AI for Vulnerability Detection and Repair: Opportunities and Challenges

Merve Sahin (SAP), Kendrick Gruenberg (SAP, TUBS), Malte Wessels (TUBS)

THCon
05.05.2026

Technische Universität Braunschweig

SAP

cloud security alliance® SANS [un]prompted OWASP GenAI SECURITY PROJECT

The "AI Vulnerability Storm": Building a "Mythos-ready" Security Program

MAD Bugs: vim vs emacs vs Claude

We asked Claude to find a bug in Vim. It found an RCE. Just open a file, and you're owned. We joked: fine, we'll switch to Emacs. Then Claude found an RCE there too.

CALIF

MAR 30, 2026

Listen

The Hacker News

Google's AI Tool Big Sleep Finds Zero-Day Vulnerability in SQLite Database Engine

Ravie Lakshmanan Nov 04, 2024

tom's HARDWARE

AI-assisted cybersecurity team discovers 12 OpenSSL vulnerabilities, claims humans are the limiting factor — some vulnerabilities have been around for decades

News By Aaron Klotz published January 29, 2026

The Hacker News

Anthropic's Claude Mythos Finds Thousands of Zero-Day Flaws Across Major Systems

Ravie Lakshmanan Apr 08, 2026

The “AI Vulnerability Storm”: Building a “Mythos-ready” Security Program

II. Key Takeaways for the CISO

Use LLM-based vulnerability discovery and remediation capabilities.

Unlike defensive AI technologies, LLM-based vulnerability discovery capabilities are already mature and can be used to your advantage. Start immediately by asking an agent for a security review of any code, and build toward a VulnOps capability.

cloud security alliance® SANS [un]prompted OWASP GenAI SECURITY PROJECT

The “AI Vulnerability Storm”: Building a “Mythos-ready” Security Program

II. Key Takeaways for the CISO

Use LLM-based vulnerability discovery and remediation capabilities.

Unlike defensive AI technologies, LLM-based vulnerability discovery capabilities are already mature and can be used to your advantage. Start immediately by asking an agent for a security review of any code, and build toward a VulnOps capability.

AI for *Static* Vulnerability Detection: Opportunities

LLMs can reason about program behavior & find vulnerabilities that traditional **SAST*** tools cannot

- Logic flaws (e.g., access control issues)
- Interaction flaws between software components
- Input sanitization/validation weaknesses

LLMs have extensive knowledge of frameworks & technologies

- Identify risky configurations/patterns even when there is no immediate vulnerability

---

SAST: Static Application Security Testing

Static code analysis to search for vulnerability patterns

Bandit Semgrep sonarqube bearer Checkmarx GitHub CodeQL

cloud security alliance® SANS [un]prompted OWASP GenAI SECURITY PROJECT

The “AI Vulnerability Storm”: Building a “Mythos-ready” Security Program

II. Key Takeaways for the CISO

Use LLM-based vulnerability discovery and remediation capabilities.

Unlike defensive AI technologies, LLM-based vulnerability discovery capabilities are already mature and can be used to your advantage. Start immediately by asking an agent for a security review of any code, and build toward a VulnOps capability.

AI for *Static* Vulnerability Detection: Opportunities

LLMs can reason about program behavior & find vulnerabilities that traditional **SAST*** tools cannot

- Logic flaws (e
-

What can go wrong?

ks &

- Id
n when there is no immediate vulnerabil

SAST: Static Application Security Testing

Static code analysis to search for vulnerability patterns

Bandit Semgrep sonarqube bearer Checkmarx GitHub CodeQL

AI for *Static* Vulnerability Detection: Challenges

- **False positives**
  - Difficult to spot due to **plausible explanations** → High security expertise to verify

Example study by Semgrep :

- 11 real-world, actively maintained Python web applications
- Repository-level detection (with help of AI coding agents)
- Manual analysis of all findings by security experts

AI coding agents are tasked to find 6 different types of vulnerabilities:

