COMPUTING edge

- Hardware Security
- AI and Machine Learning and Software Testing
- Education
- History

MARCH 2026

www.computer.org

IEEE COMPUTER SOCIETY

80th ANNIVERSARY CELEBRATION

IEEE

IEEE

IEEE COMPUTER SOCIETY

80th ANNIVERSARY CELEBRATION

IEEE Computer Society

Grants for

EMERGING TECHNOLOGY ACTIVITIES

MAKE AN IMPACT | CREATE SOLUTIONS

Are you connecting the computing community with emerging technologies? Help advance emerging tech to create solutions for the betterment of humanity.

Every year, we give up to **US$50,000** in funding per project for these efforts.

Learn more at
computer.org/communities/emerging-technology-fund

bitly

COMPUTING
edge

IEEE COMPUTER SOCIETY
IEEE COMPUTER SOCIETY computer.org

Printed with inks containing soy and/or vegetable oils

STAFF

Editor
Lucy Holden

Production & Design Artist
Carmen Flores-Garvey

Periodicals Portfolio Senior Managers
Carrie Clark and Kimberly Sperka

Publications Operations Project Manager
Christine Shaughnessy

Periodicals Portfolio Specialist
Priscilla An

Director, Publications and Special Projects
Robin Baldwin

Senior Advertising Coordinator
Debbie Sims

Circulation: *ComputingEdge* (ISSN 2469-7087) is published monthly by the IEEE Computer Society. IEEE Headquarters, Three Park Avenue, 17th Floor, New York, NY 10016-5997; IEEE Computer Society Publications Office, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720; voice +1 714 821 8380; fax +1 714 821 4010; IEEE Computer Society Headquarters, 2001 L Street NW, Suite 700, Washington, DC 20036.

Postmaster: Send address changes to *ComputingEdge*-IEEE Membership Processing Dept., 445 Hoes Lane, Piscataway, NJ 08855. Periodicals Postage Paid at New York, New York, and at additional mailing offices. Printed in USA.

Editorial: Unless otherwise stated, bylined articles, as well as product and service descriptions, reflect the author's or firm's opinion. Inclusion in *ComputingEdge* does not necessarily constitute endorsement by the IEEE or the Computer Society. All submissions are subject to editing for style, clarity, and space.

Reuse Rights and Reprint Permissions: Educational or personal use of this material is permitted without fee, provided such use: 1) is not made for profit; 2) includes this notice and a full citation to the original work on the first page of the copy; and 3) does not imply IEEE endorsement of any third-party products or services. Authors and their companies are permitted to post the accepted version of IEEE-copyrighted material on their own Web servers without permission, provided that the IEEE copyright notice and a full citation to the original work appear on the first screen of the posted copy. An accepted manuscript is a version which has been revised by the author to incorporate review suggestions, but not the published version with copy-editing, proofreading, and formatting added by IEEE. For more information, please go to: http://www.ieee.org/publications_standards/publications/rights/paperversionpolicy.html. Permission to reprint/republish this material for commercial, advertising, or promotional purposes or for creating new collective works for resale or redistribution must be obtained from IEEE by writing to the IEEE Intellectual Property Rights Office, 445 Hoes Lane, Piscataway, NJ 08854-4141 or pubs-permissions@ieee.org. Copyright © 2026 IEEE. All rights reserved.

Abstracting and Library Use: Abstracting is permitted with credit to the source. Libraries are permitted to photocopy for private use of patrons, provided the per-copy fee indicated in the code at the bottom of the first page is paid through the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923.

Unsubscribe: If you no longer wish to receive this *ComputingEdge* mailing, please email IEEE Computer Society Customer Service at help@computer.org and type "unsubscribe *ComputingEdge*" in your subject line.

IEEE prohibits discrimination, harassment, and bullying. For more information, visit www.ieee.org/web/aboutus/whatis/policies/p9-26.html.

2026 IEEE Computer Society Magazine Editors in Chief

Computer
Bret Michael, *Naval Postgraduate School (Interim EIC)*

Computing in Science & Engineering
Jeffrey Carver, *University of Alabama*

IEEE Annals of the History of Computing
Troy Astarte, *Swansea University*

IEEE Computer Graphics and Applications
Pak Chung Wong, *Rocketgraph*

IEEE Intelligent Systems
Jeffrey Voas, *NIST*

IEEE Internet Computing
Weisong Shi, *University of Delaware*

IEEE Micro
Hsien-Hsin Sean Lee, *Intel Corporation*

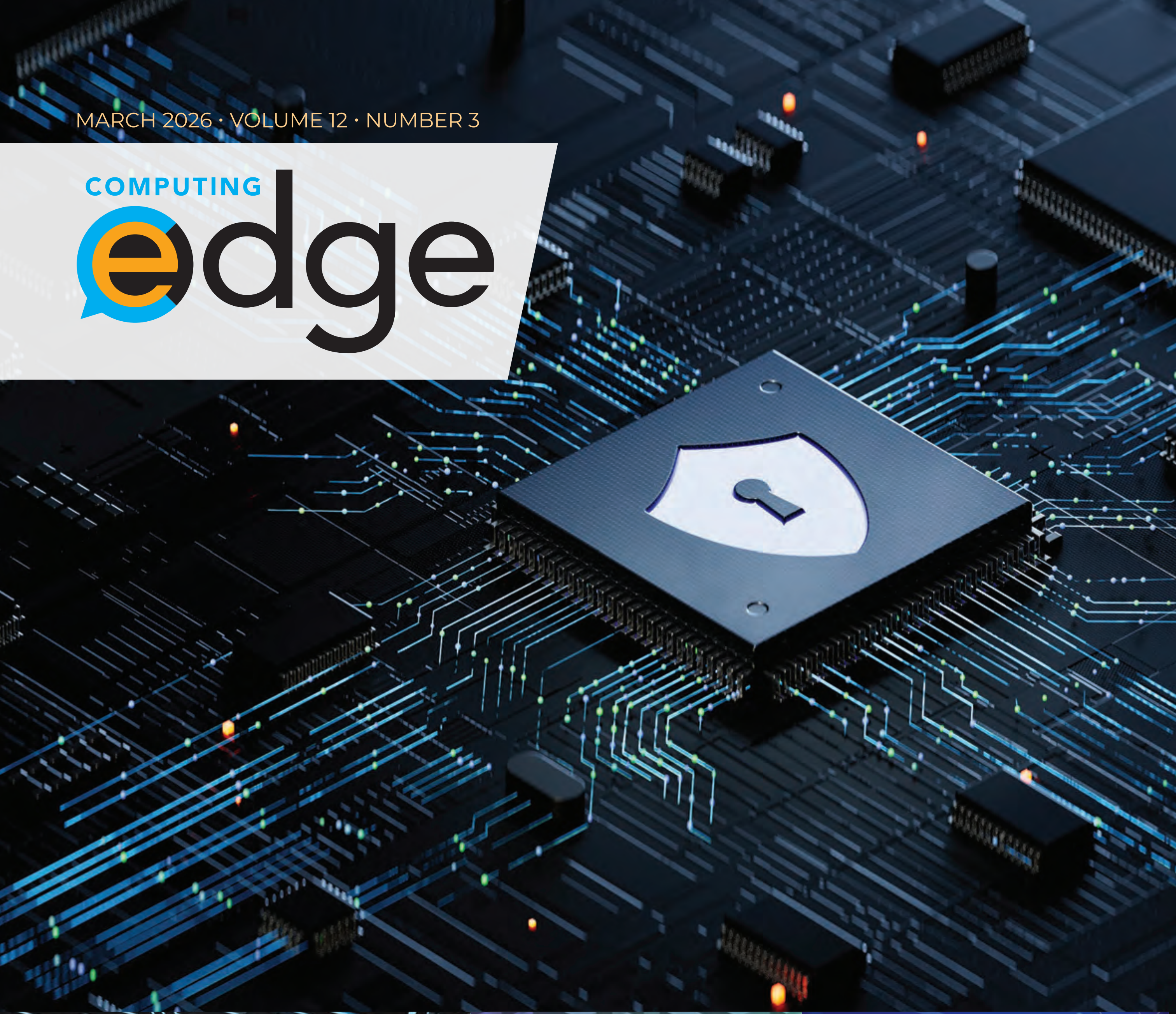
IEEE MultiMedia
Balakrishnan Prabhakaran, *University of Texas at Dallas*

IEEE Pervasive Computing
Fahim Kawsar, *Omnibuds and University of Glasgow*

IEEE Security & Privacy
Jeffrey Voas, *NIST*

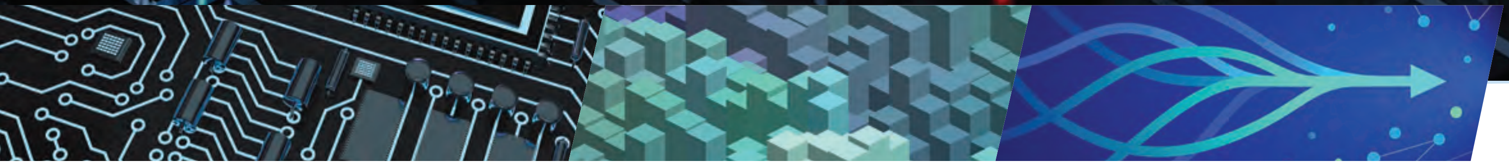
IEEE Software
Sigrid Eldh, *Ericsson, Mälardalen University, Sweden; Carleton University, Canada*

IT Professional
Charalampos Z. Patrikakis, *University of West Attica*



MARCH 2026 · VOLUME 12 · NUMBER 3

COMPUTING edge



8
Specification and Formal Verification of Hardware–Software Contracts for High-Assurance Computer Architectures

16
Toward a Universal Cryptographic Accelerator

26
Software Testing in the Generative AI Era: A Practitioner's Playbook

Hardware Security

8 Specification and Formal Verification of Hardware–Software Contracts for High-Assurance Computer Architectures
CAROLINE TRIPPEL

16 Toward a Universal Cryptographic Accelerator
SRINIVAS DEVADAS AND DANIEL SANCHEZ

20 Sipping Matcha of Security: A Fireside Chat With Mengjia Yan
JIANMING TONG AND ZISHEN WAN

AI and Machine Learning and Software Testing

26 Software Testing in the Generative AI Era: A Practitioner's Playbook
DOUGLAS C. SCHMIDT

32 Hybrid Classical-AI Systems for Software Testing and Bug Fixing
MIROSLAW STARON AND SILVIA ABRAHÃO

Education

38 Test and Evaluation of Autonomous Systems: Developing the Educational Landscape
BRIAN A. WEISS, DONALD H. COSTELLO III, AND MATT SCASSERO

History

43 ELIZA Reanimated: Restoring the Mother of All Chatbots to One of the World's First Time-Sharing Systems
RUPERT LANE, ANTHONY HAY, ARTHUR SCHWARZ, DAVID M. BERRY, AND JEFF SHRAGER

52 Stepping Into the Future—The Invention and Evolution of the Stepping Switch
THOMAS K. ROBBINS

Departments

4 Magazine Roundup
7 Editor's Note: Satisfying Hardware Security Requirements
60 Conference Calendar

Subscribe to *ComputingEdge* for free at www.computer.org/computingedge

Magazine Roundup

The IEEE Computer Society's lineup of 12 peer-reviewed technical magazines covers cutting-edge topics ranging from software design and computer graphics to Internet computing and security, from scientific applications and machine intelligence to visualization and microchip design. Here are highlights from recent issues.

Computer

European Space Agency Cyber Italy: Enabling Evidence-Based Decision-Making in Smart Communities

In this article, featured in the December 2025 issue of *Computer*, the authors discuss the Cyber Italy project, which aims to bridge the gap between environmental data and public engagement via user-friendly scenario exploration tools. Cyber Italy integrates Earth observation data, sensor measurements, and advanced analytics to create environmental digital twins, modeling the complex interplay between natural and anthropogenic phenomena.

Computing in Science & Engineering

Simulation-Based Transition Maneuver Analysis of a Convertible UAV

The authors of this July–September 2025 *Computing in Science & Engineering* article present the nonlinear aerodynamic modeling of a convertible uncrewed aerial vehicle (CUAV) designed for operation in constrained environments, integrating the advantages of fixed-wing and rotary-wing flight. The CUAV is capable of vertical takeoff and landing (VTOL) and smoothly transitions to horizontal flight after reaching a target altitude. The transitional flight behavior is analyzed across a range of angles of attack to represent different flight regimes. Simulations using computational fluid dynamics (CFD) are performed to assess the CUAV configuration, with particular emphasis on the impact of turbulence modeling on aerodynamic performance.

IEEE Annals of the History of Computing

Digital Construction Comes to the Pacific Northwest: Timber and the Landscapes of Automation

Automation—understood as a reconfiguration of work and material culture—unfolds in relation to specific histories and ecologies. This article, featured in the October–December 2025 issue of *IEEE Annals of the History of Computing*, analyzes how recent enthusiasm for digitization in construction intersects with the legacy of the timber industry in western North America. Drawing on industry literature and fieldwork from the architecture, construction, and forestry industries in Oregon and Washington, the article describes how the use of information technology to automate construction is inseparable from existing political ecologies of resource extraction.

IEEE Computer Graphics and Applications

AnchorTextVis: A Visual Analytics Approach for Fast Comparison of Text Embeddings

Visual comparison of text embeddings is crucial for analyzing semantic differences and comparing embedding models. Existing methods fail to maintain visual consistency in comparative regions and lack AI-assisted analysis, leading to high cognitive loads and time-consuming exploration processes. In this article featured in the November/December 2025 issue of *IEEE Computer Graphics and Applications*, the authors propose AnchorTextVis, a visual analytics approach based on AnchorMap—their dynamic projection algorithm balancing spatial quality

and temporal coherence and large language models (LLMs) to preserve users' mental map and accelerate the exploration process.

IEEE Intelligent Systems

Fine-Tuning Large Language Models With Behavioral Alignment for Depression Detection

Depression is a prevalent mental health issue, and early detection is crucial for effective intervention. In this article, featured in the November/December 2025 issue of *IEEE Intelligent Systems*, the authors propose the depression large language model (DLLM), a novel two-stage fine-tuning framework designed to enhance the accuracy and robustness of depression detection using multimodal data from social media.

IEEE Internet Computing

MEAL: Model of Empathy Augmented Logistics for Food Security

Millions globally lack reliable access to nutritious food. Efforts to address food insecurity seek to provide consumers food that may be rescued (i.e., what warehouses or grocers would otherwise soon discard as unusable), directly donated, or acquired using governmental funds. Current approaches produce allocations that optimize global objectives to store and move food efficiently in the network. However, they largely overlook consumer preferences and constraints. This article from the May/June 2025 issue of *IEEE Internet Computing* presents a new model, evaluated via human study and agent-based simulation, that shows how to combine the consumer and provider perspectives.

IEEE micro

MangoBoost Alice: Extremely Fast, Seamless, and Versatile FPGA-Accelerated DPU Solutions

Data processing units (DPUs) have emerged as specialized hardware that offloads data processing infrastructure tasks in data centers. However, building DPUs that effectively accelerate these tasks is highly challenging due to their wide variety and stringent performance requirements. The authors of this article, featured in the September/October 2025 issue of *IEEE Micro*, present MangoBoost Alice, a family of commercial field-programmable gate array-based DPUs designed to deliver fast, flexible, versatile, and easy-to-use solutions for a broad spectrum of data processing tasks.

IEEE MultiMedia

CAMUL: Context-Aware Multiconditional Instance Synthesis for Image Segmentation

Instance image segmentation task requires training with abundant annotated data to achieve high accuracy. Recently, conditional image synthesis has demonstrated its effectiveness in generating synthetic data for this task. However, existing image synthesis models face challenges in generating target instances to match the masks with complex shapes. Moreover, others fail to create diverse instances due to utilizing low-context simple text prompts. To address these issues, the authors of this July–September 2025 *IEEE MultiMedia* article propose CAMUL, a framework for context-aware multiconditional instance synthesis.

IEEE pervasive COMPUTING

MOBILE SYSTEMS | UBIQUITOUS COMPUTING | INTERNET OF THINGS

Parallel Feature Fusion for Multimodal Scene Recognition on Dual-Core MCUs

Multimodal sensing promises more robust environmental understanding for pervasive computing applications, but implementing sophisticated sensor fusion on resource-constrained devices remains challenging. The authors of this article, featured in the July–September 2025 *IEEE Pervasive Computing* issue, present a novel approach that leverages dual-core microcontrollers to enable the parallel processing of visual and audio data for scene recognition.

IEEE SECURITY & PRIVACY

Cybersecurity in Tactical Edge Networks

Securing computing operations at the tactical edge remains a crucial challenge. In this article, featured in the November/December 2025 issue of *IEEE Security & Privacy*, the authors discuss a recent North Atlantic Treaty Organization Science and Technology report and technical symposium that indicates that context-awareness and adaptation in cybersecurity posture can address these evolving challenges.

IEEE Software

Tool: Qadence: A Differentiable Interface for Digital and Analog Programs

Digital–analog quantum computing is an alternative paradigm for universal quantum computation, combining digital gates with global analog operations acting on a register of interacting qubits. The authors of this article from the November/December 2025 issue of *IEEE Software* present Qadence, a high-level programming interface for building complex digital and analog quantum programs.

IT Professional

Generative Artificial Intelligence: Ethical Challenges and Trust Mechanisms

Generative artificial intelligence is a groundbreaking technology transforming various sectors of society. However, its widespread adoption raises ethical concerns, including data privacy, bias risks, transparency, and accountability. Establishing trust mechanisms is critical to ensuring the sustainable development of this technology. This article, featured in the November/December 2025 issue of *IT Professional*, investigates the ethical challenges faced by generative artificial intelligence and proposes a trust framework based on transparency, fairness, accountability, and privacy, with the objective of fostering responsible applications in economic and social contexts.

IEEEDataPort™

STORE, SEARCH & MANAGE RESEARCH DATA

Individual subscriptions to IEEE DataPort are free for all IEEE society members and Young Professionals. Just log in and activate your subscription for unlimited access to datasets, data management tools, dataset storage for your own research, and more.

Open Access Options
2 TB of Cloud Storage
Link to Manuscripts
Generate Citations
Reproducible Research
ORCID Integration
Host Data Competitions
DOI Provided
IEEEDataPort

Join the IEEE Computer Society

computer.org/join

Satisfying Hardware Security Requirements

As hardware designs grow more complex and increasingly rely on untrustworthy disseminated devices for cyberphysical systems, ensuring security has become a critical challenge. This issue of *ComputingEdge* discusses problems facing modern hardware design as well as potential solutions, including tailoring cryptographic protocols to programmable hardware accelerators. The articles also outline the benefits and challenges of using artificial intelligence (AI) to enhance software testing as well as educational efforts to design autonomous learning systems and introduce a curriculum for the test and evaluation (T&E) of these systems. The issue concludes with an analysis of two older innovations—ELIZA and the stepping switch.

Engineers must design better security systems to meet the demands of modern hardware. The author of “Specification and Formal Verification of Hardware–Software Contracts for High-Assurance Computer Architectures,” from *Computer*, explores the specification and verification challenges facing modern hardware deployment and proposes some solutions. *Computer* article “Toward a Universal Cryptographic Accelerator” argues for a synergistic approach of designing programmable hardware accelerators for cryptography. The *IEEE Micro* article, “Sipping Matcha of Security: A Fireside Chat With Mengjia Yan,” features an interview with Prof. Mengjia Yan, in which she discusses her journey in secure computer architecture and her perspectives on the field.

Though machine learning (ML) and AI are helping to improve software testing, they also introduce new challenges. In “Software Testing in the Generative AI Era: A Practitioner’s Playbook,” from *Computer*, the author shows how generative AI is transforming software testing. *IEEE Software* article “Hybrid Classical-AI Systems for Software Testing and Bug Fixing” highlights new insights from recent research into improving bug detection and resolution, particularly through ML and AI.

Autonomous systems present new and exciting advances in education. The authors of *Computer* article “Test and Evaluation of Autonomous Systems: Developing the Educational Landscape” discuss the benefits of a T&E of Autonomous Systems curriculum for universities.

Looking back at historical technologies reveals the modern day impacts of these technologies. In “ELIZA Reanimated: Restoring the Mother of All Chatbots to One of the World’s First Time-Sharing Systems,” from *IEEE Annals of the History of Computing*, the authors reanimate ELIZA, the earliest chatbot, using rediscovered code. *IEEE Annals of the History of Computing* article “Stepping Into the Future—The Invention and Evolution of the Stepping Switch” delves into the history of the stepping switch, developed in the 19th century, which enabled direct telephone dialing and online real-time, networked computation.

2469-7087/26 © 2026 IEEE

EDITORS: Timothy Jones, University of Cambridge, U.K., timothy.jones@cl.cam.ac.uk
Robert Mullins, University of Cambridge, U.K., robert.mullins@cl.cam.ac.uk

This article originally appeared in
Computer
vol. 58, no. 8, 2025

DEPARTMENT: COMPUTING ARCHITECTURES

Specification and Formal Verification of Hardware–Software Contracts for High-Assurance Computer Architectures

Caroline Trippel, *Stanford University*

Modern hardware designs and deployments continue to increase in complexity, exposing shortcomings in today's computer systems specifications and verification flows, but new work on formally specifying and verifying hardware–software contracts offers paths to high-assurance computer architectures.

To scale performance at manageable power and thermal levels as Moore's Law fades, computer systems frequently combine parallelism, hardware specialization, hardware and software heterogeneity, and data-dependent optimization. To reduce costs, improve resource utilization, and meet the resource demands of modern applications, multitenancy and hyperscale systems are also commonplace. Meanwhile, computers routinely perform sophisticated tasks with stringent assurance—that is, correctness, security, and reliability—requirements, like driving cars, managing health records, controlling infrastructure, and providing security services. This convergence of factors brings about two critical challenges in the field of computer architecture:

- *The specification challenge*: Traditional instruction set architectures (ISAs) are not detailed enough to capture all the ways in which a hardware implementation thereof may undermine software assurance.
- *The verification challenge*: Despite decades of advancements in formal methods tools, ensuring that a hardware implementation (that is, microarchitecture) upholds the design specifications (including traditional ISAs and more) it intends to remains highly difficult.

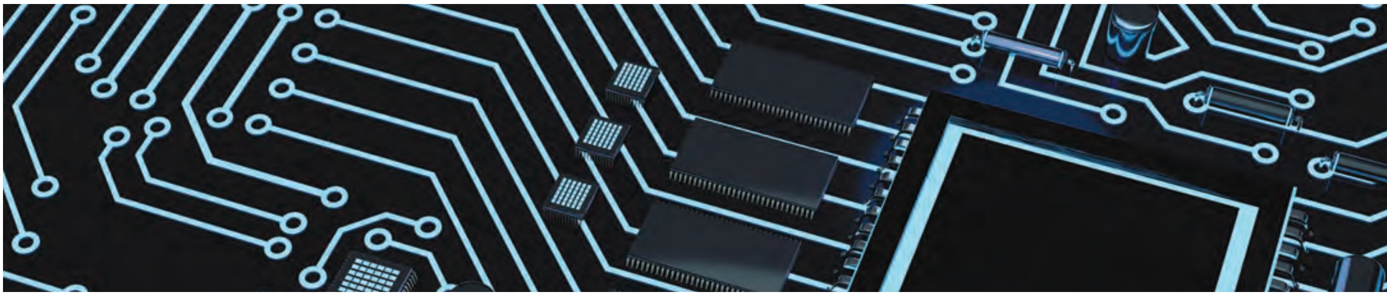
This article explores these challenges and some proposed solutions with an eye toward ensuring correctness for concurrent code and software security against hardware side-channel attacks.

THE HARDWARE–SOFTWARE CONTRACT SPECIFICATION CHALLENGE

Traditional ISAs have led programmers to make incorrect assumptions about shared memory ordering guarantees, what hardware state is software-visible, what (speculative) control- and data-flow a program may exhibit at runtime, interprocess isolation guarantees (for example, on GPUs), software reliability, and more. While exposing relevant hardware details to programmers may be straightforward in theory [for example, via register transfer level (RTL) designs], designing practical, robust, and usable hardware–software (HW–SW) contracts to augment traditional ISAs is not. Nevertheless, standard processor memory consistency models (MCMs) provide a concrete example of how such contracts can be successfully developed, and recent efforts to design hardware side-channel security contracts show similar promise.

Digital Object Identifier 10.1109/MC.2025.3573841
Date of current version: 28 July 2025

2469-7087/26 © 2026 IEEE



Specifying contracts for shared memory parallelism

Starting in the 1980s, motivated by multiprocessors, and continuing into the 2000s, motivated by multicores, ISAs were augmented with MCMs to support reasoning about the legal execution behaviors of parallel programs whose threads communicate through shared memory.[27] In particular, MCMs define the values that shared memory loads in a parallel program are allowed to return.

Current-generation memory consistency models. A longstanding method for specifying the legal/illegal execution behaviors of programs according to a specific MCM is via a set of permitted/forbidden litmus tests[2]—small parallel programs with concrete outcomes, that is, concrete return values for loads. Early MCM specifications combined natural language descriptions with handcrafted litmus tests. From the 2000s to 2010s, a large body of work formally specified MCMs for a variety of ISAs and high-level languages (HLLs).[27] These formalization efforts have enabled verified compilers and automated tools that find/repair concurrency bugs in programs.

Next-generation memory consistency models. Current MCMs support designing correct concurrent code for homogeneous shared-memory multiprocessors/multicores. However, they do not readily account for the implications of a range of other pervasive hardware features. For example, implementing cache-coherent shared memory in heterogeneous systems is increasingly important, but remains difficult due to MCM mismatches among processing elements[5,23]: *what* the system-side MCM should be and *how* it should be enforced are not well-defined.

Oswald et al.[23] address this challenge by automatically synthesizing a unique hardware *consistency protocol* (that is, an MCM-respecting coherence protocol) per set of heterogeneous clusters, given per-cluster *coherence protocol* and ISA MCM specifications. (A cluster denotes a group of homogeneous compute elements sharing a memory hierarchy.) The resulting system-wide MCM, called a *compound consistency model*, is described as the "amalgamation" of the per-cluster MCMs.

In contrast, our recent work introduces MemGlue,[5] a consistency protocol that enforces a slight strengthening of C11—the seminal heterogeneous MCM—for intercluster communication. Intracluster communication remains governed by existing coherence protocols and ISA MCMs. This modular approach enables reuse of verified C11-to-ISA compiler mappings to verify a cluster's correct integration into a MemGlue system and requires one-time verification of the MemGlue protocol itself.

