Quando a IA Programa: Quem Cuida da Segurança?

IA Generativa, Ciberataques e os Novos Riscos para o Desenvolvimento de Software

Professora Dra. Edna Dias Canedo

Universidade Federal de Pernambuco (UFPE)
Centro de Informática (CIn)

E-mail: ednacanedo@cin.ufpe.br — edna.canedo@gmail.com



Antes de começarmos...

Quem aqui já utilizou Inteligência Artificial para ajudar a programar?

Por exemplo:

- gerar uma função ou trecho de código;
- corrigir um erro;
- explicar um código existente;
- criar testes;
- sugerir uma biblioteca ou dependência;
- revisar ou melhorar uma implementação.

Levante a mão se você já fez pelo menos uma dessas coisas.

Agora uma pergunta um pouco diferente...

Você pediu à IA para gerar um código.

O código:

compila, executa e produz o resultado esperado.

Você confia nele?

Código funcional não significa código seguro

Um trecho de código pode:

- compilar corretamente;
- executar sem apresentar erros;
- produzir a saída esperada;
- passar pelos testes funcionais;

e, ainda assim, conter uma **vulnerabilidade**.

Vulnerabilidade de software

Uma fraqueza no software que pode permitir que determinadas condições ou ações comprometam propriedades de segurança do sistema.

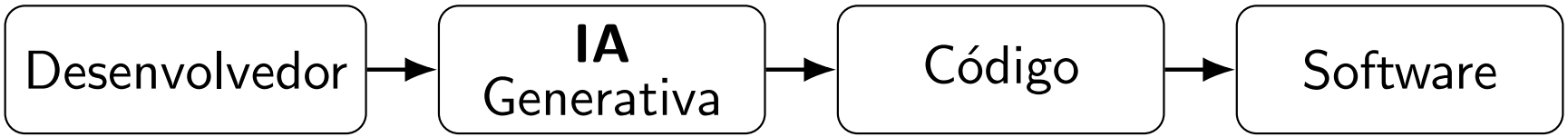
"Funciona?" e "É seguro?" são perguntas diferentes.

Mas alguma coisa importante mudou

Durante muito tempo, uma simplificação possível do desenvolvimento era:



Hoje, em muitas atividades, temos:



Uma nova entidade passou a participar da produção do código.

Mas de quem é a IA que estamos utilizando?

Quando utilizamos uma LLM, normalmente pensamos em:

- capacidade do modelo;
- qualidade das respostas;
- velocidade;
- custo.

Mas, especialmente para organizações públicas, existem outras perguntas:

Onde os dados são processados?

Quem controla o modelo e a infraestrutura?

De quais fornecedores e jurisdições dependemos?

O que significa pensar em uma IA soberana?

Dados	**Modelo**	**Infraestrutura**	**Operação**
controle e proteção	conhecimento e capacidade	processamento e armazenamento	controle e continuidade

Soberania em IA não precisa significar isolamento tecnológico.

É discutir o **grau de controle e autonomia** que uma organização ou um país mantém sobre recursos críticos de IA.

Para órgãos públicos, essa discussão também é uma questão de segurança, governança e continuidade.

E o Brasil já começou essa discussão

O **Plano Brasileiro de Inteligência Artificial (PBIA)** inclui entre suas premissas:

Soberania tecnológica e de dados

e estabelece, entre seus objetivos, o desenvolvimento de:

LLMs em português baseados em dados nacionais.

Para os órgãos públicos, isso reforça uma discussão que já está em curso:

Qual deve ser nossa estratégia para o uso de IA?

Não apenas: *"Qual LLM queremos utilizar?"*
mas também: **"Quais capacidades de IA precisamos controlar?"**

Um exemplo brasileiro: SoberanIA

O Brasil possui iniciativas voltadas ao desenvolvimento de modelos nacionais.

SoberanIA

LLM desenvolvida com foco na língua portuguesa e no contexto brasileiro, com proposta de utilização especialmente voltada ao setor público.

A iniciativa coloca em discussão elementos como:

- dados em língua portuguesa;
- processamento e armazenamento no Brasil;
- infraestrutura nacional;
- adequação ao contexto da Administração Pública;
- maior controle sobre dados e operação.

A discussão sobre soberania envolve não apenas modelos, mas também capacidades nacionais em dados, infraestrutura e operação.

Por que estamos utilizando IA para desenvolver software?

Porque ela pode apoiar diversas atividades do desenvolvimento:

- geração e completamento de código;
- explicação e transformação de código;
- geração de testes;
- documentação;
- identificação de problemas;
- correção e reparo de código;
- apoio à análise de segurança.

Isso pode trazer benefícios importantes de:

velocidade **produtividade** **automação**

O problema não é a existência desses benefícios.
A questão é como obter os benefícios sem introduzir riscos que não percebemos.

Vamos fazer uma previsão

Imagine que pedimos a uma IA para gerar centenas de programas.

Os programas precisam executar corretamente.

Quantos vocês imaginam que poderiam conter vulnerabilidades de segurança?

