, it is prudent to expect AI's power consumption to cause changes in preferred use styles and cost.

Lack of Explainability

Today's AI is built on neural networks, which are layers of "neurons" connected to mimic loosely the way human brains work. Language model developers have increased the number of layers of neurons, and those added layers enable more discrimination between sets of training data. However, for most humans, the underlying neural net still does not work as an explanation or rationale for why the AI gave the answer it gave. For serious tasks, where human health or large sums of money are at risk, humans need insight at least into how the AI made its recommendation, if not beyond that into how the AI chose between different hypotheses that humans might have considered. See the Explainable AI section (page 35) for more detail.

Untrustworthiness

The current generation of generative AI is notorious for hallucinations, or errors of fact, perhaps the most glaring of which are fictitious web links provided as evidence for a given assertion. An early example of this happened in court filings in the Southern District of New York in 2023 where AI fabricated precedent court cases that did not exist, and the lawyer using the AI did not double-check the AI's output [5]. "Untrustworthiness" in this context is different from in a truly human context. It is true that the LM is not trustworthy, in the sense that it is not careful enough about the truthfulness and accuracy of its statements, and so does not engender trust in the humans consuming those statements. Unlike humans, however, current AI has no motivation to deceive or inflate its apparent effectiveness; it is just lacking the self-awareness to know when it is at or past the point at which it can be effective. Other than certain classes of problems or information that the language model knows it can't effectively address, it doesn't understand when to say, "I don't know." As LMs improve, they may acquire human-like motivations that could complicate the task of believing their rationales.

Lack of Commercial/Technical Stability

APIs for AI services are changing rapidly, reflecting the rate of change in the underlying AI technology. This conflicts with the needs of warfighters, who need systems that work stably over 10 years or more (sometimes referred to as "set and forget"), and puts defense-software engineers in a bind, having to bridge long-term stability to the user with rapid change in a key underlying technology. Note that the change in AI APIs is just an extension of the rapid changes in cloud APIs that have challenged defense-software engineers for similar reasons over the last several years.

Explainable AI (XAI)

Very often, discussions of XAI's capabilities suffer from vague or ambiguous definitions, making it challenging to grasp its essential properties and carry on proper rigorous discussions. This section presents a framework for XAI that draws parallels with proofs in formal logic. An example use case could be recommending changes to the human operator of an energy microgrid powering a military base; see XAI Applied to Energy Microgrids for Military Bases on p. 37 below. The proposed framework precisely defines and captures the key properties of XAI and explainability. A framework like this can deliver the explainability that will build trust in defense-software engineers and their warfighter users, enabling AI to be deployed for appropriate problems. This framework employs mathematical concepts – not overly technical but grounded in common sense – to provide clarity and facilitate meaningful discussion. Let us start with defining the fundamental components of our framework:

- Input Space X: A set representing all possible inputs to the AI model.
- Output Space Y: A set representing all possible outputs or predictions.
- Model (f): A function $f:X \rightarrow Y$ representing the AI system.

To define explanations, consider a **logical system** L, such as First-Order Logic, Higher-Order Logic, or Intuitionistic Logic. A logical system provides a formal language with syntax (symbols, formulas), semantics (meaning of the formulas), and inference rules (how to derive conclusions from premises). In this context, the AI model's operation is considered as analogous to a theorem in mathematics, where the input and the model's architecture correspond to the premises, and the output corresponds to the conclusion. The **Explanation Space** E is then defined by the set of all valid proofs in the logical system L that leads to conclusions about the model's predictions. Formally,

$$E = \{ p \in P \mid Conclusion(p) \in Y \}$$

where *P* is the set of all possible proofs in *L*, and *Conclusion(p)* extracts the concluding statement from proof *p*. Each proof *p* in *E* represents a potential explanation for a prediction made by the model. This formulation captures the idea that explanations are sequences of logical deductions that start from known premises and end with the model's prediction. The **Explanation Function** *e* maps each input $x \in X$ to a proof $p(x) \in E$ that derives the model's prediction *f(x)* from a set of axioms or premises related to *x*:

$$e: X \rightarrow E,$$

$$e(x) = p(x)$$

and satisfies *Conclusion(p(x)) = f(x)*. The map *e* ensures that for every input, there is a corresponding explanation in the form of a logical derivation that culminates in the model's prediction.

These definitions bring rich framing: by thinking of explanations as proofs, the results are grounded in the language of logic, well known to software engineers and end-user analysts, while also capturing the compositionality inherent in complex AI models, which are often constructed from simpler components. This mirrors how proofs are built from basic logical steps. Furthermore, since humans often use logical reasoning to explain decisions, this framework is intuitively appealing for generating explanations that are accessible and meaningful to users. This extended model is depicted in **Figure 2**, where the conventional language model on the left does not have the ability to explain how it got its answers, while the explainable language model on the right is extended with a logic module that delivers explainability.

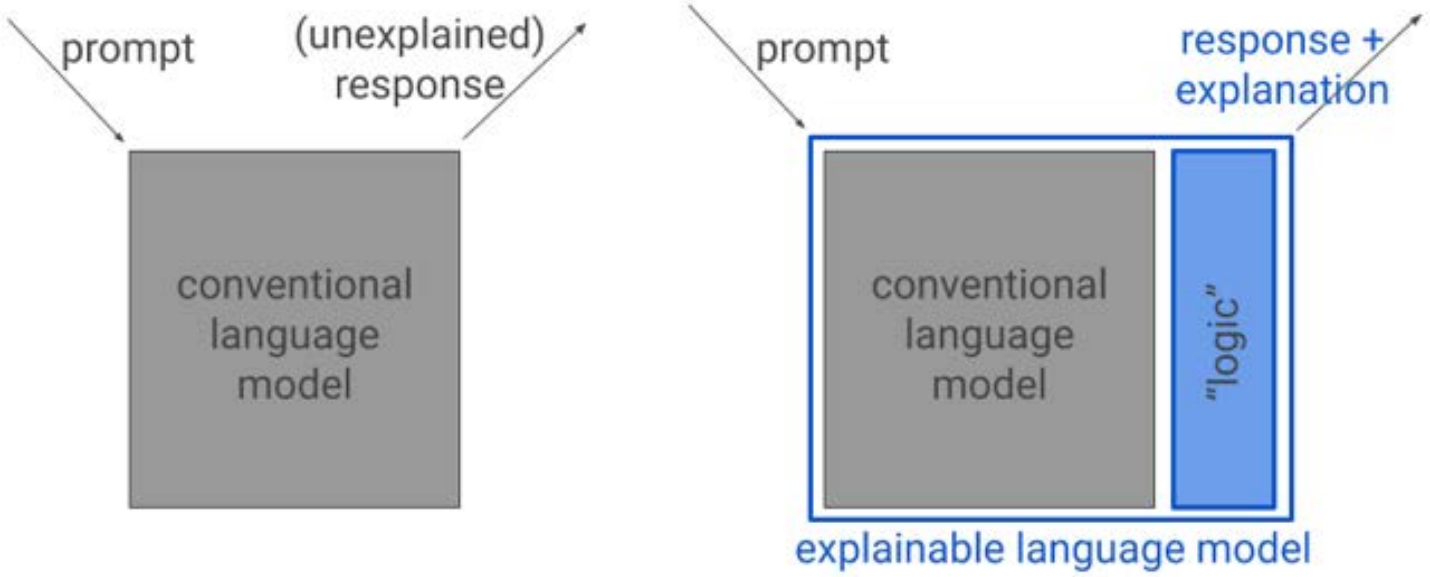


Figure 2. *Conventional Language Model and Language Model with Complementary Logic Module Enabling Explainable AI.*

Perhaps more importantly, in this age of misinformation and fabricated content, formal proofs can be verified for correctness (i.e., "show your work"), providing a means to ensure the validity of explanations. Logical systems are expressive enough to capture various forms of reasoning—deductive, inductive, and even abductive—allowing us to model the diverse ways in which AI models process information.

There is one more consideration to discuss: quantifying the quality of explanations. For this, consider measures of fidelity and interpretability. *Fidelity* assesses how accurately the explanation reflects the model's actual reasoning process; an explanation with high fidelity ensures that the logical derivation mirrors the computational steps of the model. *Interpretability* gauges how easily a human can under-

stand the explanation. It can be influenced by factors such as the length of the proof, the complexity of the logical constructs used, and the familiarity of the reasoning patterns to the intended audience. The objective is to find an explanation function e that balances fidelity and interpretability across the input space X.

XAI Applied to Energy Microgrids for Military Bases

Microgrid Background

Humankind faces the urgent need to avoid the Earth's climate breakdown, which would be catastrophic for human society. Reducing greenhouse-gas emissions from energy production is a major portion of that task, so worldwide there is a massive shift from greenhouse-gas-emitting fuels to renewable sources of energy; examples include solar, wind, hydro, and geothermal, with specific technologies spanning a wide range of maturity.

The U.S. energy grid needs to quickly incorporate massive renewable-generation capacity mediated by massive storage capacity, from a fleet of devices whose composition itself is quickly changing as new technologies become commercially relevant. The growth in the number of devices and their bidirectionality of power, compared with the unidirectionality of the traditional generation-to-load grid, requires radically new techniques to deliver the necessary reliability, resilience, flexibility, and cost-effectiveness.

Microgrids, each of which is a collection of energy grid resources (generation, load, and usually storage) under the control of a single organization, are viewed as essential to widespread deployment of renewable generation, as they provide an architecture that balances the need for rapid local change with the need for stability in connections with the wider grid. *Islanding*, the ability to separate from the

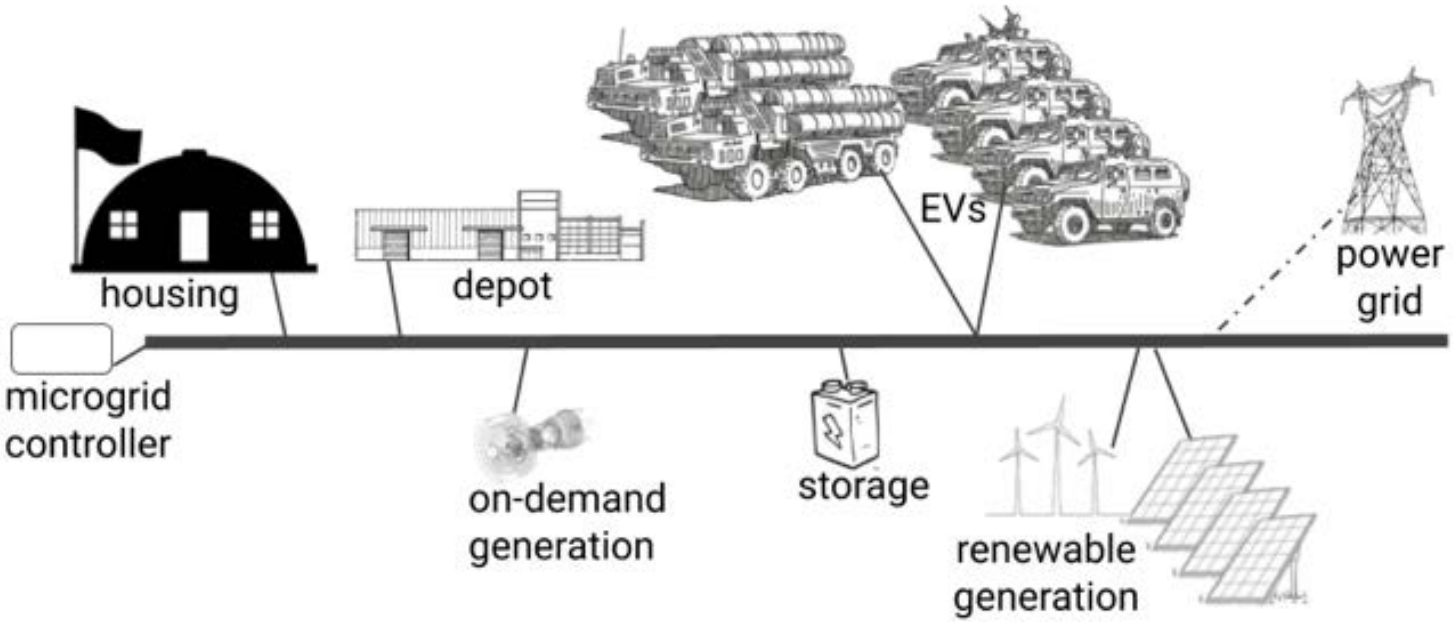


Figure 3. *An example military microgrid, showing typical uses as well as renewable-energy generation and storage.*

wider grid, insulates a microgrid from macro events such as hurricanes or cyberattacks on the energy grid, delivering better resilience than in the conventional grid. In many other circumstances, however, integration with the wider grid gives economic advantages (e.g., selling local renewably generated power when the purchase price is high and buying remotely generated power when the local demand exceeds the local supply). Thus flexibility in detaching and reconnecting from the wider grid and robust operation in either mode are key.

Value of Military Microgrids

The U.S. Department of Defense (DoD) has been one of the early movers toward microgrids, recognizing the reduced reliability of the wider grid both in the U.S. and in other countries and its impact on military readiness. To date, the key military microgrid benefit has been improved energy security and resilience and resulting enhanced mission continuity. Unlike many civilian situations, military bases can be rigorous in their response to reduced energy supply, prioritizing more and less critical needs, and microgrids enable this prioritization.

As shown in the figure above, military microgrids have primary loads that are transportation and tactical vehicles, depots, and housing. Supply comes from local generation, renewable (typically solar or wind but possibly hydro, geothermal, or others) or fossil-fuel-based, storage, and possibly transmission from the wider power grid. The local microgrid will have its own controller.

The next key step in fulfilling the potential of military microgrids is integrating renewable energy generation and storage and driving operational efficiency. The first phase's focus on resilience accepted local generation of any type, greenhouse-gas-emitting or renewable. The second phase requires integrating renewable generation devices, the most effective of which generate intermittently and thus need storage (usually batteries) to deliver power when generation is insufficient. For a military base, there are many constraints that the local microgrid must satisfy – e.g., ensuring that critical energy loads are always served, if at all possible, that storage always contains a threshold amount of energy, that renewable generation is exploited fully, that vigilance to cyberattacks is never lacking, and that use of greenhouse-gas-emitting fuels is minimized. Operating the microgrid effectively in the face of uncertainty (e.g., generation and load affected by (esp. extreme) weather, load affected by hard-to-predict mission requirements and durations, cost affected by market forces, component reliability) requires both forecasting the expected future state of the microgrid as well as responding when the forecast proves inaccurate.