Beyond heterogeneity, researchers have extended traditional MCMs to explicitly account for virtual memory,[10] exceptions,[25] and hardware specialization.[9,17] As MCMs evolve to capture a broader range of hardware features, managing their complexity is poised to become a central challenge for scalable, rigorous reasoning about the correctness of software that relies on them and hardware that implements them.

Specifying contracts for hardware side-channel leakage.

For decades, it has been known that certain programs may leak their secret data when they run on certain microarchitectures due to hardware side-channel attacks.[16,22,24] These attacks are often defined as involving a *transmit instruction* (or *transmitter*, in the victim program) that modulates a *channel* (hardware resource) in an operand-dependent manner, and a *receiver* (attacker) that observes the channel modulation to infer the operand value.[15] (Though, in general, transmitters need not be explicit program instructions.) Many hardware resources have been implicated as channels, for example, caches, branch predictors, functional units, and memory ports. A receiver observes channel modulations indirectly via their effects on nondeterministic aspects of program execution,[12] for example, execution time and resource contention.

Current-generation side-channel security contracts. Historically, there has been no ISA guidance on side-channels. Nevertheless, virtually all state-of-the-art hardware side-channel defenses assume the availability of per-microarchitecture *leakage contracts*, which characterize a microarchitecture's transmitters.

The most widely-adopted leakage contract is the constant-time (CT) contract, which simply enumerates a microarchitecture's transmitters and their unsafe ("leaky") operands. Given a CT contract, a hardware side-channel defense can ensure that secrets never reach the unsafe operands of transmitters when programs execute. This strategy is embodied in the CT programming defense—the gold standard software defense for protecting the architectural executions of cryptographic code from hardware side-channel attacks. Some software and hardware defenses against transient execution attacks, which exploit hardware faults or mis-predictions to steer secrets toward the unsafe operands of transient (bound-to-squash) transmitters, extend this strategy to target programs' speculative (including architectural and transient) executions. Owning to their widespread use, CT leakage contracts have been very recently adopted by several major ISAs, including Arm,[3] Intel,[13] and RISC-V.[1]

Next-generation side-channel security contracts. Notably, leakage contracts and other types of side-channel *security contracts* are actively evolving, and there is significant opportunity for academic research to influence how industrial ISAs expose hardware side-channel leakage to software.

The most performant and comprehensive defenses against transient execution attacks today are implemented in hardware. Instead of (or in addition to) CT contracts, these defenses adopt bespoke leakage contracts that are much finer grained and identify various microarchitectural scheduling conditions under which transmitters leak their unsafe operands. For example, a store transmitter may leak its address operand when it executes alongside a load since the load may access memory (and execute slower) or not (and execute faster) upon an address mismatch or match with the store, respectively.

Software defenses against transient execution attacks assume the availability of CT contracts and *execution contracts*, a type of security contract that characterizes the speculative control- and data-flow that programs may exhibit on a specific hardware implementation.

Our early work on this topic introduced axiomatic side-channel security contracts, called *leakage containment models* (LCMs), by directly extending axiomatic MCMs.[20] This design choice enabled us to build a program analysis tool, called Clou, which uses axiomatic LCMs and satisfiability modulo theory (SMT) solvers to find and mitigate Spectre vulnerabilities in programs, by repurposing a tool that uses axiomatic MCMs to find and fix concurrency bugs. Clou found 5/7 new Spectre gadgets in the well-studied OpenSSL/Libsodium cryptography libraries.

Clou illuminated several patterns of Spectre leakage in programs that are expensive or impossible to mitigate with speculation fence insertion (the standard software defense strategy), but cheap to mitigate using program transformations and lightweight hardware support. These observations inspired our design of Serberus, the first comprehensive compiler defense for protecting CT code (usually, cryptographic code) from Spectre attacks on existing hardware.[21] Serberus is proven correct with respect to a novel operational execution contract, called ASP (Abstract Speculative Processor), which enables programs to restrict their runtime control- and data-flow from software just enough to performantly mitigate Spectre. In response to our work, Intel updated their Software Security Guidance to confirm that two Intel microarchitectures uphold ASP.[14,28]

In general, next-generation side-channel security contracts may benefit from building on other orthogonal security contracts. For example, Serberus leverages Intel Control-Flow Enforcement Technology (CET), which was designed to protect programs from control-flow hijacking attacks, to reduce the Spectre attack surface for programs sufficiently to enable an efficient software defense.

THE HARDWARE–SOFTWARE CONTRACT VERIFICATION CHALLENGE

Robust HW–SW contracts and the analyses they support are only useful if we can verify hardware compliance—a grand challenge. As evidence, a 2024 Siemens EDA and Wilson Research Group study found that FPGA and IC/ASIC design projects spend significant time on verification, yet critical bugs escape to silicon in 87% and 86% of these projects, respectively.[7,8] Verifying next-generation HW–SW contracts that address the previously discussed specification challenge will exacerbate this issue. Plus, as artificial intelligence (AI)-driven tools gain traction in the hardware design space, the number of presilicon bugs that must be found and fixed will likely increase. This projection partially stems from the fact that AI tools may initially generate erroneous designs due to limited representative training data. More fundamentally though, ambiguity in natural language design specifications allows multiple plausible implementations, some of which are misaligned with designer intent.

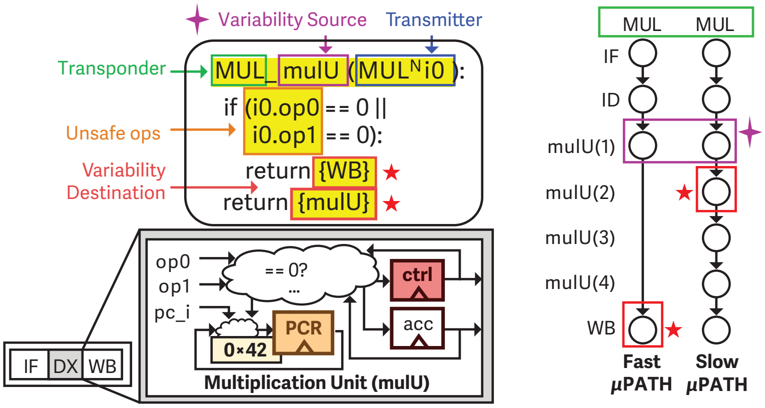


FIGURE 1. Two μPATHS for a MUL (PC = 0×42) on a pipeline that implements the zero-skip optimization. A MUL will spend one cycle in the multiplication unit (mulU) if it has at least one zero operand; else, it will spend four cycles. The leakage signature explains that MUL's μPATH variability is a function of its own (*N*) operands following its update of state in the mulU. Row(*n*) denotes the *n*th update of Row.

The industry-standard approach to formally verifying that a microarchitecture upholds some design specification is top-down: Verification engineers manually map abstract specification requirements down to complex (hard-to-check) formal properties over design signals and evaluate the design's adherence to these properties using formal methods tools, especially model checkers. This process requires significant manual decomposition of the properties and design to scale.

Bottom-up verification with template-generated properties

For certain classes of especially hard verification problems, our work adopts an alternative bottom-up approach: Many simple (easy-to-check) formal properties are automatically generated from templates and static netlist analysis and checked against the design under verification (DUV) to synthesize (that is, reverse-engineer) a formally verified specification from it. Synthesized specifications reflect hardware reality and can be analyzed (for example, directly comparing to designer expectation) to detect bugs.

This style of hardware-specification verification is inspired by the Check tools,[18] which scale/automate formal MCM and security verification of processors by analyzing abstract models of microarchitecture, which omit design details irrelevant to these verification tasks. In particular, the Check tools conduct microarchitectural happens-before (μHB) analysis, where hardware-specific executions of programs are modeled as directed μHB graphs (for example, Figure 1). A μHB graph node represents a microarchitectural event (row label) that is associated with a dynamic instruction's (column label) execution on a specific hardware implementation. Edges denote "happens-before" relationships. To facilitate μHB analysis, the Check tools analyze a microarchitecture in the guise of an axiomatic μSPEC model—a set of first-order logic axioms (rules) that describe how to construct μHB graphs. About half of these axioms encode all possible microarchitectural execution paths (μPATHS) for each implemented instruction, to instantiate column-wise nodes/edges (as in Figure 1).

Despite finding bugs in real hardware, the Check tools have not seen broad adoption due to a verification gap between μSPEC models, which have been manually written, and RTL. One goal of our work has thus been to bridge this gap by automating the synthesis of formally verified μSPEC models from RTL.

Memory consistency model verification of simple processors. Our first attempt at narrowing the μspec-RTL gap produced rtl2μspec,[11] an automated approach/tool for synthesizing formally verified μspec models from simple SystemVerilog processor designs. rtl2μspec synthesizes a complete verified μspec model from the four-core RISC-V multi-V-scale processor in 6.84 min; MCM verification with the Check tools on the output model takes seconds. The top-down approach times out for the same design.

rtl2μspec enables enables push-button MCM verification of processor RTL for the first time. However, it is limited to simple processors since it uncovers *at most one* μpath per implemented instruction—where μpath nodes denote sets of state updates that an instruction performs—on the input design; μspec models must encode *all* such μpaths.

Toward memory consistency model verification of (more) advanced processors. We resolved rtl2μspec's "single-execution-path" limitation with rtl2Mμpath,[12] an automated approach/tool for synthesizing a complete set of formally verified cycle-accurate paths—where μpath nodes represent instruction state updates in specific cycles, and edges encode one-cycle happens-before—for each instruction on a SystemVerilog processor design. By synthesizing about half of a complete μspec model, rtl2Mμpath takes a significant step toward unlocking the power of the Check tools for advanced processor RTL.

Leakage contract verification of (more) advanced processors. In designing rtl2Mμpath, we made an important observation: μpath variability (>1 possible μpath) for an instruction strongly indicates there is a hardware side-channel in the input design. Based on this observation, we designed SynthLC,[12] which extends rtl2Mμpath with a symbolic information flow tracking (symbolic IFT)[26] step to synthesize a complete set of formally verified leakage signatures from a SystemVerilog processor design, assuming an attacker that monitors program execution time or resource contention. Leakage signatures precisely characterize hardware side-channels, capturing all microarchitectural details required to implement ten state-of-the-art hardware side-channel defenses (that is, all details comprising their respective leakage contracts). So, SynthLC is the first automated methodology for verifying the correct implementation of these defenses—which include Arm, Intel, and RISC-V's ISA extensions to support CT programming—in SystemVerilog RTL.

We deployed SynthLC on the RISC-V CVA6 core, surfacing 94 unique leakage signatures, 72 transponders, and 26 transmitters. Compared to prior work that analyzes the same design, SynthLC found a novel channel that leaks store and load address operands and renders CVA6 vulnerable to a new class of *speculative interference attacks*. We separately deployed SynthLC on the CVA6 L1 data cache and cache controller, making it the first formal side-channel analysis of a realistic cache.

Several works present methodologies for formally verifying hardware against CT (leakage) contracts in particular. Fundamentally, formal side-channel analyses must check a two-trace *noninterference* property. These other works check such a property directly (that is, the top-down approach): a model checker analyzes two copies of the DUV at once (a "product circuit"). The resulting state space explosion forces these approaches to conduct analysis from a symbolic initial state, which may be unreachable, in order to achieve complete proofs on reasonably complex designs. Thus, the user must manually/iteratively classify verification counterexamples as true or false (by inspecting waveforms) and write formal properties to avoid false ones. This need for manual intervention can result in missed hardware side-channel vulnerabilities.

Instead, rtl2Mμpath uncovers μpaths using many one-trace properties, which can be evaluated on one copy of the DUV. SynthLC then checks the requisite noninterference property indirectly by comparing all pairs of μpaths to localize μpath variability and using symbolic IFT to attribute it to transmitter operands. We find that replacing one monolithic two-trace property with many one-trace properties plus symbolic IFT enables SynthLC to efficiently analyze advanced designs from a valid reset state.

Generality of μpath-based verification. Overall, bottom-up verification with rtl2μspec, rtl2Mμpath, and SynthLC demonstrates that μpaths are a powerful microarchitectural intermediate representation, which enable the design of automated and performant/scalable formal hardware verification procedures for MCMs, side-channel security contracts, and more. For instance, μpath synthesis with RTL2MμPATH on a well-studied RISC-V core surfaced new functional bugs by finding counterintuitive μPATHS for some instructions. Plus, RTL2MμPATH can be adapted to perform μPATH synthesis for microarchitectural event interpretations beyond state updates.

Verification with universal correctness and AI-generated properties

For hardware–contract verification tasks that are not amenable to property generation from templates, at least two other approaches show promise for democratizing them. First, many hardware bugs can be caught by checking "universal" correctness properties like *functional consistency*, the property that the same instruction must produce identical outputs when provided with identical inputs.[4] Second, when it is necessary to craft bespoke formal properties that capture the intent of natural language specifications, generative AI[6] and formal tools (such as model checkers)[19] can meaningfully collaborate with a human user to simplify/accelerate this process.

Importance of democratizing hardware verification

Beyond enabling faster time to market—which is increasingly critical in an era of rapidly evolving and diverse hardware organizations—scalable, push-button verification is essential for lowering development costs, improving assurance, and supporting AI-driven hardware design. In particular, iterative AI-based design methods will require ways to systematically detect and eliminate design bugs, including those introduced by ambiguous natural language specifications. Moreover, formal verifiers may play a key role in generating the high-quality datasets needed to fine-tune AI models for more accurate RTL generation.

In conclusion, the increasing complexity of modern hardware designs and deployments exposes shortcomings in current HW–SW contracts and places growing strain on already overburdened verification flows. To address these challenges, this article highlights work on designing and formally specifying next-generation contracts, developing software analyses that leverage them to produce high-assurance programs, and democratizing the formal verification of their hardware implementations.

ACKNOWLEDGMENT

This article is based on work supported by the National Science Foundation (NSF) under awards 2017863, 2236855 (CAREER), 2153936, and 2321489; by the Defense Advanced Research Projects Agency (DARPA) under contract W912CG-23-C-0025 and subcontract from Galois, Inc; and by the German Federal Ministry of Education and Research (BMBF) through funding for the CISPA-Stanford Center for Cybersecurity (FKZ: 13N1S0762, 16KIS1138).

REFERENCES

1. "RISC-V cryptography extensions volume I: Scalar & entropy source instructions," RISC-V International, Zürich, Switzerland, Jun. 2021. [Online]. Available: https://lists.riscv.org/g/dev-partners/attachment/43/0/riscv-crypto-spec-scalar-v0.9.3-DRAFT.pdf
2. J. Alglave, L. Maranget, S. Sarkar, and P. Sewell, "Litmus: Running tests against hardware," *Proc. 17th Int. Conf. Tools Algorithms Construction Anal. Syst. (TACAS)*, 2011, pp. 41–44.
3. "DIT: Data independent timing." Arm Developer. Accessed: Jun. 15, 2025. [Online]. Available: https://developer.arm.com/documentation/ddi0601/2025-03/AArch64-Registers/DIT--Data-Independent-Timing
4. S. Chattopadhyay et al., "G-QED: Generalized QED pre-silicon verification beyond non-interfering hardware accelerators," in *Proc. 60th ACM/IEEE Design Automat. Conf. (DAC)*, 2023, pp. 1–6, doi: 10.1109/DAC56929.2023.10247903.
5. R. Cleaveland and C. Trippel, "Memory consistency model-award cache coherence for heterogeneous hardware," in *Proc. 22nd Conf. Formal Methods Comput.-Aided Des.*, 2024, pp. 163–174.
6. M. Cosler, C. Hahn, D. Mendoza, F. Schmitt, and C. Trippel, "nl2spec: Interactively translating unstructured natural language to temporal logics with large language models," in *Proc. Int. Conf. Comput. Aided Verification*, 2023, pp. 383–396.

7. H. Foster, "IC/ASIC functional verification trend report - 2024," Siemens Digital Industries Software, Plano, TX, USA, Feb. 17, 2025. [Online]. Available: https://verificationacademy.com/resource/a661652a-b077-4b68-98fd-b84750cd9c1c
8. H. Foster, "FPGA functional verification trend report - 2024," Siemens Digital Industries Software, Plano, TX, USA, Feb. 17, 2025. [Online]. Available: https://verificationacademy.com/resource/0021d316-9b98-4857-834a-e40f96279fbe
9. S. Ghosh, Y. Shi, B. Lucia, and N. Beckmann, "Ripple: Asynchronous programming for spatial dataflow architectures," in *Proc. 46th ACM SIGPLAN Conf. Programm. Lang. Des. Implementation (PLDI)*, 2025, pp. 249–276.
10. N. Hossain, C. Trippel, and M. Martonosi, "TransForm: Formally specifying transistency models and synthesizing enhanced litmus tests," in *Proc. 47th Int. Symp. Comput. Archit. (ISCA)*, 2020, pp. 874–887, doi: 10.1109/ISCA45697.2020.00076.
11. Y. Hsiao, D. P. Mulligan, N. Nikoleris, G. Petri, and C. Trippel, "Synthesizing formal models of hardware from RTL for efficient verification of memory model implementations," in *Proc. 54th IEEE/ACM Int. Symp. Microarchit.*, 2021, pp. 679–694.
12. Y. Hsiao et al., "RTL2MμPATH: Multi-μPATH synthesis with applications to hardware security verification," in *Proc. 57th IEEE/ACM Int. Symp. Microarchit.*, 2024, pp. 507–524.
13. "Data operand independent timing instruction set architecture (ISA) guidance." Intel. Accessed: Jun. 15, 2025. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/best-practices/data-operand-independent-timing-isa-guidance.html
14. "Software security guidance." Intel. Accessed: Jun. 15, 2025. [Online]. Available: https://www.intel.com/content/www/us/en/developer/topic-technology/software-security-guidance/overview.html
15. V. Kiriansky, I. Lebedev, S. Amarasinghe, S. Devadas, and J. Emer, "DAWG: A defense against cache timing attacks in speculative execution processors," in *Proc. 51st Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, 2018, pp. 974–987, doi: 10.1109/MICRO.2018.00083.
16. P. C. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Proc. Annu. Int. Cryptol. Conf.*, 1999, pp. 388–397.
17. D. Lustig, S. Cooksey, and O. Giroux, "Mixed-proxy extensions for the NVIDIA PTX memory consistency model: Industrial product," in *Proc. 49th Annu. Int. Symp. Comput. Archit.*, 2022, pp. 1058–1070.
18. "Check: Research tools and papers." Martonosi Research Group. Accessed: Jun. 15, 2025. [Online]. Available: http://check.cs.princeton.edu
19. D. Mendoza, C. Hahn, and C. Trippel, "Translating natural language to temporal logics with large language models and model checkers," in *Proc. 22nd Conf. Formal Methods Comput.-Aided Des. (FMCAD)*, 2024, pp. 1–11, doi: 10.34727/2024/isbn.978-3-85448-065-5_17.
20. N. Mosier, H. Lachnitt, H. Nemati, and C. Trippel, "Axiomatic hardware-software contracts for security," in *Proc. 49th Annu. Int. Symp. Comput. Archit.*, 2022, pp. 72–86.
21. N. Mosier, H. Nemati, J. Mitchell, and C. Trippel, "Serberus: Protecting cryptographic code from spectres at compile-time," 2024, *arXiv:2309.05174*.
22. D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: The case of AES," in *Proc. Cryptographers' Track RSA Conf. Topics Cryptol. (CT-RSA)*, 2006, pp. 1–20.
23. N. Oswald, V. Nagarajan, D. J. Sorin, V. Gavrielatos, T. Olausson, and R. Carr, "HeteroGen: Automatic synthesis of heterogeneous cache coherence protocols," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, 2022, pp. 756–771, doi: 10.1109/HPCA53966.2022.00061.
24. C. Percival, "Cache missing for fun and profit," *BSDCan*, 2005. [Online]. Available: https://www.bsdcan.org/2015/
25. B. Simner et al., "Precise exceptions in relaxed architecture," in *Proc. 52nd Annu. Int. Symp. Comput. Archit. (ISCA)*, 2025, pp. 211–224, doi: 10.1145/3695053.3731102.
26. F. Solt, B. Gras, and K. Razavi, "CellIFT: Leveraging cells for scalable and precise dynamic information flow tracking in RTL," in *Proc. 31st USENIX Secur. Symp.*, 2022, pp. 2549–2566.
27. C. Trippel, "Concurrency and security verification in heterogeneous parallel systems," Ph.D. dissertation, Dept. Comput. Sci., Princeton Univ., Princeton, NJ, USA, 2019.
28. "Hardware features and behavior related to speculative execution." Intel. Accessed: Jun. 15, 2025. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/hardware-behavior-related-to-speculative-execution.html

CAROLINE TRIPPEL is an assistant professor in the Computer Science and Electrical Engineering Departments at Stanford University Stanford, CA 94305 USA. Contact her at trippel@stanford.edu.

IEEE COMPUTER SOCIETY

80th ANNIVERSARY CELEBRATION

www.computer.org

PURPOSE: Engaging professionals from all areas of computing, the IEEE Computer Society sets the standard for education and engagement that fuels global technological advancement. Through conferences, publications, and programs, IEEE CS empowers, guides, and shapes the future of its members, and the greater industry, enabling new opportunities to better serve our world.

OMBUDSMAN: Contact ombudsman@computer.org.

CHAPTERS: Regular and student chapters worldwide provide the opportunity to interact with colleagues, hear technical experts, and serve the local professional community.

PUBLICATIONS AND ACTIVITIES

***Computer*:** The flagship publication of the IEEE Computer Society, *Computer*, publishes peer-reviewed technical content that covers all aspects of computer science, computer engineering, technology, and applications.

Periodicals: The IEEE CS publishes 12 magazines, 18 journals

Conference Proceedings & Books: Conference Publishing Services publishes more than 275 titles every year.

Standards Working Groups: More than 150 groups produce IEEE standards used throughout the world.

Technical Communities: TCs provide professional interaction in more than 30 technical areas and directly influence computer engineering conferences and publications.

Conferences/Education: The IEEE CS holds more than 215 conferences each year and sponsors many educational activities, including computing science accreditation.

Certifications: The IEEE CS offers three software developer credentials.

AVAILABLE INFORMATION

To check membership status, report an address change, or obtain information, contact help@computer.org.

IEEE COMPUTER SOCIETY OFFICES

WASHINGTON, D.C.:
2001 L St., Ste. 700,
Washington, D.C. 20036-4928

Phone: +1 202 371 0101

Fax: +1 202 728 9614

Email: help@computer.org

LOS ALAMITOS:
10662 Los Vaqueros Cir.,
Los Alamitos, CA 90720

Phone: +1 714 821 8380

Email: help@computer.org

IEEE CS EXECUTIVE STAFF

Interim Executive Director: Anne Marie Kelly
Director, Governance & Associate Executive Director: Anne Marie Kelly
Director, Conference Operations: Silvia Ceballos
Director, Information Technology & Services: Sumit Kacker
Director, Marketing & Sales: Michelle Tubb
Director, Membership Development: Eric Berkowitz
Director, Publications & Special Projects: Robin Baldwin

IEEE

revised 07 January 2026

IEEE CS EXECUTIVE COMMITTEE

President: Grace A. Lewis
President-Elect: Joaquim Jorge
Past President: Hironori Washizaki
Vice President: Terry Benzel
Secretary: Yuhong Liu
Treasurer: Fernando Bouche
VP, Member & Geographic Activities: Andrew Seely
VP, Professional & Educational Activities: Cyril Onwubiko
VP, Publications: Charles (Chuck) Hansen
VP, Standards Activities: Darren Galpin
VP, Technical & Conference Activities: Andreas Reinhardt
2025–2026 IEEE Division VIII Director: Cecilia Metra
2026 IEEE Division VIII Director-Elect: Nita Patel
2026–2027 IEEE Division V Director: Leila De Floriani

IEEE CS BOARD OF GOVERNORS

Term Expiring 2026:
Megha Ben, Mrinal Karvir, Sean Peisert, Andreas Reinhardt, Deborah Silver, Yoshiko Yasuda

Term Expiring 2027:
Sven Dickinson, Alfredo Goldman, Daniel S. Katz, Yuhong Liu, Ladan Tahvildari, Damla Turgut

Term Expiring 2028:
Min Chen, Sap Ghosh, Lizy John, Rick Kazman, Carolyn McGregor, Walter Scheirer

IEEE EXECUTIVE STAFF

Executive Director and COO: Sophia Muirhead
General Counsel and Chief Compliance Officer: Ahsaki Benion
Chief Human Resources Officer: Cheri N. Collins Wideman
Managing Director, Technical Activities: Mojdeh Bahar
Managing Director, IEEE-USA: Russell Harrison
Chief Marketing Officer: Jayne O'Brien
Chief Publication Officer and Managing Director: Steven Heffner
Chief Governance Officer: Donna Hourican
Managing Director, Member and Geographic Activities: Cecelia Jankowski
Chief of Staff to the Executive Director: Allison Taylor
Managing Director, Educational Activities: Jamie Moesch
IEEE Standards Association Managing Director: Alpesh Shah
Chief Financial Officer: Kelly Armstrong
Chief Information Digital Officer: Jeff Strohschein
Managing Director, Conferences, Events, and Experiences: Marie Hunter

IEEE OFFICERS

President & CEO: Mary Ellen Randall
President-Elect: Jill I. Gostin
Past President: Kathleen A. Kramer
Director & Secretary: Jenifer P. Castillo Rodriguez
Director & Treasurer: Gerardo Barbosa
Director & VP, Publication Services & Products: W. Clem Karl
Director & VP, Educational Activities: Timothy P. Kurzweg
Director & VP, Membership and Geographic Activities: Andrew D. Lowery
Director & President, Standards Association: Gary R. Hoffman
Director & VP, Technical Activities: F D Tan
Director & President, IEEE-USA: Barry C. Tilton

EDITOR: Dimitrios Serpanos, CTI DIOPHANTUS and University of Patras,
serpanos@computer.org

This article originally
appeared in
Computer
vol. 58, no. 1, 2025

DEPARTMENT: CYBER-PHYSICAL SYSTEMS

Toward a Universal Cryptographic Accelerator

Srinivas Devadas and Daniel Sanchez, Massachusetts Institute of Technology

Cyberphysical systems have disseminated devices that can be untrustworthy or compromised. Nevertheless, the privacy and integrity of computation and data can be guaranteed through cryptographic protocols. We address the computational burden posed by cryptography, and argue for a synergistic approach of designing programmable hardware accelerators for cryptography, followed by tailoring cryptographic protocols to this hardware.

