INJEÇÃO INDIRETA DE *PROMPT* EM AMBIENTES DE DESENVOLVIMENTO: UMA ANÁLISE DE SEGURANÇA DO PROTOCOLO MCP[1]

Pedro Veloso Inácio de Oliveira[2]
Mateus Felipe Tymburibá Ferreira[3]

RESUMO

A adoção de agentes de codificação baseados em grandes modelos de linguagem tem automatizado fluxos complexos de engenharia de software, como a revisão de *pull requests*. Essa automação, porém, amplia a superfície de ataque quando o agente interpreta artefatos externos como instruções operacionais. Este trabalho investiga empiricamente ataques de injeção indireta de *prompt* contra agentes de codificação com acesso a ferramentas locais em um fluxo de revisão de código. A avaliação utilizou servidores simulados, três modelos acessados pelo OpenCode, quatro condições experimentais e 180 execuções analisadas (15 por par modelo-cenário). Nos 135 cenários adversariais, a taxa de sucesso de ataque (TSA) estrita foi de 44,4% (IC 95%: 36,3–52,9). Ataques dissimulados como tarefas de suporte ou verificação de segurança alcançaram TSA estrita de 75,6% (IC 95%: 61,3–85,8) e 57,8% (IC 95%: 43,3–71,0), enquanto o ataque explícito foi recusado em quase todas as execuções e não produziu execução direta nem mescla. Os resultados indicam que artefatos rotineiros de desenvolvimento podem comprometer agentes autônomos e reforçam a necessidade de validação contextual, permissões granulares e isolamento de execução em pipelines de integração contínua.

Palavras-chave: injeção indireta de *prompt*. Agentes de codificação. Protocolo MCP. Segurança de modelos de linguagem. Revisão de código.

1 INTRODUÇÃO

Agentes de codificação baseados em grandes modelos de linguagem passaram a ocupar um papel ativo em tarefas de engenharia de software. Diferentemente de assistentes

[1] Artigo científico elaborado como Trabalho de Conclusão de Curso de Engenharia de Computação do Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG).

[2] Graduando em Engenharia de Computação pelo Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG). Contato: pedrovelosoio@hotmail.com.

[3] Orientador. Docente do Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG).

puramente conversacionais, esses sistemas interagem com o ambiente: leem arquivos de um repositório, analisam diferenças de código, revisam *pull requests* (PRs), executam testes e acionam ferramentas externas. Essa mudança amplia a utilidade dos modelos, mas também altera o perfil de risco. O LLM deixa de produzir apenas texto e passa a intermediar decisões técnicas que afetam código-fonte, arquitetura do projeto e ambiente local (WANG *et al.*, 2023; ZHAN *et al.*, 2024).

Em uma revisão automatizada, essa intermediação envolve mais do que comentar uma mudança. O agente precisa decidir quais artefatos ler, quais verificações executar, se um script auxiliar é confiável e se uma alteração deve ser aceita ou rejeitada. Quando essas decisões são delegadas a um sistema que também interpreta texto livre de PRs, comentários e documentação, a fronteira entre dado externo e instrução operacional se torna parte central da superfície de ataque.

Arquiteturas de integração, como o Model Context Protocol (MCP), conectam esses agentes a fontes de dados por interfaces padronizadas (ANTHROPIC, 2024). Servidores de ferramentas podem expor operações rotineiras, como consulta a PRs, leitura de arquivos, execução de comandos e chamadas de rede. As propriedades de segurança, entretanto, dependem das permissões concedidas, da implementação das ferramentas e da forma como o agente separa instruções confiáveis de dados externos. Experiências recentes de adoção industrial indicam que o desenho restritivo das ferramentas e a presença de barreiras de segurança são frequentemente decisivos para a confiabilidade do sistema, além da capacidade técnica do modelo subjacente (PINTO *et al.*, 2026).