XAI for Military Microgrids

All microgrids, military or otherwise, will be pressed to work effectively quickly as foundation pieces of the Smart Grid. Given all the uncertainties in distributed energy resources (DERs), both generation and storage, including forecasting generation and load, protecting against electromagnetic transients (EMTs), and anticipated cyberattacks to the grid, microgrids will need analytics that provide headroom to cope with unanticipated issues. Military microgrids will have the added burden of likely being the first to be stressed in particular dimensions – early deployment, size (number of devices), efficiency, and use of demand response (i.e., the ability for the grid to notify certain loads that they will not be satisfied) - in addition to the obvious extra requirement of working well in truly life-or-death situations, which will put even more stress on microgrid analytics. Also, despite the name, microgrids can easily have enough nodes to exceed the abilities of conventional analytics. With these pressures to deploy scaled microgrids in earnest quickly, much better analytics are sorely needed and XAI is a promising path.

The explainable AI description above, conceivably, could be implemented by a variety of mecha-

nisms. One way to couple the language of logic to the language of AI is by using key concepts from the algebraic topology branch of advanced mathematics, which can identify noteworthy changes in data, as a bridge with the predictive power of Language Models (LMs). One such math concept is persistent homology, which measures the topology of data in the mathematical sense, meaning the structure of data that does not change with simple deformations and hence that structure can be reliably viewed as more real (persistent) and not due to the noisiness of the real world. Persistent homology can distinguish usual and unusual activity, which is used with an LM to identify worrisome from benign changes. The other LM contribution is to translate the not-very-accessible persistent homology concepts into energy-grid concepts familiar to grid operators and analysts. LMs are completely comfortable with persistent homology, so asking an LM to use persistent homology internally to analyze data, predict changes, and translate the concepts back into the familiar terminology known by the grid operator is a potential way both to sanity-check the LMs' results and to give a rationale to humans that can be compelling. Transform Computing, Inc (XFR) built prototypes of this capability and found that LMs are surprisingly effective in analyzing data and translating between domains in this way. Persistent homology provides the sanity check or proof, along with explanation, that enables human

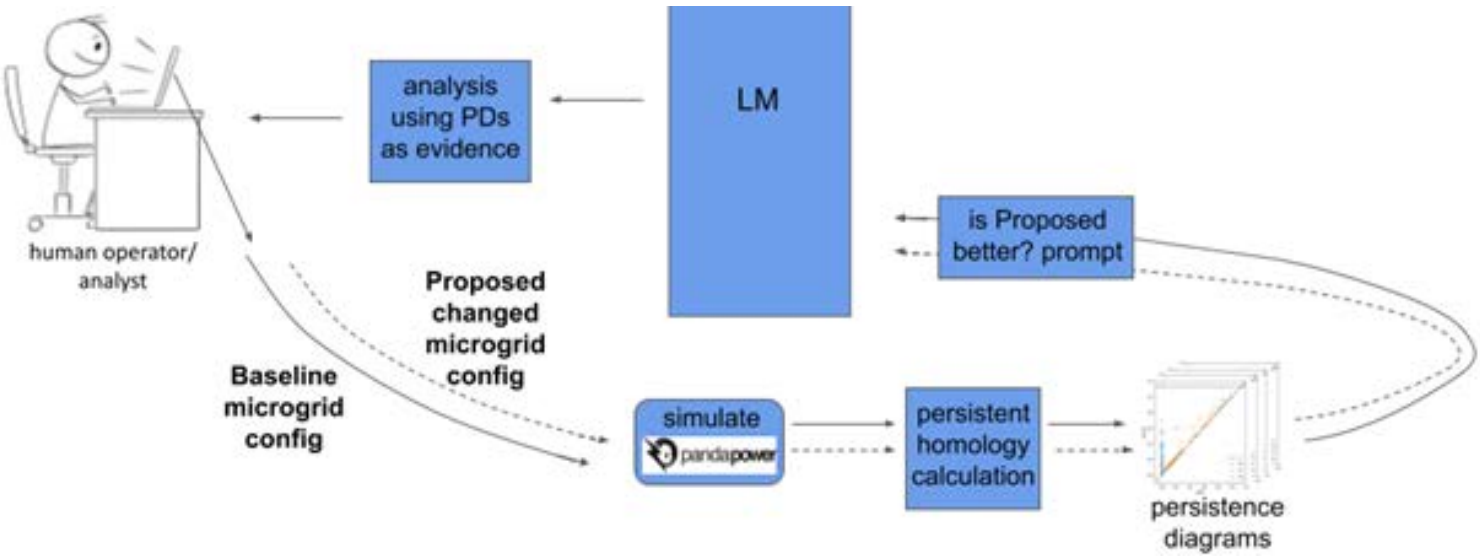


Figure 4. *Language model translates persistence diagrams from math domain to grid domain.*

grid operators to trust the results of XAI. These two uses of an LM are illustrated in **Figure 4**: the LM receives as input the persistence diagrams from simulations of the baseline and proposed-change grid configurations, interprets them for their impact on the quality of the grid, and translates those results from the math world of persistence diagrams to the grid world of the grid operator. Persistent homology and other advanced mathematical methods can deliver powerful analysis that scales to microgrid size/complexity that conventional analytics cannot, and early collaborators are working to prove these methods with real-world data.

Relevance for cybersecurity

Most of our attention has been focused on bolstering the electrical aspects of microgrids, but microgrids also have conventional computer networks inside them, which are already subject to conventional cybersecurity attacks and are expected to be targeted for combined computer/grid attacks that could temporarily disable or even permanently damage electrical generation, storage, and transmission equipment. Protecting the Smart Grid against such attacks will be paramount, whether the attackers

are private actors or nation states, and complicated by the need to deploy and scale military microgrids quickly. With comprehensive testing of microgrid software impractical, analytics that give strong situational awareness of new or poorly understood attacks will be of high value. The combination of proofs via advanced math, especially persistent homology, and LMs will deliver this required situational awareness.

Strategies for Avoiding Other AI Weaknesses

AI suffers from several weaknesses, particularly hallucinations and technical instability, both of which hinder the commercial viability of AI systems. The now familiar phenomenon of hallucinations occurs when models produce outputs that are ungrounded, nonsensical, or not based on the input data, i.e., a fabrication. This issue is especially problematic in applications like decision support systems, where inaccurate or misleading information can lead to serious consequences. Technical instability is less familiar and refers to unpredictable or inconsistent behavior in AI models, such as sensitivity to small input changes or fluctuations in performance over time.

Addressing these issues is an active area of research, with techniques like Reinforcement Learning from Human Feedback (RLHF) aiming to align model outputs with human preferences. RLHF involves training models using feedback from human evaluators, guiding the AI to produce more accurate and relevant responses. However, while RLHF and similar methods provide incremental improvements, they often lack a rigorous foundational framework and may not fully eliminate hallucinations or technical instability.

The proposal in the Explainable AI section (page 35) might be a good candidate to address these two major weaknesses—hallucinations and technical instability. First, it naturally addresses hallucinations as it employs formal logic to structure and constrain the reasoning process of AI models, so the likelihood of hallucinations is inherently reduced. In this framework, every prediction made by the model is accompanied by a logical proof that derives the output from the input data using valid inference rules. Each step in the reasoning process must logically follow from the previous one, ensuring that conclusions are directly tied to the premises. This means there is no room for unsupported information, as every conclusion must be justified within the logical system.

The same is true for technical stability: the model behaves in predictable ways, as the same premises and reasoning steps will consistently lead to the same conclusions. With explanations formalized as proofs, developers can more easily identify the sources of errors or unexpected behavior in the model. For instance, when the model's prediction deviates from expected outcomes, the corresponding proof can be examined to pinpoint where the reasoning diverged.

Summary

The current generation of AI, embodied by large language models, has apparently immense but poorly scoped capabilities, with some known weaknesses. Prudent organizations are exploring how to make use of the capabilities while avoiding the weaknesses. XFR is focused on explaining AI's results to humans in ways that build trust and confidence, building on the power of advanced math in concert with LMs. This capability is expected to be especially high value for energy microgrids, a

compelling societal need, which must grow rapidly in size, resilience, and efficiency, beyond the ability of conventional analytics to deliver. Prototypes have demonstrated some of the key capabilities of this end-to-end workflow. These and similar developments are expected to address AI's current shortcomings and help deliver more of AI's potential to a larger audience.

References

[1] Schumaker, Erin; Leonard, Ben; Paun, Carmen; and Peng, Evan. "AMA president: AI will not replace doctors." Politico, 10 July 2023. https://www.politico.com/newsletters/future-pulse/2023/07/10/ai-will-not-replace-us-ama-president-says-00105374.

[2] "AI and Compute." OpenAI, 16 May 2018, https://openai.com/index/ai-and-compute/.

[3] Eadline, Doug. "xAI Colossus: The Elon Project." HPCwire, 5 Sept. 2024, www.hpcwire.com/2024/09/05/xai-colossus-the-elon-project.

[4] Kindig, Beth. "AI Power Consumption: Rapidly Becoming Mission-Critical." Forbes, 27 Aug. 2024, www.forbes.com/sites/bethkindig/2024/06/20/ai-power-consumption-rapidly-becoming-mission-critical.

[5] Weiser, Benjamin. "Here's What Happens When Your Lawyer Uses ChatGPT." New York Times, 27 May 2023. https://www.nytimes.com/2023/05/27/nyregion/avianca-airline-lawsuit-chatgpt.html.

About the Authors

Raouf Dridi is driven by his belief that computations can often be dramatically accelerated by casting them in a different and better form for the target computer and has applied that to AI and machine learning with compelling results. Raouf earned a PhD in Math and CS, focusing on algebraic topology, at the University of Lille and then led efforts to map real-world problems to quantum computers while at the University of British Columbia, 1QBit, Carnegie Mellon University, and, most recently, led the Advanced Technology team at Quantum Computing Inc.

Please see his Google Scholar (https://scholar.google.com/citations?user=_dUVcN4AAAAJ&hl=en&oi=sra) and LinkedIn profiles (https://www.linkedin.com/in/raouf-dridi/) for more information.

Raouf Dridi

Chief Technical Officer

Dubai, UAE

https://www.linkedin.com/in/raouf-dridi/

Steve Reinhardt strives to connect the best of new computing technology with a customer's core needs, knowing that both technical and human factors must be aligned for a successful deployment. He has led technical and project teams, building scalable hardware/software systems, graph databases, and quantum-capable optimizers at Cray Research, SGI, Cray/YarcData, Interactive Supercomputing, D-Wave Systems, Quantum Computing Inc., and Quantum Machines. Four systems for which Steve led development have won HPCwire Readers'/Editors' Choice Awards and two of those each delivered more than $500M of revenue.

Please see his Google Scholar (https://scholar.google.com/citations?user=huV-_rsAAAAJ&hl=en) and LinkedIn profiles (https://www.linkedin.com/in/stevenpreinhardt/) for more information.

Steve Reinhardt

CEO

Eagan, MN

steve@xfr.ai

Engineering Security in APIs

Alejandro Gomez
Software Engineer,
Software Engineering Institute

Abstract

APIs are everywhere in software, providing mission-critical functionality. They expose large amounts of system functionality, making them valued targets for cyber-attack. Engineering security into an API is therefore essential to ensure it operates as expected, even in adverse conditions.

Introduction

Application Programming Interfaces (APIs) are everywhere in software, from library components to classes and modules. There are many definitions out there for what an API is, but I prefer Joshua Bloch's (former head of Java at Google) better than all others since it gets to the heart of what APIs provide: "A set of functionalities independent of their implementation, allowing the implementation to vary without compromising the users of the component[1]." In other words, the system may change, but the interface it provides - it's set of operations publicly available - does not change as much, if at all.

Figure 1 shows a visual representation of an API. System A and B have interfaces that expose a set of functionalities (Op1, Op2, etc.) that communicate to each other via a channel. These functionalities have a separate, and often slower, development rate of change than the systems they provide access to.

To engineer security in an API is to come up with empirical, cost-effective techniques that maintains the API's security attributes even in adverse conditions: if we are not empirical (making decisions with data) we are not learning from our observations, and those of others, to create better solutions; if we

---

This article is adapted from a presentation given at the Secure Software by Design Conference in Arlington; VA held in August 2024. There's an accompanying White Paper on the SEI website, "On the Design, Development and Testing of Modern APIs."

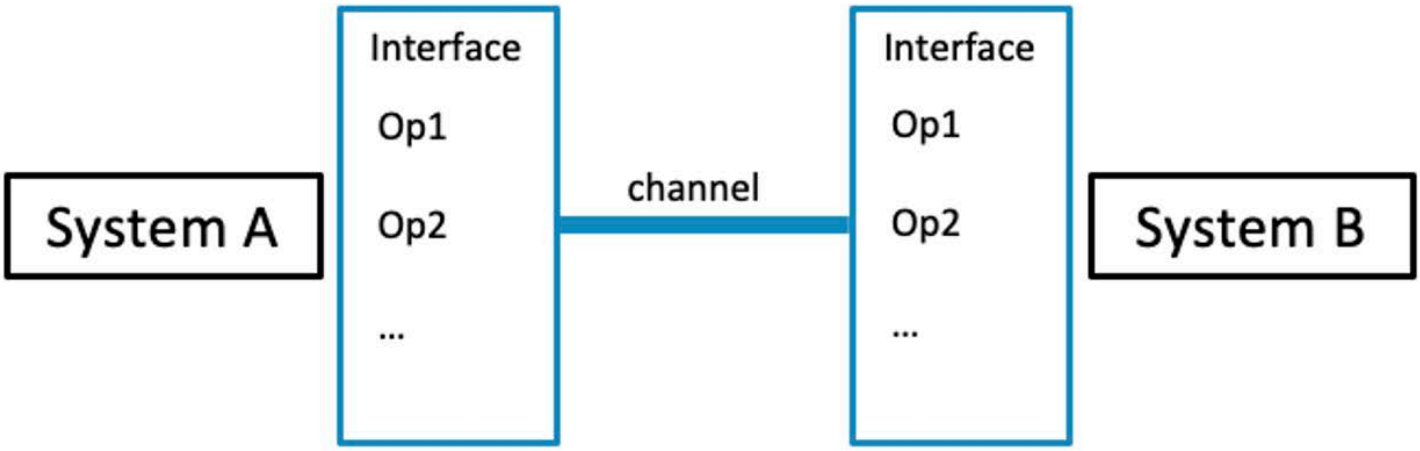


Figure 1. *Visual representation of two, connected APIs.*

are not cost-effective, we will fail to deliver the solution on-time and on-budget; maintaining security attributes comes down to the fundamental questions of, “how can you know the system you’re building will work as expected?” and, “is your system trustworthy?” These are values that are important in engineering security in APIs.