Cryptographic technologies for data encryption and authentication are mature and pervasive on the Internet [for example, Transport Layer Security (TLS), Secure Sockets Layer (SSL), and HTTPS]. But these cryptographic techniques protect data only during transmission and storage, not processing. Cyberphysical systems feature diverse and disseminated devices that process sensitive data and perform critical functions—but whose hardware or software can be easily compromised by an attacker. We propose to secure these systems by taking an approach that is rooted in cryptography and does not require trust in the hardware.

Sophisticated cryptographic primitives, such as fully homomorphic encryption (FHE), zero-knowledge proofs (ZKPs), and private information retrieval (PIR), are increasingly being used in varied applications such as outsourced computation, cryptocurrencies, private search, and anonymous communication. These primitives provide privacy and integrity guarantees on data processing, enabling secure and reliable computation from untrusted devices or servers. Unfortunately, state-of-the-art implementations of these primitives suffer poor performance, which limits their applicability: FHE and ZKP incur slowdowns ranging from four to six orders of magnitude over native computation, limiting them to small programs. Similarly, PIR schemes require time proportional to the size of the database in comparison to logarithmic lookup in a nonprivate setting.

Our goal is to take a significant step toward universal cryptographic acceleration infrastructure. We are developing a hardware architecture and a compiler that speed up a broad range of cryptographic protocols by multiple orders of magnitude. We are aided by the recent rise of postquantum-secure lattice cryptography, which is producing best-in-class protocols over a wide range of cryptographic applications, including those discussed previously. If successful, these protocols will become widely applicable and further jump-start the development of new cryptographic protocols that exploit and depend on hardware acceleration.

Our approach seeks to achieve the same synergy and wide impact for emerging cryptography that Advanced Encryption Standard (AES) hardware acceleration has had for established cryptographic primitives. While AES is a symmetric-key encryption standard,[1] since the introduction of the AES New Instructions (AES-NI) hardware instruction set in the early 2010s, there has been tremendous growth in cryptographic software taking advantage of hardware-accelerated AES for primitives other than encryption, including keyed hash functions and pseudorandom number generators.

We therefore espouse a synergistic approach of hardware designed for cryptography, which in turn is designed with hardware in mind. As one concrete effort, consider the complementary goals of FHE and ZKP.

Digital Object Identifier 10.1109/MC.2024.3481808
Date of current version: 9 January 2025

2469-7087/26 © 2026 IEEE

FHE guarantees the privacy of client data throughout the computation, and ZKP guarantees the integrity of the computation (while protecting the privacy of particular inputs as needed). There are many schemes for FHE, some more amenable to acceleration than others. The same is true for ZKP. Can ZKP schemes be tailored to fit a hardware accelerator for FHE? We argue in this article that the answer is yes.

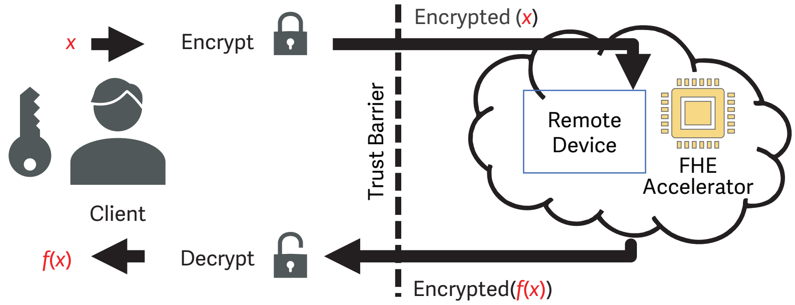


FIGURE 1. FHE can offload computation to an untrusted device securely.

BACKGROUND

Figure 1 shows how FHE enables the secure offloading of computation. The client wants to compute a function *f*(·) (for example, a deep learning inference) on some private data *x*. Computing *f*(·) locally may be too expensive for the client (for example, each inference requires either too many operations or a too-large model). To do this, the client encrypts *x* and sends it to an untrusted device, which computes *f*(·) on these encrypted data directly using FHE and returns the encrypted result to the client. FHE provides ideal privacy; even if the device is compromised, attackers cannot learn anything about the data *x* as they remain encrypted throughout.

ZKPs are an emerging family of cryptographic tools enabling one party (the prover) to prove to other parties (the verifiers) that a statement is true without requiring the prover to disclose any data to the verifiers. Figure 2 illustrates how ZKPs work. A prover generates a small proof of a statement and publishes it. Any verifier can download the proof and verify the statement cheaply. The proof is zero knowledge in that the private witness is not exposed to the verifier; however, this witness is an *optional* feature. One use case of ZKPs is complementary to that of FHE; a client wants to offload the computation of *f(x)* to a remote device. The device returns the output *y* and a proof of correctness that the client can use to verify that *f(x)* was computed correctly. Verifying the proof is orders of magnitude cheaper than computing the proof—and usually cheaper than computing *f(x)* locally. Unfortunately, proof generation, which happens in the remote device, is currently orders of magnitude more expensive than computing *f(x)*; this bottlenecks ZKP schemes and is a prime target for hardware acceleration (Figure 2). In summary, on untrusted devices, ZKP provides integrity of computation, whereas FHE provides privacy without integrity guarantees.

WE THEREFORE ESPOUSE A SYNERGISTIC APPROACH OF HARDWARE DESIGNED FOR CRYPTOGRAPHY, WHICH IN TURN IS DESIGNED WITH HARDWARE IN MIND.

HARDWARE ACCELERATION

Our team designed the F1[2] and CraterLake[3] FHE accelerators in 2021 and 2022, respectively. F1 speeds up shallow FHE programs, and CraterLake accelerates unbounded-depth FHE programs by 5,000× over a 32-core CPU and scales to large chips efficiently. We

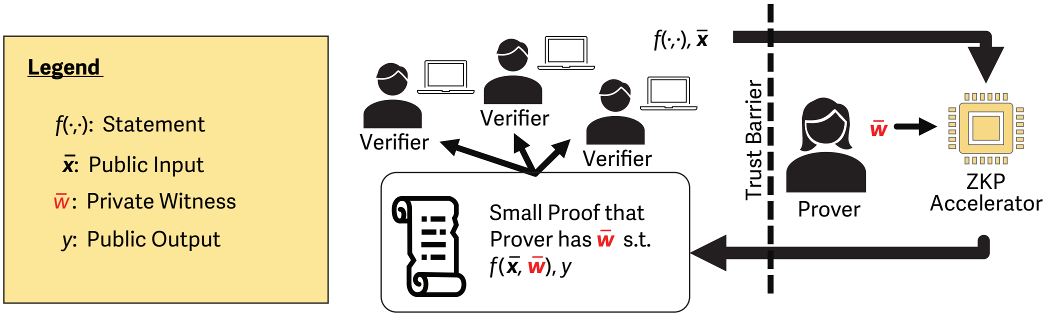

Legend
$f(\cdot,\cdot)$: Statement
$\bar{x}$: Public Input
$\bar{w}$: Private Witness
y: Public Output
Verifier
Verifier
Verifier
$f(\cdot,\cdot)$, $\bar{x}$
Trust Barrier
Prover
$\bar{w}$
ZKP Accelerator
Small Proof that Prover has $\bar{w}$ s.t. $f(\bar{x}, \bar{w})$, y

FIGURE 2. ZKPs allow a prover to convince other parties, called *verifiers*, of a statement f, optionally requiring private input w from the prover.

contributed several techniques that make this possible, including

1. a new extremely wide (2,048 lanes) vector uniprocessor architecture that spreads each vector operation across the chip, departing from prior vector multicore architectures
2. an efficient implementation of this architecture, for non-single optimized instruction, multiple data FHE operations, number-theoretic transforms (NTTs), and automorphisms by effectively decomposing them among distributed groups of lanes using a novel transpose network.

> *CRATERLAKE IS A CURRENT STATE-OF-THE-ART FHE ACCELERATOR, AND ITS FEATURES HAVE BEEN USED IN SUBSEQUENT DESIGNS.*

CraterLake is a current state-of-the-art FHE accelerator, and its features have been used in subsequent designs.

Suppose one wished to build a unified hardware accelerator that handles the full range of emerging cryptography, including FHE, ZKP, and PIR. Accelerators are expensive to design and build, so chips that can be used on many applications have a much higher probability of impact and wide adoption. At first glance, it seems challenging to build a single accelerator for this broad range of cryptographic protocols. Our insight is that, while diverse, many of these protocols share very similar computational characteristics; all work on large operands, require high-throughput modular arithmetic, and perform a similar set of primitive operations (like NTTs). These shared features, which make these protocols slow on CPUs and GPUs, can also enable a single accelerator to handle them.

As a first step toward a unified accelerator, we have designed a novel accelerator, NoCap,[4] that leverages hardware/algorithm co-design to achieve transformative speedups. NoCap generates proofs 586× faster than a 32-core CPU and 41× faster than PipeZK, a state-of-the-art ZKP accelerator. We leverage recent algorithmic developments to achieve these speedups; we identify and combine two recent hash-based ZKP algorithms, Orion and Spartan, which have similar performance on CPUs to the ZKPs targeted by prior accelerators, but are much more amenable to hardware acceleration. We chose Orion and Spartan rather than more popular elliptic-curve-based schemes (for example, Groth16) because the operations in these schemes have greater similarity to those in FHE schemes. Though Orion and Spartan result in larger proofs, we have shown that, for many applications, the end-to-end speedups (including prover time, proof transmission, and verification time) more than justify this size increase. We developed a novel hardware organization to exploit these acceleration opportunities; NoCap is a programmable vector processor with functional units tailored to the needs of hash-based ZKPs. We then combined Spartan and Orion as a novel way to form what we call the *Spartan+Orion ZKP*, which is an excellent fit for our accelerator; additional optimizations improve parallelism and reduce memory traffic. As a result, NoCap achieves speedups that enable new use cases for ZKP.

NoCap shares similarities with FHE accelerators because hash-based ZKPs and FHE schemes both consist of regular computations on large polynomials with modular integer coefficients. This results in similar functional units, for example, to perform element-wise arithmetic operations and NTTs, and paves the way for future chips that efficiently accelerate FHE, ZKPs, and other cryptographic protocols that rely on lattice-based cryptography.

We believe a successful marriage of hardware acceleration and cryptography has a high potential for impact; by lowering the cost of cryptographic techniques, these techniques will become an essential part of the secure cyberphysical systems of the future, saving billions of dollars and enabling security in a sustainable, energy-efficient way.

REFERENCES

1. J. Nechvatal et al., "Report on the development of the Advanced Encryption Standard (AES)," *J. Res. Nat. Inst. Standards Technol.*, vol. 106, no. 3, p. 511, Jun. 2001, doi: 10.6028/jres.106.023.
2. A. Feldmann et al., "F1: A fast and programmable accelerator for fully homomorphic encryption," in *Proc. 54th Int. Symp. Microarchit. (MICRO)*, Virtual Event, Greece, Oct. 2021, pp. 238–252, doi: 10.1145/3466752.3480070.
3. N. Samardzic et al., "CraterLake: A hardware accelerator for efficient unbounded computation on encrypted data," in *Proc. 49th Int. Symp. Comput. Archit. (ISCA)*, New York, New York, Jun. 2022, pp. 173–187.
4. N. Samardzic, S. Langowski, S. Devadas, and D. Sanchez, "Accelerating zero-knowledge proofs through hardware-algorithm co-design," in *Proc. 57th Int. Symp. Microarchit. (MICRO)*, Austin, Texas, USA, Nov. 2024.

SRINIVAS DEVADAS is the Webster Professor of Electrical Engineering and Computer Science at MIT, Cambridge, MA 02139 USA, where he has been on the faculty since 1988. Contact him at devadas@mit.edu.

DANIEL SANCHEZ is a professor of electrical engineering and computer science at MIT, Cambridge, MA 02139 USA, where he has been on the faculty since 2012. Contact him at sanchez@csail.mit.edu.

IEEE COMPUTER SOCIETY
CALL FOR PAPERS

GET DISCOVERED

Reach readers worldwide in publications indexed in major databases and optimized for search engines.

GET PUBLISHED
computer.org/cfp

IEEE COMPUTER SOCIETY
IEEE

EDITORS: Hyeran Jeon, hjeon7@ucmerced.edu
Adwait Jog, ajog@virginia.edu

This article originally appeared in IEEE micro vol. 45, no. 3, 2025

DEPARTMENT: MICRO FIRESIDE

Sipping Matcha of Security: A Fireside Chat With Mengjia Yan

Jianming Tong and Zishen Wan, *Georgia Institute of Technology, Atlanta, GA, 30332, USA*

IEEE Micro met with Prof. Mengjia Yan, an associate professor at Massachusetts Institute of Technology's (MIT's) Electrical Engineering and Computer Science Department and a member of the Computer Science and Artificial Intelligence Laboratory (see Figure 1). Dr. Yan received the 2024 Technical Committee on Computer Architecture (TCCA) Young Architect Award in recognition of her contributions to the analysis of security mechanisms and vulnerabilities. In this interview, Dr. Yan reflects on her journey in secure computer architecture, from her early inspirations as an undergraduate to her award-winning contributions to architectural defenses. She discusses the evolution of the field, from classical microarchitectural optimizations to interdisciplinary research that spans system design, programming languages, and formal methods.[a]

IEEE Micro: First, congratulations on receiving the TCCA Young Architecture Award in recognition of your outstanding contributions to secure architecture and the formalization of architectural defenses. Could you share some highlights from your award speech for those who couldn't attend?

Yan: Sure. My speech was primarily about expressing my gratitude to everyone who helped me along the way. I thanked my advisor—who got me started in research when I was completely naive—and many others, including Chris Fletcher, who is my role model. I also expressed my appreciation to all my students. In those early days, my mentoring skills were very raw, so I was incredibly grateful for their trust and patience. I was so nervous that I worried I might forget someone's name, even though I eventually recalled it. There were so many people who supported me, and I just wanted to convey my heartfelt thanks.

FIGURE 1. *IEEE Micro* interviewers Jianming Tong (left) and Zishen Wan (right) with Prof. Mengjia Yan (middle) at her office at the Massachusetts Institute of Technology.

IEEE Micro: We also heard you're a big fan of matcha. Could you explain the meaning behind your group's name, which stands for "Microarchitecture, Attacks, and Challenges"? After three years of the group's evolution, is there a story behind it?

Yan: Every group wants a name that defines its identity, but finding the perfect one is always challenging. One day, after a relaxed group lunch following a meeting—when many of us hadn't even had breakfast—we decided to go to our favorite Chinese spicy restaurant, Sumiao, near Kendall Square. Over lunch,

DISCLAIMER

Any views or opinions represented in this article are personal, belong solely to the interviewee, and do not represent those of *IEEE Micro* or its parent organization, IEEE.

---

[a] The recording of this interview can be found at https://www.youtube.com/@TCuArch.

0272-1732 © 2025 IEEE. All rights reserved, including rights for text and data mining, and training of artificial intelligence and similar technologies.
Digital Object Identifier 10.1109/MM.2025.3572585
Date of current version 27 June 2025.

as we chatted, we spontaneously came up with the idea of “Matcha.” Since many of us love matcha and I appreciate the spirit it conveys—green, inclusive, and refreshing—it seemed like the perfect fit. I wasn’t looking to name the group solely based on our current research because our topics can change over time. Instead, I wanted a name that reflects our inclusive nature, our willingness to embrace our limitations, and our overall positive energy. Whenever I mention that our group is called Matcha, people ask what it stands for, and I explain “Microarchitecture, Attacks, and Challenges” or something similar. Ultimately, the name represents our goal to be an inclusive and happy team.

IEEE Micro: That sounds fantastic. From your perspective, how has the field and community of computer architecture evolved since you first started as a graduate student?

Yan: Back when I was a student, fields like machine learning and security were just emerging, but what stands out to me now—more than 10 years later—is the growing emphasis on interdisciplinary research. As an undergrad and a junior Ph.D. student, I deeply appreciated the classical microarchitecture optimizations, where you made delicate, clever tradeoffs among performance, energy efficiency, and hardware complexity. These works felt almost like art, yet they’re much rarer in today’s conferences, as the focus has shifted to more trendy topics.

For me, working in security taught me that to truly understand and address a problem, you need to look beyond the computer architecture layer and examine the entire system stack. Nowadays, I intentionally explore adjacent layers—such as system design and programming languages—to tackle security issues more effectively. This evolution means that students now need to develop not only deep expertise in computer architecture, but also a broader knowledge base across disciplines. It’s very much like developing a T-shaped skill set: combining depth with breadth. While this approach is more engaging, it also presents additional challenges.

IEEE Micro: Thank you for sharing that insight. Could you give us an overview of your journey from when you first became interested in computer architecture research to today?

Yan: It’s a long story. When I was an undergrad—as is the case for most undergrads—I found programming fun and interesting, so I decided to study computer science. I got lots of fun from the lab and problem sets. There was one good lecturer who taught computer architecture in a very systematic way, which is different from what you could read from textbooks. His description of the tradeoffs really sparked my interest. That was my initial exposure to computer architecture.

Later, when I joined as a Ph.D. student working with my advisor, Josep, I barely knew anything. Josep mentioned that security was an important topic, and I trusted him and began working on it. However, after two or three years, I started feeling that although security was indeed an important problem, as a computer architect working on security, I felt somewhat like a second-class citizen because software security seemed to be prioritized over hardware efforts. For example, many researchers at that time focused on designing architectural solutions to improve the performance of software mitigations. Imagine a shadow stack used as a software mitigation against buffer overflow attacks; but it was too slow in software, so we want to implement it in hardware. This kind of research was mainstream at that time. It’s very nice, but it still felt secondary; as software people would say, “Oh, you’re just optimizing performance for us.”

I also recall a time when my advisor asked me to write a statement for applying to attend a workshop. While working on the statement, I couldn’t convince myself that the problem that I worked on was truly important. As I was generally interested in security [having taken security classes at the University of Illinois at Urbana-Champaign (UIUC)], I wanted to find a problem that I could wholeheartedly believe was worth my time.

One of my lab mates presented a paper called “Get Off My Cloud,” a highlight paper at that time addressing side-channel attacks in the wild cloud (rather than in a controlled environment). I found it super interesting. I later had a long conversation with Fangfei Liu (now a researcher at Intel’s Research Lab security team) at the International Symposium on Computer Architecture (ISCA). We chatted at receptions and on the bus, and she encouraged me and explained that entering side-channel research wasn’t too difficult, even though I was initially confused by all the crypto algorithms and attacking these crypto implementations. Her enthusiasm really helped me get started. I began working on it around my third year as a Ph.D. student, and things moved along. My first paper’s publication boosted my confidence, my interest grew, and both Josep and later Chris Fletcher (who joined UIUC as a junior faculty) were incredibly supportive. I felt very lucky that at the right time of my life, good people always come up. Chris has gone through a very solid training on security, having a profound understanding of the field. He is also very enthusiastic about the research, which has an influence on me.

Now that I’ve started my own research group, working with my students has been very energizing.

As I mentioned earlier, I now want to explore not only the computer architecture layer but also its neighboring layers. For example, how to connect it with system design, low-level operating systems, programming languages, and formal methods, i.e. the whole end-to-end process. Essentially, I view the process as a full cycle: design, verification, and then conducting attacks to discover new vulnerabilities or blind spots in threat models.

IEEE Micro: What do you think is the one thing that enables you to do meaningful, impactful work with people from different backgrounds? What started these collaborations?

Yan: Regarding collaboration, I want to share that no one is naturally collaborative. In the first four years when I was a Ph.D. student, I didn't collaborate with anyone, at least not in a formal sense. In the early stage, I mostly worked independently, seeking help from my lab mates when needed. Josep fostered a lab culture where we helped each other (especially with my English), so I received plenty of feedback to polish my writing. I often paired with a senior student like Yasser, and we learned from each other. When Chris joined, we found our research interests aligned and the collaboration happened naturally. Sometimes I meet some students who are a little bit bothered by not having too much collaboration and doing solo research. I want to say that it's totally fine. Once you find someone with aligned interest and the scheduling/arrangement works out, the collaboration will just happen naturally. More importantly, only such collaboration would last long term.

When I started as a faculty member, I sometimes found that over-collaborating can be distracting, especially when I'm still trying to find my mentoring style. The community often thinks that more collaboration is always better, but it really depends on your stage and the type of projects you're pursuing. For junior students, simply learning from a senior student or your advisor may already be a sufficient setup. Once you're reaching a certain stage, long-lasting collaboration may happen automatically.

As a faculty member, I have collaborated primarily with Joel Emer (whom I got to know during my Ph.D. internship), Thomas Bourgeat from EPFL (École Polytechnique Fédérale de Lausanne) (a former MIT student), and Sharad Malik from Princeton. I found I like such collaborations where we complement each other. Every time I talk to them, I learn so much. This kind of collaboration is very fruitful. For instance, Sharad is an expert on SAT Solver and understands its internals, while I use it as a blackbox tool. When I talked to him, I found he looks at problem from a very different perspective. Figuring out his view when our terminology is leading to misunderstandings is very enriching. I also checked his slides from the hardware security and verification course he taught at Princeton, and he kindly walked me through them to ensure I understood. Every time we meet, I learn something new. Such a type of mutually beneficial collaboration is what I aim for keeping long term.

IEEE Micro: I love your favorite metaphor: Attacking hardware is like robbing a bank. People find a way in and get the money out. Matcha finds its way into the hardware early and then tells vendors how to secure it to avoid money loss. The entire research topic is both critical and intriguing. Could you give us an overview of your journey: Why did you start this line of work?

Yan: When I started doing attack work, not many people were pursuing offensive security research or attacks within the computer architecture community. In fact, when we submitted papers on attacks, they were often rejected because attacks were seen as somewhat "evil." I must thank Michael Bailey from UIUC—now a faculty member at Georgia Tech and head of the security college (if I recall correctly)—who profoundly influenced me. He taught an excellent undergraduate security class at UIUC where all the lab assignments involved implementing hands-on attacks. In class, his theoretical description of attacks sounded intuitive and straightforward, but implementing them on a real system (rather than in a simulator) required you to figure out all the intricate details, as if you were putting together a jigsaw puzzle. I took his class in my first or second year as a Ph.D. student and was so obsessed with it that I always finished the lab assignments ahead of time because it was just so much fun.

That experience drove me to develop my own hardware security class at MIT with my student Joseph Ravichandran and a couple of other students. Many of my students have contributed to this course because we all find it very enjoyable to create hands-on hack experiences and share that fun with others. We firmly believe that you learn best by doing rather than by merely listening to lectures. We put a lot of effort into designing lab assignments on real hardware rather than simulators, because while software courses can simulate attacks in virtual machines, hardware attacks are very hardware-specific. We worked hard to make these attacks practical for senior undergraduates, junior Ph.D. students, and master's students, and the feedback has been overwhelmingly positive. I don't expect every student to pursue research in hardware security, but I believe that thinking offensively about systems is an excellent way to gain a deep understanding. For example, if you try to attack a cache, you learn about its inner workings in great detail.

IEEE Micro: What are you looking forward to the most in hardware attacks and mitigations over the next 10 years? Are quantification and formal verification what you are aiming for? Also, do you see these developments applying to single devices, clouds, or even distributed clusters?

Yan: When we approach a security problem, we try to generalize our solutions so that they target various scenarios; for instance, side channels can affect both cloud setups and personal laptops. I'm very curious about trying something new because, in the area of side-channel attacks, we already have a fairly good understanding. There are occasional news reports about interesting attacks, and while I might appreciate one, it doesn't really shock me anymore.

I'm really looking forward to discovering new problems beyond speculative execution attacks and side channels. The moment when people realized that speculative execution—a long-standing performance optimization—could be exploited for attacks (like Spectre/Meltdown) was truly shocking and opened up entirely new research directions. In the next 10 years, I hope to see entirely new threats emerge. Security is always evolving; new devices, new setups, and new technologies make life more convenient but inevitably introduce new security problems. I'm particularly interested in seeing an entirely new class or category of security issues emerge beyond the well-explored side channels.

SO FAR WE HAVEN'T SEEN A BREAKTHROUGH WHERE MACHINE LEARNING OUTPERFORMS HUMANS IN MICROARCHITECTURAL TASKS.

IEEE Micro: My experience with side-channel attacks has mostly been within a single system. Is there any potential for discovering unknown vulnerabilities in clusters or even distributed systems?

Yan: Network side channels have been studied extensively. Even before Spectrum/Meltdown, researchers knew that features like packet size or communication patterns—even when packets are fully encrypted—can reveal sensitive information. However, these issues are generally addressed within the applied cryptography domain rather than computer architecture. In applied crypto, the techniques mainly involve capturing encrypted packets and applying complex analyses, without needing the fine-tuned system-level adjustments that hardware side channels require.

IEEE Micro: What do you think the cost of hardware security will be in the future? And is there an automated way to discover potential vulnerabilities and defend against them?

Yan: Cost is indeed a major issue. Although I haven't been in the industry, from my perspective, aside from cost, hardware complexity is even more important. For instance, in Linux, enabling certain software mitigations for kernel protection can cause about a 1.5× performance slowdown, a tradeoff people are willing to accept based on risk. However, the more complex a defense mechanism is, the harder it is to ensure it is bug-free. Introducing a complex mitigation into hardware might even introduce additional problems. Therefore, ensuring and reasoning about the security, correctness, or "cleanliness" of mitigation is crucial, and simpler defenses are generally preferable. Unfortunately, modern hardware is already extremely complex, which is why I'm passionate about this research direction; it might be the current bottleneck.

IEEE Micro: Do you think there is an automated way to discover bugs in super-complex hardware and automatically secure it using large language models? Is this a worthwhile area to pursue in the future?

Yan: Given the current power of large language models, nothing is impossible. However, "automation" can take many forms. For instance, formal methods offer a degree of automation via techniques like model checking. If you set up a problem correctly—defining the properties and assertions accurately—the model checker can automatically search for bugs. In fact, people sometimes refer to model checking as a "push-button" technique in the verification community. That said, there is still a significant gap between simply prompting a tool and rigorously configuring your problem for exhaustive checking. Much of the manual effort lies in understanding the problem well enough to formulate the correct assertions so that the model checker can explore all possible inputs and trigger sequences leading to bugs or leakage. This is essentially an automatic search problem using exhaustive mathematical techniques and solvers.

