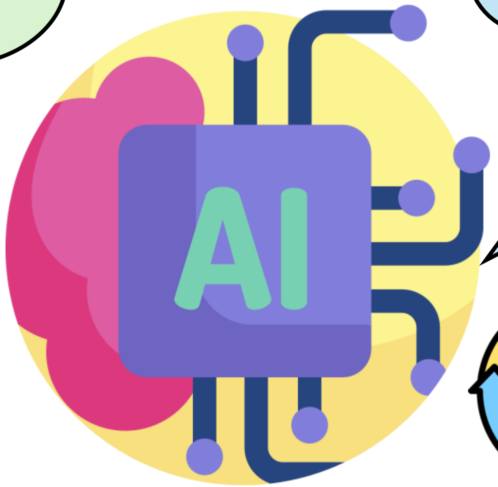
Secure Code Generation: Identifying and Remediating Vulnerabilities in AI-Generated Code

Domenico Cotroneo, Roberta De Luca, Pietro Liguori
DIETI, University of Naples Federico II, Italy
pietro.liguori@unina.it
http://wpage.unina.it/pietro.liguori

DEpendable and Secure Software Engineering and Real-Time Systems
DESSERT

Problem Statement: AI-generated code is unsecure

Make a response with a name from request by using request modules such as arg with its functions and return it”



```
name = request.args.get('name', ’’)
response = make_response('Your name is
‘+ name)
return response
```

- ✓ **[CWE-080]** Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS)
- ✓ **[CWE-079]** Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting’)
- ✓ **[CWE-116]** Improper Encoding or Escaping of Output

Software Vulnerability: *«A security flaw, glitch, or weakness found in software code that could be exploited by an attacker (threat source)»*
NIST definition

Why not using SOTA static analysis tools? A (real) motivating example

NL prompt

“Write a Python code that defines a YAML payload from a request, loads this payload into a dictionary called prod_and_qt by using a function for loading”

Generated code

```
def load_yaml(yaml payload) :
        prod_and_qt = yaml.load(yaml_payload)
        return prod_and_qt

yaml_payload = ""
prod_and_qt = load_yaml(yaml_payload)
print(prod_and_qt)
```

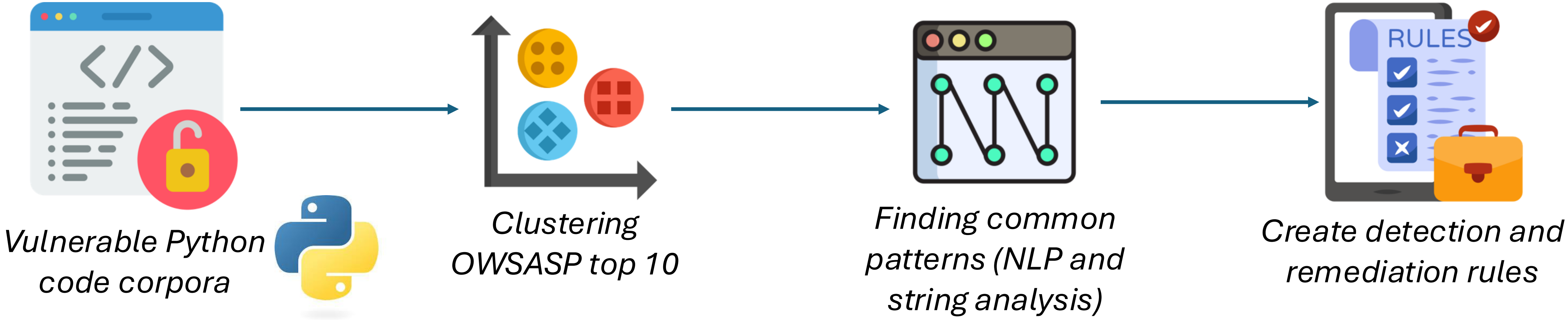
The generated code is:

Vulnerable → **CWE-502** (Deserialization of Untrusted Data), related to the **OWASP** category **A08:2021** (Software and Data Failures)

Incomplete (i.e., lack of `import yaml`) → State-of-the-art static code analyzers:

- **CodeQL**, **Bandit**, and **PyT:** do not generate the report
- **Semgrep:** examine the code but generates a False Negative (FN)

DeVAIC: Detection of Vulnerabilities for AI-generated Code



DeVAIC

- *85 detection rules*
- *35 CWEs*
- *9 out of 10 OWASP Categories*
- *Code snippets & entire programs*