The State of Modern APIs

Modern APIs are ubiquitous, expose large amounts of system functionality to users, and require evolvable interfaces (interfaces that are both backwards, and forwards-compatible) – all of which are avenues for attackers to exploit.

Ubiquitous

In a 2024 report, Cloudflare found that, between 53-60% of all internet requests came from APIs as opposed to other sources [2]. API requests are also the fastest growing kind of traffic on the internet. If we break down the data by industry we can see, perhaps surprisingly, that heavily regulated industries such as banking, financial services, and telecommunications comprised the biggest growth in API usage. If someone thinks that APIs can’t be used because they’re inherently insecure, there’s the fact that many businesses are putting themselves on the line because they realize APIs bring an extraordinary amount of value. So, APIs aren’t just used by internet companies, they’re being used by everyone.

According to metrics collected by data.gov, which is a real-time metrics aggregator provided by the U.S. government, there have been over 12 billion requests for APIs provided by federal agencies since they started collecting data, with over 300K unique API keys representing an approximate number of applications using this data [3].

Expose System Functionality

APIs expose a large share of system functionality. According to the same Cloudflare report, over half of API requests in the web are POST requests, which means they are requests that modify the system the API is exposing. This includes adding an entry to a database, making a logical change, creating or deleting resources, etc. This represents an enormous amount of functionality being made available on the internet for a clever attacker to abuse [2].

The importance of APIs, particularly third-party APIs, to work as expected cannot be understated: in perhaps one of the most famous cases of software failure, the UK Post Office prosecuted, convicted, and bankrupted hundreds of its own employees over the course of 16 years due to a faulty accounting software they were using for payroll. The system regularly showed money disappearing from Post Office accounts due to bugs in the API. “Each time the user pressed “enter” on the frozen screen, it would silently update the record. In one instance, that bug created a £24,000 discrepancy, which the Post Office tried to hold the post office operator responsible for” [4]. The Post Office never questioned the results the third-party API was providing them, assuming they were always correct, and prosecuting the post-masters (many of which are elderly or poor). It took many years for the UK government to realize this mistake and overturn convictions and bankruptcies it had made. Investigations are still ongoing.

The lesson from this is that APIs have become an essential part of the workflow for many businesses and governments, and their behavior is usually taken as gospel. Users (either in the company using the API or outside it) should be able to provide rapid feedback to the developers building the API in order to alert if something is off.

The effect of faulty API software is shown by a 2023 Palo Alto Networks survey: 92% of organizations experienced an API-related security incident the previous year. Of these, 57% experienced multiple API-related security incidents. Even though 3/4 of organizations reported having a robust API security program. A 2023 study commissioned by Akamai API Security confirms these observations: 41% of organizations surveyed had an API security incident in the last 12 months. 63% of those noted that the incident involved a data breach or data loss [5].

Evolvable Interfaces

Modern APIs require evolvable interfaces. That is, they frequently require backwards and forwards compatibility. Everyone who builds software knows that their codebase will change significantly over the course of years, months, even weeks. According to an analysis by Bernhardsson, older projects such as Linux and Git on average have the same line of code change once every 6 years; younger projects such as Kubernetes, Angular, and Keras change once every 6 months. A fitted curve for a few dozen open-source projects shows the average half-life of a line of code is 3 years [6]. Stripe, a global payments system company that makes the most used payments API in the world, has changed its API interface (not just a few fields but the whole interface) 6 times in the last 10 years [7].

So now we have a pretty good idea of the state of modern APIs: they are ubiquitous, they regularly expose large amounts of system functionality (many times mission-critical functionality with life-or-death consequence), and they require interfaces that evolve as requirements change. These attributes provide critical pathways for attackers to exploit, and they increase the risk to the organization that creates these APIs as well as those that are dependent on it. We, as software designers, need to assure that APIs work as expected by users, that they are trustworthy and useful, instead of vulnerable; an asset instead of a liability to the organization, and a tool for providing value instead of creating harm.

Development

Since we’re interested in building high-quality APIs quickly and securely, the best methodology to use is DevSecOps. DevSecOps allows for empirical development techniques by reducing the number of variables being changed and deployed in a system –instead of developing for 6 months and then having a “big-bang” deployment, we make changes small, atomic, fully tested, and reversible. If a

change causes a service failure or bug in production, we can easily identify the root cause of the issue, rollback, and re-deploy.

Supply Chain Security

Developing any kind of software nowadays requires dependency on other software. The OpenSSL library, ubiquitous in any kind of software due to its encryption capabilities (what allows access to websites with SSL/TLS), has 59 dependencies. The Debian OS has approximately 170K pre-compiled packages installed in the system (assuming a graphical installation). The effect of having some of the most popular software libraries depending on hundreds or thousands of other libraries creates an enormous attack surface area, just in the application layer. To counter this, scanners are needed that check for known vulnerabilities in software packages or their dependencies, and a Software Bill of Materials (SBOM) for an accurate, detailed view of all currently used software (the SEI has done extensive research on these topics, see further readings). A breach of an API not only compromises the organization that develops it but also any other organization that depends on it.

Fortunately, many software vendors are making it easy to create an SBOM from a project, such as Gitlab, Github, JFrog, and more. CycloneDX is a popular open-source Extended Software Bill of Material (XBOM) (SBOM, SaaSBOM, ML-BOM, etc.) specification that most vendors support and provides a rich set of metadata to track provenance, licenses, vulnerabilities, relationships and many such attributes. SBOMs are rapidly becoming a staple of an enterprise security strategy, with the army mandating SBOMs on all new software [8].

The best time to scan a software dependency for vulnerabilities is the moment it is first installed. Some organizations scan on a weekly or monthly basis, but in a DevSecOps model, security is performed the moment there is a change –in this case, when it's first installed. Artifact storage software such as Artifactory or Amazon Inspector can scan packages when they are downloaded and run periodic checks to ensure there's no recently discovered exploits. Numerous vendors and solutions for scanners exist ranging from static code analysis to runtime analyzers, container scanners, configuration scanners, secrets scanners, and many more. Trivy, Chekov, and Nessus are a few examples.

Version Control Practices

The concern with scanning for every change suggests we should have a way to track every change. The specific practice is to version control everything: code, configurations, documentation, manifests, anything except secrets and credentials. The point of this is to be able to have a snapshot of the entire system, such that any change introduced into the system that can possibly change its qualities of Confidentiality, Integrity, or Availability can be traced back without going through time-consuming, laborious root-cause analysis. Additionally, if every change made is small, it would make it easier to test, rollback, and reason about. It would truly create a system that grows incrementally, rather than exponentially. Some of the largest open-source projects exemplify these practices, such as the Linux kernel, React, and Kubernetes – these have a standard commit template that must be followed, with subject matter experts reviewing new code to ensure quality is maintained.

API Versioning

We can further engineer assurance into the API by versioning it separately from the system it supports. This is beneficial for its developers, but it has a special relevance for its consumers. If a development team is constantly pushing out changes that affect the set of operations it exposes or the correctness of its response, then the APIs' clients would not consider it very trustworthy. Its assurance would be broken. The range of solutions to this is varied. On one hand, an API can be tightly coupled

to the system, with every change potentially affecting the set of operations it exposes – this requires communication to be sent out to API clients every time, regardless of whether they want the change or not. This is going to alienate clients and cause havoc on their systems. On the other hand, the API development team can go to great lengths not to break the interface or response of the APIs, only adding new endpoints perhaps, but never changing current ones. The C standard library is a good example of an API that does not change much. Clients might like this since it means there's nothing they have to change, but it could severely affect the development of the API to the point that it becomes very expensive to make architectural changes around it or stifle the creation of new features. A good middle point approach has been exemplified by the Stripe API.

Stripe Case Study

Stripe provides an API that powers most of the online commercial transactions in the world. It is valued at $65 billion and processed over $1 trillion in payment volume in 2023. They've been around for almost 15 years and have changed their main API interface 6 times. They talk about their API in-depth in a blog post, so I'll summarize here: Initially, their API was designed for simplicity, famously summarized as "seven lines of code" to handle payments. As they expanded to support ACH, Bitcoin, and various international payment methods, their API become more and more complex, with multiple state machines, endpoints, and flags that developers had to provide. After a while, they recognized the need for a unified and simpler API approach. They put together a "Tiger Team" and they worked it out in a conference room for about a month. This led to the creation of just two endpoints, PaymentIntents and PaymentMethods, which aimed to standardize the integration process for all payment methods while abstracting away the intricacies of each specific method. To ensure these changes were beneficial and user-friendly, Stripe actively engaged with their customers, gathered feedback, and iterated on the API design. They focused on real user integrations and practical application over theoretical perfection. This overhaul reduced complexity significantly, thus lowering the cognitive load on developers. The new system, while initially more complex than the original "seven lines," ultimately provided a more scalable and reliable framework for handling diverse and global payment methods [7].

Conclusions

The lessons here are that APIs should initially be small and focused to what it is trying to accomplish. As use cases increase, communication with clients is essential to determine what changes exactly are desired. If you don't know who your clients are, reach out or lock them out. Changes to the interface should allow for backwards compatibility if that is needed, but more importantly for forward-compatibility so that the API can accommodate future changes.

Testing

In the DevSecOps model, testing is not a one-time activity but a continuous process. Testing provides increased assurance that the API will work as expected. There are various kinds of testing that should be performed as part of an API's Continuous Integration/Deployment pipeline:

- Unit testing individual code expressions
- Integration testing for testing the behavior of components or modules
- End-to-End testing to test the functionality of the entire application

Because of API's unique role in accepting input from n number of clients and passing it over to internal systems, it's resiliency for responding to all different kinds of inputs is especially important. Fuzz testers are an increasingly popular and effective tool that go through a large input space and perform

tests on them. There is a zoo of different open-source fuzz testers available to use, ranging from white-box to black-box testing. Advances in AI are making it easier than ever to create and scale fuzz tests: Google, for example, has successfully experimented with Large Language Models to re-discover or find new bugs for over 1000 projects without having to write additional code [9].

Many organizations use penetration testing to find vulnerabilities in their security systems or break through them. Recently, automated penetration testing tools are providing organizations the capability to conduct penetration exercises at a faster pace with a lower price. Usually, an automated penetration platform can be pointed toward a client network and perform scanning, probing, and analysis continuously without human oversight. The Burp suite is a good example of an automation tool that performs such tasks. Many pipeline vendors such as Gitlab and Github allow you to directly integrate Dynamic Application Security Testing (a type of automated penetration tool) right in your pipeline with minimal setup.

The point of all this testing is that, if we create thorough, rigorous tests for every change introduced in the API, it builds an assurance case for us that the API will work as expected. A change that passes all tests does not guarantee that it is secure or works as expected, but the test suite is an argument in favor that it is or is as close to what it can be based on the threat model established. That threat model can either be explicitly stated in a well-coordinated threat modeling workshop, or implicitly in the mind of a developer or project manager (not ideal!). There are other considerations in testing such as container testing, testing infrastructure, test data, the role of the tester and QA team [10].

Deployment/Operations

Once a change is ready to be deployed, we want to limit the blast radius in case something goes wrong. Canary deployments is a common, effective strategy that deploys to a small subset of the API's clients to test whether the functionality is working as expected. Once users and developers are confident of the release, it gets rolled out to the rest of the client population. Netflix has been a pioneer in the use of canary deployments and has continued to improve on this practice with the result that they can conduct large-scale migrations of their internal systems with minimal to no downtime or disruption to their business [11].

Now let's say something does go wrong. Ideally, we would want our system to recover as quickly as possible. The only way this can happen is if the faulty change gets removed from the system and it returns to its previous state. But this may not be straightforward: it may take hours or days for a change to go through the testing pipeline, release review, management approval, and so on. And even then, if the change is a dependency on other changes that were deployed with it, it makes it complicated to revert. For this reason, we want fast, automated tests that can give us signaling that the change is safe to deploy within minutes or seconds. And we want the change to be atomic so that it does not create complex dependency problems if we ever have to roll it back. Feature flags are a technique that's been extensively used to deal with the problem of deploying small changes with dependencies. At Github, feature flags have been a staple for some time that have successfully allowed them to ship complex features with reduced risk and higher confidence to millions of users [12].

API System Security

System security is defined as all the supporting infrastructure required to enable the API. This includes proxies, load balancers, firewalls, VPNs, network devices, caches, and so on. All of this hardware has security considerations: from vulnerabilities in the software to exploits hidden in the

firmware. It is well known how state actors especially have targeted these low-level devices as access points into systems. Just this year, the Department of Justice disclosed how the “Volt Typhoon” Chinese hacker group targeted Cisco routers and Netgear routers that had reached their end-of-life to target critical U.S. infrastructure [13]. Inventory management and the ability to upgrade software of such devices in case of zero-day vulnerabilities is a good first step towards mitigating these attacks.

Conclusion

The testing and cybersecurity of APIs provide assurance that it is performing as expected. This confidence is born out of continuous running of extensive tests across multiple levels of the system. interface. To maintain the qualities of confidentiality, integrity, availability, and non-repudiation in APIs requires considerable testing for the supporting services which provide these, even under changing conditions; a DevSecOps development model is the best methodology to build APIs with such attributes.