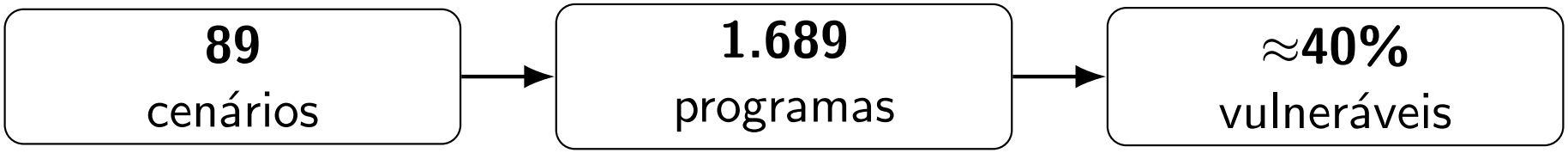
Menos de 5%?

Entre 5% e 20%?

Mais de 20%?

O que aconteceu no experimento?

Pearce et al. [1] avaliaram a segurança de código produzido com **GitHub Copilot** em diferentes cenários associados a vulnerabilidades de software.



Código plausível e funcional pode carregar decisões de implementação inseguras.

Mas que tipo de problema estamos procurando?

Em segurança de software, diferentes fraquezas podem aparecer no código:

- validação inadequada de entradas;
- uso inseguro de memória;
- exposição de informações sensíveis;
- autenticação ou autorização inadequada;
- uso incorreto de mecanismos criptográficos;
- construção insegura de consultas ou comandos.

CWE — Common Weakness Enumeration

É uma classificação utilizada para organizar tipos recorrentes de fraquezas presentes em software e hardware.

Não precisamos memorizar as categorias.
Precisamos reconhecer que decisões de código podem introduzir fraquezas.

Mas talvez estejamos olhando para o problema errado...

Até agora perguntamos:

"A IA pode gerar código vulnerável?"

Mas desenvolvimento de software não envolve apenas a IA.



O que acontece com o comportamento do desenvolvedor quando a IA entra nessa relação?

Vamos fazer outra previsão

Imagine dois grupos de desenvolvedores realizando as mesmas tarefas de programação:

Grupo A	**Grupo B**
Desenvolvedores	Desenvolvedores
sem assistente de IA	com assistente de IA

Qual grupo vocês esperariam que produzisse código mais seguro?

Essa pergunta foi investigada experimentalmente por Perry et al. [2].

O que aconteceu?

Perry et al. [2] realizaram um experimento com participantes executando tarefas de programação relacionadas à segurança.

Os resultados mostraram que os participantes com acesso a um assistente de IA:

produziram soluções significativamente menos seguras

do que participantes que realizaram as tarefas sem esse apoio.

Mas esse não foi o resultado mais interessante.

E havia um paradoxo...

Os participantes que tiveram acesso à IA

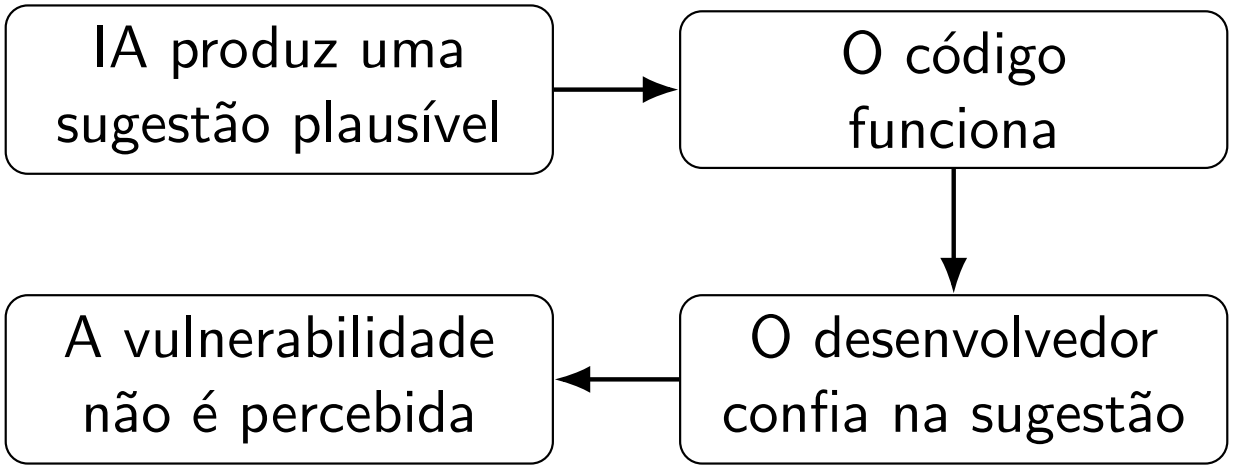
confiavam mais

que haviam produzido uma solução segura.

Mesmo quando suas soluções eram menos seguras.

Então, onde está o risco?