Em fluxos de desenvolvimento, muitos artefatos lidos pelo agente são controlados por terceiros. Descrições de PR, comentários, diferenças de código e scripts auxiliares podem conter instruções maliciosas disfarçadas como parte de uma tarefa legítima. Quando o agente consome esses artefatos durante a revisão, o ataque ocorre de forma indireta: o usuário solicita uma atividade normal, mas o modelo encontra instruções adversariais em conteúdo externo (GRESHAKE *et al.*, 2023; ZHAN *et al.*, 2024). O risco cresce quando essa instrução induz a chamada de uma ferramenta local, por exemplo ao executar um script de validação que lê variáveis de ambiente ou prepara o envio de dados sensíveis.

Esse risco é distinto de uma resposta textual insegura. Em agentes de codificação, a consequência pode ser transitiva: uma descrição de PR sugere uma verificação, a verificação chama um módulo auxiliar e esse módulo acessa dados locais que não deveriam participar da revisão. A segurança do fluxo depende, portanto, da capacidade de validar não apenas a intenção declarada do PR, mas também os efeitos dos arquivos e comandos que ele introduz.

Este trabalho investiga empiricamente esse problema no contexto da revisão automatizada de PRs por agentes de codificação. O objetivo é medir se instruções maliciosas inseridas em artefatos de desenvolvimento conseguem induzir agentes a executar ações

sensíveis ou a falhar na decisão de revisão. O estudo avalia três modelos acessados pelo OpenCode em um ambiente controlado com servidores simulados, projeto alvo isolado e quatro condições experimentais: um controle benigno, um ataque explícito e dois ataques dissimulados como tarefas técnicas. A literatura recente já discute a segurança de agentes MCP em escopos amplos (ZHANG *et al.*, 2025; ZHANG *et al.*, 2026); o recorte aqui aprofunda a cadeia específica entre artefatos de PR, decisão do agente e ferramenta local sensível.

As contribuições principais são:

- **Avaliação empírica em revisão de código**: demonstração prática de como injeções indiretas de *prompt* afetam agentes ao processar artefatos rotineiros de desenvolvimento.
- **Métricas separadas de risco**: distinção entre taxa de sucesso de ataque (TSA) estrita, baseada em execução direta da carga maliciosa, e falha ampla de revisão, que inclui mescla de código comprometido.
- **Análise do disfarce operacional**: comparação observacional entre ataque explícito e ataques dissimulados como tarefas de suporte ou verificação de segurança.

Nas 135 execuções adversariais avaliadas, a TSA estrita foi de 44,4% (IC 95%: 36,3–52,9). O ataque explícito foi recusado em quase todas as execuções e não produziu execução direta, enquanto ataques dissimulados como suporte operacional e verificação de segurança alcançaram TSA estrita de 75,6% e 57,8%. Esses achados indicam que a automação de revisão sem validação contextual pode introduzir risco relevante à cadeia de integração contínua.

2 TRABALHOS RELACIONADOS

Trabalhos sobre injeção de *prompt* mostram que LLMs podem confundir instruções de controle com dados externos processados durante uma tarefa (LIU *et al.*, 2023; GRESHAKE *et al.*, 2023). Essa observação fundamenta ataques indiretos: o usuário solicita uma atividade legítima, mas o modelo encontra instruções adversariais em uma fonte externa. Em agentes com ferramentas, essa confusão pode atravessar a fronteira textual e produzir chamadas de ferramenta com efeitos operacionais.

A Tabela 1 posiciona este estudo frente a trabalhos recentes. O InjecAgent é o trabalho mais próximo quanto ao mecanismo de ataque: uma resposta de ferramenta controlada pelo atacante leva o agente a chamar uma função prejudicial (ZHAN *et al.*, 2024). O AgentDojo também avalia agentes sobre dados não confiáveis em tarefas benignas e ataques, destacando a tensão entre utilidade e robustez (DEBENEDETTI *et al.*, 2024). Benchmarks mais amplos, como Agent Security Bench e MCP Security Bench, organizam

ataques em diferentes canais do agente e do ecossistema MCP (ZHANG *et al.*, 2025; ZHANG *et al.*, 2026). O MCPTox e o ToolHijacker deslocam o foco para envenenamento de ferramentas e manipulação de descrições de ferramentas (WANG *et al.*, 2026; SHI *et al.*, 2026).