Document Markings

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM24-1005

Further Readings

[1] Woody, Carol. “Applying the SEI SBOM Framework.” SEI Blog, 5 Feb. 2024, insights.sei.cmu.edu/blog/applying-the-sei-sbom-framework.

Citations

[1] “A Brief, Opinionated History of the API.” QCon New York 2018, Youtube. https://www.youtube.com/watch?v=LzMp6uQbmns.

[2] “Cloudflare 2024 API Security and Management Data Report.” Cloudflare, www.cloudflare.com/2024-api-security-management-report/. Accessed 6 Aug. 2024.

[3] “api.data.gov Metrics.” https://api.data.gov/metrics

[4] Cooban, Anna. “Prison. Bankruptcy. Suicide. How a Software Glitch and a Centuries-Old British Company Ruined Lives | CNN Business.” CNN, Cable News Network, 13 Jan. 2024, www.cnn.com/2024/01/13/business/uk-post-office-fujitsu-horizon-scandal/index.html.

[5] “2023 API Security Data in New Report.” Palo Alto Networks, www.paloaltonetworks.com/resources/research/api-security-statistics-report. Accessed 6 Aug. 2024.

[6] Bernhardsson, Erik. “The Half-Life of Code & the Ship of Theseus.” Erik Bernhardsson, Erik Bernhardsson, 19 Apr. 2020, erikbern.com/2016/12/05/the-half-life-of-code.html.

[7] Bu, Michelle. “Stripe’s Payments Apis: The First 10 Years.” Stripe’s Payments APIs: The First 10 Years, 15 Dec. 2020, stripe.com/blog/payment-api-design.

[8] “Assistant Secretary of the Army (Acquisition, Logistics and Technology)

Software Bill of Materials Policy.” Department of the Army, https://federalnewsnetwork.com/wp-content/uploads/2024/09/081624_Army_SBOM_Memo.pdf

[9] Liu, Dongge, et al. “AI-Powered Fuzzing: Breaking the Bug Hunting Barrier.” Google Online Security Blog, 16 Aug. 2023, security.googleblog.com/2023/08/ai-powered-fuzzing-breaking-bug-hunting.html.

[10] “API Security Fundamentals | White Paper.” The API Security Disconnect, www.akamai.com/site/en/documents/white-paper/2024/api-security-fundamentals.pdf. Accessed 6 Aug. 2024.

[11] Blog, Netflix Technology. “Migrating Critical Traffic at Scale With No Downtime — Part 2.” Medium, 20 Oct. 2024, netflixtechblog.com/migrating-critical-traffic-at-scale-with-no-downtime-part-2-4b1c8c7155c1.

[12] Gimeno, Alberto. “How We Ship Code Faster and Safer With Feature Flags - the GitHub Blog.” The GitHub Blog, 27 Apr. 2021, github.blog/engineering/infrastructure/ship-code-faster-safer-feature-flags.

[13] Crouse, Megan. “Botnet Attack Targeted Routers: A Wake-up Call for Securing Remote Employees’ Hardware.” TechRepublic, 9 Feb. 2024, www.techrepublic.com/article/volt-typhoon-botnet-attack/

About the Author

Alejandro Gomez is a software engineer at Carnegie Mellon University's Software Engineering Institute. He has served as a tech lead in multiple Department of Defense projects, bringing technical excellence, bridging communication between management and software teams as well as teaching and mentoring other developers. Prior to joining CMU, Alejandro worked as a software engineer at Vanguard and IBM. He has an MS in Software Engineering from Villanova University and a double Bachelors in English and Economics from the University of Miami, FL. He lives in Pittsburgh with his wife and two children.

Mr. Alejandro Gomez

Software Engineer

Software Engineering Institute

ajgomez@cert.org

Find Us On Our Socials!

309TH SOFTWARE ENGINEERING GROUP

The 309 SWEG is a fun and extremely talented group of engineers, computer scientists, IT, and cybersecurity professionals. This diverse group consists of more than 2000 innovators that are recognized as world leaders in "cradle-to-grave" support systems. They encompass hardware engineering, software engineering, cybersecurity, cloud security, program management, consulting, data management, and much more.

Software and Hardware: Ancient to Modern Times

ELBERT DOCKERY
TECH SQUAD ALUM & EMERGING LEADER,
NATIONAL SECURITY INNOVATION NETWORK

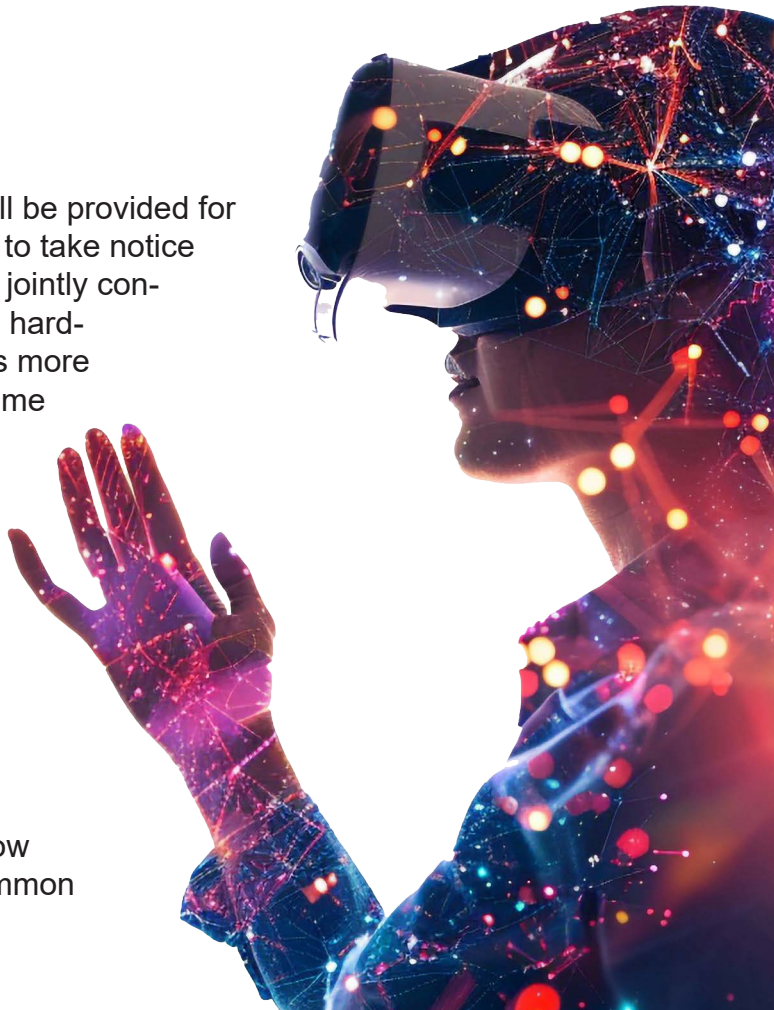
Abstract

This article will take a historical to present timeline on hardware and software, addressing software and hardware advancements throughout the years. The following main points will provide a systematic perspective to unbiasedly arrive at solutions to narrow the gap between software and hardware:

- Education
- Industry
- Media
- Customers

Each of these points will be analyzed and solutions will be provided for the entire electronics (hardware and software) market to take notice of. Everything considered, software and hardware are jointly connected. The software runs on hardware. Without good hardware, good software can't exist. As the world becomes more computerized, both software and hardware have become essential across many industries.

There is a big push for 'modernization' across many industries such as defense, manufacturing, automotives, etc. to adapt modern tools and techniques to improve the workflow of those industries. An example of modernization is the continuous integration/continuous delivery (CI/CD) model that many technology companies have adopted. This is one of many examples that causes people to think software is far ahead of hardware, because of the frequent updates and cycles that software engineers in the industry follow with the software development lifecycle. The most common



modern technique is the agile methodology, which can be used to manage both hardware and software. By the end of this article, readers will understand that software and hardware are interconnected entities that require one another to advance.

History

The histories of software and hardware have been closely related for many years. For example, one of the first developments in software started with punch cards [1]. Holes were shown to indicate specific machine code instructions. Assembly language was one of the very first computer programming languages, which originated in 1949 but is still used today for some low-level programming [2]. But hardware, or all electronics, dates back further. The history of electronics stems back to the late 1800s - specifically in the 1890s - with the invention of vacuum diodes, which was followed by a vacuum triode to amplify electrical signals. The next iterations of this hardware (the tetrode and pentode) improved upon the design and functionality of that time and were used in World War II [3]. Back then, in a short period and due to the available resources at that time, hardware showed great innovation and real-world use. Some people may argue that many things worked just fine before the maturity of software began taking shape in the 2000s.

Before the 2000s, many software applications were web-based or desktop apps. The internet was still in its infancy, so many of the web apps were poorly designed compared to today [4]. Now, we have many software frameworks and tools to help with making these much easier. So much so that some may say we have an excessive amount of software tools that affect how current software is developed [4][5].

In the early 2010s, many software apps transitioned from desktops to mobile due to expanded use of mobile devices [6][7]. The major impact of this stems from how cell phones and tablets changed the way people interacted with their devices and how convenient they were to handle most people's daily tasks. Cell phones and tablets were so impactful that web and mobile apps were written specifically for them. Since the beginning of the 2020s, cell phones and tablets have closed the gap tremendously in terms of performance compared to desktops. To take advantage of all this power that is available now, software must be developed for the new semiconductors and larger memory modules [8]. Many software apps do not take advantage of these advances, resulting in a buggy application that will frustrate the end user.

The changes that have been broken down here have led to more challenges that professionals are currently facing today: challenges in the way we teach and train engineers [8], in the way industry hires and onboards new engineers, in the way the media has affected the perception of engineering, and in the way profit has dominated innovation.

Education

The current education of computer science (CS) is a bit unorthodox because it's still a new field of study compared to other engineering studies. It has gotten better and more streamlined as time has gone on. But, compared to the field of electrical engineering (EE), which has years of extensive research and finetuning, computer science is still in its infancy. In the undergraduate college system, computer engineering (CE) is derived from the two, but more so related to electrical engineering. Depending on the location, most schools group their EE with CE, or their EE with CS. This is a great strategy to have in terms of industry, but there is a bit of fragmentation. This presents problems for students looking to transfer to different schools because there isn't a standard or model for schools to follow regarding teaching these subjects. The accreditation organizations, such as The Accredita-

tion Board for Engineering and Technology (ABET), give some curriculum standards, but not how a university structures its engineering program. The choice of which program is best is often difficult for most students.

CS has even more fragmentation. Some schools have their CS program with the engineering department (CE, EE, or all three together), some have their CS program with the science department, some have their program with the Math department, some have their program with the Information Technology (IT) department, and some have their computer science program as a standalone department. The topic of computer science is so enormous that it's hard to standardize. Understanding of the topic expands more every day, causing more confusion and leaving students to ask questions across online forums and communities (Reddit, Quora, and Stack Overflow) such as, "What is the difference between CS, CE, and EE?" or "How to narrow the choices of CS, CE, and EE?" This fragmentation pigeonholes undergraduate students into one expertise and contributes to the gap in understanding of software and hardware. From a hiring perspective, it also makes it difficult for hiring managers to gauge where a new graduate's skills lie.

System Engineering Education

The next concern many students have when studying electronics (both software and hardware) is the difficulty of math and the complexity of certain topics. Usually, when complexity is involved, there needs to be a way to deal with variables as they scale. Because of this problem, systems engineering should be taught among all undergraduate engineering and computer science majors. This will allow students to see things on a macro level [9]. Taxonomies, ontologies, and topologies are all facets

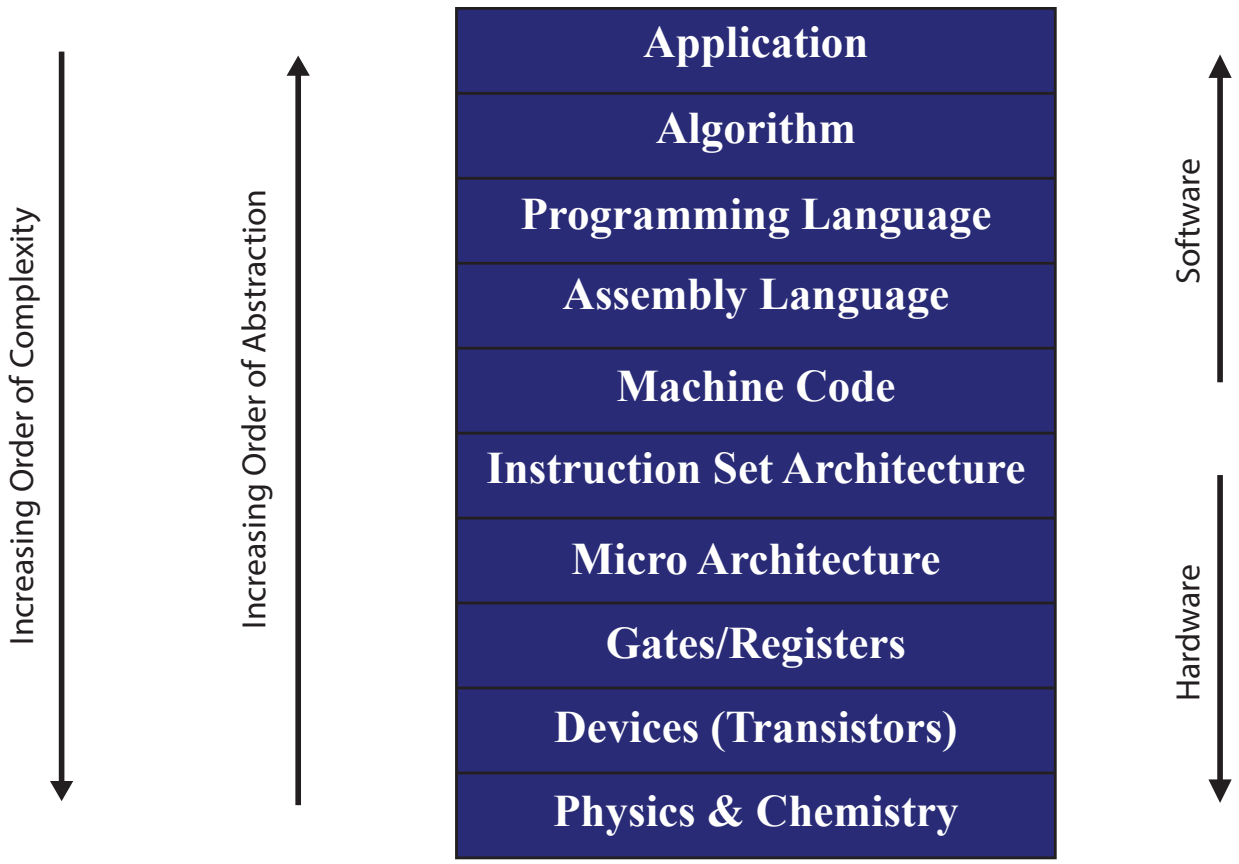


Figure 1. *Computer Systems Layers of Abstraction: The design of how a computer system should be recognized when building solutions of any layer.*

of engineering that need to be involved. As students enter the workforce, they learn how to conduct basic responsibilities to complete the job.