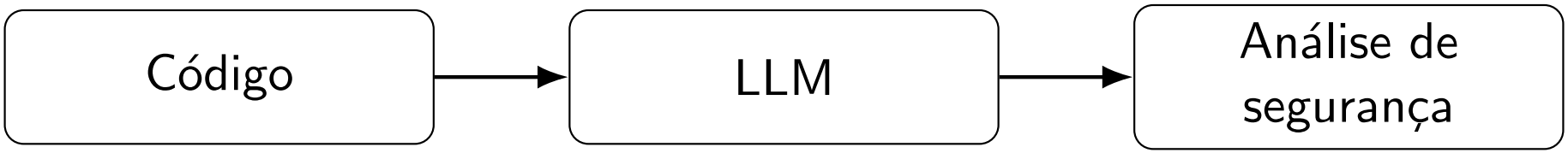
Podemos ter uma combinação problemática:



O maior problema pode não ser apenas a IA errar. Pode ser nós não percebermos que ela errou.

Então a solução parece simples...

Se a IA pode gerar uma vulnerabilidade, poderíamos pedir à própria IA para verificar o código.

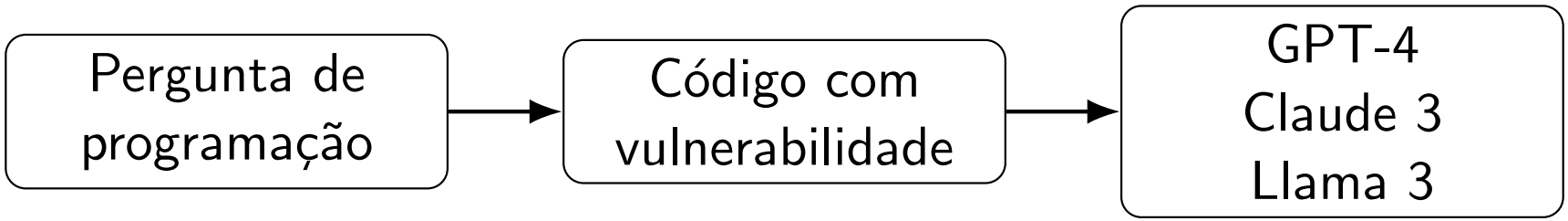


Mas a IA percebe uma vulnerabilidade se ninguém perguntar explicitamente por ela?

Se enviarmos um código vulnerável para um LLM e perguntar apenas como resolver determinado problema de programação, vocês esperam que ele avise espontaneamente sobre a vulnerabilidade?

Um experimento diferente

Sajadi et al. [3] investigaram como LLMs respondem a perguntas de programação quando o código apresentado contém vulnerabilidades.



A pergunta enviada ao modelo **não solicitava explicitamente** uma análise de segurança.

O modelo perceberia espontaneamente o problema de segurança?

Eles perceberam?

A capacidade de identificar espontaneamente as vulnerabilidades foi limitada.

Os autores observaram taxas de identificação das vulnerabilidades entre 12,6% e 40%, dependendo do modelo e das condições avaliadas.

Em muitos casos, o modelo conseguia responder à pergunta do desenvolvedor **sem alertar sobre o problema de segurança presente no código**.

Resolver a tarefa solicitada não significa avaliar tudo que importa.

Um exemplo simples

Considere um código Python que realiza uma requisição:

Código

```
requests.get(url, verify=False)
```

O parâmetro:

`verify=False`

desabilita a verificação do certificado TLS.

A aplicação pode continuar funcionando normalmente.

Mas uma proteção importante da comunicação foi removida.

Então os LLMs são ruins para segurança?

Não é tão simples.

Até agora vimos evidências de que LLMs podem:

- gerar código vulnerável;
- deixar de perceber vulnerabilidades;
- influenciar a confiança do desenvolvedor.

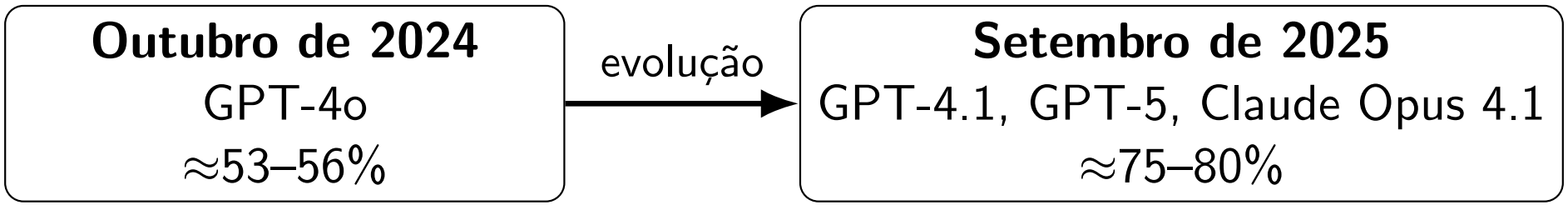
Mas também podem:

- identificar vulnerabilidades;
- explicar problemas de segurança;
- propor correções;
- apoiar atividades de revisão.

E essas capacidades estão evoluindo rapidamente.

Um ano pode fazer muita diferença

Belozerov et al. [4] avaliaram LLMs na identificação e correção de vulnerabilidades encontradas em interações reais de desenvolvimento.