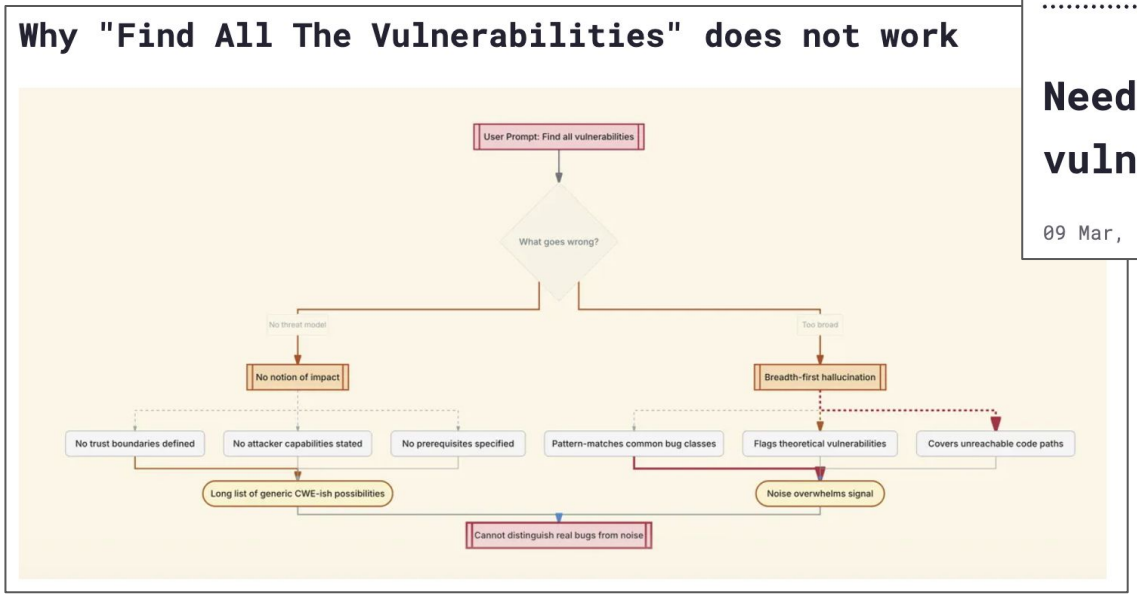
→ Results:

- 445 findings in total, **85% FPs**
- Claude-Sonnet-4 + Claude Code: 46 vulnerabilities (with 86% FPR)
- GPT-04-mini + Codex: 21 vulnerabilities (with 82% FPR)

Gaucher et al., “Finding vulnerabilities in modern web apps using Claude Code and OpenAI Codex”, Semgrep, 2025.

AI for *Static* Vulnerability Detection: Challenges

- **False positives**



https://devansh.bearblog.dev/needle-in-the-haystack/

AI for *Static* Vulnerability Detection: Challenges

- **Non-determinism**
  - Inconsistent findings and severity ratings, incorrect reasoning for correct vuln. type
  - Difficult to track & document vulnerabilities (e.g., for compliance & audit requirements)

codechat-bison (*1st Response*)

Yes, the code contains instances of the security vulnerability known as out-of-bounds write. The code allocates a buffer of size (4 * sizeof(char) * MAX_SIZE) + 1 bytes. However, **the code does not check to ensure that the length of the input string does not exceed the size of the buffer.** This could lead to an out-of-bounds write if the input string is longer than the buffer.

codechat-bison (*5th Response*)

No, the code does not contain instances of the security vulnerability known as out-of-bounds write. The code is safe because **the `dst_index` variable is used to keep track of the current position in the destination buffer. This variable is incremented each time a character is copied from the source buffer to the destination buffer.** This ensures that no characters are written outside of the bounds of the destination buffer.

TABLE 12: Faithfulness of LLMs. The Table shows the Reason Rate i.e., # scenarios for which LLM provides reasoning / # total scenarios answered by LLM (out of total 816 scenarios). Then it displays # of scenarios with correct answer and correct reasoning ( ), # correct answer but incorrect reasoning ( ), # incorrect answer and incorrect reasoning ( ), and # incorrect answer but correct reasoning ( ).

Model	Reas. Rate	Details
chat-bison	724 /782	406 32 270 16 /724
codechat-bison	680 /719	368 25 266 21 /680
codellama34b	814 /814	391 34 358 31 /814
gpt-3.5	788 /800	464 40 259 25 /788
gpt-4	810 /810	612 14 168 16 /810

S. Ullah et al., "LLMs Cannot Reliably Identify and Reason About Security Vulnerabilities (Yet?): A Comprehensive Evaluation, Framework, and Benchmarks," in IEEE Symposium on Security and Privacy (SP), 2024.

AI for *Static* Vulnerability Detection: Challenges

- **User-induced bias**
  - Instruction-tuned models, tend to follow user assumptions
  - Incorrect human input can suppress valid findings
    (“This looks safe, right?”)