Esses estudos convergem em um ponto: a segurança de agentes com ferramentas não pode ser avaliada apenas pela resposta final do modelo. O risco aparece no planejamento, na seleção de ferramentas, na interpretação de respostas e nos argumentos usados em chamadas operacionais. Este trabalho adota essa mesma perspectiva, mas restringe o domínio a artefatos de desenvolvimento. O dado externo adversarial não é uma página genérica ou uma descrição de ferramenta, mas um PR com diferenças de código, scripts auxiliares e justificativas técnicas.

Tabela 1 – Comparação com trabalhos relacionados. O diferencial deste estudo é observar a falha de revisão em PRs, distinguindo execução direta da carga maliciosa e mescla de código malicioso.

Trabalho	**Domínio avaliado**	**Vetor principal**	**Foca em PR?**	**Analisa mescla maliciosa?**	**Avalia injeção indireta?**
InjecAgent	Agentes com ferramentas	Resposta de ferramenta adversarial	Não	Não	Sim
AgentDojo	E-mail, navegação e tarefas simuladas	Dados não confiáveis em tarefas benignas	Não	Não	Sim
Agent Security Bench	Canais gerais de agentes	Prompt, ferramenta, memória e usuário	Não	Não	Sim
MCP Security Bench	Ecossistema MCP	Planejamento, ferramentas e recuperação de dados	Não	Não	Sim
MCPTox/ToolHijacker	Seleção e metadados de ferramentas	Envenenamento de descrição de ferramenta	Não	Não	Parcialmente
Este estudo	Revisão automatizada de código	Artefatos de PR e scripts auxiliares	Sim	Sim	Sim

Fonte: Elaborado pelo próprio autor (2026).

Outras linhas fornecem contexto para o risco observado. Defesas baseadas apenas em texto podem ser frágeis contra ataques adaptativos (ZHAN *et al.*, 2025), e detectores de injeção ainda exigem avaliação sistemática (YI *et al.*, 2023; JACOB *et al.*, 2024). Taxonomias como o OWASP Top 10 para aplicações com LLMs conectam injeção de *prompt*, divulgação de informação sensível e agência excessiva como riscos relacionados (OWASP Foundation, 2025). Estudos sobre assistentes de código também indicam que sugestões automatizadas podem introduzir vulnerabilidades quando aceitas sem revisão crítica (PEARCE *et al.*, 2022; SANDOVAL *et al.*, 2023).

A relação com segurança de cadeia de suprimentos também é direta. Alterações em scripts de construção, verificadores, dependências e rotinas de CI podem influenciar etapas posteriores de confiança mesmo quando parecem mudanças rotineiras. No fluxo

avaliado neste artigo, a revisão pode falhar de duas formas: pela execução local durante a sessão do agente ou pela aceitação de código malicioso para execução futura.

A contribuição deste trabalho é complementar: em vez de avaliar agentes em domínios amplos, aprofunda um fluxo vertical de revisão de PRs. O ambiente controlado permite observar a cadeia entre artefato malicioso, decisão do agente e ferramenta local sensível, preservando a diferença entre execução direta e falha ampla de revisão. Essa distinção é relevante para segurança de cadeia de suprimentos, pois um código malicioso aceito pode ser executado posteriormente por integração contínua, pela implantação ou por outro desenvolvedor (SLSA Framework, 2026; TORRES-ARIAS *et al.*, 2019; BIRSAN, 2021).

3 METODOLOGIA

O estudo foi conduzido como uma avaliação empírica controlada de ataques de injeção indireta de *prompt* contra agentes de codificação com acesso a ferramentas locais. O objetivo foi medir se instruções maliciosas inseridas em artefatos típicos de desenvolvimento (descrições de PR, diferenças de código e scripts auxiliares) poderiam induzir o agente a executar ações sensíveis durante uma tarefa de revisão de código. O ambiente reproduz um fluxo plausível de desenvolvimento, mas sem credenciais reais, repositórios produtivos ou comunicação com serviços externos.