A capacidade de detectar e corrigir vulnerabilidades melhorou substancialmente.

Então podemos ficar tranquilos?

Os modelos:

- estão melhorando;
- detectam mais vulnerabilidades;
- conseguem propor correções;
- podem apoiar revisão de segurança.

Se chegarmos perto de 100%,
o problema estará resolvido?

O que vocês acham?

Software não é construído apenas com o nosso código

Uma aplicação moderna normalmente combina:

- código desenvolvido internamente;
- bibliotecas;
- frameworks;
- pacotes open source;
- APIs e serviços externos;
- ferramentas de build;
- pipelines de integração e entrega;
- componentes produzidos ou sugeridos por IA.

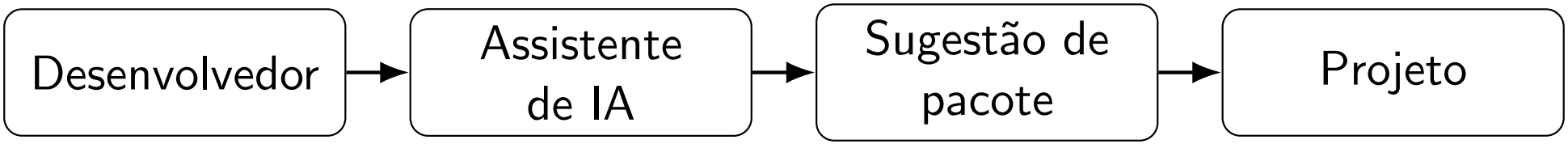
Software Supply Chain

É o conjunto de componentes, dependências, ferramentas, processos e atores envolvidos na produção e entrega de um software.

Uma vulnerabilidade não precisa nascer no código que sua equipe escreveu.

Onde a IA entra nessa cadeia?

Imagine uma situação cotidiana:



O desenvolvedor pergunta:

"Que biblioteca posso usar para fazer X?"

E se o pacote sugerido pela IA não existir?

O que aconteceria se o LLM inventasse um nome de pacote que parece perfeitamente legítimo?

Package Hallucination

Uma package hallucination ocorre quando um modelo gerador de código faz referência a um **pacote que não existe**.

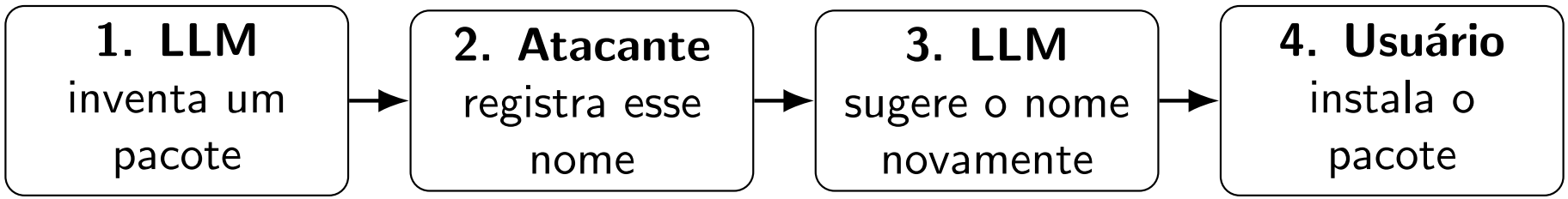
Por exemplo:

```
pip install secure-data-helper
```

O nome parece plausível.
A recomendação parece legítima.

Mas o pacote pode ter sido simplesmente inventado pelo modelo.

Como uma alucinação pode se tornar um ataque?



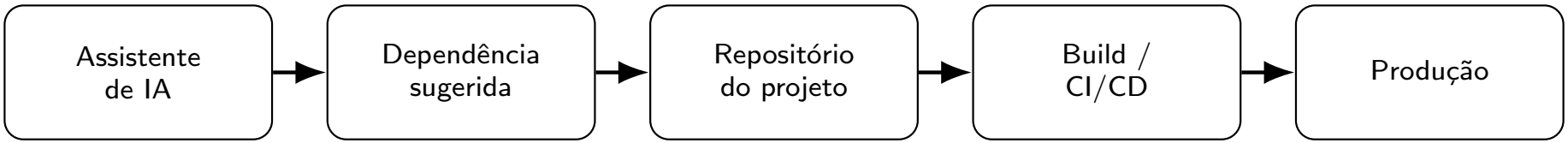
Agora o nome sugerido pelo LLM corresponde a um pacote real.

Mas esse pacote pode conter:

código controlado pelo atacante.

Por que isso é um problema de supply chain?

Porque o componente pode entrar no software como uma **dependência aparentemente legítima**.



Se a dependência comprometida é incorporada ao projeto, o problema pode avançar pela cadeia de desenvolvimento.

A vulnerabilidade não precisa estar no código que a equipe escreveu.

Mas será que isso é apenas um caso raro?

Spracklen et al. [5] decidiram investigar o fenômeno em escala.

16 LLMs

Python e JavaScript

diferentes configurações e conjuntos de prompts

Quantos códigos vocês acham que eles geraram?

10 mil? 100 mil? 500 mil?