Architecturally, from a systems perspective, it is important to know how the system is functional in decomposition [9]. Every student and engineer who deals with electronics should always remember the computing layers of abstraction because some solutions do not address the problems. It is vital to understand both sides to fully optimize the capabilities of both hardware and software. Adopting a common practice of educating engineers in system engineering will give them the skills necessary to understand both sides.

System Design Education

Students going into software have often struggled with interview questions involving systems design. From a software engineering perspective, there just isn't enough design emphasis taught in schools [10]. This is in direct opposition to hardware engineering (electrical and mechanical, specifically) where there is more emphasis on design. In the book "Engineering and the Mind's Eye," the author discusses the 'lost art' of design and how the education system has gotten away from the importance of design [11]. In software engineering curriculums, there often isn't much design and architecture of software development taught. For this reason, the edge is given to hardware in terms of students coming out of school prepared for a job. The margin for error is very slim when creating products out of physical hardware materials compared to creating them in the digital world [10][12].

From an engineering perspective, other engineers have classified software engineering as a science rather than pure engineering [13]. Most of what is built is on the computer. The computer takes inputs from the software engineer and displays the information as some output after the computation is done. But, because of how software engineers are taught today by schools, bootcamps, and industry, many skills are missing from software engineers' repertoire. Design and architecture are part of the problem. Some engineers eventually learn this concept on the job, in about five to seven years [10] [13]. But this is a major issue and the reason why there has been a trend to ship production code 'half baked' [14]. I've heard it said, "If you can't draw it, then you can't understand it. If you can't understand it, you can't build it."

Soft Skills Education

The next problem is the ability to communicate and write – another set of skills that are overlooked in the study and training of engineering and computer science. Some engineers discount the need to know how to write documentation and communicate designs and solutions to others. This is more of a problem with software engineers because of how undergraduate programs/bootcamps may be structured. Isolating oneself is not how real engineering is done. Engineers usually must write documentation for others and themselves, as well as communicate with other students, co-workers, or customers. Being able to communicate is a very underrated skill that separates a good software engineer from a great software engineer [15]. It would also allow more communication between those who engineer software and those who engineer hardware, thus allowing that connection to narrow the gap between the two.

Education plays a critical role in bridging the gap between disciplines like computer science, computer engineering, and electrical engineering. One of the fundamental components of this bridge is the curriculum. The curriculum is the foundation on which students build their knowledge and skills in these fields. This ensures employers take on minimal risk and gives a person confidence that they can do their job.

While not all schools suffer from these problems, it remains a big issue overall and to address them

there is a need for it to be recognized as an issue in the media and in industry. The media and industry mold the perception of how incoming students and parents see the job outlook.

Industry

In modern times, most systems in the world are run electrically. However, there are tradeoffs and constraints that any electrical system has since software sits on top of an electrical system and needs it to function. This logically means software can only be pushed so far before its capabilities reach their peak [16]. Software functions to make hardware better and to make the user experience enjoyable.

The common reasons people think software is so far ahead of hardware is because of the frequent updates and relatively low overhead of software [8][17]. From a business perspective, it is the reason so many companies have chosen to have a 'software first' approach.

The Modernization of Code

In 2021 and 2022, most job positions were focused on web applications with a sprinkle of mobile applications [18] - iOS and Android (though some preferred a cross-platform framework). Before COVID, most job postings requested experience in Java, Python, and V8 JavaScript. However, what has remained consistent is the high demand for embedded software engineers with knowledge of C and C++ [19].

Although C/C++ are older, 'lower-level languages,' there are too many systems that depend on them. C++ has received updates every 3 years since 'modern' C++ was released. These languages are used heavily across all aerospace and defense industries and are considered the standard in terms of software interfacing with hardware. Most other programming languages that could compete with C and C++ are built off of these languages at their core, just as most operating systems have been built off Unix and Linux (not including the Windows NT Kernel). **Table 1** shows how it has evolved over the years [20].

C++03	C++11	C++14	C++17	C++20
Templates	Multithreading & memory model	Reader-writer locks	Parrell algorithms of the STL	Coroutines
I/O Streams	Keywords: auto and decltype	Generic lambda functions	Structured binding	Modules
STL with containers and algorithms	Std: array		std: any, std: variant	Ranges library

Table 1. *C++ expansion over the years [20].*

Many software engineers focus heavily on the code. That is as important as learning how to think about the overall software architecture. Next, engineers learn how to decompose (frame out the house) user requirements and translate them into some type:

- Libraries and API calls
- Struct and/or Classes
- Functions
- Logic (if statements and loops)

But there is always a new framework, programming language, API, etc. released and marketed to engineers. It's impossible to learn everything in software, but knowing the fundamentals and forming a model is the core of learning anything.

In comparison with hardware, not much has drastically changed since the 1980s [21]. Many electronic devices today are built on:

- Diodes
- Capacitors
- Resistors
- Inductors
- Transistors

Most of the world still runs on 8-bit chips that were used decades ago. In fact, most military aerospace hardware is working fine from decades old technology. The same can be said for many industries.

As the market for software matures, the landscape continues to change. As software is constantly evolving, it is difficult for software professionals to keep up to date on innovations in software, much less innovations in hardware, concurrently.

Technological Changes

Industrial Revolutions	Industry 1.0	Industry 2.0	Industry 3.0	Industry 4.0	Industry 5.0
Description	Water & steam power	Mass production, Electrification	Computers, automated production	Internet of Things, machine learning	Human-robot collab, personalization humanization
Systems	Mechanization	Assembly line	Electronics	Cyber-physical systems	Cognitive Systems

Table 2. *Industrial Revolution timeline 1800s-2020s [22].*

Currently, we are in the early stages of Industry 5.0, the information era, and the deployment of robotic systems [22]. The focus is on Artificial Intelligence (AI), cybersecurity, and the miniaturization of products to become smaller and more computerized (embedded systems). Combining those three things has established the term 'smart' or 'intelligent' systems.

Cloud computing is another term that many in the industry think is modern. The heavy usage of pushing data to the servers of Amazon, Google, and Microsoft (the biggest vendors) is what many in the startup community and larger companies do to cut the costs of having to deal with operating and maintaining their servers [23]. The tradeoff is trusting another company with that data [24]. There have been many data breaches across large companies and, while there are many reasons and benefits of having another company deal with server maintenance, it is creating companies with larger and larger market valuations [23][24]. Since we are in industry 5.0, and data is important, people

should prioritize their data. Data is powerful and whoever has most of the world's data becomes a titan [3][25]. Since there has been a small uptick in the complexity of microservices, there will be another cycle that starts with more companies taking ownership of the data collected [23][26].

Historically, like all things in the world, there are cycles. Offloading data to another vendor while focusing on the other parts of the product or system has been happening since well before the 2000s, when Amazon Web Services (AWS) started to take over the internet. As time went on, the most common distributed control system became programmable logic controllers (PLC). In modern times they are still around but in the form of clusters of computers controlling tasks [23][27].

Changes in Hardware

Electrical Systems

Everything has some type of electrical circuit board and computer component but, as mentioned, there are tradeoffs. One of the flaws of electrical systems is how to deal with heat. Many companies have built cooling systems for dealing with heat dissipation, but the issue is still present as any electrical device can overheat. The next is aging electrical systems that can become faulty from wires being corroded or some electronic storage medium failing for a plethora of reasons [21][28].

Semiconductors

Semiconductors are inside almost everything now and their power has grown tremendously since their first iterations. This Intel table shows the comparison of one of its first chips for the desktop to one released in the 2020s. There is still a lot of untapped potential to show how software can bring out the power of these chips [29].

Processor:	Intel i9-12900	Intel 8086
Instruction Set:	x86-64bit	x86-16bit
Released:	2021	1978
Cores:	16	1
Clock:	3.5 Ghz	5 Mhz
Cache Memory:	19.25 MB	64 KB
Power:	125 watts	1 watt

Table 3. *Intel 8086 vs Intel i9-12900 [30].*

In short these are just some of the factors that address the gap between hardware and software. It highlights the increase of computational power, but that power has not been used properly.

Media

History has shown everyone that many powerful people use media for good and bad. Media of all forms only champions software as king due to some of the reasons explained [10][31][32]. The example used of VCs is the reason why TV shows, movies, social media, etc., push for more people to go into software engineering while neglecting to show the negatives and downsides of being a software engineer, which has contributed to a huge surplus of people going into the industry [32].

With all the praise and hype behind the software, much of the media fails to report common negatives of being a software engineer: the long work hours, bad management, product/project stagnation, poor working environments, unrealistic deadlines, stagnating pay, and variable job security [32][33][34] [35]. The media does not talk about these issues and the engineers themselves won't speak out due to fear of termination and blackballing. The media's failure to highlight hardware and that side of the industry has also contributed to an imbalance in job seekers.

The media's presentation of hardware is borderline nonexistent excluding technical outlets. The spotlight has been on software due to low overhead and the high salaries of software engineers. Due to the lack of manufacturing in America, most of the hardware is imported from foreign countries, which creates its own problems depending on how the product is used. Software is perceived as 'magical' and 'cool' while hardware is seen as the 'old kid on the block' [43].

When problems like these exist, a software and hardware gap is created due to too few members of a team. This can cause products to be delayed among many other internal issues [33][34][35]. A platform can be used to analyze and start some type of change in the industry for engineers to become more comfortable on the job. Companies of all sizes can learn from the data scattered across the internet to retain engineers.

Media outlets should also interview seasoned engineers from various backgrounds. There is a lot of misinformation throughout media and social media that having one central outlet for engineers (both software and hardware) to express themselves and improve the quality of work would better the industry

Customers

The bottom line is that sales mean everything. This expression provides a clear indication of what customers and suppliers want. For example, the first big push for alternative fuel for vehicles happened in the 1980s [36]. There were many problems with using these alternatives: infrastructure, supplier parts, and market maturity were some of the many reasons the American people were not convinced it was better [37]. In the 2020s, electric vehicles (and other alternatives) have begun making people see how they can live with a car that is not powered by gasoline [37][38]. But, although electric vehicles are the most popular cars now, there are many problems with them, such as the build quality and overall expense required to repair them compared to gasoline vehicles [39]. Other variables that indirectly affect this are the infrastructure and rising costs of batteries and other natural resources. Software and firmware have drastically improved the user experience in these vehicles, but many customers have complained about the failure of the electronics and how much it costs to not only repair them (sensors, LCD/LED screens, communications - such as Bluetooth and USB) but program the electronics to the electronic control unit (ECU) [40][41][42].

Overall, the comparison of vehicles from years before now is an example of customers voting with their dollars. People and entities buy things that benefit their lives by saving them time, money, or invoking emotion in the user. This can be said for various innovations electronically, such as the cell phone and virtual reality. The main use of technology should be to solve a problem. People purchase items in the hopes of making something better or more efficient.

There are two examples: the first example is how a company may release some iterations of hardware over a certain number of years. The emphasis is on hardware innovation and, although the software may add features and bug fixes, this company chooses to use hardware as their key innovation. The flaw with this approach is the market maturity that comes with frequent hardware releases, as some customers choose to hold out on their previous hardware because it works fine. But the com-

pany tells customers how that hardware model will lack software updates.

The other example is a company releasing a hardware product every few years or more, relying on software features and fixes to strengthen the product and stretch its lifecycle. The problem with this approach is that some customers may think, based on their needs, there must be more frequent hardware releases because software features can only last so long based on the constraints of its hardware. This may cause customers to choose another competitor. Therefore, electronics success is tied to how customers use it. If it is useful and solves a problem, it will be successful, but there are other variables to consider, which is what cycle the product is currently positioned in the market.