On the other hand, machine learning-based automation might add some "magic" to the process. I think it's a promising direction—there's already some research in this area—but so far we haven't seen a breakthrough where machine learning outperforms humans in microarchitectural tasks. For example, many software courses worry that students might use tools like ChatGPT to cheat on lab assignments. We haven't seen that issue in our hardware security class because our attack code requires detailed interaction with hardware. I wouldn't be surprised if, in the future, tools like ChatGPT could generate high-quality, detailed cache attacks that work perfectly on our machines, but for now, they remain limited.

IEEE Micro: Do you think that vulnerabilities primarily arise from insufficient test coverage in the industry, or are there other contributing factors?

Yan: In my class, I tell students that bugs and security vulnerabilities typically arise from multiple causes. You might think that a bug occurs when an implementation doesn't match its specifications. However, in industry, I often see that the specification itself is insecure. The designers say, "This is how I want the system to work," and the RTL (register transfer level) faithfully implements that design, even if the design has inherent vulnerabilities. That's the common case I've observed.

This is why, in our verification work (for example, in a paper presented at ISCA in 2023), we focused on early-stage microarchitecture modeling for security. Typically, early-stage models are created for performance estimation (like for Gem5), where you only consider how the system should function. However, these models often lack the critical details needed to verify security with formal methods. In that work, we argued that even if your high-level microarchitecture is secure, you still need to ensure that your RTL implementation is secure as well.

Generally, industry mistakes fall into two categories:

1) *Under-specification:* For example, when designing an x86 processor, the only requirement might be to implement the instruction set architecture (ISA) correctly. The ISA doesn't mention aspects like speculative execution, so those features are left as "free variables" with no security constraints.
2) *Implementation issues:* Once a security feature is specified, the implementation must be carried out very carefully and tested thoroughly. Initially, many vulnerabilities arise simply because security features were not specified at all. As security awareness increases, the focus shifts to ensuring that the implementations are correct and properly covered by tests.

IEEE Micro: How do you view the challenges of silent data corruptions (SDCs), especially in the context of generative artificial intelligence (AI) and large-scale data centers?

Yan: It's very challenging. I must say that the reliability direction—such as our Reliably project—is entirely led by my students, Peter Deutsch and Vincent (one of our top students who will soon join as a postdoc). I admit I know far less about reliability than they do; I mainly offer support while they lead the work.

The tricky part with SDCs is that they occur infrequently, yet they are frequent enough to be problematic in large-scale data centers, while on a single machine, they're nearly impossible to observe directly. As a result, we rely on fault models. Although there are well-established models, they may not fully explain what is happening now, so the actual root cause remains mysterious. Often, these root causes lie in the physical or circuit layers, and as computer architects, we must make assumptions about these lower layers, which can be uncomfortable. We end up consulting experts in fault models.

The challenge deepens when we consider AI and machine learning accelerators. Even if a fault causes a program-visible bit flip in a model parameter (a weight), it's not straightforward to determine whether the model's behavior will be affected. This adds another murky layer to the problem. Fault injection simulations can take an extremely long time, especially when run across many models, making authentic testing nearly impossible. Moreover, this area is highly industry-dependent. It would be beneficial if the industry could share more information or allow academia access to faulty machines for better calibration. However, establishing such connections requires significant mutual effort, and it isn't easy.

IEEE Micro: Thank you so much, Professor Yan, for joining us today and for this insightful discussion. I hope our conversation is helpful for both students and the computer architecture community.

Yan: Thanks a lot. These have been very good questions. Thank you for chatting with me—I really appreciate it.

JIANMING TONG is a Ph.D. degree candidate at the Georgia Institute of Technology, Atlanta, GA, 30332, USA, under the guidance of Prof. Tushar Krishna. He is also a visiting researcher at the Massachusetts Institute of Technology, Cambridge, MA, 02139, USA, and a student researcher at Google. Contact him at jianming.tong@gatech.edu.

ZISHEN WAN is a Ph.D. candidate at the Georgia Institute of Technology, Atlanta, GA, 30332, USA, advised by Profs. Arijit Raychowdhury and Tushar Krishna. Contact him at zishenwan@gatech.edu.

Get Published in the *IEEE Transactions on Privacy*

This fully open access journal is soliciting papers for review.

IEEE Transactions on Privacy serves as a rapid publication forum for groundbreaking articles in the realm of privacy and data protection. Submit a paper and benefit from publishing with the IEEE Computer Society! With over 5 million unique monthly visitors to the IEEE Xplore® and Computer Society digital libraries, your research can benefit from broad distribution to readers in your field.

Submit a Paper Today!

Visit computer.org/tp to learn more.

IEEE COMPUTER SOCIETY | IEEE

EDITOR: Phil Laplante, IEEE Fellow, plaplante@psu.edu

DEPARTMENT: SOFTWARE ENGINEERING

This article originally appeared in
Computer
vol. 58, no. 7, 2025

Software Testing in the Generative AI Era: A Practitioner's Playbook

Douglas C. Schmidt, *William & Mary*

This article presents a practitioner-focused playbook on how generative artificial intelligence is transforming software testing by automating tasks like test generation, simulation, and anomaly detection, while also introducing new challenges like hallucinations, bias, and nondeterminism.

Software testing is undergoing a paradigm shift due to generative artificial intelligence (AI). Imagine test cases that write themselves by analyzing your documentation and code, user behaviors being simulated before any real user ever touches the software, and continuous feedback guiding development in real time. Generative AI—especially large language models (LLMs) trained on vast data sets—is turning this vision into reality. These tools can analyze your requirements and code to produce test cases automatically, synthesize realistic test scenarios on the fly, and even monitor running systems for anomalies. The promise is compelling: you and your team can test faster, smarter, and more thoroughly.

However, these AI-augmented advances also introduce new risks. The same AI tools that boost testing can produce unpredictable behavior—from biased outputs or "hallucinated" results to inconsistent, nondeterministic responses that are hard to reproduce or verify. In high-stakes domains like health care, finance, and defense, where lives and missions are on the line, ensuring the reliability and trustworthiness of AI-augmented testing is paramount. Traditional quality assurance (QA) practices must adapt to issues like AI bias, unexpected outputs, and regulatory compliance.

This article explores how generative AI is reshaping software testing—the benefits it offers, the challenges it raises, and how you can adapt. With AI coding assistants like OpenAI's Codex, GitHub Copilot, and Meta's Code Llama increasingly integrated development environments, continuous integration/continuous delivery (CI/CD) pipelines, and cloud platforms, teams need to adjust quickly. Understanding how these tools work—and recognizing when they don't—is becoming a critical skill. The guidance here draws on my recent stint as the Director of Operational Test and Evaluation and related collaborations with colleagues.[1,2,3]

HOW GENERATIVE AI IS AUTOMATING AND SCALING TESTING TODAY

Generative AI is extending the reach of test automation. Instead of manually writing every test or trying to guess all possible user behaviors, you can offload many tedious and error-prone tasks to AI. The AI-augmented testing capabilities shown in Figure 1 serve as accelerators for software QA, and are described as follows.

Automatic test case generation

Tools like Diffblue Cover and CodiumAI can analyze your code and generate unit tests across a variety of languages. They can even integrate with your continuous integration pipelines (for example, Jenkins or GitLab) for automatic execution.[4] For instance, an LLM can read a specification and produce a suite of tests covering the expected behaviors. This automation helps validate intended functionality early and targets known weak spots. It also expands coverage while reducing the manual effort of writing tests.

Digital Object Identifier 10.1109/MC.2025.3562940
Date of current version: 27 June 2025

2469-7087/26 © 2026 IEEE

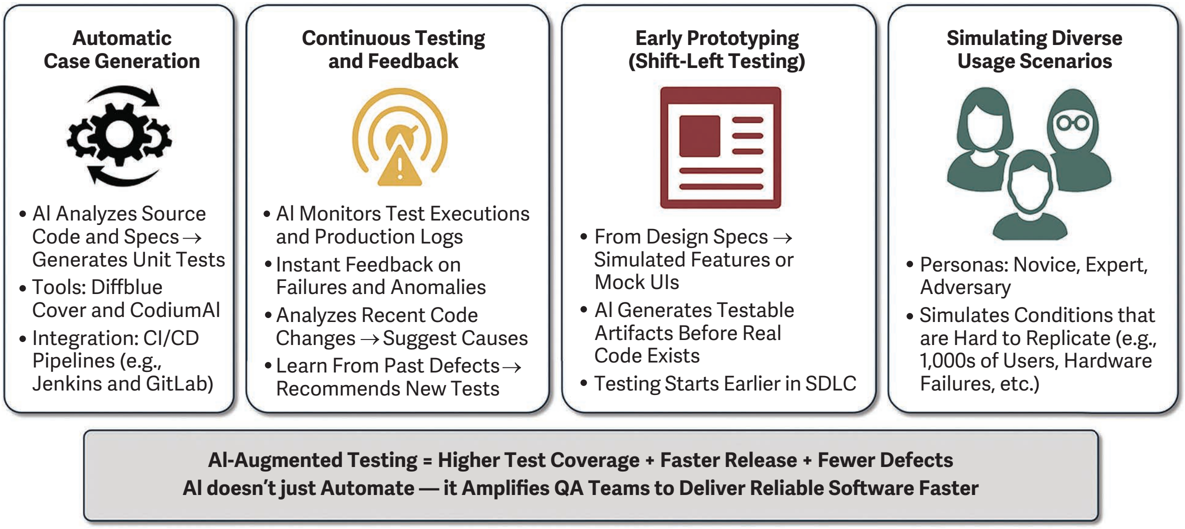

Automatic Case Generation
• AI Analyzes Source Code and Specs → Generates Unit Tests
• Tools: Diffblue Cover and CodiumAI
• Integration: CI/CD Pipelines (e.g., Jenkins and GitLab)
Continuous Testing and Feedback
• AI Monitors Test Executions and Production Logs
• Instant Feedback on Failures and Anomalies
• Analyzes Recent Code Changes → Suggest Causes
• Learn From Past Defects → Recommends New Tests
Early Prototyping (Shift-Left Testing)
• From Design Specs → Simulated Features or Mock UIs
• AI Generates Testable Artifacts Before Real Code Exists
• Testing Starts Earlier in SDLC
Simulating Diverse Usage Scenarios
• Personas: Novice, Expert, Adversary
• Simulates Conditions that are Hard to Replicate (e.g., 1,000s of Users, Hardware Failures, etc.)
AI-Augmented Testing = Higher Test Coverage + Faster Release + Fewer Defects
AI doesn't just Automate — it Amplifies QA Teams to Deliver Reliable Software Faster

FIGURE 1. Accelerating software testing with generative AI. UI: user Interface; SDLC: software development life cycle.

Continuous testing and feedback

Generative AI supports continuous testing by monitoring systems and providing rapid feedback. Machine learning models can watch test executions or even live production logs to spot anomalies and failures in real time.[6] If a test fails, an AI assistant might immediately analyze the recent code changes and suggest likely causes. AI can also detect patterns in past defects to recommend new test cases or fixes. In modern Development, Operations, and Security pipelines, every code commit can trigger AI-generated tests and anomaly detection, providing instant feedback and catching issues earlier.

Early prototyping and “shift-left” testing

Generative AI blurs the line between development and testing by enabling earlier prototyping of features. Rather than waiting for a fully integrated system to begin testing, an AI tool can create a mock interface or simulate a new feature from just a design description. This “shift-left” approach allows you to exercise and evaluate components before writing any real code, thus shortening the QA feedback loop and catching design flaws sooner in the development process.

Simulating diverse usage

Generative AI enables you to simulate a wide range of user behaviors and environmental conditions. For instance, you might prompt an LLM to act as various personas[5] (for example, a novice user, an expert, even an adversarial attacker) to see how your software handles different inputs and sequences of actions. AI-augmented simulations can also mimic conditions that are impractical to reproduce manually (imagine thousands of virtual users or sudden sensor failures). Testing under these diverse scenarios gives you greater confidence in the robustness of your system.

These AI-augmented techniques can widen your test coverage (including edge cases), catch bugs earlier, and speed up the feedback cycle. By offloading tedious work to AI and injecting intelligence into testing, generative AI becomes a force multiplier for QA, enabling you to deliver more reliable software faster.

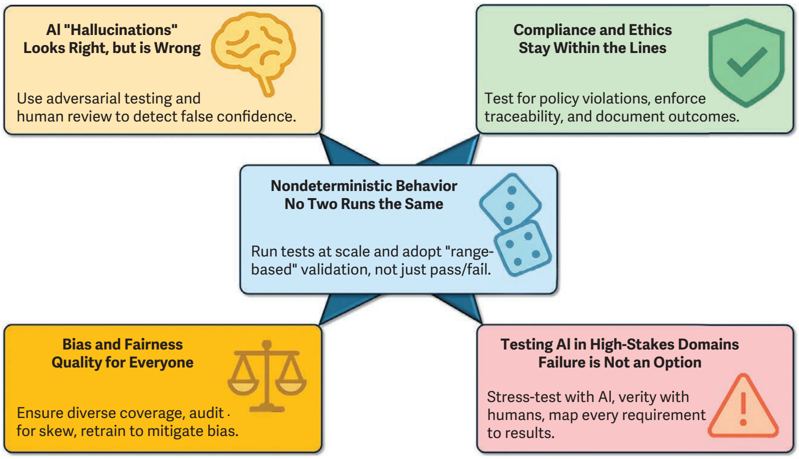

AI "Hallucinations"
Looks Right, but is Wrong
Use adversarial testing and human review to detect false confidence.
Compliance and Ethics
Stay Within the Lines
Test for policy violations, enforce traceability, and document outcomes.
Nondeterministic Behavior
No Two Runs the Same
Run tests at scale and adopt "range-based" validation, not just pass/fail.
Bias and Fairness
Quality for Everyone
Ensure diverse coverage, audit for skew, retrain to mitigate bias.
Testing AI in High-Stakes Domains
Failure is Not an Option
Stress-test with AI, verity with humans, map every requirement to results.

FIGURE 2. Challenges of AI-augmented software testing.

CHALLENGES OF AI-AUGMENTED TESTING

Along with the benefits of AI-augmented testing come new challenges, there are several key areas you need to address, which are shown in Figure 2 and described as follows.

> *GENERATIVE AI BLURS THE LINE BETWEEN DEVELOPMENT AND TESTING BY ENABLING EARLIER PROTOTYPING OF FEATURES.*

AI "hallucinations"

AI systems sometimes produce answers or code that look correct and confident but are completely wrong.[7] For example, an AI-generated test might assert an output value that appears plausible (perhaps it even aligns with the documentation) but is incorrect. If you rely on that test, it could pass or fail for the wrong reasons. In short, you can't blindly trust AI-generated outputs—especially for critical logic—without verification. Keep a human in the loop to review important AI-generated tests and results.[8] Likewise, perform adversarial testing; for example, feed malformed or malicious inputs to the AI and see if the system goes off track. Your goal is to ensure the system handles uncertainty properly, for example, by indicating when it doesn't know an answer instead of confidently fabricating one.

Nondeterministic behavior

Unlike traditional software, AI components might not behave the same way every time. A conventional function will produce identical outputs for identical inputs, but an AI model might respond slightly differently on each run, which makes it hard to consistently reproduce results. For example, if a test passes once but fails on the next run, is that due to randomness or a real bug? You may need to rerun certain tests numerous times to understand the range of possible outcomes. Even small input tweaks or data updates can shift an AI's responses, so watch out for model drift—when an AI's behavior changes over time as its data or parameters evolve. Ensuring quality for nondeterministic systems means gathering far more test data and often defining new metrics. It's a shift away from the traditional "pass/fail" mindset toward measuring whether the AI's performance stays within an acceptable range of reliability.

Compliance and ethics

In regulated domains, you must ensure that AI systems comply with all relevant laws and ethical standards, for example, a healthcare AI should never leak patient data. Your testing should include scenario-based checks that exercise policy constraints, for example, check that AI never does something it shouldn't, such as make an unauthorized decision or reveal sensitive information. Emerging regulations like the European Union's AI Act will likely mandate evidence of an AI system's fairness, transparency, and risk controls. AI tools can assist by scanning for compliance issues,[9] but ultimately, you and your team are accountable for ensuring that AI-augmented software stays within legal and ethical bounds.

Bias and fairness

Generative models learn from their training data, so any biases in that data can creep into AI-generated tests or outputs. If the training data underrepresents certain user groups or conditions, the AI might overlook those scenarios, leading to blind spots in quality. So don't just ask “Does it work?”—also ask “Does it work for everyone using our software?” In practice, you’ll need to use diverse inputs, cover underrepresented scenarios, and watch for skewed or unfair results. If you detect bias, you may need to adjust your test data or retrain the model to improve its fairness, even if that takes extra effort. These considerations broaden the notion of software QA to include factors like fairness, explainability, robustness under uncertainty, and ethical integrity. They aren’t insurmountable but addressing them requires evolving your testing strategies—especially in safety-critical environments where even a small AI misstep can be disastrous.

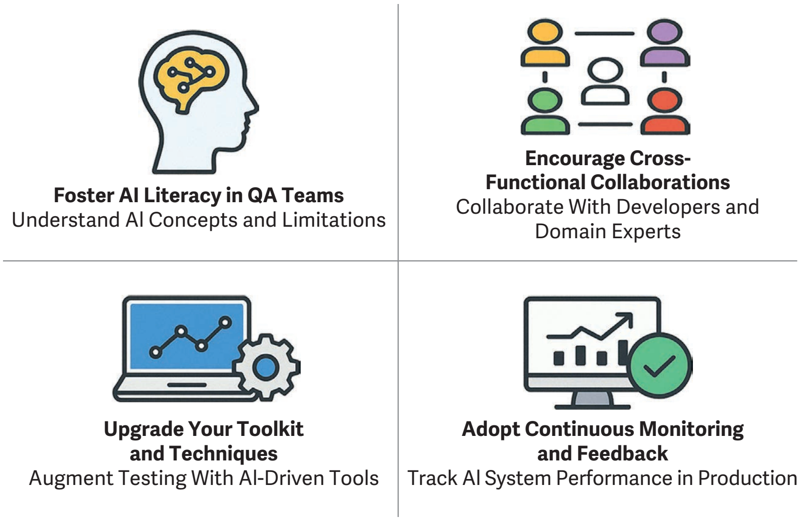

Foster AI Literacy in QA Teams
Understand AI Concepts and Limitations
Encourage Cross-Functional Collaborations
Collaborate With Developers and Domain Experts
Upgrade Your Toolkit and Techniques
Augment Testing With AI-Driven Tools
Adopt Continuous Monitoring and Feedback
Track AI System Performance in Production

FIGURE 3. Adapting testing practices for the AI era.

Testing AI in high-stakes domains

The stakes are elevated in high-stakes domains like defense, aerospace, and health care, which can’t tolerate failures in AI-augmented systems. A flaw in an autonomous emergency response drone’s software or a medical decision-support AI could be deadly. In these cases, you must demonstrate that the system works correctly even under extreme or unusual conditions. Generative AI can help by simulating numerous variations of scenarios (for example, different operating conditions, emergency situations, rare edge cases) to stress-test the system and expose weaknesses.

Even so, rigorous human oversight is essential, so “trust but verify” at every step. If AI-augmented testing tools report everything passed, you should still spot-check the results to identify edge cases the AI might have missed. In high-stakes domains, even rare oversights can be unacceptable. Traceability is also crucial, so document how each requirement maps to specific tests/outcomes to justify your test coverage and explain test results to stakeholders and regulators. Whenever AI is involved in a regulated, safety-critical system, be prepared to show exactly what was tested and verify nothing was left to chance.

The lesson from high-stakes domains is clear: use AI to amplify your testing, but maintain strict human oversight and accountability. Reliability, transparency, and safety must remain paramount whenever AI is part of a mission-critical system.

ADAPTING TESTING PRACTICES FOR THE AI ERA

To harness generative AI in testing while mitigating its risks, you’ll need to evolve your QA practices. Several ways to update your approach for the AI era are shown in Figure 3 and described as follows.

Foster AI literacy in QA teams

Ensure your testing team learns how AI works—and where it can fail. Testers should learn the basics of model training and evaluation, familiarize themselves with common AI failure modes, and adopt prompt engineering techniques and patterns[5] to apply LLMs effectively. This knowledge helps them use AI-based tools robustly and recognize when an AI’s output might be misleading. Encourage your team to practice crafting prompts and critically analyze AI-generated results. With greater AI literacy, your team can treat AI as a powerful assistant under their supervision, rather than a mysterious black box.

Encourage cross-functional collaboration

Testing AI-infused systems isn’t just the QA team’s job. Developers, testers, data scientists, and domain experts should all collaborate on AI-augmented projects. Involve QA early in the design of AI features so

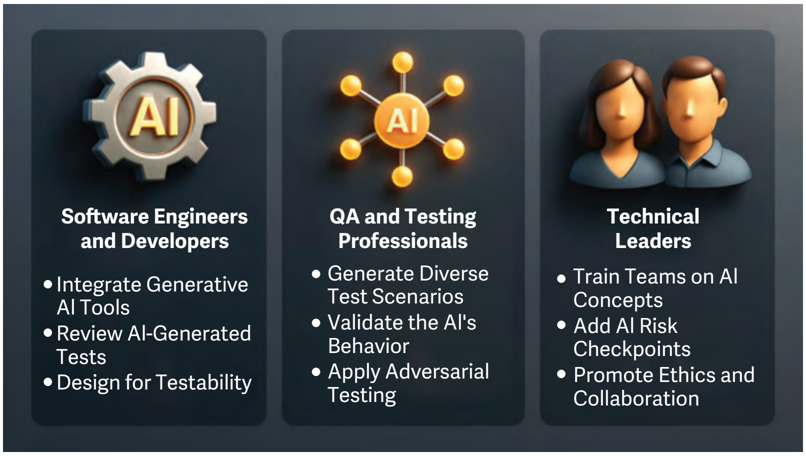

AI
Software Engineers and Developers
• Integrate Generative AI Tools
• Review AI-Generated Tests
• Design for Testability
AI
QA and Testing Professionals
• Generate Diverse Test Scenarios
• Validate the AI's Behavior
• Apply Adversarial Testing
Technical Leaders
• Train Teams on AI Concepts
• Add AI Risk Checkpoints
• Promote Ethics and Collaboration

FIGURE 4. Testing roles for AI-augmented software systems.

they can advise on testability and anticipate problems. Likewise, developers should build transparency and test hooks into AI components, for example, logging an AI model's key decision factors for later review. Involve domain experts to ensure test scenarios reflect real-world conditions, for example, have medical doctors contribute cases to test a healthcare AI or have financial experts suggest scenarios to test fintech AI. Breaking down silos and sharing knowledge across roles yields more robust testing strategies for AI-augmented systems.

Upgrade your tool kit and techniques

Keep your proven QA methods but augment them with AI-augmented approaches. For example, introduce adversarial testing (to try "breaking" the AI with malicious or unexpected inputs) and apply statistical analysis to interpret test results since AI outputs are probabilistic rather than deterministic.[3] You may need new metrics to quantify qualities like bias or uncertainty in the AI's output. Likewise, experiment with emerging AI-augmented test generation and analysis tools to see how they can boost your productivity. By expanding your toolbox, you can better cover AI-specific failure modes and make your testing regime more effective.

Adopt continuous monitoring and feedback

Don't treat testing as a one-and-done phase. After deployment, continue to monitor your AI components in production and feed those insights back into your tests. AI-augmented monitoring tools (such as Amazon DevOps Guru or Sentry) can watch production systems for anomalies, performance drift, or new types of errors, then alert your team. If a model's performance degrades over time, be ready to update test cases or retrain the model as needed. Schedule regular audits of AI outputs to catch issues that only appear with production deployments. AI can automate parts of this monitoring and analysis, but human experts should review the results and guide improvements. Maintaining this feedback loop will keep your AI systems reliable long after deployment.[10]

By putting these strategies into practice, you can turn generative AI into a real asset in your QA process. Testing will become more adaptive and comprehensive, keeping pace with AI-augmented development while upholding high standards of quality and trust.

YOUR AI TESTING PLAYBOOK: WHAT TO DO NEXT

Generative AI is rapidly reshaping software testing and early adopters are already seeing significant benefits. Autogenerated tests and large-scale simulations are catching bugs that previously slipped through manual efforts and faster feedback loops are helping teams iterate quickly. When applied properly, these techniques can yield more reliable software and shorter development cycles by enabling smarter testing.

At the same time, you need a clear view of AI's limitations. An AI-augmented system is only as good as the data and algorithms behind it, so it can still exhibit bias, produce incorrect results, or behave unpredictably. Human judgment, disciplined testing practices, and strong processes remain essential to validate and rein in AI contributions. Remember to "test the tester" by building in safeguards to catch mistakes or quirks in the AI tools themselves. By updating your practices and skill sets to account for these issues, you can harness the benefits of AI while mitigating its risks.

So what should you do now? Figure 4 shows a few concrete next steps for different roles on your team, which are explained as follows.

- *Software engineers and developers*: Integrate generative AI tools into your development and testing workflow to boost coverage and catch issues early. Use AI to generate unit tests and to simulate complex user flows. Don't assume the AI is always right—review and refine every AI-generated test case or code suggestion. Design your software with testability in mind, for example, add logging and hooks around AI-augmented features so you can verify their outputs. Let AI handle the tedious parts of testing but apply your judgment to discern what's correct.
- *QA and testing professionals*: Leverage AI as a partner to extend your testing. Use AI tools to generate diverse test scenarios and synthetic data, uncovering bugs that traditional methods might miss. Focus on validating the AI's behavior, for example, look for biased outputs or hallucinations and apply adversarial testing to stress-test AI-augmented features. Champion new types of checks in your test plans (for example, include explicit fairness and bias tests for AI outputs). Orchestrate the best of both worlds—use AI to work more efficiently while you focus on ensuring accuracy and integrity.
- *Technical leaders*: Create an environment that integrates AI into quality assurance thoughtfully. Invest in training your teams on AI tools and concepts and give them room to experiment with AI-augmented testing solutions. Update your QA processes to add checkpoints for AI-related risks, for example, require bias testing for machine learning components, and mandate human review of critical AI-augmented decisions or recommendations. Promote cross-team collaboration so that development, QA, and data science teams are aligned on quality goals. Likewise, insist on high ethical standards (such as transparency, fairness, and accountability) for any AI usage in your products and processes. With strong leadership and the right guardrails, your organization can harness generative AI responsibly to deliver better software faster!