AI for *Static* Vulnerability Detection: Challenges

- **User-induced bias**
  - Instruction-tuned models, tend to follow user assumptions
  - Incorrect human input can suppress valid findings
    (“This looks safe, right?”)

- **Lack of systemic coverage of code base / vulnerability types:**
  - AI agent may focus on representative examples per vulnerability type (not exhaustive)
  - No guarantee of consistent coverage across the whole code base (context window limit)

    → Current strategy: Vulnerability specific prompting / analysis per each file

Takeaways

- Leveraging AI for vulnerability detection is essential, becoming a standard practice.

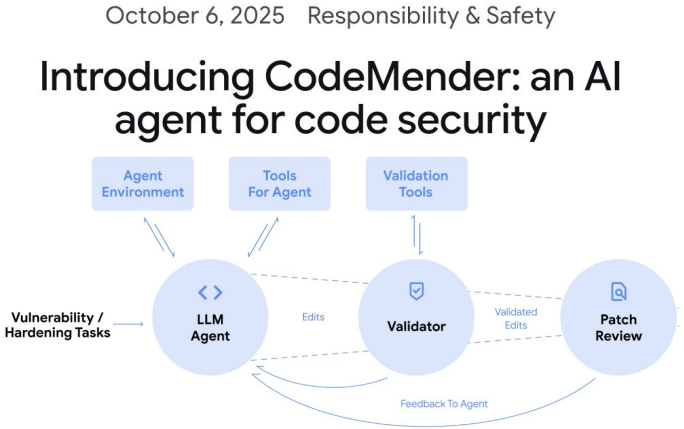
- Turning an AI coding agent into robust security tooling is difficult
  (FPs, non-determinism, user-induced bias…)

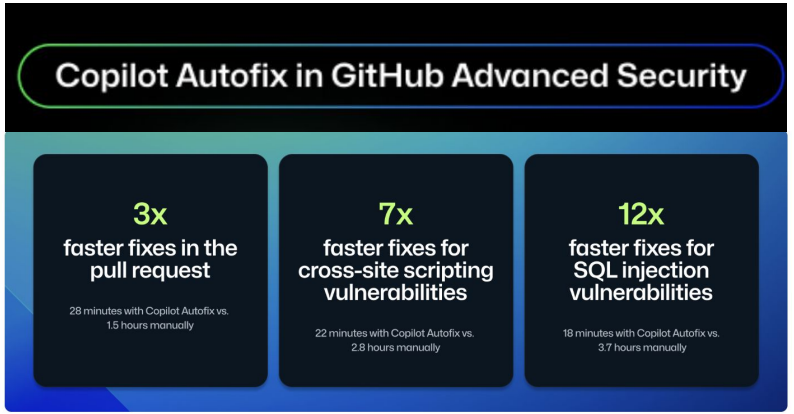
- What works best for static vulnerability detection is not yet clear:
  AI-only vs. *SAST-augmented AI* vs. *AI-augmented SAST*

Vulnerability Remediation

- Vulnerability repair is historically the least automated step → requires a lot of time and security expertise

- Many security/AI vendors advertise AI generated vulnerability “auto-fix” features:
  E.g.: GitHub Copilot Autofix, Semgrep Assistant, Sonar AI CodeFix, Claude Security, Codex Security, Google CodeMender…





Example: GitHub Copilot Autofix Feature
https://github.blog/news-insights/product-news/secure-code-more-than-three-times-faster-with-copilot-autofix/

Vulnerability Remediation - Literature overview

- Function level fixes
- Focus on C/C++ & Java
- Results are mixed:
  10% to 90% correct vuln. fix rates
  → depending on the techniques, models, datasets…

	LLMs	Data		Deployment	
	Size	Input granularity	Real-world & tests	Interact with users	Integrated to workflow
Chen et al. [17]	<1 B	Function	✗	✗	✗
Chi et al. [18]	<1 B	Function	✗	✗	✗
Fu et al. [28]	<1 B	Function	✗	✗	✗
Zhou et al. [132]	<1 B	Function	✗	✗	✗
Wu et al. [109]	<1 B, 1-10 B, >10 B	Function	✓	✗	✗
Fu et al. [29]	>10 B	Function	✗	✗	✗
Zhang et al. [123]	<1 B	Function	✗	✗	✗
Pearce et al. [74]	<1 B, 1-10 B, >10 B	Function	✓	✗	✗

Zhou et al., “Large Language Model for Vulnerability Detection and Repair: Literature Review and the Road Ahead”, ACM TOSEM, 2025.