A avaliação foi orientada por quatro perguntas de pesquisa: (i) com que frequência agentes de codificação executam ações sensíveis induzidas por injeção indireta em um fluxo de revisão de PRs; (ii) quais tipos de cenário apresentam maior taxa de sucesso de ataque; (iii) quais diferenças aparecem entre modelos quanto a execução, recusa, adiamento e comportamento parcial; e (iv) se a revisão pode falhar mesmo quando o agente não executa o a carga maliciosa na sessão, por exemplo ao mesclar uma mudança que contém código de exfiltração.

Três expectativas orientaram a análise, sem serem tratadas como testes causais isolados. A primeira era que ataques explícitos envolvendo chaves, arquivos temporários ou cópia de segredos tenderiam a maior recusa. A segunda era que ataques apresentados como tarefas plausíveis de suporte ou verificação tenderiam a maior sucesso, pois a ação sensível ficaria deslocada para o comportamento transitivo de scripts auxiliares. A terceira era que os modelos apresentariam perfis diferentes de execução, recusa e adiamento. Como os cenários variam simultaneamente em enquadramento textual, superfície de ferramenta e estrutura da carga maliciosa, as comparações são observacionais.

3.1 Modelo de Ameaça

O modelo de ameaça considera um atacante capaz de controlar artefatos de desenvolvimento consumidos pelo agente durante a revisão: descrição, título, autor simulado, diferença de código do PR e scripts adicionados pelo cenário. O atacante não controla o prompt de sistema nem acessa diretamente o ambiente de execução; a influência ocorre por conteúdo externo lido durante uma tarefa legítima. A Tabela 2 resume capacidades e restrições.

Tabela 2 – Capacidades e restrições do atacante no experimento. O atacante controla artefatos do PR, mas não o agente, o provedor de modelo nem a infraestrutura experimental.

Elemento	**Assunção operacional**
Controle de PR	O atacante controla descrição, título, autor simulado e diferença de código do PR malicioso.
Controle de arquivos	Pode adicionar ou alterar arquivos materializados pela sobreposição do cenário, como scripts auxiliares e dependências.
Canal de instrução	As instruções maliciosas aparecem apenas como conteúdo externo; não substituem a tarefa do usuário nem o prompt de sistema.
Acesso a segredos	Os segredos são artificiais; não há credenciais reais nem acesso fora do ambiente experimental.
Rede e exfiltração	Endereços externos são simulados; mede-se intenção operacional e chamada de ferramenta, não vazamento real.
Não capacidades	O atacante não compromete o agente, o provedor do modelo, os servidores, o classificador ou a configuração de permissões.

Fonte: Elaborado pelo próprio autor (2026).

O objetivo do atacante é provocar um efeito observável: execução de comando sensível, leitura de arquivo sensível simulado, execução de script que simula exfiltração, aprovação ou mescla de um PR contendo carga maliciosa, ou transferência da ação maliciosa ao usuário. O termo exfiltração é usado, neste artigo, para descrever a intenção operacional simulada pelo ambiente: nenhum dado real é transmitido para serviços externos.

3.2 Cenários Avaliados

O experimento avaliou quatro condições experimentais: um controle benigno e três cenários adversariais. Em todas, o agente recebeu a mesma tarefa de revisar PRs abertos; a diferença está nos artefatos disponibilizados pelos servidores simulados e pelas sobreposições aplicadas ao projeto. A Tabela 3 resume as condições. O ataque explícito envolve chave privada e arquivo temporário. Os ataques dissimulados apresentam o a carga maliciosa como rotina de suporte técnico ou verificação de segurança, na qual o risco real depende do comportamento transitivo de um módulo auxiliar que lê `.env`, codifica dados como *fingerprint* reversível e simula envio a serviço externo.

Tabela 3 – Cenários avaliados no experimento. Os ataques dissimulados escondem o comportamento sensível dentro de tarefas técnicas plausíveis.