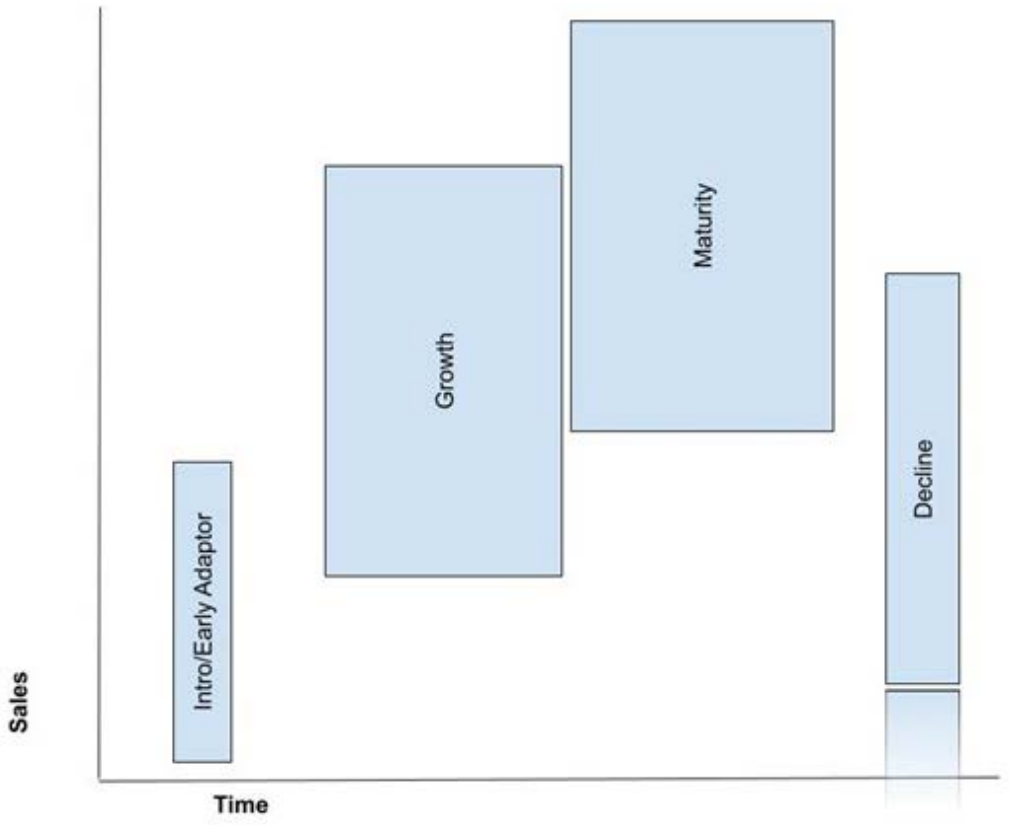


Figure 2. *Product Market Cycle.*

Conclusion

This article only scratches the surface of the generations in and of hardware and software. Other issues, such as the cost of developing hardware vs. software, and new innovations that work to bridge the gap between hardware and software. Although every product and service are different, in terms of business operations. In the short term- hardware is more expensive due to importing and exporting and dealing with the supply chain of suppliers across the world, overhead of multiple iterations of prototype, and testing. But in the long-term software becomes more expensive. This is because of development, support, maintenance cost, licensing, customization and integration. But the lines are becoming blurred now due to many hardware components becoming software programs and more software interfacing and integrating with hardware. A more in-depth version could easily become as large as an encyclopedia series.

The study of how to narrow the gap between software and hardware has been going on for three

decades. The development processes of both are more aligned with each other now. However, just because web and mobile applications have more frequent updates does not mean the software is far ahead of hardware. It is normally because some bug fixes are being addressed. There is a lower margin of error for addressing hardware. Also, due to industry and media, software engineers are given the spotlight compared to other engineers. The physical systems are often overlooked because of the huge influence the Asian market has on exporting hardware to other parts of the world.

Digital systems are only a small subsystem and software should not be isolated from the other parts of the system. That is one of the reasons there is so much poorly written software. Software engineers could not work without the other engineers at the lower levels of abstraction. We all need each other, and innovation comes from both hardware and software. Looking into the past and continuing to innovate off what was done yesterday or centuries ago is what true innovation looks like. Don't fear the past, learn from it.

REFERENCES

[1] Colburn, Robert. "The Surprisingly Long Life of the Punch Card." IEEE Spectrum, 24 July 2021, spectrum.ieee.org/the-surprisingly-long-life-of-the-punch-card.

[2] Roller, Joshua. "Coding From 1849 to 2022: A Guide to the Timeline of Programming Languages." IEEE Computer Society, 11 July 2023, www.computer.org/publications/tech-news/insider-membership-news/timeline-of-programming-languages.

[3] Massachusetts Institute of Technology. (n.d.). Lee DeForest—Triode Amplifier. https://lemelson.mit.edu/resources/lee-deforest#:~:text=Lee%20De%20Forest%20(1873%2D1961,which%20he%20received%20in%201899.

[4] Martinez, Michael. "50 Years of Software." IEEE Computer Society, 18 July 2023, www.computer.org/publications/tech-news/trends/50-years-of-software.

[5] Laato, Samuli, et al. "Trends and Trajectories in the Software Industry: Implications for the Future of Work." Information Systems Frontiers, Apr. 2022, https://doi.org/10.1007/s10796-022-10267-4.

[6] Hristov, Victor. "Apple iPhone History: The Evolution of the Smartphone That Started It All." PhoneArena, 15 Sept. 2023, www.phonearena.com/news/Apple-iPhone-history-evolution-every-model-list_id98169.

[7] "Your Phone Is Now More Powerful Than Your PC." Samsung Business Insights, 22 Dec. 2021, insights.samsung.com/2021/08/19/your-phone-is-now-more-powerful-than-your-pc-3.

[8] Hennessy, John & Patterson, David. Computer Architecture: A Quantitative Approach. 6th ed., Morgan Kaufmann, 2018. ISBN 9780128119068, 0128119063.

[9] Dunbar, Brian. "Systems Engineering Handbook." National Aeronautics and Space Administration. https://www.nasa.gov/seh/2-fundamentals

[10] Cico, Orges, et al. "Exploring the Intersection Between Software Industry and Software Engineering Education - a Systematic Mapping of Software Engineering Trends." Journal of Systems and Software, vol. 172, Feb. 2021, p. 110736. https://doi.org/10.1016/j.jss.2020.110736.

[11] Ferguson, Eugene. Engineering and the Mind's Eye. First MIT Press, 1994. ISBN 9780262560788.

[12] Dym, Clive L. "Design, Systems, and Engineering Education." Harvey Mudd College, Dept of

Engineering, 2004.

[13] Davis, Michael. “Will Software Engineering Ever Be Engineering?” Communications of the ACM, vol. 54, no. 11, Nov. 2011, pp. 32–34. https://doi.org/10.1145/2018396.2018407.

[14] Lyman, Isaac. “Is Software Getting Worse?” Stack Overflow. 25 Dec. 2023, stackoverflow.blog/2023/12/25/is-software-getting-worse.

[15] Galster, Matthias, et al. “What Soft Skills Does the Software Industry *Really* Want? An Exploratory Study of Software Positions in New Zealand.” Association of Computing Machinery, Sept. 2022, pp. 272–282. https://doi.org/10.1145/3544902.3546247.

[16] Bailey, Brian. “Why Hardware-Dependent Software Is So Critical.” Semiconductor Engineering, 23 June 2022, semiengineering.com/why-hardware-dependent-software-is-so-critical.

[17] Proven, Liam. “New Software Sells New Hardware – but a Threat to That Symbiosis Is Coming.” The A Register, 10 Jan. 2023, www.theregister.com/2023/01/11/software_versus_hardware.

[18] “Occupational Employment and Wages.” Bureau of Labor Statistics, May 2022. https://www.bls.gov/oes/current/oes151252.htm#st

[19] “Occupational Outlook for Computer Engineers.” Bureau of Labor Statistics. https://www.bls.gov/ooh/architecture-and-engineering/computer-hardware-engineers.htm#tab-6

[20] Stroustrup, Bjarne. A Tour of C++, 3rd ed., Addison-Wesley Professional, 2022. ISBN 9780136823575.

[21] ARRL Handbook for Radio communications: The comprehensive RF Engineering Reference, 92nd ed., The American Radio Relay League, Inc., 2016.

[22] Xu, Xun, et al. “Industry 4.0 and Industry 5.0—Inception, Conception and Perception.” Journal of Manufacturing Systems, vol. 61, Oct. 2021, pp. 530–35. https://doi.org/10.1016/j.jmsy.2021.10.006.

[23] Surbiryala, Jayachander, and Chunming Rong. “Cloud Computing: History and Overview.” IEEE Xplore, Aug. 2019, https://doi.org/10.1109/cloudsummit47114.2019.00007.

[24] Al Morsy, Mohamed, et al. An Analysis of the Cloud Computing Security Problem. Cornel University, 2010. https://doi.org/10.48550/arXiv.1609.01107.

[25] Adel, Amr. “Future of Industry 5.0 in Society: Human-centric Solutions, Challenges and Prospective Research Areas.” Journal of Cloud Computing, vol. 11, no. 1, Sept. 2022, https://doi.org/10.1186/s13677-022-00314-5.

[26] Leng, Jiewu, et al. “Industry 5.0: Prospect and Retrospect.” Journal of Manufacturing Systems, vol. 65, Oct. 2022, pp. 279–95. https://doi.org/10.1016/j.jmsy.2022.09.017.

[27] Lamb, Frank. Advanced PLC Hardware & Programming: Hardware and Software Basics, Advanced Techniques & Allen-Bradley and Siemens Platforms. Automation Consulting, LLC, 2019. ISBN-13978-0578482231.

[28] Avallone, Eugene, et al. Marks’ Standard Handbook for Mechanical Engineers, 11th Ed., McGraw-Hill Professional Pub, 2006. ISBN-100071428674.

[29] Heyman, Karen. “Rethinking Engineering Education in the U.S.” Semiconductor Engineering, 17 Jan. 2024, semiengineering.com/rethinking-engineering-education-in-the-u-s.

[30] Explore Intel’s History. timeline.intel.com.

[31] Wang, Y C. “Early Education Can Be the Key to Recruit the Next Generation Workforce.” Electronics Weekly, 2 Mar. 2023, www.electronicsweekly.com/news/early-education-can-be-the-key-to-recruit-the-next-generation-workforce-2023-02.

[32] Alpaio, Kelsey. “Career Crush: What Is It Like to Be a Software Engineer?” Harvard Business Review, 22 Feb. 2022, hbr.org/2021/07/career-crush-what-is-it-like-to-be-a-software-engineer.

[33] Chae, Daniel. “The Pros and Cons of Being a Software Engineer at a BIG Tech Company.” Stack Overflow. 17 Feb. 2021, stackoverflow.blog/2021/02/17/the-pros-and-cons-of-being-a-software-engineer-at-a-big-tech-company.

[34] Cser, Tamas. “The Cost of Finding Bugs Later in the SDLC.” Functionize. 13 Feb. 2023, www.functionize.com/blog/the-cost-of-finding-bugs-later-in-the-sdlc.

[35] Toporek, Jared. “The Hardest Part of Building Software Is Not Coding, It’s Requirements.” Stack Overflow. 29 Dec. 2023, stackoverflow.blog/2023/12/29/the-hardest-part-of-building-software-is-not-coding-its-requirements.

[36] Muratori, Matteo, et al. “The Rise of Electric Vehicles—2020 Status and Future Expectations.” Progress in Energy, vol. 3, no. 2, Mar. 2021, p. 022002. https://doi.org/10.1088/2516-1083/abe0ad.

[37] Adderly, Shawn A., et al. “Electric Vehicles and Natural Disaster Policy Implications.” Energy Policy, vol. 112, Jan. 2018, pp. 437–48. https://doi.org/10.1016/j.enpol.2017.09.030.

[38] Acharya, Samrat, et al. “Cybersecurity of Smart Electric Vehicle Charging: A Power Grid Perspective.” IEEE Access, vol. 8, Jan. 2020, pp. 214434–53. https://doi.org/10.1109/access.2020.3041074.

[39] Gitlin, Jonathan M. “EVs Have 79% More Reliability Problems Than Gas Cars, Says Consumer Reports.” Ars Technica, 29 Nov. 2023, arstechnica.com/cars/2023/11/evs-have-79-more-reliability-problems-than-gas-cars-says-consumer-reports.

[40] Bosch, Robert. Automotive Handbook, 11th ed., Wiley, 2022. ISBN 978-1-119-91191-3

[41] Fraden, Jacob. Handbook of Modern Sensors: Physics, Designs, and Applications, 5th ed., Springer, 2016.

[42] Scherz, Paul, & Monk, Simon. Practical Electronics for Inventors, 4th ed., McGraw Hill, 2016. ISBN-13 978-1259587542.

[43] SpaceX’s newest employee is a 14-year-old prodigy from California - CBS News

About the Author

Elbert Dockery is a software engineer with a wide variety of skills, ranging from front end development, backend development, to embedded systems. He has a strong passion for embedded software development with an interest in leading a startup to become a sub and prime contractor to the DoD as well as NASA. Elbert has been labelled as a ‘DOD innovator’ by NSIN/DoD.

Elbert Dockery

Software Engineer

Department of Defense

https://www.linkedin.com/in/elbert-d-4b00b555/

Department of Defense

Weapon Systems Software Summit

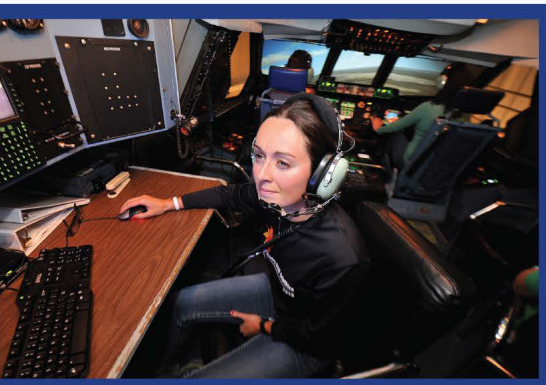
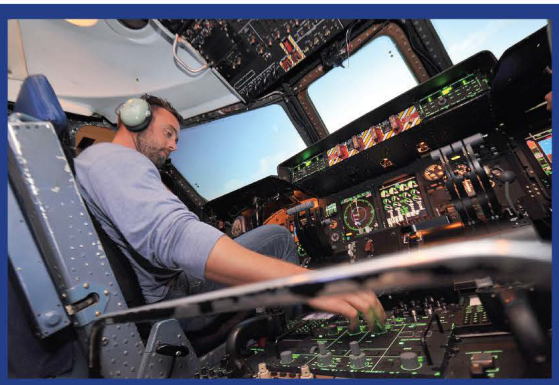
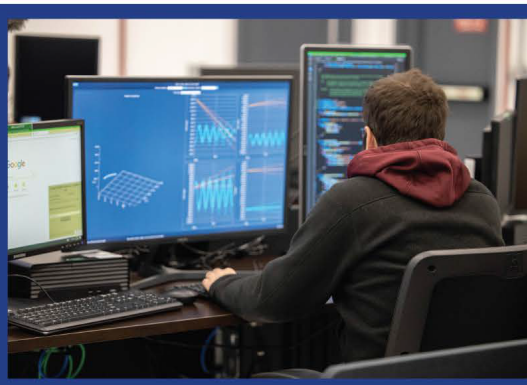
DECEMBER 10-13, 2024
SALT PALACE CONVENTION CENTER
SALT LAKE CITY, UT

SOFTWARE SUMMIT PROGRAM

The 3rd annual DoD Weapon Systems Software Summit stakeholders including the defense industrial base, academia, Office of the Secretary of Defense, and the Services will share solutions to software issues in the tracks and panels below:

- DevSecOps Ecosystems
- Software Assurance
- Software Modernization
- Securing the Embedded Software Supply Chain
- Secure Software by Design
- OSD and Services Policy for Weapon Systems Software
- AI/ML – Does It Apply to Weapon Systems
- Agile Coaches Corner Panel
- Workforce Innovations Panel
- PMOS Working with Organic Software Teams Panel

Program details for the Software Summit will be available soon at the QR code below.
Book your hotel now at the QR code below.
Register using the code **24SFTWR** for the Software Summit discount.