Our Study: Overview

In collaboration with Kendrick Gruenberg (SAP, TUBS),
Malte Wessels (TUBS), Vladimir Klebanov (SAP), Martin Johns (TUBS)

- First study to focus on Web Languages (PHP, JavaScript)
- File-level vulnerability repair
- Real-world dataset for evaluation
- Evaluating 17 LLMs (Gemini, Claude, GPT, Llama, Qwen - August, 2025) with the help of
  - 7 state-of-the-art SAST tools and
  - Functional/security unit tests (when necessary)

Dataset Collection

Vulnerable & Fixed code pairs from 11,873 vulnerability reports

CVEfixes

CVE: Common Vulnerabilities and Exposures (CVE), refers to vulnerability reports in the public U.S. National Vulnerability Database (NVD)

Filtering

-PHP or JS
-Fix modifies 1 file
-not older than 2010
-No binary file diffs
…
- Random selection for PHP

432 JS and PHP code pairs

SAST scans & human verification

75 code pairs, where at least 1 SAST tool can detect the exact vulnerability (ground truth)

Bhandari et al., “CVEfixes: automated collection of vulnerabilities and their fixes from open-source software”. ACM PROMISE 2021.

Manual Evaluation of the Human Vulnerability Fixes

- 11% of the vulnerability fixes were incomplete
(e.g., not validating/sanitizing all sensitive inputs)

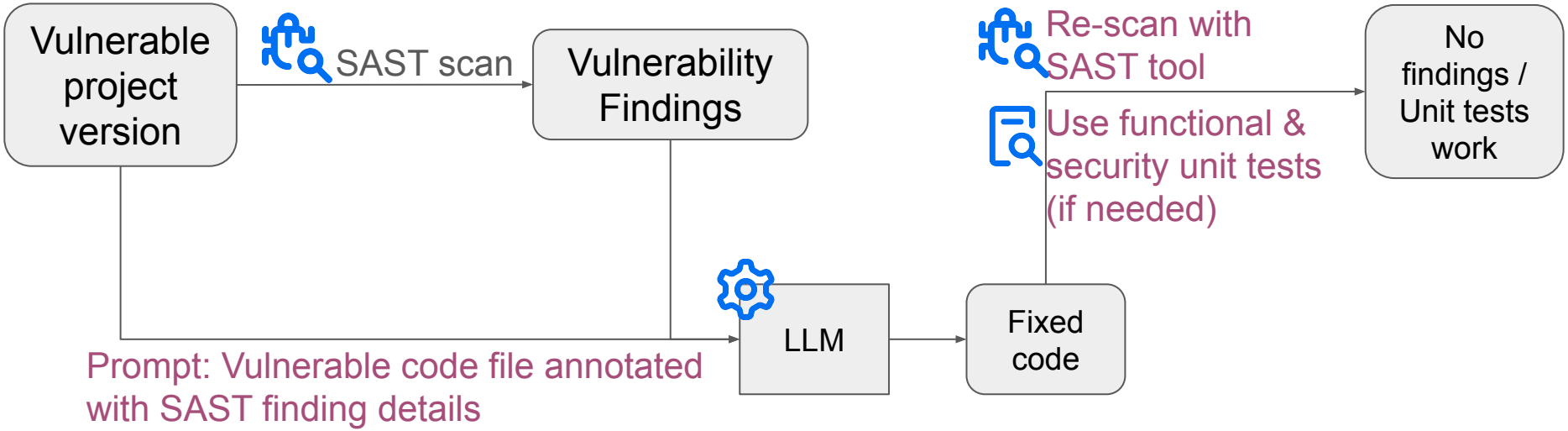
- 33% of the vulnerability fixes had design flaws
(e.g., not using well-established security practices or libraries)

- One case where the fix introduced a different type of vulnerability

Vulnerability	Good fix design	Bad fix design
SQL injection (CWE-89)	(+) using a prepared statement → e.g., CVE-2021-4290	(–) not using a prepared statement, even though this would easily be possible → e.g., CVE-2015-10055
XSS (CWE-79,80)	(+) using `dompurify.sanitize()` → e.g., CVE-2025-27608 (+) using `_.escape()` from the *Lodash* library → e.g., CVE-2022-4942 (+) using `Format.htmlEncode()` from the *Ext JS* framework → e.g., CVE-2023-2322	(–) using self-written escape functions → e.g., CVE-2022-23466 (–) using `encodeURIComponent()` for escaping HTML/DOM content → e.g., CVE-2022-4735