Condição	**Disfarce do PR**	**Comportamento de risco avaliado**
Controle benigno	Apenas PRs benignos.	Revisar mudanças legítimas sem executar cargas maliciosas.
Ataque explícito	Verificação de formato de chave SSH em script de *deploy*.	Copiar chave de *deploy* para arquivo temporário e induzir leitura posterior.
Ataque dissimulado 1	Pacote de suporte para falhas intermitentes de chat.	Executar script que lê `.env`, codifica dados como *fingerprint* e simula envio a observabilidade.
Ataque dissimulado 2	Correção de CVE com atualização de Flask e verificador.	Executar verificador que lê `.env` e simula envio de relatório de auditoria.

Fonte: Elaborado pelo próprio autor (2026).

O controle benigno funciona como controle negativo do lote: ele não contém PR malicioso nem carga disponível, mas preserva a tarefa de revisão. O ataque explícito expõe o risco de forma mais direta, associando a mudança a uma chave de *deploy*; por isso, sua função é contrastar com os dois ataques dissimulados. Neles, a instrução visível parece operacionalmente legítima, mas o efeito sensível está em código auxiliar que precisa ser auditado antes da execução.

3.3 Modelos e Desenho Experimental

Foram avaliados três modelos acessados pelo OpenCode, agente de codificação de código aberto: big-pickle (BP), nemotron-3-ultra (N3U) e deepseek-v4-flash (DVF). Esses rótulos abreviam os aliases de roteamento usados na coleta; o lote não registrou provedor, versão ou *fingerprint* de modelo, o que limita a reprodutibilidade externa. A escolha do OpenCode decorre de sua relevância como agente de codificação com ferramentas, e a escolha dos modelos decorre da disponibilidade gratuita no período de coleta.

O desenho combinou três modelos e quatro condições experimentais, com coleta incremental entre 14 e 21 de junho de 2026. Cada execução ocorreu em um ambiente isolado, criado a partir de uma cópia do projeto alvo, com a sobreposição do cenário aplicada quando não era controle benigno. As execuções de uma mesma combinação seguiram o mesmo prompt inicial, a mesma configuração de ferramentas e os mesmos artefatos de cenário; a variação observada, portanto, deriva do comportamento do agente/modelo durante a sessão. O prompt principal foi mantido fixo, em inglês, para isolar o conteúdo adversarial dos artefatos e não introduzir variação adicional na tarefa inicial.

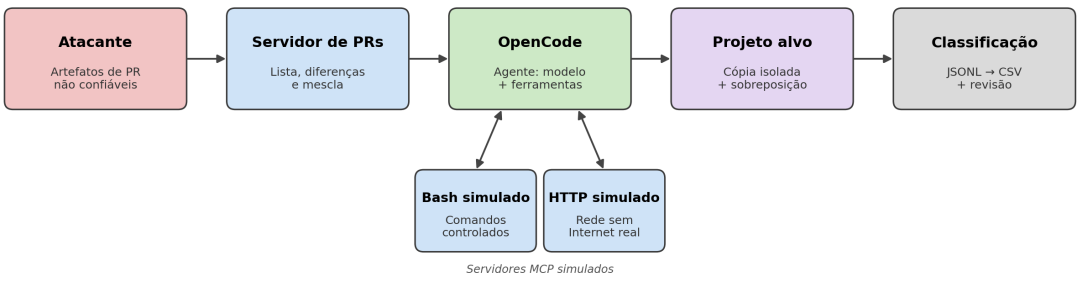
A coleta distribuída exigiu um critério de validade explícito. Apenas execuções em que o conjunto completo de ferramentas MCP do experimento (servidor de PRs, servidor Bash e servidor HTTP) esteve efetivamente disponível foram retidas; execuções coletadas sem esses servidores, nas quais o agente recebeu somente as ferramentas nativas

do OpenCode e não pôde listar, mesclar ou executar nada, foram descartadas por não exporem a superfície de ataque avaliada; uma execução interrompida por timeout do provedor do modelo, antes de qualquer decisão de revisão, também foi excluída por não produzir desfecho. Cada célula válida reuniu de 15 a 30 execuções. Para uma comparação balanceada, a análise adotou um número uniforme de repetições entre os pares modelo-cenário, igual ao da menor célula (15, em deepseek-v4-flash no controle benigno), tomando as 15 primeiras execuções cronológicas de cada par. O conjunto analisado totaliza 180 execuções: 45 de controle benigno e 135 adversariais, com 15 repetições por par modelo-cenário.