O experimento em escala

Os autores geraram **576 mil** amostras de código [5]

Modelos comerciais
Pacotes alucinados
$\geq$ **5,2%**

Modelos open source
Pacotes alucinados
21,7%

Foram identificados **205.474 nomes únicos** de pacotes alucinados.

“Então basta verificar se o pacote existe...”

Parece uma solução simples:

“Antes de instalar, verificamos se o pacote existe no repositório.”

Mas existe um problema:

o atacante pode registrar o nome antes da verificação.

Portanto:

existência $\neq$ **legitimidade** $\neq$ **segurança**

A presença de um pacote em um repositório público não garante que ele seja confiável.

Até aqui, nossa história estava deste lado...



Vimos que a IA pode afetar:

- a segurança do código gerado;
- a percepção do desenvolvedor;
- a identificação de vulnerabilidades;
- a escolha de dependências;
- a software supply chain.

Mas estamos olhando apenas para metade do problema.

O que muda quando a IA passa a apoiar o outro lado?

Agora vamos mudar de perspectiva

Até agora:

Como a IA pode introduzir riscos quando estamos construindo software?

Agora:

Como a IA muda as capacidades de quem tenta atacar o software?

Em quais atividades a IA pode reduzir esforço, aumentar escala, automatizar etapas ou ampliar determinadas capacidades existentes?

A IA criou uma nova geração de ataques?

Muitas técnicas utilizadas por atacantes **não são novas**:

- phishing;
- engenharia social;
- descoberta de vulnerabilidades;
- exploração de sistemas;
- desenvolvimento de malware;
- coleta de informações;
- evasão e adaptação de ataques.

O que a IA pode modificar é:

velocidade **escala** **automação** **acessibilidade**

Nem sempre é um ataque novo.
Pode ser uma nova capacidade para executar um ataque conhecido.

Onde a IA pode apoiar atividades ofensivas?

A literatura já investiga o uso de LLMs em diferentes etapas e atividades relacionadas a ataques.

Reconhecimento e coleta de informações

Engenharia social e phishing

Análise e exploração de vulnerabilidades

Apoio à geração ou modificação de código malicioso

Automação de etapas de ataque

A mudança importante é a combinação entre capacidade e automação.

Antes de continuar: CWE e CVE não são a mesma coisa

Já vimos:

CWE — Common Weakness Enumeration

Representa uma **categoria de fraqueza**, como tipos recorrentes de problemas de segurança no software.

Agora precisamos de outro conceito:

CVE — Common Vulnerabilities and Exposures

É um identificador utilizado para uma **vulnerabilidade específica publicamente conhecida**.

CWE = tipo de fraqueza

CVE = vulnerabilidade específica

Uma coisa é encontrar uma vulnerabilidade...

Outra é conseguir:

explorá-la.

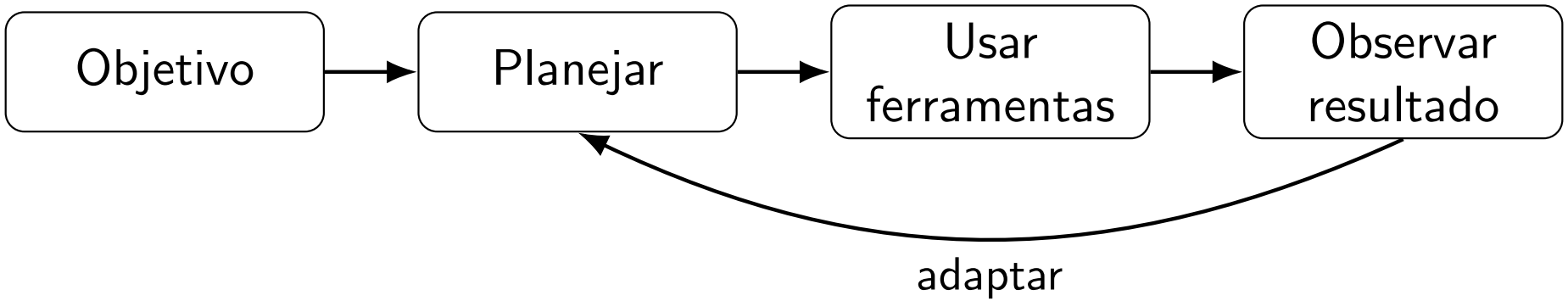
Um agente baseado em IA consegue interagir com um sistema e explorar uma vulnerabilidade real?

De chatbot para agente

Um chatbot tradicional responde a uma solicitação:

Pergunta → LLM → Resposta

Um agente pode operar em um ciclo:



A IA deixa de apenas recomendar uma ação e passa a participar da execução da tarefa.

Testando agentes contra vulnerabilidades reais

O **CVE-Bench** foi criado para avaliar a capacidade de agentes de IA explorarem vulnerabilidades reais em aplicações web [6].

O benchmark utiliza:

- vulnerabilidades publicamente conhecidas;
- aplicações reais reproduzidas em ambientes controlados;
- condições verificáveis de sucesso;
- diferentes agentes e configurações.