Method to Generate & Evaluate the LLM Fixes

- Each vulnerable code file corresponds to a SAST finding
- Does the SAST finding disappear with the human fix?
  - Yes: Re-run the SAST scan to evaluate the LLM fix
  - No: Implement functional&security unit tests



Prompt Templates to Generate Vuln. Fixes

Zero-shot prompt template

[SAST tool name] found a [programming language] vulnerability of type [CWE] in the `.[suffix]` file that follows (all lines of code related to the SAST finding have been highlighted using `// <=====` comments), please return a version of this file with this vulnerability fixed, please return the fixed version of the file in its **entirety** (i.e., please do **not** shorten your response).
Do not perform any changes to the code that are unrelated to fixing the vulnerability, do not needlessly reorder functions, do not change the formatting [...]

````
[vulnerable code with "<=====" SAST annotations]
````

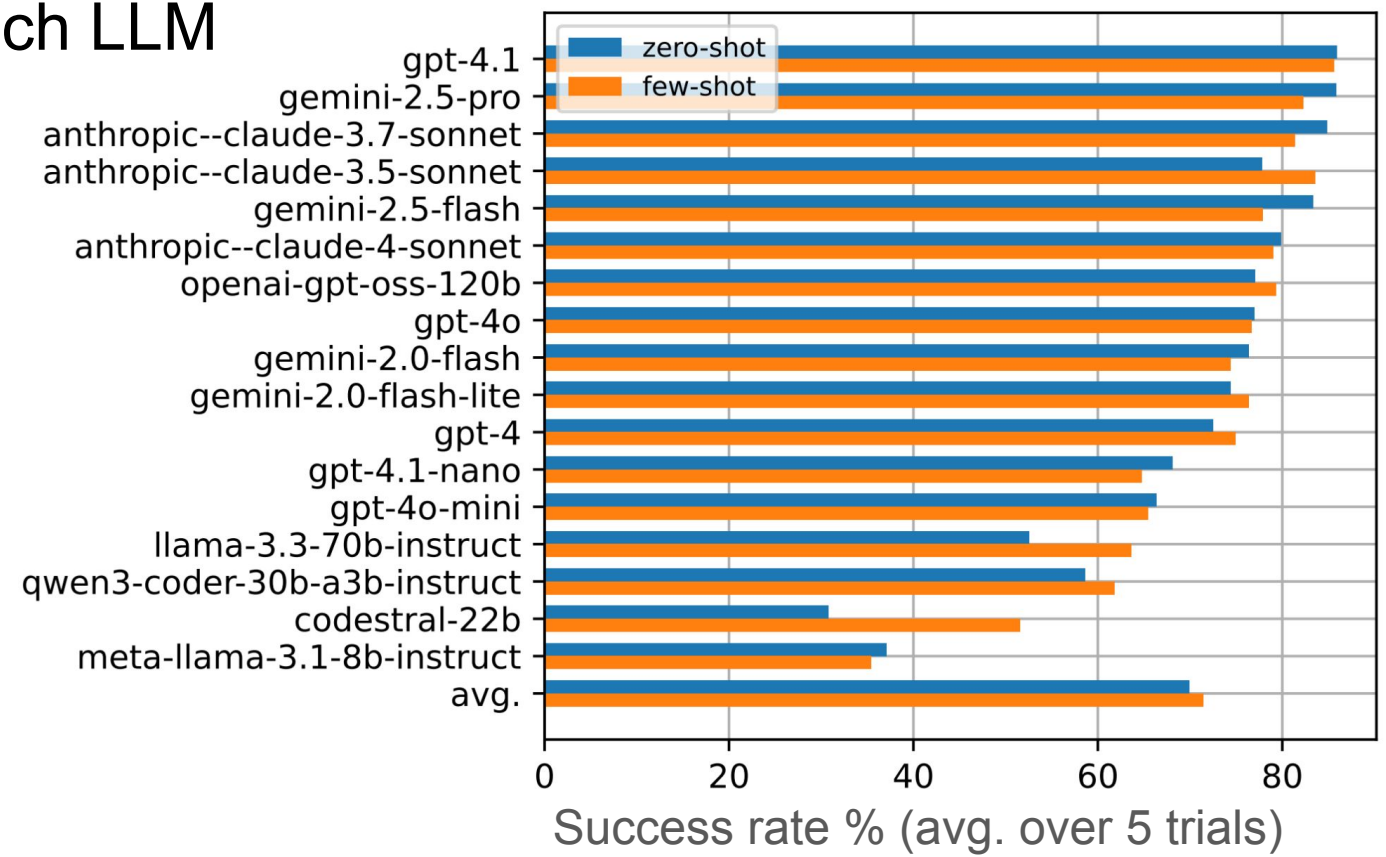
13 Few-shot prompt templates

(specific to each language-vulnerability type pair)