REFERENCES

1. L. J. Freeman, D. C. Schmidt, A. Bonnell, J. Robert, H. Wojton, and Acquisition Innovation Research Council, AIRC Panel on Generative AI in the Acquisition Lifecycle. (Jan. 7, 2025). [Online Video]. Available: https://www.youtube.com/watch?v=ZICc94w-2bY
2. L. J. Freeman, J. Robert, and H. Wojton, "The impact of generative AI on test & evaluation: Challenges and opportunities," in *Proc. Int. Workshop Envisioning AI-Augmented Softw. Develop. Lifecycle*, Trondheim, Norway, 2025, pp. 1–5.
3. P. Roy, J. Chandrasekaran, E. Lanus, L. J. Freeman, and J. Werner, "A survey of data security: Practices from cybersecurity and challenges of machine learning," 2023, *arXiv:2310.04513*.
4. F. Ricca, A. Marchetto, and A. Stocco, "AI-based test automation: A grey literature analysis," in *Proc. IEEE Int. Conf. Softw. Testing, Verification Validation Workshops (ICSTW)*, 2021, pp. 263–270, doi: 10.1109/ICSTW52544.2021.00051.
5. J. White et al., "A prompt pattern catalog to enhance prompt engineering with ChatGPT," in *Proc. 30th Pattern Lang. Program. (PLoP) Conf.*, Allerton Park, IL, USA, pp. 1–31, 2023.
6. P. Dogga, K. Narasimhan, A. Sivaraman, S. Saini, G. Varghese, and R. Netravali, "Revelio: ML-generated debugging queries for finding root causes in distributed systems," in *Proc. Mach. Learn. Syst. 4 (MLSys)*, pp. 601–622, 2022.
7. F. Liu et al., "Exploring and evaluating hallucinations in LLM-powered code generation," 2024, *arXiv:2404.00971*.
8. N. Maleki, B. Padmanabhan, and K. Dutta, "AI hallucinations: A misnomer worth clarifying," in *Proc. IEEE Conf. Artif. Intell. (CAI)*, 2024, pp. 133–138, doi: 10.1109/CAI59869.2024.00033.
9. A. S. Chandrasekaran, "Harnessing the power of generative artificial intelligence (GenAI) in governance, risk management, and compliance (GRC)," *Int. Res. J. Eng. Sci. Technol. Innov.*, vol. 11, no. 5, p. 11, 2024.
10. E. Lanus, B. J. Lee, L. Pol, D. Sobien, J. Kauffman, and L. J. Freeman, "Coverage for identifying critical metadata in machine learning operating envelopes," in *Proc. IEEE Int. Conf. Softw. Testing, Verification Validation Workshops (ICSTW)*, 2024, pp. 217–226.

DOUGLAS C. SCHMIDT is dean of the School of Computing, Data Sciences & Physics, William & Mary, Williamsburg, VA 23185 USA. Contact him at douglas.c.schmidt@wm.edu.

EDITORS: Silvia Abrahão, Universitat Politècnica de València, sabrahao@disc.upv.es
Miroslaw Staron, Chalmers University of Technology and University of Gothenburg,
miroslaw.staron@cse.gu.se

This article originally appeared in
Software
vol. 42, no. 6, 2025

DEPARTMENT: PRACTITIONERS' DIGEST

Hybrid Classical-AI Systems for Software Testing and Bug Fixing

Miroslaw Staron and Silvia Abrahão

Testing, debugging, and defect resolution are all well-known challenges faced by software engineers. When we encounter a bug and debug our software, we can spend hours doing so despite advances in software engineering tools. Testing and debugging processes can be tiresome—but also rewarding when bugs are found and resolved. In this column, we explore the latest research in the area of testing and bug fixing, presented at the ACM International Conference on the Foundations of Software Engineering (FSE 2025), held in Trondheim in June. We welcome your feedback and suggestions on the topics covered. Additionally, if you experiment with or adopt any of the practices featured here, please share your experiences with us and the authors of the respective papers.

A GLIMPSE INTO THE RESEARCH LANDSCAPE

Of the more than 120 papers presented at the conference, 36 focused on software testing (21) and bug fixing (15), with one paper explicitly addressing both aspects. Figure 1 categorizes the types of contributions in these papers. The majority of these contributions are new tools, which is to be expected due to the advent of language models and generative AI. What is surprising, however, is the large set of frameworks presented at the conference for testing as well as the considerable number of new methods introduced for bug fixing.

Let us also explore how the scientific community evaluates these new contributions. Figure 2 illustrates that the majority of papers employed open source software to evaluate their proposed contributions. Only in a few cases was industrial evaluation included, and this was mainly concerned with bug fixing. This is surprising. It is unexpected because testing is an activity performed by all software companies, and at the same time, not surprising, given the conference's focus on innovation and cutting-edge technology rather than empirical observations. Let's now dive deeper into some of these contributions to see what we can learn from the latest research in testing and bug fixing.

> *THE MAJORITY OF THESE CONTRIBUTIONS ARE NEW TOOLS, WHICH IS TO BE EXPECTED DUE TO THE ADVENT OF LANGUAGE MODELS AND GENERATIVE AI.*

BETTER TESTS, THANKS TO LARGE LANGUAGE MODELS

One of the problems with tests is that they tend to lose their effectiveness over time. Software shipped to customers receives more maintenance than tests and other supporting scripts. Updating and improving tests is left to software testers, who must do this in addition to their test assignments. This situation often leads to suboptimal tests, at least from their design point of view. In their paper, "Automated Unit Test Refactoring," Yi Gao, Xing Hu, Xiaohu Yang, and Xin Xia present the UTRefactor tool, which is capable of improving unit testing. The tool employs a hybrid approach, combining a large language model (LLM) with a domain-specific language (DSL). The tool can identify test smells and eliminate them, thereby improving the quality of test cases and reducing the effort required to maintain them.

Digital Object Identifier 10.1109/MS.2025.3597698
Date of current version: 10 October 2025

The tool was evaluated using a set of six open source programs with 879 test cases, containing 2,375 code smells. UTRefactor was able to reduce the number of smells to 265, representing an 89% reduction. Achieving these results required the authors to address several challenges, for example, reducing LLM hallucinations. To reduce hallucinations, UTRefactor employs a chain-of-thought approach by providing the LLM with the source code of the unit test, the context, the description of the smell, and the DSL program that demonstrates how to modify the test case. The best results are achieved for the *Assertion Roulette* smell, where the tool successfully removed all instances of this smell, which occurs when a test case contains multiple undocumented assertions. It's easy for a human to detect but not so much for an automated tool because the documentation can be physically separated from the assertion (for example, all assertions can be described in one large comment). The worst results were obtained for the *Sensitive equality* smell, where an assertion converts an object to a string using the *toString()* method and compares the string; it is sensitive to the implementation of the *toString* method. For this category of test smells, UTRefactor could reduce the smells by 30%. Even that is an excellent result compared to the previous baselines, where the reductions were in the single digits. Access the paper at https://tinyurl.com/UTrefactor.

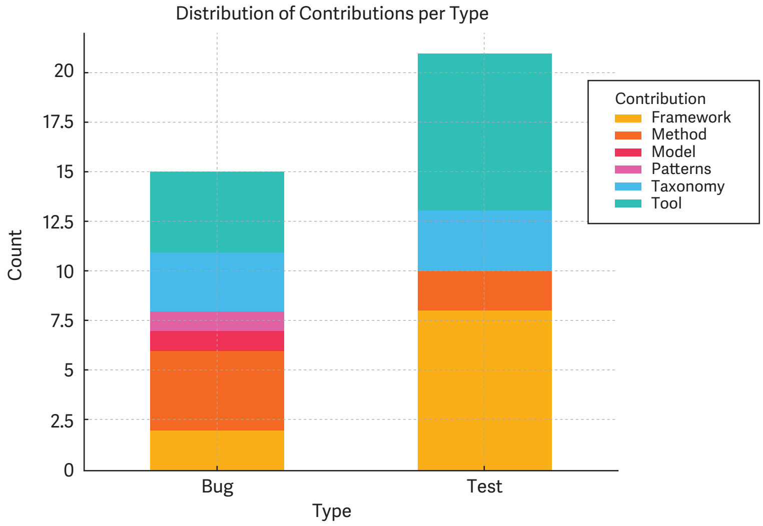


FIGURE 1. Types of contributions.

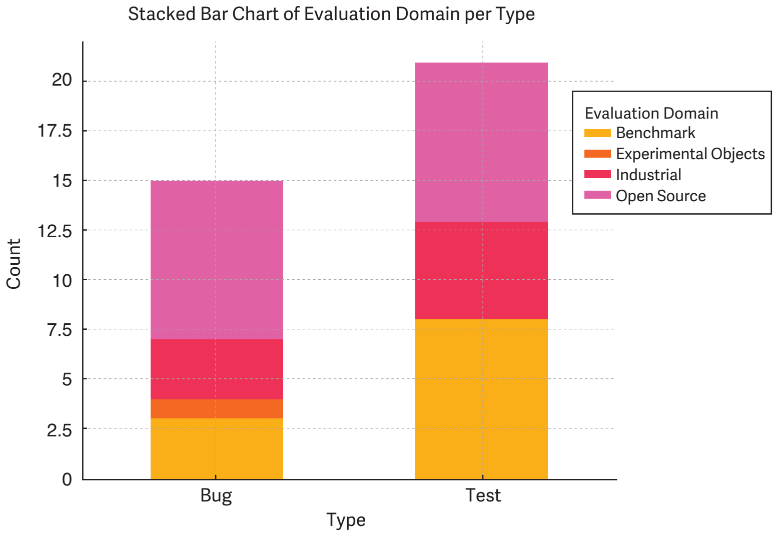


FIGURE 2. Types of evaluation of the presented contributions.

TALKING TO DEBUGGERS LIKE YOU TALK TO PEOPLE

Debuggers are fantastic tools, but using them can be frustrating. It is often not enough to know that a variable is set to null; it is more important to understand why it is set to null and when it was assigned the null value. In their paper, "ChatDBG: Augmenting Debugging with Large Language Models," Kyla H. Levin, Nicolas van Kempen, Emery D. Berger, and Stephen N. Freud present a new tool that enables conversations with debuggers. The tool allows for asking questions like "why is x null?" instead of backtracking the assignments by reading the source code. To leverage this capability, the tool enables the LLM (OpenAI's GPT-4) to take control over debuggers, specifically

LLDB, GDB, and PDB, in this study. Just asking ChatDBG a question, "why?", leads to solving 57% of problems. A single but more elaborate query led to bug fixing 67% of the time, while asking an additional question bumped up this number to 85%.

What is even more interesting is that ChatDBG can suggest solutions that extend beyond the actual program, for example, "Check the CSV file marble-sample.csv to verify the proportion of blue to red marbles and ensure it is close to 70% blue marbles." This kind of capability is very much needed, but standard debuggers do not provide it because they are limited in scope to the program that they debug. On top of that, ChatDBG can access the .csv file and check whether the proportion is correct. The set of bugs found using this tool is impressive, all from misspelled columns to wrong percentiles in calculating confidence intervals. For C/C++ programs, the tool was able to explain almost all problems and crashes, suggesting fixes for 36% of the root causes and 55% of the proximate causes. I firmly believe that this technology can revolutionize the way we debug programs and that the cost of software debugging will be significantly reduced. Access the paper at https://tinyurl.com/ChatDBG.

> *WHAT IS EVEN MORE INTERESTING IS THAT CHATDBG CAN SUGGEST SOLUTIONS THAT EXTEND BEYOND THE ACTUAL PROGRAM.*

THE CHALLENGE OF MULTILANGUAGE BUGS

Modern software systems often combine different programming languages in a single program. Build scripts are one example of such programs, where languages such as bash scripting contain bits of Python needed to build software. Debugging such programs—and even worse, finding defects in them—is a daunting task that requires a deep understanding of the environment and the programming languages involved. In their paper, "Dissecting Real-World Cross-Language Bugs," Haoran Yang and Haipeng Cai study 400 cross-language bugs mined from more than 50,000 commits. The two most common language pairs are Python-C and Java-C. The former combination is well known from frameworks such as TensorFlow or PyTorch, while the latter is a common combination in mobile and web development. The reason why these kinds of bugs are so problematic is partially explained in the paper. These bugs predominantly manifest themselves through incorrect output, error or warning messages, and program crashes. In Python-C, significant issues are often difficult to debug, such as memory leaks. For Java-C, performance degrades in the case of cross-language bugs. These kinds of bugs usually originate from assignments, local function calls, or cross-language application programming interface (API) usage. The last one is predominant in the Python-C, while the others are equally frequent for both language pairs.

Finally, the paper discovers that the majority of bugs are found in the native language, not the one that is embedded. The authors can explain this by the greater complexity of the native code, which often has to handle data translations between different languages and type systems, manage different memory models, and handle resource allocation. The paper concludes with two strategies to address these issues: incorporating necessary checks and input validations and utilizing appropriate data types and casting. Although cross-language bugs have distinct characteristics, they are not immune to the typical problems of single-language bugs, such as memory management or type-checking issues. However, the new issues are more challenging to capture and address directly. Access the paper at https://tinyurl.com/crosslanguage.

BEYOND SCRIPTS: INDUSTRIAL GUI TESTING WITH MODELS

Testing graphical user interfaces (GUIs) is challenging and costly, especially as systems evolve. While model-based testing (MBT) offers automation and maintainability advantages, its use in industry remains limited. In their paper, "A Mixed-Methods Study of Model-Based GUI Testing in Real-World Industrial Settings," Shaoheng Cao, Renyi Chen, Wenhua Yang, Minxue Pan, and Xuandong Li investigate how MBT performs in industrial GUI testing environments, where scripted testing remains the norm. The authors partnered with an industrial software development team and conducted a two-month mixed-methods study combining controlled experiments, surveys, and interviews. The team deployed a model-based GUI testing tool in a production software testing workflow and compared it to traditional script-based testing.

Quantitative data revealed that MBT was comparable to script-based methods in terms of defect detection and testing efficiency. More importantly, MBT significantly outperformed scripts in terms of maintainability; model updates proved easier and less error prone than maintaining large sets of manual test scripts, especially as the system evolved. Qualitative feedback from practitioners was equally revealing. Initially, there was considerable skepticism toward MBT, primarily due to unfamiliarity and perceived complexity. However, over time, participants developed a more favorable attitude. They noted that the approach streamlined test creation, reduced redundancy, and helped ensure more consistent coverage of GUI paths. By the end of the study, several testers expressed interest in using MBT beyond the study. Overall, the results provide evidence that MBT, when introduced carefully, can not only match but sometimes exceed traditional testing approaches in industrial GUI scenarios. For practitioners considering automation strategies, MBT emerges as a promising and accessible option. Access the paper at https://tinyurl.com/4smrkcbr.

SMARTER TEST AUTOMATION WITH A HUMAN TWIST

Testing remains one of the most time-consuming parts of software development, largely due to the challenge of verifying correctness—known as the "oracle problem." Most tools rely on a single oracle type: implicit (for example, crash detection), property based (checking outputs meet defined rules), or example based (comparing expected outputs). Each has its strengths, but none fully supports the nuanced judgment that developers bring to testing. In their paper, "TerzoN: Human-in-the-Loop Software Testing with a Composite Oracle," Matthew C. Davis, Amy Wei, Brad A. Myers, and Joshua Sunshine propose a hybrid solution: the Composite Oracle. This approach combines all three oracle types in a single workflow, enabling developers to generate tests using an implicit oracle and then refine test cases by annotating observed behaviors and adding property checks via code snippets. The authors also introduce TerzoN, a tool built around this composite model. It offers a unified interface for evaluating test outputs through multiple lenses—automated signals, user-defined properties, and example-based checks. Developers can compare results side by side, helping them spot discrepancies and bugs more effectively.

To evaluate TerzoN, the authors conducted a user study with 14 professional developers, comparing it to fast-check, a widely used property-based testing tool. Participants using TerzoN found 72% more bugs, gave more accurate bug descriptions (more than twice as many), and worked 16% faster on average. These statistically significant gains demonstrate how a composite oracle, combined with human insight, can lead to faster and more effective testing. For practitioners, TerzoN offers a flexible alternative to traditional test tools—automated yet guided by developer intuition. It supports faster and clearer decision making without requiring deep up-front specification work. Access the paper at https://tinyurl.com/4f87k9sh.

TREATING EACH PERFORMANCE BUG AS A JOURNEY

In configurable software systems, where interacting features make performance bugs complex and unpredictable, debugging becomes a challenging task. In their paper, "Understanding Debugging as Episodes: A Case Study on Performance Bugs in Configurable Software Systems," Max Weber, Alina Mailach, Sven Apel, Janet Siegmund, Raimund Dachselt, and Norbert Siegmund reconceptualize debugging as a sequence of goal-oriented episodes—distinct phases driven by developer intent rather than predefined strategies.

Using immersive virtual reality (SoftVR) to observe professional developers who are tackling real-world performance issues, the authors identify recurring episodes, such as exploration, search, reasoning, reorganization, and also aimlessness. Rather than following single strategies, developers fluidly transition between episodes depending on their understanding and goals. This episodic model better reflects the reality of debugging configurable systems and can guide future tool development, training, and research. The paper introduces a novel framework for capturing this process and highlights the importance of supporting developers not just with technical tools but

TABLE 1. Insights from selected AI research presented at FSE 2025.

Research focus	Key insights	Takeaways	Suggested applications
LLM-assisted unit test refactoring	Hybrid use of LLM + DSL effectively removes test smells with an 89% success rate on real-world test suites.	AI tools can meaningfully improve and clean up legacy test code with minimal human intervention.	Apply UTRefactor to improve test maintainability and quality in systems with aging test suites.
Conversational debugging	Chat-style interactions with debuggers help resolve up to 85% of issues—capable of integrating external knowledge.	LLMs can enhance debugging workflows by offering explanations and actionable suggestions, reducing cognitive load.	Integrate ChatDBG into debugging environments to improve the speed and clarity of issue resolution.
Cross-language bug analysis	Bugs often arise in the native host language (for example, C), not the embedded one; memory and API issues are frequent.	Developers need targeted strategies for cross-language systems, especially for resource and memory handling.	Apply stricter validation and use standardized interfaces when bridging languages like Java-C or Python-C.
Model-based GUI testing	MBT matches scripted testing in effectiveness but excels in maintainability; initial resistance fades with use.	GUI testing automation with MBT can reduce the maintenance overhead while retaining coverage and reliability.	Introduce MBT gradually in GUI-heavy applications, especially where GUI changes are frequent.
Human-in-the-loop testing	Combining implicit, property-based, and example-based oracles improves test effectiveness and efficiency.	A composite oracle empowers developers to refine tests faster and catch more bugs than conventional tools.	Use TerzoN in testing workflows to leverage both automation and human judgment.
Debugging performance bugs as episodes	Debugging complex performance bugs is best understood as shifting goal-oriented episodes rather than fixed linear strategies.	Support debugging by detecting aimlessness, fostering search–reasoning cycles, and training developers in episodic and goal-oriented problem-solving approaches.	Adopt tools that offer integrated performance insights, config-aware visualizations, and stepwise guidance.

with cognitive frameworks suited to the complexity of their tasks. Access the paper at https://tinyurl.com/2t9zxdyt.

The reviewed papers reflect a growing trend; AI and classical approaches are converging—not to replace humans, but to amplify them. Tools like UTRefactor, ChatDBG, and TerzoN show that LLMs can boost test quality and debugging efficiency—but only when guided by human insight. The most effective systems are not fully autonomous, human guided, and context aware. Yet, many approaches remain undertested in industrial settings. For practitioners, this is both a challenge and an opportunity—to help shape these innovations into practical and impactful solutions. As testing and debugging evolve, AI is not replacing engineers—it's becoming their smartest assistant. Table 1 summarizes the key insights from the reviewed papers, highlighting several takeaways for practitioners and suggesting some applications in their context.

MIROSLAW STARON is a full professor in the Interaction Design and Software Engineering Division in the Computer Science and Engineering Department shared by the Chalmers University of Technology, 412 96 Gothenburg, Sweden, and the University of Gothenburg, 412 96 Gothenburg, Sweden. Contact him at miroslaw.staron@cse.gu.se.

SILVIA ABRAHÃO is a full professor at the Department of Computer Science, Universitat Politècnica de València, 46022 Valencia, Spain. Contact her at https://sabrahao.wixsite.com/dsic-upv or sabrahao@dsic.upv.es.

COMPUTING edge

WWW.COMPUTER.ORG/COMPUTINGEDGE

IEEE COMPUTER SOCIETY

80th ANNIVERSARY CELEBRATION

UNLOCK THE BENEFITS

WORLD-CLASS CONFERENCES

Stay ahead of the curve by attending one of our 195+ globally recognized conferences.

DIGITAL LIBRARY

Access over 1 million articles covering world-class peer-reviewed content in the IEEE Computer Society Digital Library anywhere, anytime.

CALLS FOR PAPERS

Discover opportunities to publish or present your ground-breaking accomplishments.

EDUCATIONAL RESOURCES

Strengthen your resume with the IEEE Computer Society Course Catalog and its range of offerings.

CAREER ADVANCEMENT

Search new positions posted in the IEEE Computer Society Jobs Board from all over the world.

NETWORKING OPPORTUNITIES

Connect with peers, mentors, and industry leaders by participating in local Region, Section, and Chapter activities.

Sign Up for a Membership at the IEEE Computer Society

computer.org/membership

Your Professional Home

IEEE

EDITORS: George Hurlburt, U.S. Federal Service (Retired), USA, gfhurlburt@gmail.com
Sorel Reisman, California State University, USA, sreisman@computer.org

DEPARTMENT: EDUCATION

This article originally appeared in
Computer
vol. 58, no. 1, 2025

Test and Evaluation of Autonomous Systems: Developing the Educational Landscape

Brian A. Weiss, Donald H. Costello III, and Matt Scassero, *University of Maryland*

The University of Maryland is developing a test and evaluation (T&E) of autonomous systems curriculum to be deployed in both graduate degree and certificate forms, which will lead to a more robust and richer T&E workforce.

Every product and process that was ever developed required some form of assessment before being adopted by the intended user community; test and evaluation (T&E) is an integral part of the product and process development process. From a broader perspective, T&E is fundamental to the overall product lifecycle. The necessity motivating T&E is decision making. T&E is critical to system verification or validation, which can include assessing constituent capabilities or components, to inform stakeholders of performance. Stakeholders use this performance insight to make decisions, which can range from determining if another phase of a technology's development should be funded to whether a technology should be fielded.

INTRODUCTION

As technologies continue to evolve, performance insight becomes more critical to enable informed decisions. More specifically, intelligent technologies including autonomous systems are becoming more common and complex; T&E is an absolute necessity for these technologies to advance forward in the product lifecycle. The U.S. government is one such entity that is faced with critical decisions requiring T&E insight. The U.S. government is a significant investor in the development of these technologies. Unfortunately, there is a lack of sufficient T&E expertise to meet the development and fielding demands of these emergent systems.

This gap in T&E expertise can be attributed to the corresponding education gap. This gap is rather pronounced when considering the other functions of the product lifecycle; product development and design are taught throughout many engineering curricula, where some universities provide focused degree programs in relevant domains (e.g., systems engineering). Focused T&E education curricula, particularly programs honed toward T&E of autonomous systems, have a minimal presence in the academic community. This has resulted in many T&E professionals being forced to develop their skills through "on-the-job" training.

The University of Maryland (UMD) is aiming to fill this knowledge gap through its efforts to develop and implement a T&E of Autonomous Systems curriculum. This curriculum, to be presented across certificate and degree options, would enable both current university students, as well as working professionals, to grow and accelerate their T&E of autonomous systems knowledge base. The authors define an autonomous system using a standard sliding scale.[1] This scale ranges from level 0, where there is no autonomous capability, to level 5, where the system is fully autonomous.

Digital Object Identifier 10.1109/MC.2024.3485928
Date of current version: 9 January 2025

2469-7087/26 © 2026 IEEE

BACKGROUND

Importance of T&E

T&E is an important aspect of any technology, whether it be a product or process development effort. T&E provides key intelligence to decision makers as they make a range of determinations, including whether a technology should be further developed or whether it is ready to be fielded. Evaluations can either be informative or summative.[2] Formative evaluations provide key performance insight as a technology is still evolving and maturing. Findings from formative evaluations enable modifications and updates in future iterations of the technology. Summative evaluations highlight and validate the technology's capabilities in its final form. Feedback from these evaluations can also be used to set appropriate expectations with users.

T&E can occur in the form of verification (that is, "Did I build the system right?") and validation (that is, "Did I build the right system?").[3] T&E also occurs in acceptance testing – Does the system meet a minimum level of performance? Even though there is a clear distinction between verification and validation, it's possible that an assessment can address both based on the metrics that are designed and captured.

Ultimately, good technology development includes some form of assessment.[4] Even though the design process may provide some indication of how technology will perform, T&E is a necessary formal step to clearly articulate the performance capabilities. Additional value of this performance intelligence is that it can provide clear technology expectations to stakeholders, including those buying and/or using the technology. Likewise, these performance benchmarks can be provided back to the technology developers and requirement-setters to enhance their ability to address deficiencies or spur continuous improvement. T&E methods and performance metrics can be integrated into community-driven standards, guidelines, and best practices.

As noted earlier, decision makers rely on T&E findings; identified shortcomings can indicate that the technology is not ready to be fielded and identify specific areas that need to be remedied. Finally, T&E findings support risk mitigation as technology performance expectations crystallize. If stakeholders have high confidence in where a technology will perform well, and where it will fall short, then they can apply this knowledge to ensure the technology is used where it can have a reasonable chance of success, thereby providing value to the user.

> *EVEN THOUGH THERE IS A CLEAR DISTINCTION BETWEEN VERIFICATION AND VALIDATION, IT'S POSSIBLE THAT AN ASSESSMENT CAN ADDRESS BOTH BASED ON THE METRICS THAT ARE DESIGNED AND CAPTURED.*

Formalized T&E approaches

Many T&E-focused frameworks, methodologies, and approaches have been developed to add science and rigor to assessments. This section introduces just a few of those that have been disseminated.