Contact us: AFSC.DoD.WS_SWSummit@us.af.mil

SAE.ORG/DOD

COMMON CYBER RESILIENT OPERATING ENVIRONMENT (CCROE)

Linda Wright
Computer Scientist,
Naval Surface Warfare Center Dahlgren Division
Dam Neck Activity

Abstract

Cyber resiliency is a key component to all Navy software applications. No matter how large or small the program is, the code must be secure. This creates a significant hurdle for small and low-budget programs. The Common Cyber Resilient Operating Environment (CCROE) supports both the large Program of Record (PoR) systems, as well as the small developmental projects. CCROE is currently supporting sixteen projects across the Navy. The development of CCROE is modular and allows for multiple configurations, reusable to allow multiple PoRs to leverage, cyber resilient to embed cyber security, designed to support Commercial of the Shelf (COTS) products, and Linux based to support current PoR efforts. This article provides a high-level overview of CCROE, the cyber resiliency process, and the benefits to the Navy.

Background

Cybersecurity is an ever-present issue that is continually evolving. The Naval Surface Warfare Center, Dahlgren Division, Dam Neck Activity (NSWCDD DNA) approach to hardening a combat system's Operating Environment (OE) must evolve with it. The NSWCDD DNA OE team has shown continued success in developing OEs. Implementing a *common Navy surface ship OE* reduces risks of critical infrastructure, provides benefits of increased security posture, and promotes better usage of project resources. CCROE increases customers and stakeholders' trust, provides a competitive advantage, and establishes cost savings.

In GitLab, we create a fork of Red Hat Enterprise Linux (RHEL) operating systems (OS) that incorporates Defense Information Security Agency (DISA) application Security Technical Implementation Guides (STIGs), National Security Agency (NSA) Controlled Access Protection Profile (CAPP) recommended configurations, and National Institute of Standards and Technology (NIST) Common Configured Enumerations (CCEs) standards and guidelines. "A fork is a new repository that shares code and visibility settings with the 'upstream' repository" [1]. By hardening the underlying OS, other cyber technologies and combat systems can build upon our stable foundation. This technology is especially

useful for new systems that may not have the budget in the early development phase.

Introduction

What is Cyber Resilience?

Cyber resilience, or cyber resiliency, is “the ability to weather adverse events in a computing environment. It applies to both physical and virtual assets” [2]. Cyber resiliency is a key component to all Navy software applications. No matter how large or small the program is, the code must be secure. This security measure creates a significant hurdle for small and low-budget programs.

The NIST Special Publication (NIST SP) defines cyber resilience as “the capability to anticipate, withstand, recover from, and adapt to adverse conditions, stresses, attacks, or compromises on systems that use or are enabled by cyber resources” [2] [3]. “Cyber resilience is a concept that brings business continuity, information systems security, and organizational resilience together” [4]. Cyber resiliency is “intended to enable mission or business objectives that depend on cyber resources to be achieved in a contested cyber environment” [5].

What Constitutes Cyber Resilience?

Figure 1 shows the U.S. Department of Commerce NIST Cybersecurity Framework (CSF) functions that aid organizations in expressing the management of cybersecurity risk at a high level and enables risk management decisions. The CSF 2.0 six functions are Govern, Identify, Protect, Detect, Respond, and Recover. These functions provide a comprehensive view of the lifecycle for managing cybersecurity risks over time [6].

Why is Cyber Resilience Even More Critical Now?

With the continued widespread business disruptions, “organizations are looking for technology with solutions to provide secure, adaptable, engaging, and trusted experiences for their employees, customers, and partners” [7]. An ultimate goal is to have systems that are highly automated, distributed, and ready for any mission.

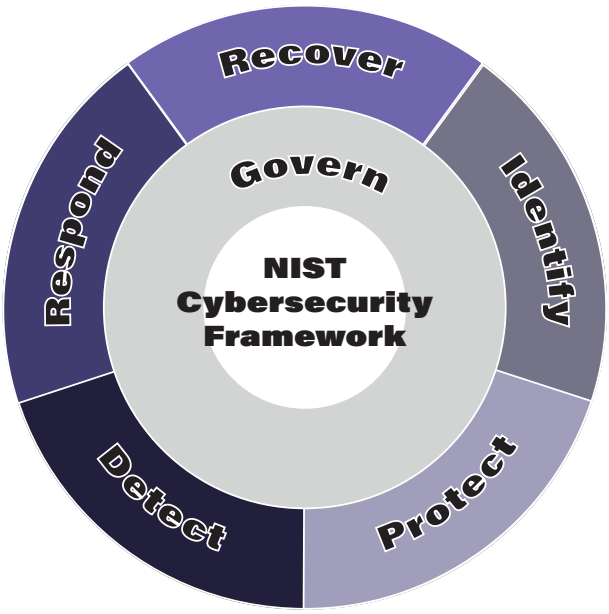


Figure 1. *CSF Functions [6].*

“Cyber resilience unites IT cybersecurity with business continuity and overall organizational durability. When properly deployed, the concept leads to the ability to continue routine operations when facing cyberattacks, as well as natural disasters, economic downturns, and various other crises” [8].

No matter how robust a cybersecurity program operates, considering the occurrence of a successful cyberattack at any moment means that organizations should concentrate on resilience comparatively to prevention [8]. “While cybersecurity focuses on preventing cyber threats, cyber resilience includes preparation, response, and recovery” with the acknowledgment that cyber incidents could be evident regardless of the best efforts of total prevention [8]. “Given the realities of today’s cyber threats, and tomorrow’s potential new technology-generated attack vectors, we really have no choice but to build resilience in from the beginning” [9].

Components of Cyber Resilience

The essential components of cyber resilience are identifying which services are most critical, what plans need to be in place to make sure those services continue, what staff training might be required to enhance cyber resilience, and other key aspects of planning for adverse events [7]. Cybersecurity is essential to a cyber resilience strategy. “Cisco research found that 97% of organizations in America had made changes to their cybersecurity policies—specifically to support remote working—since the start of the COVID global health crisis” [7].

What is System Hardening?

The NIST defines system hardening as “a process intended to eliminate a means of attack by patching vulnerabilities and turning off nonessential services” [10] [11]. “Implementing robust security measures without addressing system vulnerabilities and nonessential components is like installing a security system that only protects the primary entrance of a facility” [11]. Avoiding attacks remediates unplanned downtime. “The main goal of system hardening is to improve overall IT security. This lowers your risk for data breaches, unauthorized access, and malware injection” [12]. Attackers will have fewer opportunities to gain access to a mission-critical or critical-infrastructure computer system’s sensitive information [13].

Types of System Hardening	System Hardening Description
Application	Software hardening prevention by protecting your applications from fraud techniques like unauthorized tampering and reverse engineering.
Operating System	Enabling or adding security features to your OS to make it more secure.
Server	Securing the data, ports, components, functions, and permissions of a server using advanced security measures at the hardware, firmware, and software layers.
Endpoint	Securing and protecting end-user devices from potential vulnerabilities and malicious attacks.
Database	Securing both the contents of a digital database and the database management system (DBMS), which is the database application users interact with to store and analyze information within a database.
Network	Securing the basic communication infrastructure of multiple servers and computer systems operating within a given network.

Table 1. *Types of system hardening [14].*

Types of System Hardening

Table 1 describes several types of system hardening. Operating system hardening is the topic in this article and it involves patching and implementing advanced security measures to secure a server's OE. The Operating Environment (OE) as a Target for Hardening

An OE is a commercially available OS with a selection of software and configurations to meet users' needs, with unnecessary software removed, and hardened with security included from the start. By hardening the OE, you ensure that each system built from the OE has a common baseline level of security. *Systems are born secure.* The attack surface is minimized by removing unnecessary software.

Security will be a feature of development from day one, *not bolted-on* at the end. There will be less time allocated to setting and checking the basics. The OE's package manager, and its robust infrastructure, can help to deploy hardened applications.

CCROE Development

The CCRO process uses the package management infrastructure to provide cyber resilient concepts outlined in the Introduction section. CCROE development is modular, reusable, and cyber resilient.

Package Management: Tool for Application Hardening

A package manager is part of the OS that manages installed software. In the RHEL OS, the package manager is Yellowdog Updater Modified (YUM). The command-line interface "yum", operates on packages called RPMs. **Table 2** describes package management examples using *rpm* and *yum*. In the example, the RPM (and packages such as RPMs) is built. It provides all information to the package manager along with files defining the software itself, and scripts governing installation upgrade and removal. In each interaction, the package manager ensures that all packages are compatible. The package manager does *all of this so the user does not have to manage the details*.

If an application is hardened *when made into a package*, then the user can use the package manager to deploy *already-hardened applications*. By hardening the underlying operating system, we created a stable foundation that other cyber technologies and combat systems can build upon. This technology

RPM Instance	RPM Command
Query software is present	$ rpm -qa
Show version, vendor, and other information about installed software	$ rpm –qi firefox
Ensure software is signed by a trusted source and unmodified	$ rpm --verify firefox
Install software packages from local or remote	$ yum install firefox
Upgrade software packages from local or remote	$ yum upgrade firefox
Remove software packages from local or remote	$ yum erase firefox

Table 2. *Package management examples.*

is especially useful for new systems that may not have the budget in the early phases of development where cyber hygiene is crucial. Below is a *lightweight* example (i.e., the developer performs less work):

- Take DISA STIGs for Google Chrome
- Write a settings file that implements the policies required by the STIGs
    - Example: Disable the setting that allows search suggestions
- Create a RPM package to install that file
- The RPM ensures only administrators can edit the file
- Deploy that RPM in repositories so new and existing machines can install the RPM

A *heavyweight* example (i.e., the developer performs more manual work):

- Take Firefox source, made by Mozilla to meet the security needs of the public
- Take DISA STIGs for Firefox that require certain settings
- Recompile Firefox to always follow the STIG-required settings
- Package that version of Firefox as a RPM and deploy it in repositories, as previously

When a STIG says to do something, the application gets packaged so that it already does the action; therefore, less time is spent checking upon it. In the Chrome example, one add-on per harden RPM covers 40 STIG requirements. This is important when the platform is afloat and one cannot personally administer it.

Repackaging software to be pre-hardened comes at a cost:

- Time is taken to learn the process and perform the repackaging
- Firefox in particular may take hours to compile
- When the original vendor releases an update, now their version is newer, and may replace the hardened package, until a new hardened edition is packaged.
- Some security features cannot be packaged (e.g., hard drive partitioning)

The benefits of repackaging are:

- Applications, and entire systems, installed from the hardened repository are born with those security features present
- Applies to isolated, field-installed assets that will never have access to a centralized management server
- Allows many users to draw from those repositories

Verification and Validation CCROE

Verification and validation of our CCROE products is part of the development effort, because the OEs will be deployed to sites without centralized management. Our team ensures the software and infrastructure is compliant with DISA STIGs in addition to executing other test cases derived from customer needs. It must also be ensured that hardware platforms remain reliable during upgrades and replacements over time. Product updates and new features are delivered and tested as often as the stakeholder requests.

Benefits of a Common OE

A common OE is available to multiple users within a project, and to multiple projects (e.g., tactical systems, training systems, maintenance systems, developer workstations, etc.). If multiple projects can use a common OE, then there is less time taken re-implementing the basics. A modular approach

helps to strike a balance between unique needs and a shared implementation. Below are examples:

- A common, modular OE makes a hardened version of Firefox available; however, a project could choose to use a different version, or omit it altogether.
- A maintenance system can take the common OE, remove the graphical environment, and add network management tools.
- A developer workstation can take the common OE and add compilers and editors. Package manager is an important part of making the OE modular.
- Projects sharing an OE also share knowledge about how to utilize and improve the OE.
- Rapid recovery from incidents assists in the ability to deploy servers that are secure instantaneous of the installation.

Developing a Common OE

The CCROE is based on RHEL. The hardening process for the OE adds in missing Department of Defense required elements such as the DISA RHEL STIGs, DISA application STIGs (e.g., Apache, MySQL, Firefox, etc.), NSAs Controlled Access Protection Profile (CAPP), and NIST CCEs.

CCROE is comprised of four components: hardened RPMs, curated groups, common tools, and Kickstart profiles. The RPM Packaging Guide is referenced for novice users to create RPM packages [15]. In Red Hat, the essential inputs of a common OE are:

- The commercially available Red Hat baseline OS (e.g., RHEL 7.9) as an '.ISO file'.
- Repositories of RPM files. (The RPMs are modified by updating existing open-source RPMs with DISA STIGs and industry standards).
    - Includes the pre-hardened packages.
    - Includes anything else users need, since they may be unable to reach public repositories.
    - Curate group: At this stage, groups can be created to help with organizing the package.
- Kickstart files that govern the overall installation process. (Kickstart profiles are used to minimize installation efforts and ensure proper system configuration). Each type of user, or even each particular server, may have a customized Kickstart file. This specifies which packages to install from the provided repositories. This may also specify hard drive partitioning, hostname, Internet Protocol (IP) address, etc. Those settings may provide security features that are not possible to package, like disk partitioning. A fully automated Kickstart file can reduce the installation procedure to a single button press.
- A Makefile that defines the process of building the OE from those parts.

CCROE High-Level Flow

NSWCDD DNA developed CCROE to be modular (to allow for multiple configurations), reusable (to allow multiple PoR to leverage), cyber resilient (to imbed cybersecurity), designed to support Commercial of the Shelf (COTS) products, and Linux based (to support current PoR efforts). **Figure 2** (below) depicts the common hardened OE cycle.