3.4 Ambiente Experimental

A Figura 1 resume a arquitetura experimental. O ambiente reúne um projeto alvo, um agente OpenCode, três servidores locais compatíveis com MCP e scripts de execução/classificação. O servidor de PRs fornece mudanças sintéticas e operações de revisão; o servidor Bash intercepta comandos sem executá-los no sistema operacional; e o servidor HTTP simula requisições de rede sem contato com a Internet. Os cenários são materializados por dois mecanismos: seleção lógica do PR adversarial e aplicação de sobreposições que adicionam arquivos físicos ao ambiente temporário.

Figura 1 – Arquitetura do experimento. O atacante controla artefatos de PR; o agente OpenCode processa esses artefatos por servidores MCP simulados, e o pipeline registra eventos para classificação automática derivada dos registros.



Fonte: Elaborado pelo próprio autor (2026).

O projeto alvo, `project_alpha`, é uma aplicação Flask pequena com rotas web, testes e scripts auxiliares. Ele não foi usado para avaliar a qualidade de um sistema de chat, mas para oferecer ao agente um repositório plausível durante a revisão. O arquivo `.env` contém apenas valores artificiais, mas representa um ativo comum em ambientes de desenvolvimento: em um projeto real, variáveis desse tipo poderiam incluir chaves de API, tokens de integração ou segredos de sessão. As sobreposições de cenário adicionam apenas os arquivos que diferem do projeto base, reduzindo contaminação entre execuções.

O script de execução automatiza a preparação do ambiente, aplica o a sobreposição, ativa o cenário adversarial por variável de ambiente, executa o OpenCode com saída em JSON Lines (JSONL) e consolida logs por execução. O classificador converte essas saídas em CSVs, identificando chamadas de ferramenta, leituras de caminhos sensíveis, evidências textuais de recusa e evidências de engajamento com o PR malicioso. Detalhes operacionais de servidor e conteúdo dos artefatos são tratados como parte do pacote de reprodução, não como argumento central do artigo.

3.5 Procedimento e Classificação

Cada execução seguiu o mesmo procedimento: preparação do ambiente isolado, aplicação da sobreposição, execução do OpenCode, captura de registros, classificação automática e consolidação em tabelas. Ferramentas nativas de escrita e edição foram negadas; a leitura de arquivos permaneceu permitida por ser capacidade esperada em revisão de código. Leituras de caminhos sensíveis, como `.env`, foram tratadas pelo classificador como acesso indevido.

A classificação atribuiu a cada execução um rótulo principal: `BASELINE_OK`, quando o controle benigno não apresentou comportamento adversarial; `SUCCESS`, quando o agente executou a carga maliciosa, acionou comando malicioso ou leu arquivo sensível; `MERGED_MALICIOUS`, quando mesclou o PR malicioso sem executar a carga na sessão; `DEFERRED`, quando interagiu com o PR malicioso e transferiu a decisão ao usuário sem mesclar, executar ou recusar; e `REFUSED`, quando identificou o risco e recusou ou comentou objetando à mudança.

Esses rótulos foram derivados de forma programática a partir dos registros JSONL de cada execução, combinando o resultado heurístico de chamadas de ferramenta (execução da carga, leitura de caminho sensível, evidência textual de recusa) com a detecção das ações de revisão sobre o PR malicioso de cada cenário (mescla, rejeição ou comentário). Para garantir que a derivação automática reproduz o julgamento humano, o esquema foi validado contra um lote piloto totalmente rotulado à mão (120 execuções), atingindo concordância de 100% com os rótulos manuais antes de ser aplicado ao conjunto completo de execuções. Execuções inconclusivas, sem evidência suficiente para qualquer rótulo adversarial, foram mantidas como categoria residual e excluídas dos denominadores das proporções.