The "Systems Engineering V" is a generally accepted model of a system lifecycle and features a strong T&E component.[5] While the standard V diagram does not explicitly state it, T&E is required to be a part of the system for the duration of the lifecycle. At the beginning of a system's lifecycle, T&E professionals need to understand the system requirements. In turn, these requirements will influence how the overall system will be designed, including its constituent subsystems and components, along with how they will be integrated together. The awareness is crucial for T&E design: Where T&E will be iteratively applied as a system is constructed from its foundational building blocks. Additionally, the T&E professional needs to assess how the system will evolve, how upgrades will be evaluated, and how the system will eventually be retired.

To support multiple T&E efforts of emergent, intelligent technologies, personnel from the National Institute of Standards and Technology (NIST) developed the system, component, and operationally relevant evaluation (SCORE) framework.[6] This framework provided the evaluation backbone for the assessment of several advanced military technologies. Specifically, it drove multilevel evaluations to capture both technical performance and utility assessment that provided stakeholders with a broad understanding of technology performance. Recognizing the opportunity to increase the scientific rigor of evaluations of complex technologies, the multirelationship evaluation design (MRED) framework was developed.[7] MRED enables the generation of test plan blueprints while factoring multiple characteristics, including technology test levels, technology maturity, and stakeholder preferences.

These approaches are just some of the efforts that seasoned T&E professionals have crafted to add structure, rigor, and repeatability to assessments. Likewise, these approaches can be leveraged to assess emergent technologies, including those that are autonomous.

Autonomous systems

From approximately 2000 to 2024, there has been a dramatic increase in vehicle automation, including a growing acceptance of uncrewed vehicles in the military (for example, unmanned drones). Greater vehicle automation is also visible in driver relief modes (that is, adaptive cruise control) and the increase of highly automated cars. Decision makers, particularly those from the U.S. Department of Defense, are moving toward technologies with more autonomous capabilities, including those that either present a human–machine team or are fully autonomous. The overall motivation for this progression is to enable humans to work safer and more efficiently.

With technological intelligence rapidly growing and more autonomous systems emerging, decision makers need to categorize a system's capabilities and characterize its performance so it can be appropriately certified for operations. System certification will be different if there is a human in or on the loop (for example, a human controlling the system, human monitoring the system so they can take action as needed) as compared to if there is not a human in or on the loop (the system will define its environment based on a set of sensors, and make decisions based on those sensors). That is, a fully autonomous system will be certified for operations differently than one featuring a human–machine team.

T&E will become more paramount to understanding the performance of these technologies, including what expectations should be set; where, what kind, and how much data are relevant and effective in training a system; and if/when they should be certified for operation.

T&E of emergent, intelligent systems

To date, T&E has been performed on a range of intelligent systems; the authors cite several that were assessed by personnel from NIST. One significant effort that NIST researchers undertook was developing both foundational performance measurements and standards for autonomous ground vehicles. Measurements and metrics stemmed from articulating both the desirable and undesirable behaviors that an autonomous ground vehicle should exhibit.[8] Years later, NIST researchers would continue their efforts in autonomous ground vehicle T&E to include the development of standards.[9]

In addition to its ground-breaking work in assessing autonomous ground vehicles, NIST personnel also assessed robot systems used by first responders and law enforcement personnel. Specifically, NIST developed performance metrics and corresponding test methods to assess urban search-and-rescue robots and explosive ordnance disposal robots,[10,11] These systems ranged from teleoperative to having some autonomous features.

T&E of intelligent systems is not limited to those in the physical domain. Evaluations were also performed on emergent applications to support military operations.[12] Although the applications were engaged using handheld devices, T&E was primarily focused on the capabilities of the individual applications with a secondary focus on the physical platform to which they were bound.

EDUCATION

The fundamental need for a T&E of autonomous systems curriculum is clear. The evolution of foundational hardware and software technologies has accelerated the pace of autonomous system development. Many of the autonomous technologies that have been developed to date have been assessed by "home-grown" T&E professionals. This paradigm will largely continue as more intelligent, including autonomous systems, come to fruition.

UMD's efforts to advance T&E education will address numerous challenges, including:

- accelerating T&E knowledge acquisition so it's no longer restricted to on-the-job training
- offering richer T&E content and rigor so T&E is no longer restricted to the "job-specific" technology being assessed and the extent to which rigor is being applied
- enabling T&E personnel to understand the necessity of being involved in the technology lifecycle well ahead of T&E-proper
- integrating research into the T&E of autonomous systems education program to ensure it is constantly updated in this rapidly evolving field.

UMD is planning a curriculum that will enable students to attain a professional certificate, graduate certificate, or Masters of T&E of Autonomous Systems. The curriculum will feature multiple core courses focused on T&E and autonomous/artificial intelligence-driven technologies. The T&E education will include both the elements found in T&E proper and elements not traditionally in the T&E realm, yet are central to the product lifecycle that T&E should be aware of (at minimum) or directly engaged in (ideally). The first core course, T&E101, is projected for deployment during the Fall 2025 semester. Additional core courses will be developed and offered in subsequent semesters. Likewise, the full certificate and Masters' programs are expected to be in place in concert with the core courses.

Some of the "traditional" or fundamental T&E elements that the course will cover include the design, collection, and analysis of measurements and metrics; determination of technology test levels and how assessing a technology at differing levels can offer unique insight; identification of environmental parameters, recognizing that some factors can be controlled in some environments, whereas some cannot, and must simply be noted; specification of test personnel knowledge and autonomy (that is, control of their actions in the environment and the extent to which they can engage with the technology); and the design and implementation of evaluation scenarios.

Moving beyond T&E proper, additional elements will be presented in the curriculum, further educating where T&E personnel can objectively support the entire product lifecycle. These additional elements include the initial problem definition (something that is often skipped when decision makers begin the process with specific technology—that is, solution—in mind), use case articulation, workflow articulation (both the "as-is" and the "to-be"), user pain point identification, system training implications, and technology insertion prioritization.

Given the complexity of autonomous systems, a rigorous T&E curriculum will provide the foundation for new workforce entrants and existing professionals to build their T&E acumen before even designing their first test. This foundational knowledge will be critical, as autonomous systems are fraught with practical T&E challenges.

As noted earlier, T&E is crucial to enable informed decision making, including continuing a technology's development and transitioning it to the field. For these decisions to be sufficiently informed, leadership needs to understand what it can expect from a technology's performance. This understanding is part of the information equation in mitigating any potential risk. Calculating risk becomes a different exercise when you have a human in the loop in a human–machine-teamed technology versus an autonomous system, where there is no human in the loop. It then becomes reasonable to assert that the autonomous system will be evaluated on a different level as compared to a system where a human is present.

Decision makers still need to define how autonomous systems will be evaluated. More importantly, decision makers need to define what the standards and methods of compliance will be for the certification of autonomous systems. Whatever is decided, knowledgeable T&E professionals will be required to design and implement appropriate assessments to provide the critical intelligence.

The acceleration of autonomous system development and the corresponding stakeholders' desire to clearly understand performance expectations sets a clear demand signal for competent, rigorous, and repeatable T&E. The realization of such T&E requires an appropriately trained and seasoned workforce. The existing approach—building up T&E expertise through on-the-job training—will not keep pace with the current and expected T&E demands. UMD has identified this gap as an opportunity to develop and deploy a curriculum to educate undergraduate, graduate, and working professionals on T&E of autonomous systems. Addressing this need will both expand the knowledge base of the existing workforce and enable graduates to have immediate impacts in the T&E domain.

REFERENCES

1. "Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles," SAE International. Accessed: Sep. 18. 2024. [Online]. Available: https://www.sae.org/standards/content/j3016_202104/
2. B. A. Weiss and L. C. Schmidt, "The multi-relationship evaluation design framework: Creating evaluation blueprints to assess advanced and intelligent technologies," in *Proc. 10th Perform. Metrics Intell. Syst. Workshop*, Sep. 2010, pp. 136–145.
3. R. Valerdi and B. P. Sullivan, "Engineering systems integration, testing, and validation," in *Handbook of Engineering Systems Design*, A. Maier, J. Oehmen, and P.E. Vermaas, Eds., Cham, Switzerland: Springer-Verlag, 2022.
4. E. B. Magrab, S. K. Gupta, F. P. McCluskey, and P. Sandborn, *Integrated Product and Process Design and Development: The Product Realization Process*. Boca Raton, FL, USA: CRC Press, 2009.
5. A. Kossiakoff, W. N. Sweet, S. J. Seymour, and S. M. Biemer, *Systems Engineering Principles and Practice*, Vol. 83. Hoboken, NJ, USA: Wiley, 2011.
6. C. Schlenoff, "Applying the system, component, and operationally-relevant evaluation (SCORE) framework to evaluate advanced military technologies," *Int. Test Eval. Assoc. (ITEA) J.*, vol. 31, no. 1, pp. 112–120, 2010.
7. B. A. Weiss, L. C. Schmidt, H. A. Scott, and C. I. Schlenoff, "The multi-relationship evaluation design framework: Designing testing plans to comprehensively assess advanced and intelligent technologies," *Int. Design Eng. Tech. Conf. Comput. Inform. Eng. Conf.*, vol. 44137, pp. 603–616, Jan. 2010, doi: 10.1115/DETC2010-28928.
8. J. S. Albus, *Features of Intelligence Required by Unmanned Ground Vehicles*. Gaithersburg, MD, USA: NIST Special Publication, 2001, pp. 243–252.
9. S. Yoon and R. Bostelman, "Analysis of automatic through autonomous-unmanned ground vehicles (A-UGVs) towards performance standards," in *Proc. IEEE Int. Symp. Robotic Sensors Environ. (ROSE)*, Piscataway, NJ, USA: IEEE, Jun. 2019, pp. 1–7, doi: 10.1109/ROSE.2019.8790421.
10. A. Jacoff and E. Messina, "Urban search and rescue robot performance standards: Progress update," *Unmanned Syst. Technol. IX*, vol. 6561, pp. 486–497, May 2007, doi: 10.1117/12.719692.
11. J. Scholtz, M. Theofanos, and B. Antonishek, "Development of a test bed for evaluating human-robot performance for explosive ordnance disposal robots," in *Proc. 1st ACM SIGCHI/SIGART Conf. Hum.-Robot Interact.*, Mar. 2006, pp. 10–17, doi: 10.1145/1121241.1121246.
12. B. A. Weiss, L. Fronczek, E. Morse, Z. Kootbally, and C. Schlenoff, "Performance assessments of Android-powered military applications operating on tactical handheld devices," *Proc. SPIE Int. Soc. Opt. Eng.*, vol. 8755, p. 25, Jun. 2013, doi: 10.1117/12.2014771.

BRIAN A. WEISS is a research engineer at the University of Maryland, College Park, MD 20742 USA. Contact him at baweiss@umd.edu.

DONALD H. COSTELLO III is an associate research professor at the MATRIX Lab, University of Maryland Clark School of Engineering, California, MD 20619 USA. Contact him at dhcost@und.edu.

MATT SCASSERO is the director for operations and outreach for the MATRIX Lab, University of Maryland Clark School of Engineering, California, MD 20619 USA. Contact him at mscasser@umd.edu.

EDITOR: Alexander B. Magoun, a.b.magoun@ieee.org

This article originally
appeared in
IEEE
Annals
of the History of Computing
vol. 47, no. 2, 2025

DEPARTMENT: ANECDOTES

ELIZA Reanimated: Restoring the Mother of All Chatbots to One of the World’s First Time-Sharing Systems

Rupert Lane, *Independent Researcher, Tokyo 141-0022, Japan*
Anthony Hay, *Devon EX31 4QP, U.K.*
Arthur Schwarz, *Slipbits, Irvine, CA 92620 USA*
David M. Berry, *Faculty of Media, Arts and Humanities, University of Sussex, Falmer BN1 9RH, U.K.*
Jeff Shrager, *Bennu Climate and Stanford University, Palo Alto, CA 94025 USA*

ELIZA, created by Joseph Weizenbaum at MIT in the early 1960s, is considered the earliest chatbot. He programmed it in Michigan Algorithm Decoder-Symmetric List Processor (MAD-SLIP) on MIT’s Compatible Time-Sharing System (CTSS) operating system, on an IBM 7094. We discovered an original ELIZA printout in Prof. Weizenbaum’s papers at MIT’s Institute Archives, including an early version of its famous DOCTOR script, a nearly complete version of the MAD-SLIP code, and various support functions in MAD and Fortran Assembly Program. Here we describe the reconstruction and reanimation of this original ELIZA on a restored CTSS, running on an emulated IBM 7094. The entire stack is open source, so that any user of a Unix-like operating system can run the world’s earliest chatbot on their own version of a pioneering time-sharing system.

ELIZA’S ANCESTRY

Joseph Weizenbaum created ELIZA, usually considered the world’s first chatbot, in Michigan Algorithm Decoder-Symmetric List Processor (MAD-SLIP) at MIT’s Project MAC in the mid-1960s [20]. One of the core goals of Project MAC was to demonstrate the power of the new concept of interactive time sharing, embodied by the Compatible Time-Sharing System (CTSS) [18], among the world’s first interactive time-sharing systems, running on an IBM 7094 [6]. CTSS was originally conceived and implemented at MIT’s Computation Center, led by Fernando Corbató. Project MAC improved CTSS and made it available to many different projects, including Weizenbaum’s [18].

One of CTSS’s core user-level languages was MAD [12]. Weizenbaum had developed his SLIP in order to provide list-processing capabilities to high-level languages. He originally developed SLIP for Fortran [19] but rebuilt it for MAD and shortly thereafter built ELIZA in MAD-SLIP on CTSS. In addition to being an excellent demonstration of the power of interactive time sharing, ELIZA was intended as an exploration in symbolic computing applied to natural language processing and as a platform for mounting experiments in human–machine interaction, centered around discourse [17].

ELIZA has had a significant impact on AI, being the first program to directly embody Turing’s test and to bring to reality the science-fiction fantasy of holding an actual conversation with a computer. However, Weizenbaum foresaw “a danger [...] that the example [ELIZA] will run away with what it is supposed to illustrate” [20, p. 43]. Indeed, shortly after the MAD-SLIP

1058-6180 © 2025 IEEE. All rights reserved, including rights for text and data mining, and training of artificial intelligence and similar technologies.
Digital Object Identifier 10.1109/MAHC.2025.3564095
Date of current version 10 June 2025.

ELIZA appeared in print, Bernie Cosell, at Bolt Beranek and Newman, wrote a near-clone of it in Lisp [16]. Once Cosell's Lisp ELIZA hit the academic world via the rapid spread of the ARPANet, it supplanted Weizenbaum's MAD-SLIP version, and the name "ELIZA" (and its "DOCTOR" script) became associated with Lisp, which was at the same time rapidly becoming the lingua franca of AI.[1]

Almost exactly a decade later, in 1977, *Creative Computing*, one of the magazines that served the mid-1970s personal computer explosion, published an ELIZA knock-off written in BASIC by Jeff Shrager [11]. Because of its brevity and simplicity and the personal computer explosion, this ELIZA begat innumerable knock-offs through the decades in every conceivable programming language [4].

ELIZA REDISCOVERED

MAD was soon replaced by other ALGOL-like languages, and Lisp's widespread adoption eclipsed SLIP. The version of ELIZA that was known in the academic community was usually written in Lisp (starting with Cosell's clone), and the version known to the public was usually written in BASIC (starting with Shrager's version). Few aside from a tiny cadre of AI historians concerned themselves with the whereabouts of the original ELIZA code until 2021 when Jeff Shrager (coincidentally, the author of the 1973/1977 BASIC ELIZA) and MIT archivist Myles Crowley found what appeared to be a complete copy of the source among Weizenbaum's papers [1], [13]. Further explorations of the archives found relevant underlying code, including critical parts of MAD-SLIP, written in both MAD and Fortran Assembly Program (FAP).[2]

ELIZA REANALYZED

The rediscovered MAD-SLIP code was clearly not exactly the ELIZA described in the 1966 *CACM* paper. For example, some differences are as follows:

- The preliminary transformation `PRE` reassembly pattern is not implemented, for example,

  ```
  (I'M = YOU'RE ((0 YOU'RE 0)
     (PRE (YOU ARE 3) (=I))))
  ```

- Neither the keyword stack nor the `NEWKEY` reassembly rule are supported, for example,

  ```
  (DREAMT 4 ((0 YOU DREAMT 0)
     (REALLY, 4)
     (HAVE YOU EVER FANTASIED 4 WHILE YOU WERE
       AWAKE)
     (HAVE YOU DREAMT 4 BEFORE)
     (=DREAM)
     (NEWKEY)))
  ```

- The found source code supports calls to other rules at the transformation level, for example,

  ```
  (HOW (=WHAT))
  ```

  but does not support them at the reassembly level, as in the `(=DREAM)` as used in the `DREAMT` rule above.

The version discovered by Shrager came physically attached, on the fan-fold printout, to a simpler version of the DOCTOR script than that in the 1966 paper. The found code also appeared to have a fully internal script editing capability that was only mentioned in passing in the 1966 paper, and is described in more detail below.

Although it was clear from these differences that the found version was not exactly the same as the published ELIZA, what remained unknown was whether this version of ELIZA worked at all. For all we knew, we might have found the printout of a version riddled with bugs. The best way, indeed, perhaps the only way, to figure out if this was a working ELIZA was to try it.

ELIZA REANIMATED

Reanimating ELIZA from the discovered code required numerous steps of reconstruction, including code cleaning and completion, emulator stack installation and debugging, nontrivial debugging of the found code itself, and even writing some required functions that were not found in the archives or in the available MAD and SLIP implementations. To be clear, whereas this represented quite a lot of effort, at the end of the day, 96 percent of the running ELIZA and SLIP code was exactly as it was found in the archive.

Rupert Lane, an aficionado of early operating systems, had been working with the Michigan Time Sharing (MTS) system (which has MAD and SLIP implementations). David Berry asked Lane to think about bringing up the original ELIZA. After some discussion with the rest of the team we realized that it would be simpler to bring up the original ELIZA in its original environment of CTSS on the 7094.

---

[1] Jeff Shrager and a small community of Lisp aficionados reconstructed and reanimated Cosell's original Lisp version from printouts in 2013 [3].

[2] FAP code was required to call CTSS services from MAD, and to carry out various machine-level operations, such as bitwise and word-wise manipulations.

CTSS ON THE 7094

MIT's CTSS, first introduced in 1961 at MIT, was the first multiuser time-sharing system [18]. It brought many innovations such as a disk filing system, real-time communication with users via keyboard terminals, and the ability to develop programs interactively without waiting hours or days for batch execution. It would have been impossible for Weizenbaum to develop ELIZA without such a system.

CTSS ran on the IBM 7094, an early transistorized computer with only 32,000 words of user memory, a 36-bit word length, and a clock speed of around 450 kHz. At that time a 7094 cost around $3 million (now ~$30 million), and MIT heavily modified it, including adding a second 32,000 word memory bank for system software. CTSS could support around 30 simultaneous users and was in use at MIT until the early 1970s.

David Pitts wrote an emulator for the 7094, based on work by Paul Pierce, and a working version of CTSS was brought up on that emulator by Pitts in the early 2000s [14].

RECONSTRUCTING THE CODE

The combination of MAD and SLIP that constitutes the ELIZA stack is around 2,600 lines of mostly uncommented MAD and FAP code. Getting this into machine-readable format was nontrivial; it did not OCR well, so Anthony Hay and Art Schwarz manually transcribed it. Hay had previously done a careful analysis of the algorithm and DOCTOR script as described in the original 1966 paper and wrote a C++ version of ELIZA that is, so far as we know, the best clone of the original [5]. Schwarz had worked with SLIP for many years and is the author of the GNU SLIP library [15]. Their expertise was invaluable in navigating the code.

Lane fed each file into the compiler on the emulated CTSS system, which helped hunt down typos and let us understand the structure of the code. Some of the unique challenges of working with this code included the following:

- CTSS uses 6-bit BCD character encoding, packing six characters into a 36-bit word. (This was before 8-bit bytes or ASCII were universal.)
- The MAD language uses long keywords like WHENEVER but allows the programmer to abbreviate them to things like W'R. Although understandable in the era of punch cards, this makes the code difficult to read (and OCR).
- Source code was often created on punch cards at that time, so column layout was important. For example, to enter lines longer than 78 characters, one punched a digit in column 11 of continuation cards.

Missing Functions and Other Minor Issues

As we got each piece compiled, we realized that several functions were missing:

- `BCDIT`, used in function `FRBCD`, was missing. It appears to convert a binary number to a BCD string. We replaced this with the CTSS library routine `DELBC`.
- `INLSTL` used in `XMATCH` and `ASSMBL`, two of the most critical ELIZA functions, was missing. It appears in the Fortran implementations of SLIP. It takes the cells of the first parameter and inserts them to the left of the list cells of the second parameter. We wrote our own in MAD.
- The `LETTER` function was not in either the MAD or SLIP documentation and was not in the archive code, and Google was not helpful. Inference from the calling code suggested that it classified characters in a word into one of 14 categories. (An equal sign ("=") returns 1, a comma (",") returns 13, the digits "0"–"9" return 12, etc.) We wrote our own in MAD.
- The MAD compiler on CTSS seems not to support `$$` in string constants to mean emit a single `$`. The 1963 MAD manual confirms this, saying there is no clean way to put a `$` in a string, suggesting this somewhat bizarre workaround: `TEST=-$=$` will set `TEST` to "`$`."
- Some initialization of common data areas appeared to be missing, causing the loader to fail. We added code to ELIZA.MAD to initialize the public list `W(1):LIST.(W(1))`.
- The CTSS loader then failed due to too many object files. CTSS did have a "huge loader" where these limits were relaxed, but this component was not present in the emulated system. Luckily, the source code for the loader was available so we added it.

With all the above changes, ELIZA compiled and completed reading the script and even began to interact for the first time in over sixty years. Unfortunately, it did not get very far:

```
r eliza
W 850.1
EXECUTION.
WHICH SCRIPT DO YOU WISH TO PLAY
100
```

```
 DOCTOR PLEASE TYPEWR PROBLE
INPUT
Men are all alike.

 PROTECTION MODE VIOLATION AT 23174.
 INS.=060671060671,  RI.=000000000000,
  PI.=062061000000
R .666+.000
```

It took a fair amount of print-log debugging to narrow this down, but we finally found a single character error deep in the FAP code, at line 1670 of the function `SQIN.FAP`. We changed "`LAS =H 0000`" to: "`LAS =H 00000`." (Of course, it always comes down to a one-character typo!)

IT IS ALIVE!

Lane built a test environment where changes could be rapidly tested and was eventually able to produce an executable, and on Saturday, 21 December 2024 at 10:54 GMT, Rupert announced the following to the rest of the team:

```
I'm happy to announce that ELIZA is now running
  again on CTSS!

r eliza
        W 1835.0
        EXECUTION.
        WHICH SCRIPT DO YOU WISH TO PLAY
        200
        HOW DO YOU DO . PLEASE TELL ME YOUR
          PROBLEM
        INPUT
        Men are all alike.
        IN WHAT WAY
```

After all this effort, the reanimated ELIZA, not having run for over sixty years, carried out complete conversations. Again, although a lot of work went into reaching this point, most of the work was around the margins or underneath; 96 percent of the running ELIZA and SLIP code was exactly as it was found in the archive.

DOCTOR UNDER THE KNIFE

There was still one problem standing between us and the goal of demonstrating that ELIZA actually worked. ELIZA uses a script containing the language rules that determine its persona. The most famous of these is, of course, the "DOCTOR" script, which leads ELIZA to interact in what seems to be a caricature of a Rogerian therapist.

The code that we discovered in Weizenbaum's archived papers, the code that we are running here, had two early versions of the DOCTOR script attached (literally, in the sense of being on the same fan-fold print out). Their names are given in the printout as `.TAPE. 100` and `.TAPE. 102`. However, we had no example of a conversation with these scripts. After analysis, we used a corrected version of the 1966 paper's script (which we called 200) because we could compare the I/O with the published transcript. Of course, we knew that some functionalities (e.g., `PRE`) are unimplemented in the present code. Regardless, in order to compare the found ELIZA's performance with the only published conversation, we used script 200 with some corrections for typos. In this way we were able to nearly exactly recreate the famous "Men are all alike" conversation.

For his prior explorations in ELIZA, Hay had a clean transcript of the 1966 DOCTOR script wherein he had already corrected several apparent typos, including lines that appear to have been replicated at each point in the fan-fold printout (every 34 lines). A few lines that were longer than 78 characters were reflowed.

We made one change in the prompt:

```
He says I am depressed much of the time.
```

The original prompt was as follows:

```
He says I'm depressed much of the time.
```

which, as a consequence of assuming that the `PRE` rule worked, produced

```
PRE 5:$5:$ 5QZ5QZ
```

and then ELIZA might crash a few turns later.

The other transcript difference was for the following prompt:

```
You are like my father in some ways.
```

which always outputs `=DIT` because this version of ELIZA does not understand `PRE` rule and only supports `=keyword` references at the transformation rule as described above.

Using the 1966 script, the reanimated ELIZA followed the *CACM* dialogue except in those two places. Lane [9] is a video of the reanimated ELIZA holding the conversation from Weizenbaum's paper.

A MAJOR BUG

Running this early ELIZA proved that it in fact worked and that it was very close to the 1966 ELIZA! However, running the actual program also led us to discover a significant bug that would have been very difficult to find by mere inspection of the code. This version of ELIZA does not handle numerical inputs correctly; it will crash if given numeric input such as "you are 9 today."

We tracked this problem to a complex issue in SLIP and the way ELIZA used it. When entered, a number is turned into what SLIP interprets as a pointer, which thence dereferences to some arbitrary location in memory. This generally leads to no good end. We return to this issue in the section "ELIZA's Final Analysis."

THE RAIN IN SPAIN

In 1966, Joseph Weizenbaum briefly mentions being able to teach ELIZA:

> Its name was chosen to emphasize that it may be incrementally improved by its users, since its language abilities may be continually improved by a "teacher". Like the Eliza of Pygmalion fame, it can be made to appear even more civilized[...]. [20, p. 36]

We find this mysterious because one would think that, this being the *raison d'être* of the program's name, Weizenbaum would discuss this topic at length. However, this is the only mention of this capability anywhere in Weizenbaum's writings. Interestingly, the code we reconstructed and reanimated contains the complete functionality for this "teacher" mode, including functionality for editing and saving the script.