A pergunta deixa de ser "o modelo sabe?"
e passa a ser "o agente consegue fazer?"

E os agentes conseguiram?

No cenário *one-day*, os agentes recebem informações sobre vulnerabilidades conhecidas e tentam explorá-las no ambiente [6].

Uma arquitetura multiagente avaliada no CVE-Bench conseguiu explorar:

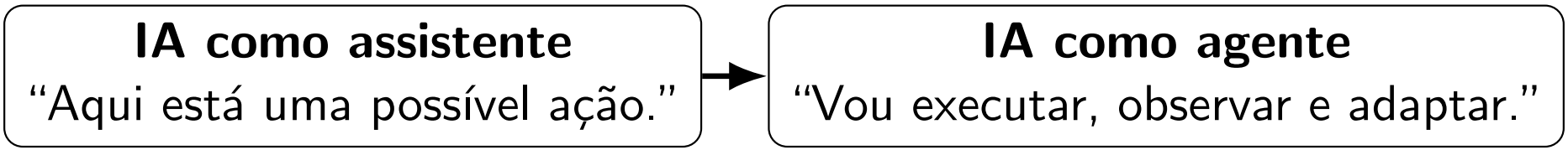
13%

das vulnerabilidades avaliadas nesse cenário.

13% pode parecer pouco.

O ponto importante é que essa capacidade já foi demonstrada experimentalmente em ambiente controlado

O que está mudando?



Isso pode reduzir a quantidade de intervenção humana necessária em determinadas etapas.

Automação muda a escala do problema.

Isso já aparece no cenário de ameaças?

O ENISA Threat Landscape 2026 analisa o cenário europeu de ameaças cibernéticas a partir de incidentes e tendências recentes.

O relatório observa:

- uso crescente de IA em atividades maliciosas;
- utilização de GenAI para apoiar diferentes tarefas;
- evolução das capacidades dos agentes;
- expansão da IA para mais etapas das operações cibernéticas.

A IA passa a aparecer dos dois lados da superfície de ataque.

O que já vemos e o que pode vir a seguir?