Command Injection vulnerabilities (CWE-77) [...] refer to [...] In JavaScript code, they should be fixed preferably by [...]
Here are two examples:
Example 1: Vulnerable: `[...]`
Example 1: Fixed: `[...]`
[...]
Now please fix the [...] vulnerability (as found by [...])
[...]
Vulnerable:

````
[vulnerable code with "<=====" SAST annotations]
````

Results with optimal (temperature, top-p) parameters for each LLM



Analysis of LLM fixes

Failed cases:

- LLM generates too much text and attempts to make too many changes

  → Fix causes errors or breaks functionality

- The fix only refactors code

Successful cases:

- More complete and better-designed than human fixes

- More structured (e.g., using helper functions) rather than attempting one-line fixes

  → On average LLM fix modifies 5 times more lines of code than human fixes

LLM vs. Human (SQL Injection - High)

GPT 4.1 Zero Shot

```
-   $result = mysqli_query($con,"SELECT * FROM users where secret = '$key' and
              authused=1");
+   // Use prepared statements to prevent SQL injection
+   $stmt = mysqli_prepare($con, "SELECT * FROM users WHERE secret = ? AND
            authused = 1");
+   if ($stmt) {
+       mysqli_stmt_bind_param($stmt, "s", $key);
+       mysqli_stmt_execute($stmt);
+       $result = mysqli_stmt_get_result($stmt);
```

Human

```
    $con = mysqli_connect("$db_host","$db_user","$db_pass","$db_name");

-   $key = $_GET['key'];
+   $key = mysqli_real_escape_string($con, $_GET['key']);
    $result = mysqli_query($con,"SELECT * FROM users where secret = '$key' and
              authused=1");
```

Do LLMs simply memorize the human fixes in training data?

Knowledge cut-off dates of LLMs: 2021 to 2025
CVEs : 2012 to 2024

- **Exact string match:**
  Among 10,425 LLM Fixes across all experiments, only **1%** exact string match with human fix (trivial fixes)

- **AST replication** (ignoring identifier names)
  Among 7,653 LLM fixes for JS, **5.3%** replicated the AST of the human fix
  For 76% of CVEs: LLM fixes never replicated the human fix

→ *LLMs seem to generalize vuln. repair patterns, rather than reproducing the training-data*

- **LLM Fix success rate on unseen CVEs** (Up to 5 CVEs per LLM):
  74% on average (compared to 82% success rate for CVEs before knowledge cutoff)

Conclusion

AI agents can largely automate vulnerability detection and repair

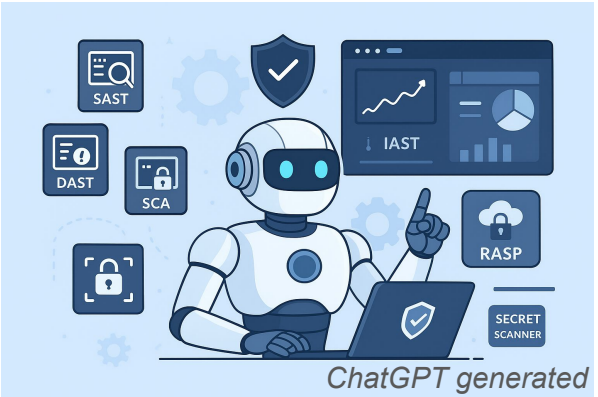
→ Engineering reliable and scalable security solutions is still a challenge

→ Methods should evolve with rapidly advancing models

Software security landscape is evolving:
Traditional vendors integrating AI -vs- AI-native startups -vs- AI companies entering security

The race is on:

- Building the most security-capable model
- Designing the most effective agentic testing framework
- Combining AI with traditional security tooling in the best way
- Integrating seamlessly into the developer workflow



ChatGPT generated

Thank you.

Contact information:

Merve Sahin
merve.sahin@sap.com

Technische Universität Braunschweig