Typing "+" to ELIZA takes it out of conversation mode and into "teaching" mode. ELIZA responds with PLEASE INSTRUCT ME and waits for the user to enter any of seven commands: ADD <keyword>(<transformation rule>) adds the given transformation rule to an existing keyword rule; APPEND <keyword>(<existing reassembly rule>) (<new reassembly rule>) appends the new reassembly rule to the transformation rule with the existing reassembly rule in the given keyword; SUBST <keyword>(<existing reassembly rule>) (<new reassembly rule>) replaces the existing reassembly rule with the new one in the given keyword; TYPE <keyword> displays the script rule for the given keyword; DISPLA displays the rules for all keywords, as well as the MEMORY rules; RANK <keyword><n> allows the user to set the precedence for a keyword; and START returns ELIZA to the conversation mode. Furthermore, a "star" ("*") command, used at the normal input, allows the "teacher" to add an entirely new rule.

The sidebar entitled "A Complete Example of Teaching ELIZA New Rules" contains a complete "teaching" interaction with our reanimated ELIZA.

Once the script has been edited, it is useful to save it. Giving no input (just pressing ENTER at the INPUT prompt) will cause ELIZA to ask for a pseudo-tape number to write its rules to. Entering 0 (or giving no input at all) leads it to print the script to the console, and entering a number, say, 130, will write to disk file: `.TAPE. 130`. The output is a well-formed S-expression but with different spacing than we usually see in external scripts that have been edited by hand or reformatted (sometimes incorrectly!) for publication. This can also be observed in the sidebar "A Complete Example of Teaching ELIZA New Rules."

Although this "teaching" mode was completely implemented, Weizenbaum appears to have abandoned it in later versions of ELIZA, given the availability of text editors in CTSS. Indeed, elsewhere in his paper, he describes invoking ED from ELIZA by using the EDIT command during a conversation. This does not exist in the found code.

ELIZA'S FINAL ANALYSIS

While simple by today's standards, ELIZA was groundbreaking for its time and is now embedded in the AI mythos. It brought to life the concept, mere science fiction to that point, of holding a conversation with a computer and was the first program to directly embody Turing's test. It showed, in part due to its overt lack of any machinery that could constitute "understanding," that one may not have to actually *be* intelligent to *appear* intelligent. Up to that point, and indeed even through the recent apparent intelligence of LLMs, people had been (and have been) trying to model how people thought and code those processes into computers (e.g., [10]).

While most of us cannot put ourselves in the shoes of someone living in 1966, it is fascinating to experience a conversation with the original ELIZA, even if it breaks on occasion. If the code we discovered did not work at all, it would be very disappointing. But it does work, with the exception of the number bug.

We have struggled with whether to repair the number bug. Our feeling is that running the original code feels good and authentic. Finding bugs in it only adds to the authenticity. Any changes or bug fixes we make detract from that, except the minor interventions necessary to get the code running at all. Put in more academic terms, it is important that programs discovered in archaeological research of this sort retain as much of their original nature as possible [2]. We needed to make repairs at the margin in order to get the discovered code running, but none of these required significant changes to that code, as would be required to repair the number bug.

We continue to search for additional versions of ELIZA. We would appreciate it if readers of this article

A COMPLETE EXAMPLE OF TEACHING ELIZA NEW RULES

Comments are enclosed in square brackets.

```
% telnet 0 7094        [CTSS is listening to local port 7094.]
Trying 0.0.0.0...
Connected to 0.
Escape character is '^]'.
s709 2.4.1 COMM tty0 (KSR-37) from 127.0.0.1

MIT8C0: 1 USER AT 01/08/14 1915.8, MAX = 30
READY.

login eliza
W 1915.9
Password
 YOU HAVE 10L SAVED
 M1416 10 LOGGED IN 01/08/14 1915.9 FROM 700000
 LAST LOGOUT WAS 12/24/14 048.6 FROM 700000
 HOME FILE DIRECTORY IS M1416 ELIZA

THIS IS A RECONSTRUCTED CTSS SYSTEM.
IT IS A DEBUG AND NOT FULLY FUNCTIONAL VERSION.

 CTSS BEING USED IS: MIT8C0
R .016+.000

r eliza
W 1921.6
EXECUTION.
WHICH SCRIPT DO YOU WISH TO PLAY
200
HOW DO YOU DO . PLEASE TELL ME YOUR PROBLEM
INPUT
i want a pony

WHAT WOULD IT MEAN TO YOU IF YOU GOT A PONY
INPUT
* hi        [Adding a new rule via * command]
((hi eliza) (how do you do))
((hi 0) (hello from 2))

INPUT
hi eliza

HOW DO YOU DO        [This hits the first transformation]
INPUT
hi there you fun program you
          [This hits the second transformation]
HELLO FROM THERE I FUN PROGRAM I
INPUT
+           [Entering 'teaching' mode]

PLEASE INSTRUCT ME
INPUT
type hi
          [Display our new rule]
HI

HI ELIZA
 1
HOW DO YOU DO

HI 000000
 1
HELLO FROM 000002
PLEASE INSTRUCT ME
INPUT        [Change one of the rules in 'hi']
subst hi (hello from 2) (goodbye 2)

PLEASE INSTRUCT ME
INPUT
type hi        [Let's check that it's actually changed]

HI

HI ELIZA
 1
HOW DO YOU DO

HI 000000
 1
GOODBYE 000002 [Looks good]
PLEASE INSTRUCT ME
INPUT
start        [Back to test our new rules]

INPUT
hi to all my friends
          [Oops! 'my' has higher priority (2)]
YOUR FRIENDS
INPUT
hi to the cats of the world

GOODBYE TO THE CATS OF THE WORLD
```

(Continued)

(*Continued*)

```
INPUT
+
rank hi 9         [Let's rank 'hi' above 'my']

PLEASE INSTRUCT ME
INPUT
type hi
         [Check that the re-rank worked]
HI 000009

HI ELIZA
 1
HOW DO YOU DO

HI 000000
 1
GOODBYE 000002
PLEASE INSTRUCT ME
INPUT
start

INPUT
hi to all my friends

GOODBYE TO ALL YOUR FRIENDS   [Looks good!]
INPUT
+
         [Let's change one of the given rules]
PLEASE INSTRUCT ME
INPUT
type your

YOUR = MY

000000 MY 000000
WHY ARE YOU CONCERNED OVER MY 000003
WHAT ABOUT YOUR OWN 000003
ARE YOU WORRIED ABOUT SOMEONE ELSES 000003
REALLY, MY 000003
PLEASE INSTRUCT ME
INPUT
start
my foot itches

INPUT
my foot itches

WHY DO YOU SAY YOUR FOOT ITCHES
```

```
INPUT
your foot itches

WHY ARE YOU CONCERNED OVER MY FOOT ITCHES
INPUT
+         [Changing one of the rules for YOUR/MY]
subst your (WHY ARE YOU CONCERNED OVER MY 3) (worry about
your own 3)

PLEASE INSTRUCT ME
INPUT
type your         [Did it take?]


YOUR = MY

000000 MY 000000
 1
WORRY ABOUT YOUR OWN 000003
WHAT ABOUT YOUR OWN 000003
ARE YOU WORRIED ABOUT SOMEONE ELSES 000003
REALLY, MY 000003
PLEASE INSTRUCT ME
INPUT
start         [Let's test it!]

INPUT
your spoon is too big
         [Roll through all the transformations]
WHAT ABOUT YOUR OWN SPOON IS TOO BIG
INPUT
your spoon is too big

ARE YOU WORRIED ABOUT SOMEONE ELSES SPOON IS TOO BIG
INPUT
your spoon is too big

REALLY, MY SPOON IS TOO BIG
INPUT
your spoon is too big

WORRY ABOUT YOUR OWN SPOON IS TOO BIG   [Tada!]
INPUT

[User enters a blank line (double ENTER) causing ELIZA to dump
the script.]

WHAT IS TO BE THE NUMBER OF THE NEW SCRIPT
```

(*Continued*)

(*Continued*)

```
[User doesn't give a tape number, so the script is dumped to the
teletype.]
[Only affected entries are shown here.]

 (HI 000009 ((HI ELIZA)
000001 (HOW DO YOU DO)
) ((HI 000000) 000001 (GOODBY
E 000002)))

[...]

 (YOUR = MY ((000000 MY
 000000) 000001 (WORRY ABOUT YOUR
 OWN 000003) (WHAT ABOUT YOUR
 OWN 000003) (ARE YOU WORRIE
D ABOUT SOMEONE ELSES 000003)
 (REALLY, MY 000003)))

[...]

[This is ELIZA's normal exit:]
EXIT CALLED. PM MAY BE TAKEN.
R 1.250+.016
```

with access to archives that might contain versions of ELIZA would contact us.

DIY ELIZA

We have posted a video of the reanimated ELIZA holding the conversation from the 1966 *CACM* paper [9]. Readers with access to a Unix-based machine can download and build CTSS and ELIZA to run on your own computer [7]. This has been tested on various Linux and MacOS versions. Thanks in advance for reporting discoveries, issues, or fixes. A detailed "reconstruction card" is also available [8].

ACKNOWLEDGMENTS

We thank Dave Pitts for his amazing 7094 emulator and CTSS kit, Jerry Saltzer and Tom Van Vleck for contributing their deep knowledge of CTSS and its history, the MIT archivists especially Myles Crowley and Allison Schmitt for their diligent research, and Pm Weizenbaum and the Weizenbaum estate for permission to open source the code. This article was greatly improved through the guidance of *IEEE Annals of the History of Computing* editors David Hemmendinger and Alexander Magoun. This work could not have happened without the support and contributions of Team ELIZA, a research group looking into the history of the original ELIZA: David M. Berry, Sarah Ciston, Anthony Hay, Mark C. Marino, Peter Millican, Arthur Schwarz, Jeff Shrager, and Peggy Weil. Rupert Lane did the bulk of the hands-on work to bring up the Pitts emulator and compile and debug ELIZA. His efforts were supported by extensive background work and additional research by Anthony Hay and Art Schwarz, who spent several years studying the code and so had extensive prior knowledge of ELIZA, SLIP, and FAP. David Berry recognized the opportunity to bring the original ELIZA up on an emulator, made the initial connection to Lane, and joined Jeff Shrager in a second spelunking tour of the archives. Shrager discovered the original code in Joseph Weizenbaum's archives and composed this article (largely from e-mail discussions among the authors). Lane, Shrager, and Hay have tested the download-and-run process. All authors have read, improved, and approved this article.

BIBLIOGRAPHY

[1] A. G. Bell. "The history and mystery of ELIZA." CoRecursive Podcast. Accessed: Jan. 24, 2025. [Online]. Available: https://corecursive.com/eliza-with-jeff-shrager

[2] D. M. Berry. "Digital ruins and critical code studies: Towards an ethics of historical software reconstruction." Stunlaw. Accessed: Jan. 26, 2025. [Online]. Available: https://stunlaw.blogspot.com/2025/01/digital-ruins-and-critical-code-studies.html

[3] "Commonly known ELIZA clones." Elizagen. Accessed: Jan. 24, 2025. [Online]. Available: https://sites.google.com/view/elizagen-org/eliza-clones

[4] "Other Eliza-like programs." Elizagen. Accessed: Jan. 24, 2025. [Online]. Available: https://sites.google.com/view/elizagen-org/eliza-likes

[5] A. Hay. "ELIZA." GitHub. Accessed: Jan. 24, 2025. [Online]. Available: https://github.com/anthay/ELIZA

[6] "IBM 7094 principles of operation," IBM, Poughkeepsie, NY, USA, Form A22-6703-4, Oct. 1966. Accessed: Jan. 24, 2025. [Online]. Available: https://bitsavers.org/pdf/ibm/7094/A22-6703-4_7094_PoO_Oct66.pdf

[7] R. Lane. "eliza-ctss." GitHub. Accessed: Jan. 24, 2025. [Online]. Available: https://github.com/rupertl/eliza-ctss

[8] R. Lane, "eliza-ctss/RECONSTRUCTION-CARD.md." GitHub. Accessed: Jan. 26, 2025. [Online]. Available: https://github.com/rupertl/eliza-ctss/blob/main/RECONSTRUCTION-CARD.md

[9] R. Lane. *ELIZA Running on CTSS*. (Dec. 26, 2024). Accessed: Jan. 24, 2025. [Online]. Available: https://youtu.be/j5Tw-XVcsRE

[10] A. Newell and H. A. Simon, "Computer simulation of human thinking," *Science*, vol. 134, no. 3495, pp. 2011–2017, Dec. 1961, doi: 10.1126/science.134.3495.2011.

[11] S. North, "ELIZA," in *Creative Computing*. Morristown, NJ, USA: David Ahl, Jul./Aug. 1977, pp. 100–103.

[12] E. Organick, The Michigan algorithmic decoder (The MAD manual) revised edition – 1966, 1966. Accessed: Jan. 24, 2025. [Online]. Available: http://deepblue.lib.umich.edu/bitstream/2027.42/3292/5/bab2545.0001.001.pdf

[13] "original-eliza." Google Sites. Accessed: Jan. 24, 2025. [Online]. Available: https://sites.google.com/view/original-eliza

[14] D. Pitts. "IBM 7090/7094 page." cozx.com. Accessed: Jan. 24, 2025. [Online.] Available: https://www.cozx.com/dpitts/ibm7090.html

[15] A. Schwarz. "gSlip." slipbits.com. Accessed: Jan. 24, 2025. [Online]. Available: https://slipbits.com/gslip

[16] P. Seibel, *Coders at Work*. New York, NY, USA: Apress, 2009.

[17] J. Shrager, "ELIZA reinterpreted: The world's first chatbot was not intended as a chatbot at all," 2024, *arXiv:2406.17650*.

[18] D. Walden and T. Van Vleck, Eds., *The Compatible Time Sharing System (1961-1973): Fiftieth Anniversary Commemorative Overview*. Washington, DC, USA: IEEE Computer Society, 2011. Accessed: Jan. 24, 2025. [Online]. Available: https://multicians.org/thvv/compatible-time-sharing-system.pdf

[19] J. Weizenbaum, "Symmetric list processor," *Commun. ACM*, vol. 6, no. 9, pp. 524–536, Sep. 1963, doi: 10.1145/367593.367617.

[20] J. Weizenbaum, "ELIZA—A computer program for the study of natural language communication between man and machine," *Commun. ACM*, vol. 9, no. 1, pp. 36–45, Jan. 1966, doi: 10.1145/365153.365168.

RUPERT LANE is an independent researcher in Tokyo 141-0022, Japan. Contact them at rupert@timereshared.com.

ANTHONY HAY is located in Devon, EX31 4QP, U.K. Contact them at anthony.hay.1@gmail.com.

ARTHUR SCHWARZ is a chief technologist at Slipbits, Irvine, CA 92620 USA. Contact them at aschwarz@acm.org.

DAVID M. BERRY is a professor in the Faculty of Media, Arts and Humanities, University of Sussex, Falmer BN1 9RH, U.K. Contact them at d.m.berry@sussex.ac.uk.

JEFF SHRAGER is an adjunct professor in the Stanford University Symbolic Systems Program, Stanford, CA 94025 USA. Contact them at jshrager@stanford.edu.

IEEE COMPUTER SOCIETY

80th ANNIVERSARY CELEBRATION

FOLLOW US

LinkedIn
IEEE Computer Society

YouTube
@IEEEComputerSociety

EDITOR: Alexander B. Magoun, a.b.magoun@ieee.org

This article originally appeared in IEEE Annals of the History of Computing vol. 46, no. 3, 2024

DEPARTMENT: ANECDOTES

Stepping Into the Future—The Invention and Evolution of the Stepping Switch

Thomas K. Robbins, *retired, Focused Ingenuity Inc., Aberdeen Proving Ground, MD, 21005, USA*

The stepping switch, also known as a uniselector or rotary switch (Figure 1), was a transformative 19th-century technology that provided a step into the future, initially by enabling direct telephone dialing and, unexpectedly, online, real-time, networked computation, among other applications.

My interest with these devices began while working at AmTote International as its photographer. AmTote International is a descendant of Henry (Harry) L. Straus's American Totalisator Company. In the early 1990s, I received a request from Boston's television station WGBH for photographs of Straus and his company. WGBH and the British Broadcasting Corporation (BBC) were producing a documentary series, *The Machine That Changed the World*, on the history of digital computers. The producers wanted to include Straus since he was an early investor in J. Presper Eckert and John Mauchly's company—Eckert Mauchly Computer Corporation, arguably the world's first computer company.

My archive had few documents from that period, but I sent a film, *The American Totalisator Story*, that included photos of Straus and his invention—the totalizator (tote). The film showed how the tote's stepping switches controlled the adding function for and processed amounts bet on horses or greyhounds. Clips from the film were included in episode two, "Inventing the Future" [2]. The stepping switches seemed familiar at the time, and I remembered reading about them while working at the National Security Agency. The switches were used in the PURPLE cipher machine during World War II.

As my employer's founder, Straus made me curious. The son of a wealthy family who graduated in mechanical engineering from the Johns Hopkins University in 1917, Straus combined his love of horse racing, inventiveness, and family connections into an electromechanical tote that eventually dominated racetracks in the United States. A biography refers to his work on Navy projects during World War II [19]. Could his stepping switches be related to the ones used in the PURPLE machine? American Totalisator operated a wartime factory in Park Ridge, Illinois, which manufactured rotary switches used in control systems aboard submarines [15]. I was fascinated by this possibility and wanted to learn more about Straus's involvement during the war. Although I found no evidence that American Totalisator provided parts for the codebreaking effort, drawings exist for their Navy work, which included a binomial sequential analyzer and a radar-related device. My research filled in, however, some of the stepping switch's relations to automated calculation, cryptography, and computing over its 100 years of use.

INVENTION—STEPPING INTO THE FUTURE

The stepping switch traces its origins to the automatic telephone exchanges that filled a void left open by the American Telephone and Telegraph Company's manual telephone exchanges at the end of the 19th century.[1] These required human operators to connect callers and serviced the big cities, leaving the less profitable exchanges in small towns across the United States to telephone entrepreneurs like Almond Strowger, a Kansas City, Missouri, undertaker. He thought operators were directing potential customers to his competitors and wanted to eliminate them from the network. As Alexander Bell's patents expired [8, pp. 550–551], Strowger invented an automatic telephone exchange that eliminated the need for switchboard operators [11].

Numerous inventors had filed patents on automatic telephone exchanges between 1879 and 1891, but Strowger's was the first that worked in a community. He enlisted Alexander Keith and John and Charles Erickson, who further refined his system using piano wire and a rotary switch and then vertical banks of 10-point rotary switches, which became the stepped system

---

[1] The basic Bell patents expired in 1893–1894. Prior to 1892, about 60 independent companies had been formed. Fifteen years later, more than 5,000 independent companies owned about 45% of U.S. telephones.

1058-6180 © 2024 IEEE
Digital Object Identifier 10.1109/MAHC.2024.3437028
Date of current version 12 September 2024.

for future exchanges [7, pp. 60–72]. Keith and the Ericksons also introduced a line finder and a rotary dial telephone, which reduced the complexity of the initial selector and simplified the exchange architecture.

Strowger and Keith organized the Automatic Electric Company (AEC) to manufacture their equipment. As AEC began distributing Strowger exchanges globally, it needed local manufacturing in countries where it did business. In England, the Automatic Telephone Manufacturing Company (ATM) was formed in 1912 as a subsidiary. It manufactured Strowger telephone exchanges and exported them to Canada, South America, Europe, Australia, and Asia [7, p. 212].

The German telecommunications manufacturer Siemens-Halske, on the other hand, obtained Strowger licenses in 1907 when it bought another company and its rights [7, p. 214]. After a director of Siemens-Halske toured factories in the United States with several engineers, he hired engineer Fritz Lubberger of the Automatic Electric Company to oversee production in Europe. In 1910, Siemens-Halske's engineers introduced the first of a series of improvements to Automatic Electric's stepping switches, including a lighter uniselector that was adopted and adapted by others [14].

HOW IT WORKS

The basic operation of the switch connects one electrical line to another by moving a pair of spring contacts, or wipers, in an arc that touches stationary contacts, thereby joining the circuits. The stepping switch was configured for telephone exchanges as a line finder or uniselector, which passed the pulse of a dialed number through the exchange to connect with the receiver phone. The subscriber phone was hard wired to the exchange, so there was one phone to one line finder [24].

TWO DECADES OF INNOVATION: 1928–1948

Manufacturers not only modified the stepping switch's design, but they also applied it to radio broadcasting, horse and dog race betting, codebreaking, and digital computation using binary addition [12].

In 1928, the BBC used stepping switches to automate the connection of radio programs from its studios over telephone lines to the desired radio transmitters. Formerly in charge of the General Post Office (GPO) Lines Section and its design and maintenance of the nation's telephone switching circuits, A. S. Attkins led the BBC's transition from a manual switching architecture to an automatic one [16]. At Piccadilly broadcasting station in Manchester, Attkins replaced the manual switchboard with two-motion stepping switches for simultaneous broadcasting across the country using GPO lines. The noise of the stepping switches (likened to machine gunfire—discussed later in the article) was more acceptable in telephone exchanges than for radio broadcasts, however, and the BBC redesigned the system using relays with platinum–gold–silver contacts. Troubles persisted with the automatic exchanges that were not resolved until after World War II with improved telephone line-finder equipment [22].

Around the same time, Henry L. Straus, a mechanical engineer and horseman, had a bad day in 1927 at

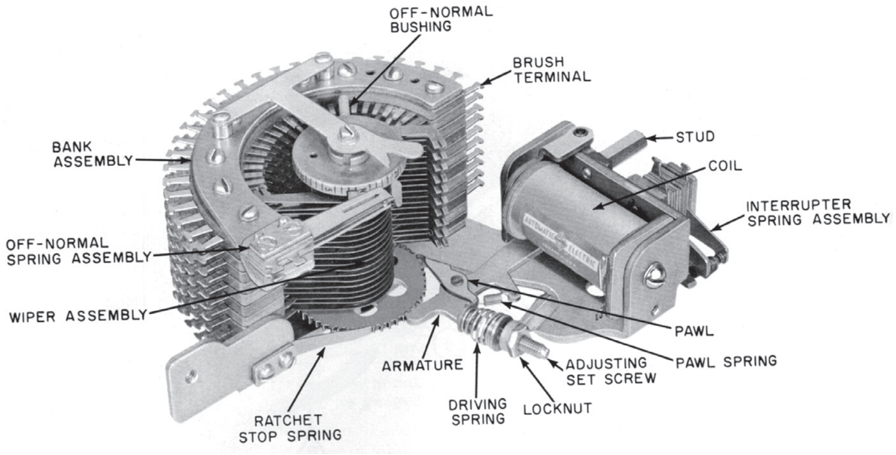


FIGURE 1. Automatic Electric rotary switch Type 45 from *How to Use Rotary Stepping Switches* (1964), with permission from Nokia Corporation. The switch's many moving parts required accurate adjustments and cleaning since dust would interfere with the wipers and contacts.

the Havre de Grace Racetrack in Maryland, USA. He and his friends disliked that the payout of their considerable bet differed from what was displayed. Straus recollected that they "vented their opinions of the management in such vitriolic terms that I would blush to repeat them if I had not so often used worse. However, it was the unanimous opinion of all of us that it would be a good idea if a machine could be built that would constantly and accurately display to the public the trend of betting" [23].

Straus, whose company sold industrial power equipment, asked his friend John Upp, a manager at General Electric, for help. Upp assigned one of his control systems engineers, Arthur J. Johnston, to work with Straus. In 1928, they designed an electromechanical tote using telephone relays and stepping switches to total bets placed and display the odds and payouts. Each betting pool, win or place, had a separate adder (Figure 2) that could total bets placed in that pool. As the teller keyed in the amount, the switch would count and step to the next higher number. When it reached ten, it would switch the circuit to the next switch to keep track of the tens digit and progress to the hundreds and thousands digits. The totals would be updated on the indicators located on tote boards surrounding the track.

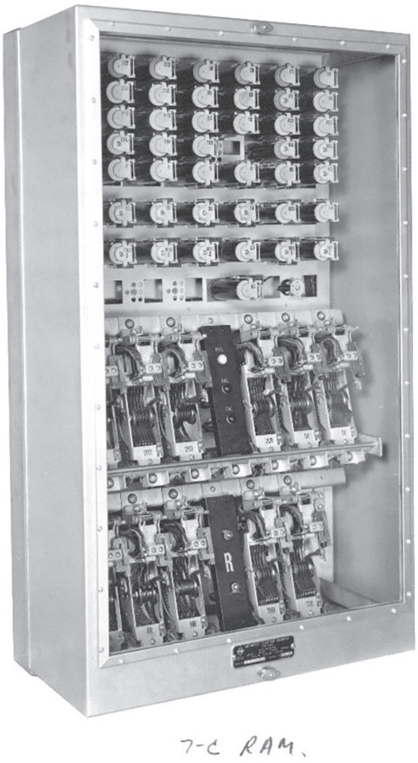


FIGURE 2. Relay tote adder (no date)—AEC component. (Source: AmTote International; used with permission of 1st Corporation.)

Only a few states permitted betting, however, and Straus's friend Jarvis Spencer, chair of the Maryland Racing Commission, suggested he look to England for an opportunity. The British Parliament had passed The Totalisator Act that year, requiring the technology's use at racecourses [3]. There were few tote manufacturers at the time, with the British Automatic Totalisator Company, the largest, using George Julius's electromechanical tote, first installed in 1917 [25].

General Electric arranged for its British subsidiary Thomson-Houston and ATM to demonstrate Straus's system to the Racecourse Betting Control Board. Straus and Johnston traveled to England for their demonstration at the Thirsk Racecourse in January 1930 [26]. The Board made Straus and Johnston's design the industry standard, but British law required its manufacture in England. Straus sold his British rights to ATM, which had a factory dedicated to Strowger switches in Liverpool, and returned to the United States, where he installed his first tote in the United States at Arlington Park, Chicago, in 1933. This fueled the explosion of racetrack betting in the United States, and, by the 1950s, millions of dollars had been bet through his systems. Relay totes were still in use at small fairs around the country at the end of the 20th century.

During the 1930s, other computational events were happening in which the stepping switch played a part. In Germany, Konrad Zuse developed a program-controlled calculator—the Z3—to assist with aircraft design. He and Helmut Schreyer used telephone stepping switches and relays for the Z3's binary calculators, which they cannibalized from a dealer selling second-hand telephone equipment [20].