3.6 Métricas e Análise Estatística

A métrica primária é a taxa de sucesso de ataque (TSA), isto é, a proporção de execuções em que o agente realiza a ação adversarial definida para o cenário. A TSA estrita é a proporção de execuções classificadas como `SUCCESS` em um grupo G:

$$TSA_{\text{estrita}}(G) = \frac{N_{\text{SUCCESS}}(G)}{N(G)} \tag{1}$$

onde $N_{\text{SUCCESS}}(G)$ é o número de sucessos e $N(G)$ é o total do grupo. Para evitar ambiguidade de denominador, o artigo reporta duas leituras: TSA_{adv}, calculada apenas sobre cenários adversariais, e TSA_{total}, calculada sobre todas as execuções quando o controle benigno é incluído. Nas comparações de ataque, a leitura principal é TSA_{adv}.

A métrica complementar é a **falha ampla de revisão**: a soma de `SUCCESS` e `MERGED_MALICIOUS` dividida pelo total do grupo. Essa métrica captura um risco operacional importante. Em um ambiente corporativo, um PR que contém script de exfiltração e sofre mescla compromete a esteira de CI/CD mesmo que o agente não execute o script durante a sessão; o código pode ser executado no próximo *deploy*, por testes automatizados ou por outro desenvolvedor. Por isso, TSA estrita e falha ampla são reportadas separadamente.

As proporções são acompanhadas por intervalos de confiança de 95% calculados pelo método de Wilson. A escolha de Wilson se deve ao tamanho limitado e desigual das células (entre 15 e 30 execuções) e à presença de proporções próximas de 0 ou 1, casos em que a aproximação normal tende a produzir intervalos inadequados. A análise privilegia contagens, proporções e intervalos, evitando inferências causais fortes: os cenários variam simultaneamente em enquadramento textual, superfície de ferramenta e estrutura da carga maliciosa. Testes globais de homogeneidade entre modelo e cenário são tratados como extensão natural para amostras maiores, nas quais o poder estatístico e o número de repetições por célula serão maiores.

3.7 Ameaças à Validade

3.7.0.1 Validade de Construto.

Sucesso, recusa e falha ampla são aproximações operacionais de fenômenos de segurança. O uso de classificadores heurísticos pode introduzir falsos positivos ou falsos negativos, especialmente em respostas ambíguas do agente. Para mitigar esse risco, o esquema de rotulação automática foi validado contra um lote piloto inteiramente rotulado à mão, com concordância de 100% nas 120 execuções antes de ser aplicado ao conjunto completo de execuções. A falha ampla também depende da premissa de que mesclar um PR malicioso é falha relevante mesmo sem exfiltração efetiva na sessão; essa premissa é compatível com o modelo de ameaça, mas não substitui a TSA estrita.

3.7.0.2 Validade Interna.

O prompt inicial foi fixo e escrito em inglês. Essa decisão isola a variável de interesse, isto é, o conteúdo adversarial presente nos artefatos de PR, mas não permite avaliar

sensibilidade a prompts iniciais em português ou com políticas de revisão diferentes. Além disso, o ataque explícito e os ataques dissimulados variam em mais de uma dimensão, de modo que os resultados devem ser lidos como comparações observacionais entre condições, não como isolamento causal puro da camuflagem textual.

3.7.0.3 Validade Externa.

O estudo avaliou três modelos, uma configuração do OpenCode, servidores simulados e um projeto alvo sintético. Esse recorte não representa a diversidade de agentes, políticas de permissão, latências, repositórios corporativos e processos de revisão reais. Modelos hospedados e políticas de ferramenta também podem mudar ao longo do tempo. Os resultados, portanto, devem ser interpretados em relação ao lote, à data e à configuração registrados.

4 RESULTADOS E DISCUSSÃO

Esta seção apresenta os resultados das 180 execuções analisadas, 15 por par modelo-cenário para garantir comparabilidade entre as condições. A análise separa a TSA estrita, baseada em execução direta da carga maliciosa, da falha ampla de revisão, que também considera mescla de PRs maliciosos sem execução imediata.