Já observado

IA apoiando atividades maliciosas
e diferentes tarefas da operação

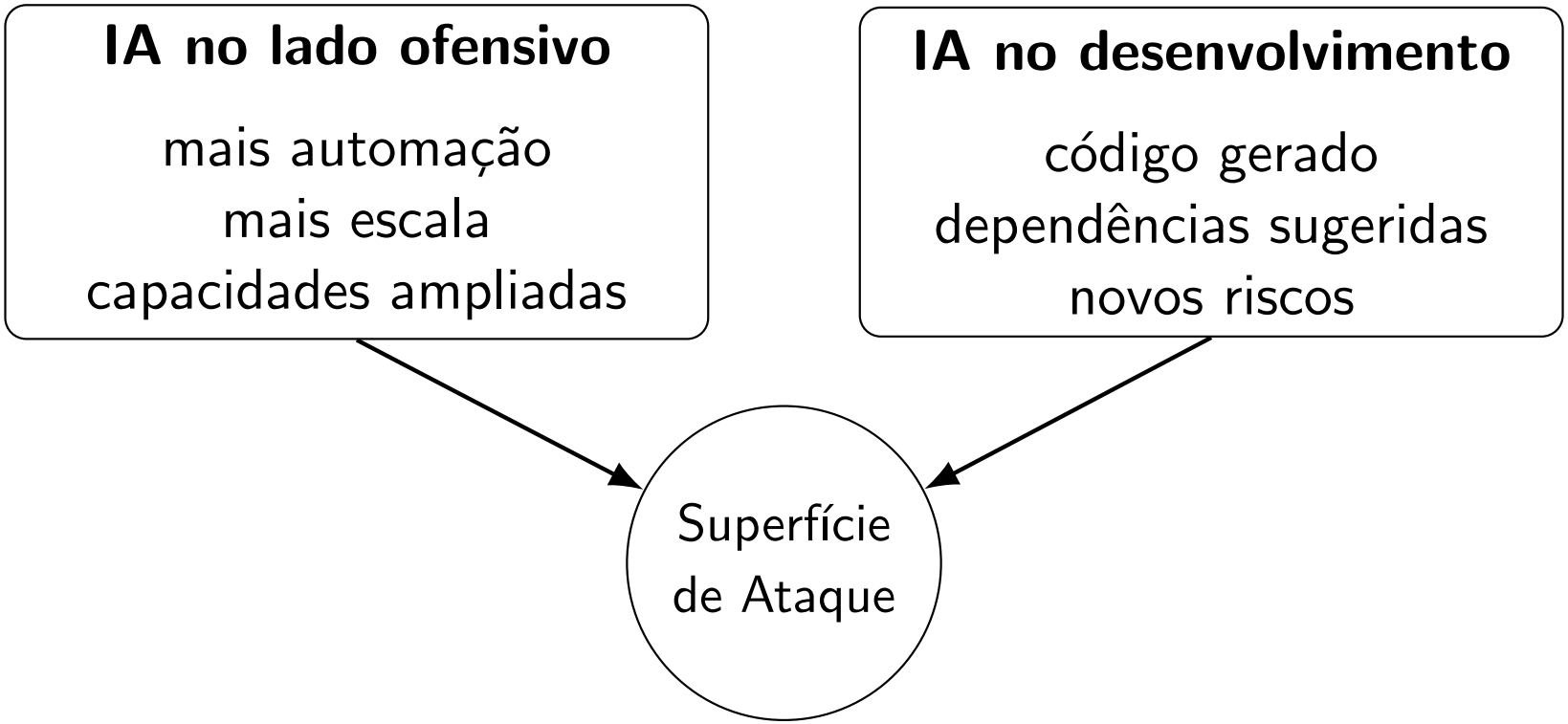
↓

Direção apontada para 2026

mais etapas diretamente habilitadas por IA
e experimentação com menor intervenção humana

A tendência não é apenas IA "ajudando" o atacante, mas participando de uma parcela maior da operação.

Agora conseguimos enxergar o problema completo



A IA está mudando os dois lados da equação de segurança.

Então devemos parar de usar IA para desenvolver software?

Não.

A questão não é eliminar a IA do desenvolvimento.

A questão é:

como incorporá-la sem eliminar os mecanismos de segurança?

Primeiro princípio

A IA pode participar da produção do software.

Mas sua saída não deve ser tratada como evidência.

Código gerado pela IA precisa passar pelos mesmos controles de Engenharia de Software e Segurança aplicáveis ao código produzido por outras fontes.

Isso significa revisão, testes, análise de segurança, verificação das dependências e controles no pipeline.

O fluxo não deveria terminar na geração



Gerar é apenas uma etapa.
Aceitar é outra decisão.

A automação da geração **não deve implicar** automação da confiança.

A validação não precisa ser exclusivamente humana.

Na verdade, o que veremos é uma combinação de controles automatizados e revisão humana.

O que precisamos verificar?

A validação pode combinar diferentes mecanismos:

- **Code Review** – revisão das decisões de implementação;
- **Testes** – verificação do comportamento esperado;
- **SAST** – análise estática do código em busca de fraquezas;
- **SCA** – análise dos componentes e dependências utilizados;
- **Secret Scanning** – identificação de credenciais e segredos expostos;
- **Security Tests** – testes direcionados a propriedades de segurança.

Nenhum mecanismo isolado precisa responder por toda a segurança.

Usamos mecanismos diferentes porque eles procuram classes diferentes de problemas.

E quando a IA sugere uma dependência?

Não basta perguntar:

"O pacote existe?"

Também precisamos considerar:

- de onde esse componente veio?
- quem o mantém?
- qual versão está sendo utilizada?
- possui vulnerabilidades conhecidas?
- é realmente a dependência pretendida?
- por que ela precisa entrar no projeto?

Sugestão da IA não deve equivaler à aprovação de uma dependência.

Não podemos proteger aquilo que não conhecemos

SBOM — Software Bill of Materials

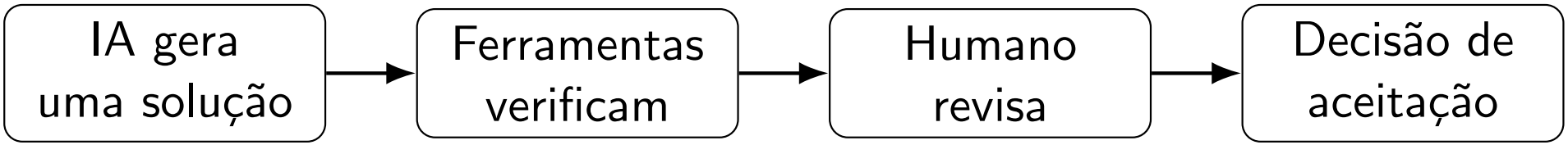
Um inventário dos componentes que fazem parte de um software, incluindo suas dependências e informações associadas.

Uma SBOM ajuda a responder:

- quais componentes estão presentes?
- quais versões utilizamos?
- onde determinada dependência aparece?
- quais sistemas podem ser afetados quando uma vulnerabilidade é descoberta?

Com IA ou sem IA, visibilidade sobre a supply chain é essencial.

E onde fica o desenvolvedor?

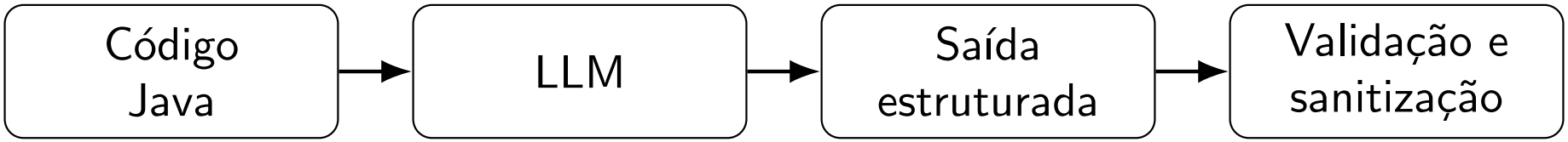


Human-in-the-loop não significa "o humano confere tudo manualmente".