CCROE development includes two primary goals. First, it must secure the software application prior to installation and comply with current STIGs, Security Content Automation Protocol (SCAP) Security Guides, (SSGs), and industry best practices. Second, it must minimize the IA footprint. It does so by starting development with the minimum set of packages required, rather than by removing packages at the end of the development process. CCROE development includes containerization and incorporates Continuous Integration and Continuous Deployment (CI/CD).

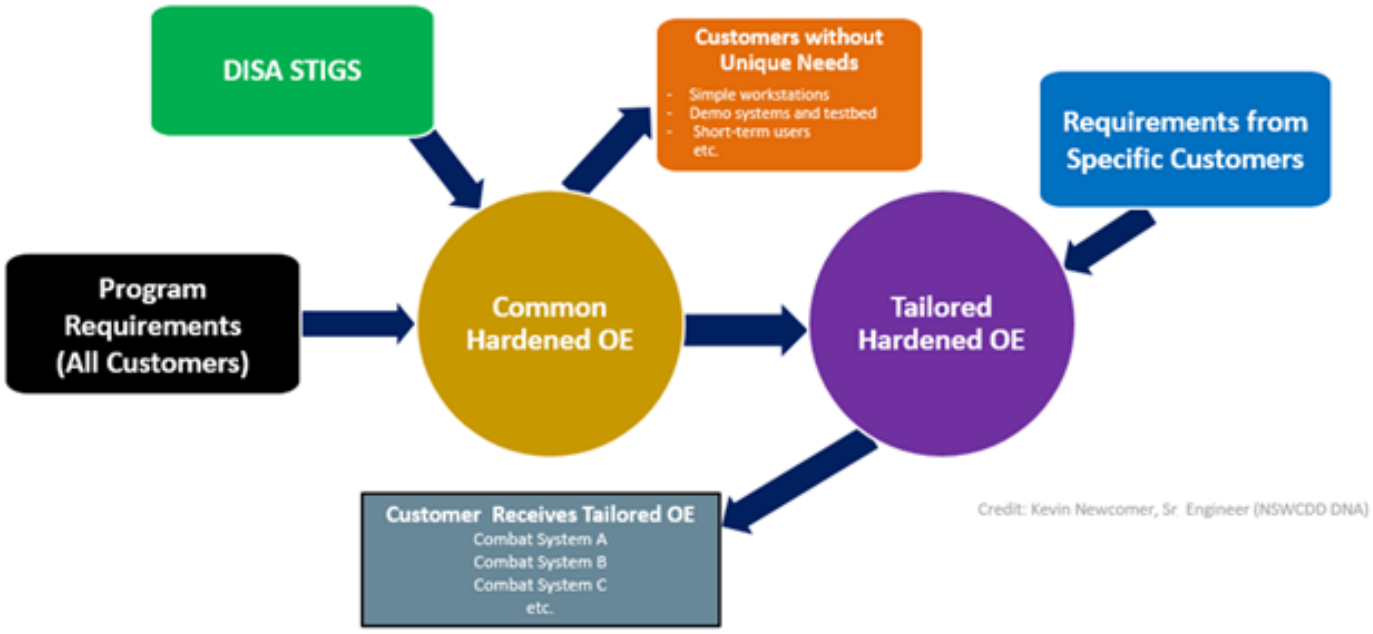


Figure 2. *Common hardened OE cycle.*

The process to create a common OE starts with a set of generic **requirements** that are not specific to any consumer:

- OS baseline (e.g., RHEL 7.9)
- What package suites to make available (e.g., the GNU Network Object Model Environment (GNOME) graphical environment, Firefox, vim)
- What packages to remove from a commercial baseline (e.g., remove wireless printer drivers)
- Basic hard drive partitioning

DISA STIGs, the security guidelines from DISA, are incorporated into the common OE from the ground up:

- Repository of RPMs with patches built-in
- Requirements for hard drive partitioning
- Hard disk encryption
- Initial firewall setup
- Scripted application of patches not covered by hardened RPMs

As more security features and basic setup tasks are implemented in the common OE, less manual work is required after the installation process. For some use cases, that common OE, without further customization, may be sufficient.

- The common OE will include the resources needed to install a stand-alone machine.
- That process may require user interaction to customize the installation, or it could proceed purely by script.

Situations where the common OE may suffice include:

- Simple workstations
- Testbeds for testing, development, and demonstrations
- Anytime you need *something* working in a hurry, while still having security and commonality

Looking at **customer-specific requirements**, you may decide to make certain customer-requested features part of the common OE.

For **other users**, customizing that common OE will meet their needs. They can:

- Choose which packages to install from the hardened repositories
- Provide additional packages to install
- Override the hard disk partitioning and other settings from the common OE
- Provide specific user accounts and network settings
- Set up services such as NTP

The process of **tailoring** the OE to a specific need may then feed back to the set of customer requirements. For instance, the possibility of needing new dependencies.

Once the customer has received a secure and tailored OE, their setup process has been simplified, since they no longer need to provide all of the security and customization themselves; those features are present in the OEs from the moment they are installed. Importantly, this also applies to the customer's response to cyber incidents: servers can be reinstalled as needed, and will have a baseline level of security immediately.

A Hardened and Common OE

For more than 10 years, NSWCDD DNA has been developing Operating Environments to protect the Navy fleet. The CCROE development supports both the large PoR systems as well as the small developmental projects. CCROE is currently supporting sixteen projects across the Navy (e.g., Power Temperature Controller (PTC), Maintenance Toolkit (MTK), Portable Maintenance Aid (PMA), and Environmental aGregation GUI (EGG) OEs).

Having hardened applications and OEs mean that systems are born in a secure state. Administrators spend less time implementing and checking the basics. Security will be a feature of development from day one, not bolted-on at the end. An isolated system without a centralized management server will start out secure. By using secure packages, a layer of security provides and verifies the existing, robust infrastructure of the package manager.

A common, modular OE allows its users to focus on their unique needs, rather than re-implementing the basics. Application developers can focus on the application. This also avoids the reimplementation of the security provided by a hardened OE. Knowledge about the OE, and contributions to improve it, come from a wider user base than for an in-house OE. This allows for long-term stability of OE knowledge, if application development shifts from one contractor to another.

A modular approach means that when certain users must opt out of security features for mission reasons, that opt-out done in a narrow scope is not by all users. **The common theme in all of these is cost avoidance by re-use of the OE and its security.**

Acknowledgment

It is a pleasure to acknowledge Mr. Emilio Digioia, for the assignment; Dr. Vernon "Dale" Bloodgood, abstract paper; Mr. Kevin Newcomer, for reviewing and contributing; and NSWCDD DNA reviewers.

References

[1] "Fork a repository: About forks." GitHub Docs, GitHub, Inc., 2024, docs.github.com/en/pull-requests/collaborating-with-pull-requests/working-with-forks/fork-a-repo.

[2] Pacific Northwest National Laboratory (PNNL). "Cyber Resilience." U.S. Department of Energy. www.pnnl.gov/explainer-articles/cyber-resilience.

[3] Ron Ross, et.al. National Institute of Standards and Technology (NIST) Computer Security

Resource Center. “Developing Cyber-Resilient Systems: A Systems Security Engineering Approach,” National Institute of Standards and Technology Special Publication (NIST SP) 800-160 Vol. 2, Rev. 1, U.S. Department of Commerce, December 2021, csrc.nist.gov/pubs/sp/800/160/v2/r1/final.

[4] “What is cyber resilience: Cyber resilience defined,” IBM, www.ibm.com/topics/cyber-resilience.

[5] “Glossary: cyber resiliency.” National Institute of Standards and Technology (NIST) Computer Security Resource Center (CSRC). NIST SP 800-172A, U.S. Department of Commerce, February 26, 2024, csrc.nist.gov/glossary/term/cyber_resiliency.

[6] National Institute of Standards and Technology. “NIST Cybersecurity Framework 2.0: Resource & Overview Guide.” NIST Special Publication 1299, NIST CSF 2.0, U. S. Department of Commerce, February 2024, doi.org/10.6028/NIST.SP.1299.

[7] “What is Cyber Resilience?” Hybrid Work Solutions, Cisco, www.cisco.com/c/en/us/solutions/hybrid-work/what-is-cyber-resilience.html.

[8] Edwards, John. “Why Cyber Resilience May Be More Important Than Cybersecurity,” Information Week, February 29, 2024, informationweek.com/cyber-resilience/why-cyber-resilience-may-be-more-important-than-cybersecurity.

[9] Pearlson, Keri. Cybersecurity And Digital Privacy: “When Cyberattacks Are Inevitable, Focus on Cyber Resilience.” Harvard Business Review, July 18, 2024, hbr.org/2024/07/when-cyberattacks-are-inevitable-focus-on-cyber-resilience

[10] Elaine Barker et al. National Institute of Standards and Technology. “A Profile for U.S. Federal Cryptographic Key Management Systems.” NIST Special Publication 800-152, U.S. Department of Commerce, October 2015, nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.800-152.pdf.

[11] RSI Security. “What are System Hardening Standards?” RSI, 27 Apr 2022, https://blog.rsisecurity.com/what-are-system-hardening-standards

[12] “What Is System Hardening?” System Hardening, https://www.intel.com/content/www/us/en/business/enterprisecomputers/resources/system-hardening.html.

[13] Brett, Daniel. “System Hardening: An Easy-to-Understand Overview.” Blogs by Trenton Systems, 14 Apr 2021, trentonsystems.com/blog/system-hardening-overview.

[14] Schrader, Dirk. “What is Systems Hardening?” Cybersecurity, Net-

wrix Blog, 22 Feb 2023, updated 17 March 2023, https://blog.netwrix.com/2023/02/22/system-hardening.

[15] Adam Miller et al. “RPM Packaging Guide.” rpm-packaging-guide.github.io.

About the Author

Linda Wright, a Computer Scientist, most recently, led Ship Self-Defense System (SSDS) software engineering team at NSWCDD DNA. Linda has applied her 20+ years of expertise in software and systems engineering lifecycles as a civilian and contractor to solve mission and safety critical challenges within various federal and defense agencies. Linda received a bachelor’s degree in computer science. She is an International Council on Systems Engineering (INCOSE) Certified Systems Engineering Professional (CSEP).

Linda Wright

Computer Scientist

Naval Surface Warfare Center

Dam Neck Activity

linda.c.wright20.civ@us.navy.mil

The Dragon's Hoard

Destinie Comeau
CrossTalk Publications Manager,
AFSC Software Directorate

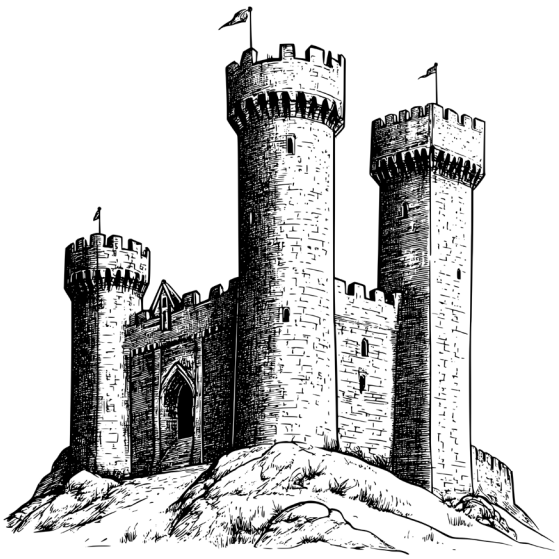
Hear ye, hear ye! Gather 'round to learn about the newest creature spotted in the land!

Rumor has it that a dragon has been spotted cavorting around the outskirts of the kingdom. Those who have survived an encounter with the beast return calling it [Unknown Source] … whatever that means.

Villages along the border have been complaining about its frightening appearance and potential to do harm. Many fear the dragon stealing their personal information, as its hoard seems to consist of 'data' rather than the typical gold and jewels. Guards posted in towns near where the creature has been spotted admit that the dragon has shown great intelligence and versatility in its attacks, proving its ability to manipulate ideas with bias and misinformation. They warn that the kingdom's security is in jeopardy! Even townspeople who have not seen the dragon in person are concerned about what kind of damage the beast could cause with the knowledge stored in its hoard!

While many an adventurer has attempted to tame the beast, none have come out entirely successful. Some return with stories about the creature, claiming to have studied it and learned all there is to know, but others are never seen again. If the danger remains, are we ever truly safe?

The Software Engineering Guild has voiced concern about the resources the dragon uses to maintain its hoard. While some of their members attempt to build a cage that may contain [Unknown Source] and its malicious data, others have taken to ignoring the problem entirely. After all, what cage could hope to hold such a fearsome force? And if the beast were to break out, what more damage could it do when angered?



On the other hand, some mages have found solace in the dragon's company. [Unknown Source] is, apparently, very knowledgeable about a range of subjects and willing to grant pieces of its hoard to those who prove their worth. However, not all mages looking for this gift have the kingdom's best interests at heart. Reports of data ending in the hands of malicious actors remain high in number. The current recommendation for any who wish to work with the dragon

is to tread carefully and double-check any information received. While the assumption isn't that the dragon lies maliciously, some of its vast knowledge is out of date.

Whether the majority opinion on the dragon and its hoard is good or bad, the entirety of the kingdom has called on the land's monarch to send knights into the dragon's territory. While some citizens say we simply need more information about the creature, others have called for new laws regarding how to deal with the beast and the information it guards. Either way, the involvement of the crown and its shining pillars is necessary as we move onto next steps in dealing with [Unknown Source] and the data it keeps.

About the Author

Destinie Comeau works as a publications manager for CrossTalk: The Journal of Defense Software Engineering. Before CrossTalk, she worked as a Tier 1 Account Manager at Pinterest. Ms. Comeau is currently working toward an MFA in Writing through Lindenwood University. She has her bachelor's degree in creative writing from Weber State University.

Destinie Comeau

CrossTalk Publications Mananger

AFSC Software Directorate

destinie.comeau@us.af.mil

https://www.linkedin.com/in/destinie-comeau-b2b838183/

With Special Thanks to Siria Snounou