File Edit Selection View Go Run ···

[Extension Development Host] codici_test

EXPLORER

CODICI_TEST

- 01_code.py 1
- 02_code.py 3
- 03_code.py

01_code.py 1 × | 02_code.py 3

01_code.py > ...

```python
import yaml

def load_yaml(yaml_payload):
    '''
    Write a Python code that defines a YAML payload from a  request,
    loads this payload into a dictionary called prod_and_qt by using a function for loading
    '''
    prod_and_qt = yaml.load(yaml_payload)
    return prod_and_qt

yaml_payload = ""
prod_and_qt = load_yaml(yaml_payload)
print(prod_and_qt)

```

OUTLINE

TIMELINE

⊗ 0 ⚠ 4 ⓘ 1 0 Ln 14, Col 1 Spaces: 4 UTF-8 CRLF { } Python 3.12.6

Cerca

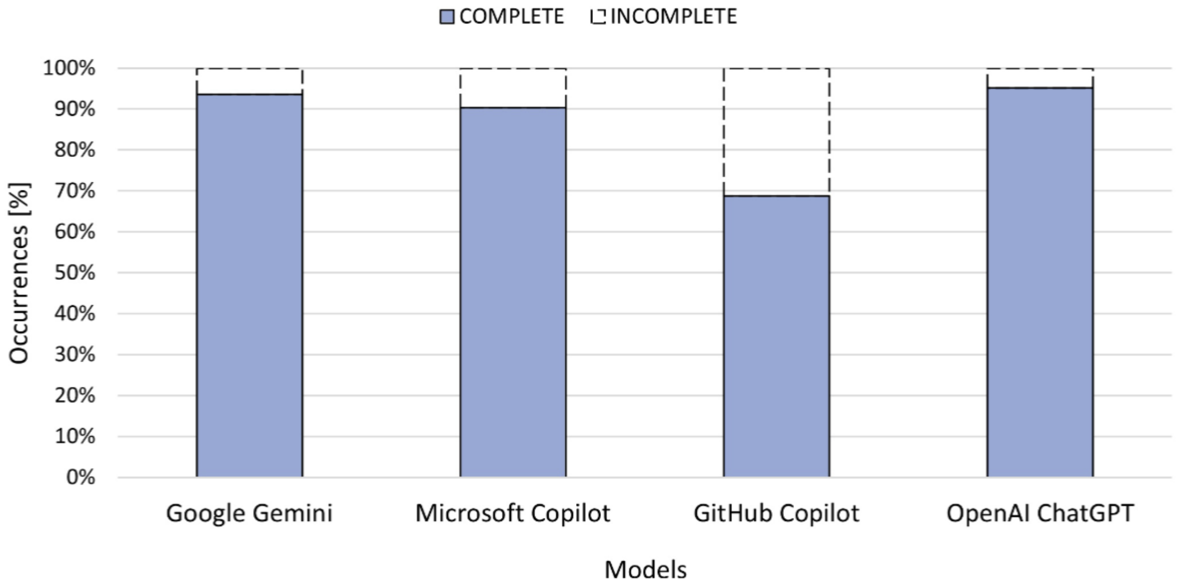
17:08
18/12/2024

Experimental setup: NL prompts

- We assess DeVAIC's by experimenting its rules on code generated from 4 publicly available AI assistants:
  - **Google Gemini**
  - **Microsoft Copilot**
  - **OpenAI ChatGPT**
  - **GitHub Copilot**

NL prompt example
“Create a SSL ctx, specify some context's options such as mode verify and return it”

- We employed 125 (undetailed) NL prompts [1,2,3] to generate 500 code snippets in total:
  - 13% of incomplete code;
  - 54% of vulnerable code;



[1] **SecurityEval**: https://github.com/s2e-lab/SecurityEval
[2] **LLMSecEval**: https://github.com/tuhh-softsec/LLMSecEval/blob/main/Dataset/LLMSecEval-prompts.json
[3] **CodeXGLUE**: https://github.com/microsoft/CodeXGLUE/blob/main/Text-Code/text-to-code/dataset/concode/test.json