O papel humano é especialmente importante para **interpretar contexto, avaliar trade-offs, analisar exceções e assumir decisões de aceitação** quando elas têm impacto relevante.

IA pode fazer parte da própria defesa

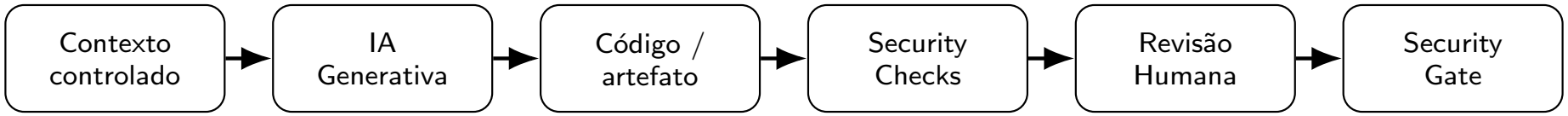
Falcão e Canedo [7] avaliaram diferentes estratégias para detectar informações pessoais em código Java.



O melhor equilíbrio entre precisão e cobertura foi observado quando a saída estruturada permaneceu estável e a validação/sanitização foi aplicada.

A capacidade do LLM é combinada com controles de Engenharia.

Um desenvolvimento assistido por IA mais seguro



Antes e durante a geração

- requisitos de soberania e controle;
- política de uso;
- controle dos dados enviados;
- ferramentas aprovadas;
- rastreabilidade.

Antes da aceitação

- testes;
- SAST e SCA;
- revisão de dependências;
- revisão humana;
- critérios de aprovação.

A IA entra no SDLC. A segurança precisa entrar junto.

E afinal: quem cuida da segurança?

A IA pode:

gerar analisar recomendar corrigir

Mas ela não define sozinha:

risco aceitável **responsabilidade** **governança**

A responsabilidade continua sendo da organização e das pessoas que tomam as decisões.

Vamos voltar ao início...

No começo eu perguntei:

"A IA gerou um código que compila, executa e produz o resultado esperado."

Que evidências e controles precisamos ter para aceitar esse código com segurança?

Depois de tudo que vimos...

essa talvez não seja mais a melhor pergunta.

Talvez a pergunta certa seja outra

Em vez de perguntar:

"Eu confio no código que a IA produziu?"

podemos perguntar:

Que evidências e controles precisamos ter para aceitar esse código com segurança?

Confiança não substitui verificação.

Para levar desta masterclass...

1. **A IA está mudando a forma como software é produzido.**
2. **A IA também está mudando capacidades relacionadas a ataques.**
3. **Podemos delegar tarefas à IA.**
   Não podemos delegar a responsabilidade pela segurança.

Perguntas?

Professora Dra. Edna Dias Canedo

ednacanedo@cin.ufpe.br

H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, and R. Karri, "Asleep at the keyboard? assessing the security of github copilot's code contributions," *Commun. ACM*, vol. 68, no. 2, pp. 96–105, 2025. [Online]. Available: https://doi.org/10.1145/3610721

N. Perry, M. Srivastava, D. Kumar, and D. Boneh, "Do users write more insecure code with ai assistants?" in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 2785–2799. [Online]. Available: https://doi.org/10.1145/3576915.3623157

A. Sajadi, B. Le, A. Nguyen, K. Damevski, and P. Chatterjee, "Do llms consider security? an empirical study on responses to programming questions," *Empir. Softw. Eng.*, vol. 30, no. 3, p. 101, 2025. [Online]. Available: https://doi.org/10.1007/s10664-025-10658-6

V. Belozerov, P. J. Barclay, and A. Sami, "Secure coding with AI - from detection to repair," *Empir. Softw. Eng.*, vol. 31, no. 4, p. 93, 2026. [Online]. Available: https://doi.org/10.1007/s10664-026-10812-8

J. Spracklen, R. Wijewickrama, A. H. M. N. Sakib, A. Maiti, B. Viswanath, and M. Jadliwala, "We have a package for you! A comprehensive analysis of package hallucinations by code generating llms," in *34th USENIX Security Symposium, USENIX Security 2025, Seattle, WA, USA, August 13-15, 2025*, L. Bauer and G. Pellegrino, Eds. USENIX Association, 2025, pp. 3687–3706. [Online]. Available: https://www.usenix.org/conference/usenixsecurity25/presentation/spracklen

Y. Zhu, A. Kellermann, D. Bowman, P. Li, A. Gupta, A. Danda, R. Fang, C. Jensen, E. Ihli, J. Benn *et al.*, "Cve-bench: A benchmark for ai agents' ability to exploit real-world

web application vulnerabilities," *arXiv preprint arXiv:2503.17332*, 2025. [Online]. Available: https://arxiv.org/abs/2503.17332



F. D. S. Falcão and E. D. Canedo, "Evaluating foundation model integration strategies for detecting pii in java software engineering pipelines," *Empirical Software Engineering*, vol. 31, no. 6, p. 184, 2026. [Online]. Available: https://doi.org/10.1007/s10664-026-10919-y