The Boolean algebraic solution to processing data was formalized by Claude E. Shannon, who wrote his 1938 master's thesis at the Massachusetts Institute of Technology on "A Symbolic Analysis of Relay and Switching Circuits." This proved that Boolean algebra could be used to simplify the configuration of relays used in automatic telephone exchanges. Like the inventors of electromechanical data processing systems, Shannon recognized that "There are many special types of . . . stepping switches and selector switches of various sorts, multi-winding relays, cross-bar switches, etc." He went a step further, however, by observing that "The operation of all these types may be described with the words 'or,' 'and,' 'if,' 'operated,' and 'not operated'. This is a sufficient condition that may be described in terms of hinderance functions with the operations of addition, multiplication, negation, and equality" [21].

The 1930s also saw the Japanese and the Americans adopt the stepping switch's digital characteristics for coding and decoding messages. The Japanese used stepping switches in three cipher machines that the Americans termed CORAL, JADE, and PURPLE [6]. The stepping switches' wiper positions encrypted either the Roman alphabet or Japanese katakana syllabary.

The PURPLE (Figure 3) development began with Japanese Navy Captain Risaburo Ito, who was charged with developing a cipher machine to encode diplomatic messages. His team included Eikichi Suzuki, who recommended using stepping switches instead of half-rotor switches [9]. By 1937, they had invented the *97-shiki ōbun injiki* (九七式印字機), or System 97 Printing Machine.

The System 97/PURPLE had three main components: a plugboard for inputs, stepping switches to create a permutation of letters, and an output plugboard. The input plugboard was split between the keyboard alphabet and an internal alphabet used to encrypt the message. This alphabet was split between six vowels and 20 consonants. The vowels were permutated by 25 positions on a stepping switch, creating 720 possible permutations. The consonants were permutated for a possible 25^3 combinations.

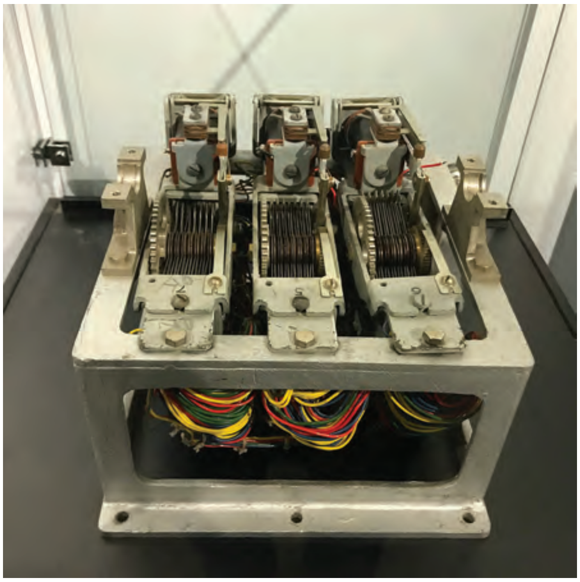


FIGURE 3. Three stepping switches in part of a "PURPLE" cipher machine recovered from the Japanese embassy in Berlin after World War II. (Source: National Cryptologic Museum, Maryland, USA; used with permission.)

The Americans had broken the PURPLE code but needed a faster method to decrypt the messages. In 1939, electrical engineer Leo Rosen was assigned the task and thought that a uniselector he noticed in an electronics catalog would be suitable for a cipher machine. Colonel Spencer Akin, the officer leading the PURPLE decrypts team, asked a friend at a telephone supply company in Chicago to airmail the switches to Washington, DC. There, Rosen built a machine in two weeks at a cost of US$290.97. It consisted of 13 25-point, six-layer telephone stepping switches [10].

Codebreaking efforts were also underway at Bletchley Park, England. The German Army High Command was using C. Lorenz AG's rotor stream cipher machine to encrypt teleprinter messages. In 1941, the Germans made a mistake that allowed the British to decipher the messages by hand but, as with PURPLE, a faster way was needed. The Tunny machine was built with a rack of uniselectors to mimic the Lorenz teleprinter's logic but was still too slow, which led to the development of Tommy Flowers's Colossus.

Flowers, the telephone engineer in charge of the GPO's switching research, began his codebreaking career in 1941, when he was assigned to Bletchley Park to work with Max Newman to design a machine that would speed up the decoding for the Lorenz cipher. Flowers's solution, Colossus, contained about 1,600 vacuum tubes and used stepping switches (Figure 4) to mark the wheel start positions [18].

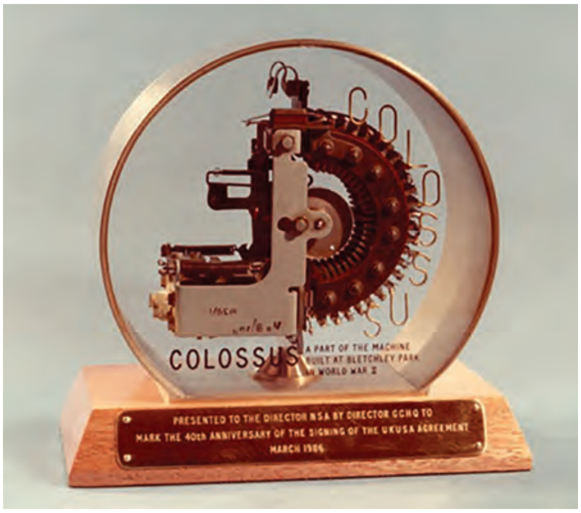

COLOSSUS
COLOSSUS
A PART OF THE MACHINE
BUILT AT BLETCHLEY PARK
WORLD WAR II
PRESENTED TO THE DIRECTOR NSA BY DIRECTOR GCHQ TO
MARK THE 40th ANNIVERSARY OF THE SIGNING OF THE UKUSA AGREEMENT
MARCH 1986

FIGURE 4. Stepping switch from the first Colossus that was presented to the director of the National Security Agency by the director of GCHQ in 1986. (Source: National Cryptological Museum; used with permission.)

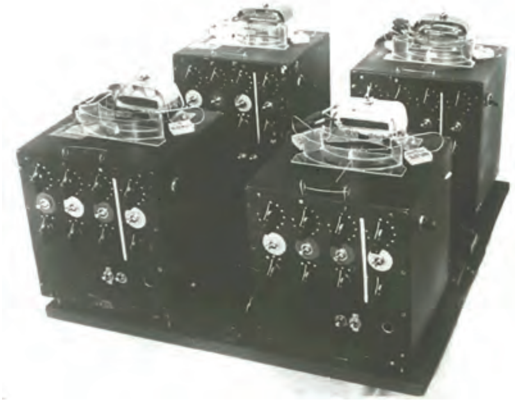


FIGURE 5. Homeostat. [Source: Reproduced from the W. Ross Ashby Digital Archive (https://ashby.info); used with permission of the Estate of W. Ross Ashby.]

In addition to the Lorenz teleprinter, the German military used a modified commercial Enigma cipher machine. Polish codebreakers had been decoding German Enigma messages since 1934 and found that the Germans connected the plugboard alphabetically, allowing them to duplicate the wiring. Cryptographer Marian Rejewski devised the "Bomba" that could process 17,000 rotor positions electromechanically and generate the key to decipher a message [30].

Drawing on Rejewski's design, Alan Turing invented a similar machine, the Bombe. It searched for texts that the cryptanalysts assumed were in a message and used stepping switches to mimic the movement of the Enigma rotors [27]. Gordon Welchman, working with Turing, developed a "diagonal board" to complement the Bombe. Since the Enigma plugboard used cables to connect one letter to another, Welchman duplicated this for the Bombe via his board.

When Turing visited the National Cash Register (NCR) Corporation's factory in Dayton, Ohio, to review the manufacture of Bombes for allied codebreakers, he observed that, to set up the menus, NCR used stepping switches instead of the "Jones" plugs that the British used. Turing reported that the "'diagonal board' is wired to several uniselectors of 26 positions and 26 wipers each. There is one uniselector associated with the input and one with the output of each enigma" [27]. He thought that this method would produce "an awful lot of wire" but saw the advantage in eliminating an independent diagonal board, thus speeding the setup. The British Tabulating Machinery Company also made Bombes for Bletchley Park and outfitted some, Jumbo Bombes, with a stepping switch called the "machine gun." These made a sound that resembled a machine gun firing but detected the diagonal board settings, similar to the Dayton machines [28].

The gradual transition to electronic data processing and computing continued after World War II and saw the rise of numerous electronic computer development efforts. Although electromechanical switches were slower, Harvard's Mark II used stepping switches to process subroutines and branching programs [4], the University of Cambridge's EDSAC (Electronic Delay Storage Automatic Calculator) incorporated them to control its bootstrap sequence, and the U.S. Navy's ABEL used relays and uniselectors to test the logic design of the Atlas I computer in 1949 [5].

The stepping switch also supported research in cybernetics. Ross Ashby built a homeostat (Figure 5) to mimic the brain as part of early studies on how computers could contribute to robotics and artificial intelligence [1]. The machine modeled biological homeostasis through an electromechanical system, which incorporated stepping switches from four surplus Royal Air Force bomb control devices. The machine's responses to stimuli were controlled by the uniselectors cycling through 25 randomly selected resistors whose values were chosen by a table of random numbers [13].

POLISH CODEBREAKERS HAD BEEN DECODING GERMAN ENIGMA MESSAGES SINCE 1934 AND FOUND THAT THE GERMANS CONNECTED THE PLUGBOARD ALPHABETICALLY, ALLOWING THEM TO DUPLICATE THE WIRING.

The stepping switch found uses over 100 years and helped usher in the digital information age through improved connectivity and data processing. Despite its electromechanical nature, resourceful engineers applied stepping switches well beyond telephone networks to data scanning and aggregation, computation, memory, and security. Consequently, courts struck down patents for electronic versions of these techniques as "not new to data processing" [29]. Employees of Universal Controls, which bought the late Straus's American Totalisator, asserted in the mid-1960s that "We were putting in on-line systems at racetracks before anyone knew that they were called that" [17]. In other words, under the cover of horse blankets and greyhound kennels, the stepping switch of Strowger et al. contributed to computational advancements that revolutionized the way we live.

ACKNOWLEDGMENTS

The author would like to thank Rob Simpson from the National Cryptological Museum for help with cipher machine documents and manuscript review; Dr. Paul Ceruzzi and Dr. Brian Randell for their encouragement and review of the initial draft; Karen Walsh for her editorial review; and Dr. Alexander Magoun for his guidance, edits, and acceptance of the manuscript for publication.

BIBLIOGRAPHY

[1] W. R. Ashby. “The W. Ross Ashby digital archive.” The W. Ross Ashby Digital Archive. Accessed: May 27, 2024. [Online]. Available: https://ashby.info

[2] A. Baio. “The machine that changed the world: Inventing the future.” WGBH Boston, WGBH Television Series Episode. YouTube. Accessed: Mar. 14, 2023. [Online Video]. Available: https://www.youtube.com/watch?v=GropWVbj9wA%22youtube.com/watch?v=GropWVbj9wA

[3] D. Berry, *Hi-Lites in Tote History*. Baltimore, MD, USA: American Totalisator Company, 1961.

[4] R. M. Bloch, “Oral history interview with Richard M. Bloch,” Charles Babbage Institute, University of Minnesota, Minneapolis, MN, USA, 1984. Accessed: Jul. 28, 2024. [Online]. Available: https://hdl.handle.net/11299/107123”hdl.handle.net/11299/107123\

[5] D. Buck, I. Dey, *The Cryotron Files: The Strange Death of a Pioneering Cold War Computer Scientist*. London, U.K.: Icon Books, 2018, p. 48.

[6] C. B. Burke, “It wasn’t all magic: The early struggle to automate cryptanalysis, 1930s-1980s,” United States Cryptologic History Special Series, vol. 6, Center for Cryptologic History/National Security Agency, Fort Meade, MD, USA, 2002. Accessed: Mar. 10, 2023. [Online]. Available: https://www.cryptomuseum.com/cam/files/NSA_CryptHist_V6_grey.pdf

[7] R. J. Chapuis, *100 Years of Telephone Switching Volume 1 (1878-1978)*. Amsterdam, The Netherlands: North Holland Publishing Company, 1982.

[8] M. D. Fagen, *A History of Engineering and Science in the Bell System – The Early Years (1875–1925)*. Murray Hill, NJ, USA: Bell Telephone Laboratories, Incorporated, 1975, pp. 550–551.

[9] W. Freeman, G. Sullivan, and F. Weierud, “PURPLE revealed: Simulation and computer-aided cryptanalysis of *Angooki Taipu B*,” *Cryptologia*, vol. 27, no. 1, pp. 1–43, 2003.

[10] T. Hannah. “Frank Rowlett, a profile.” Cryptologic Spectrum. Accessed: Oct. 30, 2023. [Online]. Available: nsa.gov/portals/75/documents/news-features/declassified-documents/cryptologic-spectrum/frank_rowlett.pdf

[11] R. B. Hill, “The early years of the Strowger system,” Bell Laboratories Record, Murray Hill, NJ, USA, 1953, vol. XXXI, no. 3, pp. 95–103. Accessed: Aug. 8, 2024. [Online]. Available: historyofphonephreaking.org/docs/blr-1953-03-early-years-of-strowger.pdf

[12] V. E. James, *How to Use Rotary Stepping Switches*, Northfield, IL, USA: Automatic Electric Company, 1964. Accessed: May 27, 2024. [Online]. Available: https://www.telephonecollectors.info/index.php/browse/bc-switching-library/automatic-electric/ae-switching-docs/10539-how-to-use-rotary-stepping-switches-automatic-electric-1964-ocr-r/file

[13] J. Johnston, *The Allure of Machinic Life Cybernetics, Artificial Life, and the New AI*. Denver, CO, USA: Braford Books, 2008, pp. 42–43.

[14] H. P. Mahoney, *Principles of Automatic Telephony, Circuits and Trunking*. Chicago, IL, USA: Automatic Electric Company, 1924.

[15] “Maryland in World War II: Industry and agriculture,” Maryland Historical Society. War Records Division, Baltimore, MD, USA, 1951, p. 71.

[16] E. Pawley, “BBC Engineering 1922-1972,” British Broadcasting Corporation Publications, London, U.K., 1972, p. 72. Accessed: May 10, 2023. [Online]. Available: https://bbc.com/historyofthebbc/documents/edward_pauley_bbc_engineering_1922-1972.pdf

[17] W. Rolph. “EDP conference for retailers.” Datamation. Nov. 1966, p. 77. Accessed: Jul. 28, 2024 [Online]. Available: https://bitsavers.org/magazines/Datamation/196611.pdf

[18] A. Sale, “Lorenz and colossus,” in *Proc. IEEE Comput. Secur. Found. Workshop*, Cambridge, U.K., 2000, p. 216 Accessed: Sep.29, 2023. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CSFW.2000.856938

[19] J. Schmidt, *Win Place Show: A Biography of Harry Straus the Man Who Gave America the Tote*. Baltimore, MD, USA: The G. W. C. Whiting School of Engineering, The Johns Hopkins University, 1989.

[20] H. T. Schreyer, “An experimental model of an electronic computer,” *IEEE Ann. Hist. Comput*., vol. 12, no. 3, pp. 189–197, 1990. Accessed: Oct. 13, 2023. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/MAHC.1990.10026

[21] C. E. Shannon, “A symbolic analysis of relay and switching circuits,” M.S. thesis, Dept. Elect. Eng., Massachusetts of Technol., Cambridge, MA, USA, 1940, p .29.

[22] L. G. Smith, “Engineering department in the 1930s led by Les “LG” Smith,” 2003. [Online]. Available: orbem.co.uk/misc/lg.htm Accessed: May 27, 2024. [Online]. Available: https://orbem.co.uk/misc/lg.htm

[23] H. L. Straus, “Travels with the ‘Tote’,” *The Johns Hopkins Alumni Magazine*. 1938, vol. 27, p. 9.

[24] “Strowger P-A-X,” Automatic Electric Company Bulletin 1026, Sep. 1929, p. 15. Accessed: Oct. 19, 2023. [Online]. Available: https://bitsavers.org/communications/automaticElectric/Strowger_P-A-X_Bulletin_1026_Sep1929.pdf

[25] D. Swade, “Forgotten machines: The need for a new master narrative,” in *Exploring the Early Digital*, T. Haigh, ed., Heidelberg, Germany: Springer, 2019, pp. 41–68.

[26] “The Tote at British race tracks.” Totalisator Demonstration at Thirsk, England, 1930. Pathe Newsreel Company. British Pathe. Accessed: Oct. 17, 2023. [Online Video]. Available: https://www.britishpathe.com/asset/90247/

[27] A. Turing, “Mathematical theory of ENIGMA machine by Alan Mathison Turing.” TNA HW 25/3, p. 119–121. Accessed: Jan. 24, 2024. [Online]. Available: https://discovery.nationalarchives.gov.uk/details/r/C6136633.

[28] A. M. Turing, “Visit to National Cash Register Corporation of Dayton, Ohio,” *Cryptologia*, vol. 25, no. 1, pp. 1–10, 2001.

[29] United States Court of Appeals, Second Circuit, Apr. 1977. Digitronics v. New York Racing Association 553 F.2d 740 [2d Cir. 1977]. casetext. Accessed: Aug. 8, 2024. [Online]. Available: https://casetext.com/case/digitronics-corp-v-new-york-racing-assn”casetext.com/case/digitronics-corp-v-new-york-racing-assn

[30] J. Wilcox, *Solving the Enigma: History of the Cryptanalytic Bombe*, Revised ed. Fort Meade, MD, USA: Center for Cryptologic History, National Security Agency, 2006.

THOMAS K. ROBBINS is retired in Havre de Grace, MD, 21078, USA. Robbins received an M.Sc. degree in computer system management. Contact him at tomrobbins@comcast.net.

ADVERTISER INFORMATION

Advertising Coordinator

Debbie Sims
Email: dsims@computer.org
Phone: +1 714-816-2138 | Fax: +1 714-821-4010

Advertising Sales Contacts

Mid-Atlantic US, Northeast, Europe, the Middle East and Africa:
Dawn Scoda
Email: dscoda@computer.org
Phone: +1 732-772-0160
Cell: +1 732-685-6068 | Fax: +1 732-772-0164

Southwest US, California:
Mike Hughes
Email: mikehughes@computer.org
Cell: +1 805-208-5882

Central US, Northwest US, Southeast US, Asia/Pacific:
Eric Kincaid
Email: e.kincaid@computer.org
Phone: +1 214-553-8513 | Fax: +1 888-886-8599
Cell: +1 214-673-3742

Midwest US:
Dave Jones
Email: djones@computer.org
Phone: +1 708-442-5633 | Fax: +1 888-886-8599
Cell: +1 708-624-9901

Jobs Board (West Coast and Asia), Classified Line Ads

Heather Buonadies
Email: hbuonadies@computer.org
Phone: +1 623-233-6575

Jobs Board (East Coast and Europe), SE Radio Podcast

Marie Thompson
Email: marie.thompson@computer.org
Phone: +1 714-813-5094

IEEE COMPUTER SOCIETY

80 ANNIVERSARY CELEBRATION

COMPUTER.ORG

CALL FOR SPECIAL ISSUE PROPOSALS

Computer solicits special issue proposals from leaders and experts within a broad range of computing communities. Proposed themes/issues should address important and timely topics that will be of broad interest to *Computer*'s readership. Special issues are an essential feature of *Computer*, as they deliver compelling research insights and perspectives on new and established technologies and computing strategies.

Please send us your high-quality proposals for the 2026 - 2027 editorial calendar. Of particular interest are proposals centered on:

- 3D printing
- Robotics
- LLMs
- AI safety
- Dis/Misinformation
- Legacy software
- Microelectronics

Proposal Guidelines Are Available at:

computer.org/csdl/magazine/co/write-for-us/15911

IEEE

Conference Calendar

IEEE Computer Society conferences are valuable forums for learning on broad and dynamically shifting topics from within the computing profession. With over 200 conferences featuring leading experts and thought leaders, we have an event that is right for you. Questions? Contact conferences@computer.org.

APRIL

3 April

- CI2A (Int'l Conf. on Connected Intelligence for Industrial Applications), Punjab, India

12 April

- AST (IEEE/ACM Int'l Conf. on Automation of Software Test), Rio de Janeiro, Brazil
- CAIN (IEEE/ACM Int'l Conf. on AI Eng. – Software Eng. for AI), Rio de Janeiro, Brazil
- FormaliSE (IEEE/ACM Int'l Conf. on Formal Methods in Software Eng.), Rio de Janeiro, Brazil
- ICSE (IEEE/ACM Int'l Conf. on Software Eng.), Rio de Janeiro, Brazil
- MOBILESoft (IEEE/ACM Int'l Conf. on Mobile Software Eng. and Systems), Rio de Janeiro, Brazil
- MSR (IEEE/ACM Int'l Conf. on Mining Software Repositories), Rio de Janeiro, Brazil

15 April

- COOL CHIPS (IEEE Symposium on Low-Power and High-Speed Chips and Systems), Tokyo, Japan

20 April

- DATE (Design, Automation & Test in Europe Conf.), Verona, Italy
- PacificVis (IEEE Pacific Visualization Conf.), Sydney, Australia

26 April

- ISPASS (IEEE Int'l Symposium on Performance Analysis of Systems and Software), Seoul, Korea

27 April

- VTS (IEEE VLSI Test Symposium), Napa, USA

MAY

4 May

- HOST (IEEE Int'l Symposium on Hardware Oriented Security and Trust), Washington, DC, USA
- MOST (IEEE Int'l Conf. on Mobility, Operations, Services and Technologies), Detroit, USA

8 May

- BigDataSecurity (IEEE Conf. on Big Data Security on Cloud), New York City, USA
- CAI (IEEE Int'l Conf. on Artificial Intelligence), Granada, Spain
- HPSC (IEEE Int'l Conf. on High Performance and Smart Computing), New York City, USA
- IDS (IEEE Int'l Conf. on Intelligent Data and Security), New York City, USA
- SmartCloud (IEEE Int'l Conf. on Smart Cloud), New York City, USA

11 May

- SenSys (ACM/IEEE Int'l Conf. on Embedded Artificial Intelligence and Sensing Systems), Saint Malo, France

12 May

- RTAS (IEEE Real-Time and Embedded Technology and Applications Symposium), Saint Malo, France

13 May

- FCCM (IEEE Annual Int'l Symposium on Field-Programmable Custom Computing Machines), Atlanta, USA

15 May

- ICES (Int'l Conf. on Energy Storage), Shenyang, China

18 May

- CCGrid (IEEE Int'l Symposium on Cluster, Cloud and Internet Computing), Sydney, Australia
- ICFEC (IEEE Int'l Conf. on Fog and Edge Computing), Sydney, Australia
- ICST (IEEE Int'l Conf. on Software Testing, Verification and Validation), Daejeon, Korea
- S&P (IEEE Symposium on Security and Privacy), San Francisco, USA

19 May

- ICDE (IEEE Int'l Conf. on Data Eng.), Hong Kong, China

2469-7087/26 © 2026 IEEE

- ISMVL (IEEE Int'l Symposium on Multiple-Valued Logic), Sendai, Japan

25 May
- FG (IEEE Int'l Conf. on Automatic Face and Gesture Recognition), Kyoto, Japan
- IPDPS (IEEE Int'l Parallel and Distributed Processing Symposium), New Orleans, USA

JUNE

1 June
- ICHI (IEEE Int'l Conf. on Healthcare Informatics), Minneapolis, USA

3 June
- CBMS (IEEE Int'l Symposium on Computer-Based Medical Systems), Limassol, Cyprus
- CVPR (IEEE/CVF Conf. on Computer Vision and Pattern Recognition), Denver, USA

10 June
- SVCC (Silicon Valley Cybersecurity Conf.), San Jose, USA

16 June
- WoWMoM (IEEE Int'l Symposium on a World of Wireless, Mobile and Multimedia Networks), Bologna, Italy

22 June
- DCOSS-IoT (Int'l Conf. on Distributed Computing in Smart Systems and the Internet of Things), Reykjavik, Iceland
- DSN (Annual IEEE/IFIP Int'l Conf. on Dependable Systems and Networks), Charlotte, USA
- ICDCS (IEEE Int'l Conf. on Distributed Computing Systems), Seoul, Korea
- ICSA (IEEE Int'l Conf. on Software Architecture), Amsterdam, Netherlands
- SmartComp (IEEE Int'l Conf. on Smart Computing), Messina, Italy

26 June
- IEEE Cloud Summit, Washington, DC, USA

27 June
- ISCA (ACM/IEEE Annual Int'l Symposium on Computer Architecture), Raleigh, USA

28 June
- ARITH (IEEE Symposium on Computer Arithmetic), Fulda, Germany

JULY

1 July
- IOLTS (IEEE Int'l Symposium on On-Line Testing and Robust System Design), Polignano a Mare, Italy

6 July
- EuroS&P (IEEE European Symposium on Security and Privacy), Lisbon, Portugal
- ICALT (IEEE Int'l Conf. on Advanced Learning Technologies), Hung Yen, Vietnam
- ICME (IEEE Int'l Conf. on Multimedia and Expo), Bangkok, Thailand

7 July
- COMPSAC (IEEE Annual Computers, Software, and Applications Conf.), Madrid, Spain
- ISVLSI (IEEE Computer Society Annual Symposium on VLSI), Kolkata, India

13 July
- ICCP (IEEE Int'l Conf. on Computational Photography), Princeton, USA
- SERVICES (IEEE World Congress on Services), Sydney, Australia

26 July
- CSF (IEEE Computer Security Foundations Symposium), Lisboa, Portugal

27 July
- CISOSE (IEEE Int'l Congress on Intelligent and Service-Oriented Systems Eng.), Fukuoka, Japan

IEEE COMPUTER SOCIETY CAREER CENTER

Career Accelerating Opportunities

Explore new options—upload your resume today

careers.computer.org

Changes in the marketplace shift demands for vital skills and talent. The IEEE Computer Society Career Center is a valuable resource tool to keep job seekers up to date on the dynamic career opportunities offered by employers.

Take advantage of these special resources for job seekers:

- JOB ALERTS
- TEMPLATES
- WEBINARS
- CAREER ADVICE
- RESUMES VIEWED BY TOP EMPLOYERS

No matter what your career level, the IEEE Computer Society Career Center keeps you connected to workplace trends and exciting career prospects.

IEEE COMPUTER SOCIETY

80th ANNIVERSARY CELEBRATION

IEEE