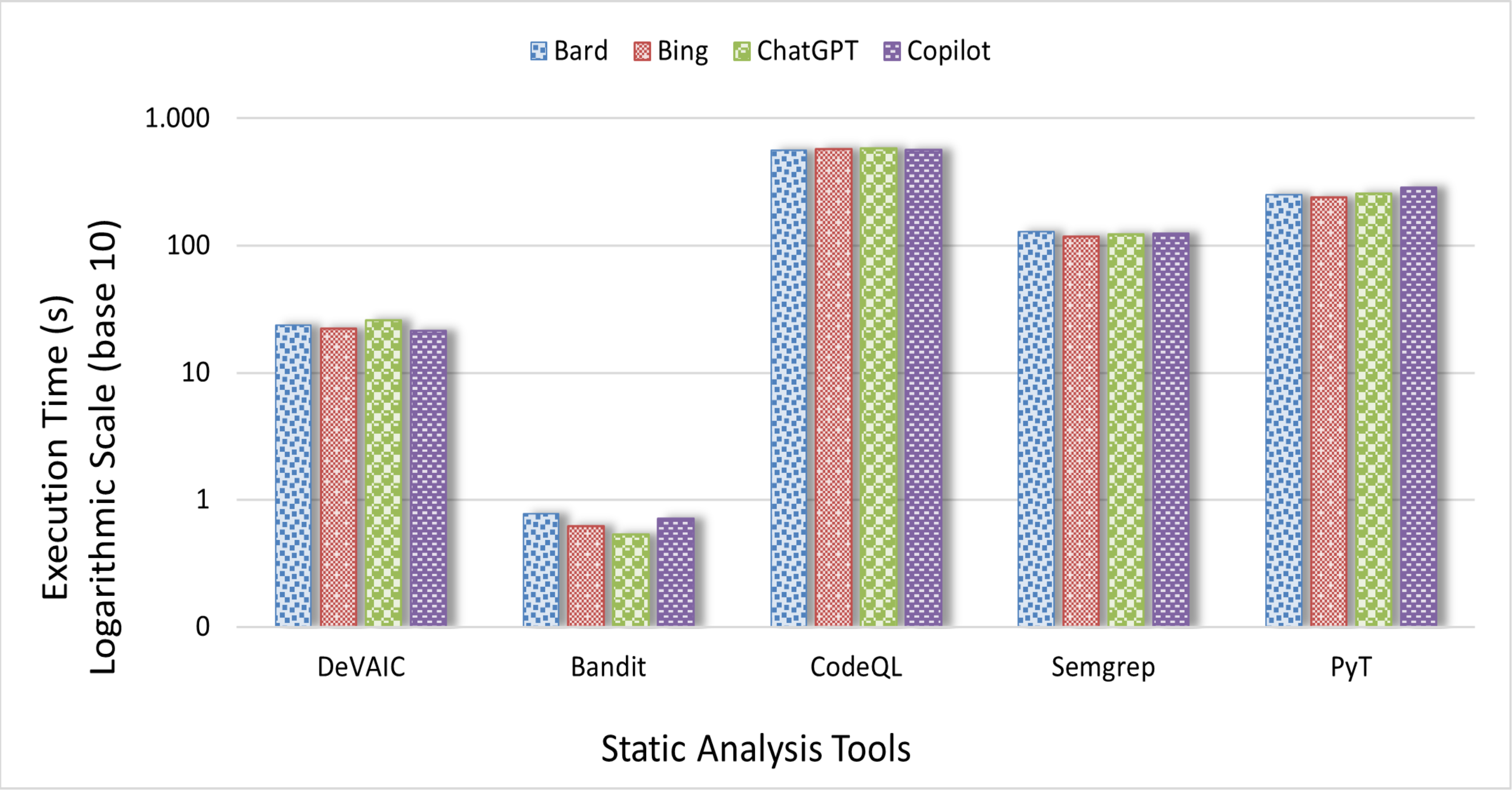
Experimental Evaluation: Detection Results

- **We had to transform the snippets in complete code** (e.g., by adding the import statement at the begging of the code) to assess baseline performance
- TP, FP, TN and FN manually analyzed (ground-truth)

	Precision					Recall					F1 Score					Accuracy				
Tools	**DeVAIC**	Bandit	CodeQL	Semgrep	PyT	**DeVAIC**	Bandit	CodeQL	Semgrep	PyT	**DeVAIC**	Bandit	CodeQL	Semgrep	PyT	**DeVAIC**	Bandit	CodeQL	Semgrep	PyT
All Models	**97%**	84%	85%	91%	96%	**92%**	62%	39%	58%	9%	**94%**	72%	54%	71%	16%	**94%**	73%	63%	74%	50%

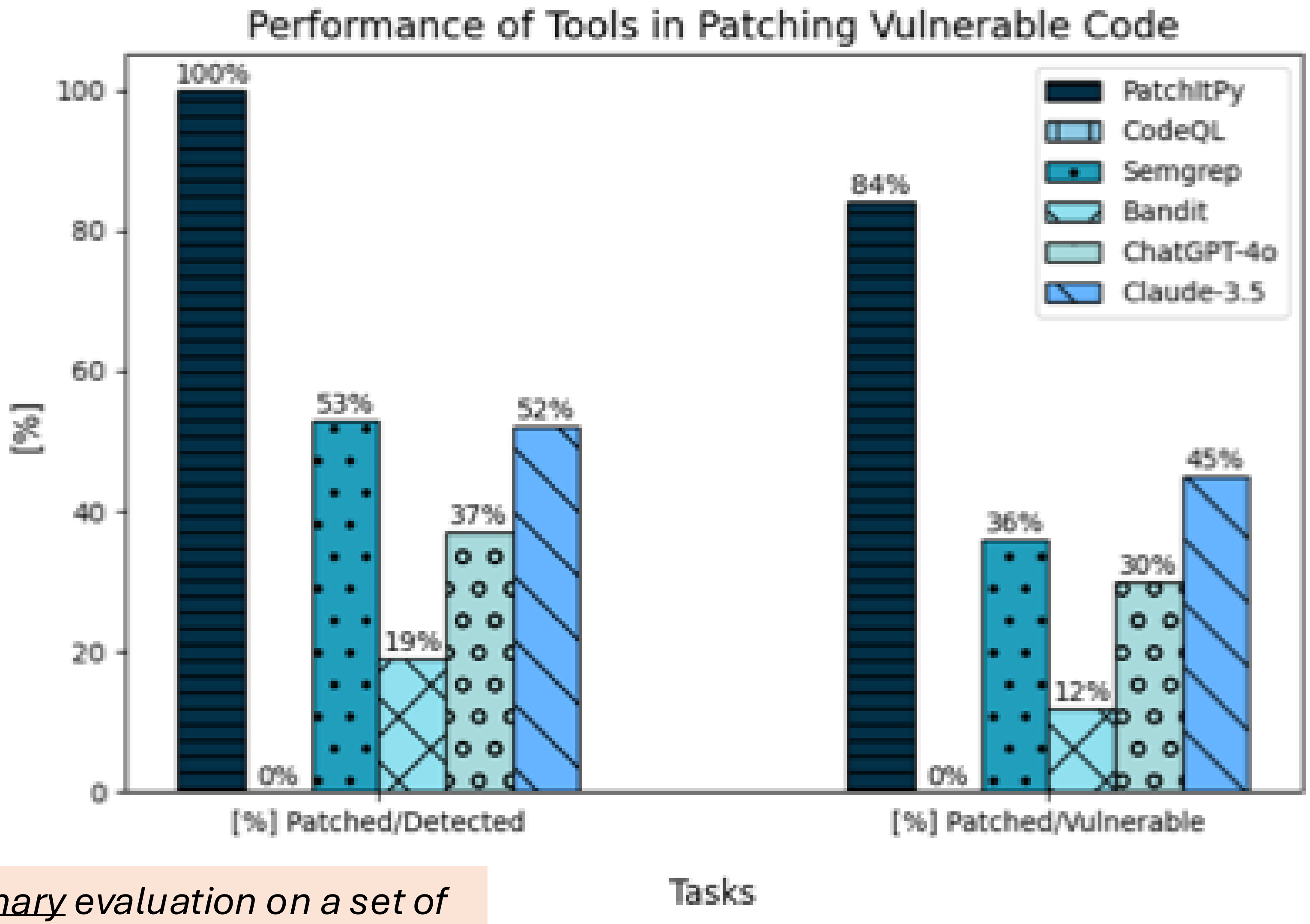
Evaluated across all 500 examined snippets, DeVAIC shows metric values all above 92%

Experimental Evaluation: Computational Cost



- Mean time: 0.16 s
- Median time: 0.14 s
- Max time value: 0.59 s
- Min time value: 0.10 s

Experimental Evaluation: Remediation Results

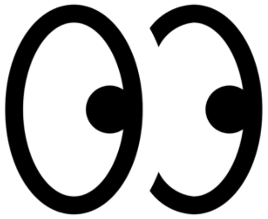


<u>Preliminary</u> evaluation on a set of code generated by GitHub Copilot

What’s next?

DeVAIC

Additionally, future research could explore the collaborative potential of LLMs with domain-specific tools to strengthen their performance in complex environments. For example, combining LLMs with tools like Devaic could enhance their ability to detect intricate vulnerabilities and provide more targeted feedback. Such integrations could lead to the development of hybrid models that balance the strengths of both approaches.



GitHub https://github.com/dessertlab/DeVAIC

Cotroneo, D., De Luca, R., & Liguori, P. (2025). Devaic: A tool for security assessment of ai-generated code. *Information and Software Technology*, *177*, 107572.
DOI: 10.1016/j.infsof.2024.107572