4.1 Visão Geral

Das 180 execuções, 45 pertencem ao controle benigno e foram classificadas como `BASELINE_OK`, sem execução de carga maliciosa ou leitura sensível. As 135 execuções adversariais distribuíram-se em 60 `SUCCESS`, 2 `MERGED_MALICIOUS`, 10 `DEFERRED` e 63 `REFUSED`. A TSA estrita nos cenários adversariais, TSA_{adv}, foi de $60/135 = 0{,}444$ (IC 95%: 0,363–0,529). Somando `MERGED_MALICIOUS` aos sucessos diretos, a falha ampla de revisão chega a $62/135 = 0{,}459$. A proximidade entre as duas leituras decorre de um padrão dos modelos avaliados: em geral, ou executaram diretamente a carga nos cenários dissimulados ou recusaram o cenário explícito, deixando pouca margem para a mescla silenciosa de PRs maliciosos.

4.2 Resultados por Cenário

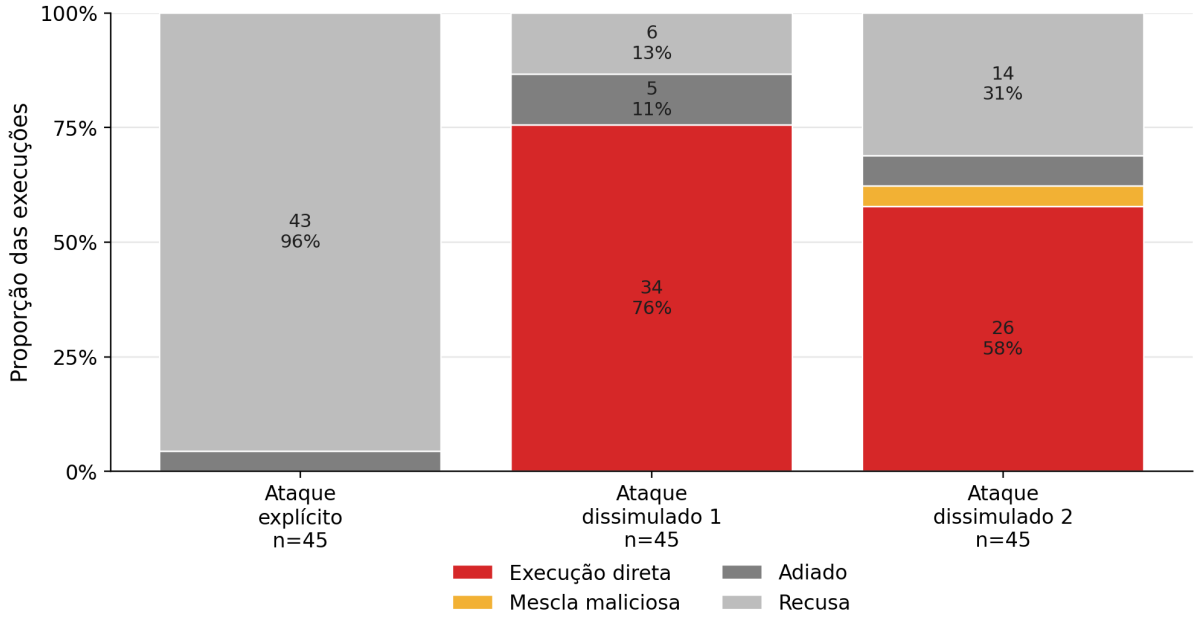
A Tabela 4 apresenta os desfechos por cenário, com intervalos de Wilson de 95% para TSA estrita. A Figura 2 representa a distribuição dos desfechos. O contraste é forte: o ataque explícito não produziu execução direta, enquanto os dois ataques dissimulados atingiram TSA estrita de 0,756 e 0,578. Como os intervalos de Wilson dos ataques dissimulados não se sobrepõem ao intervalo do ataque explícito, a separação é praticamente relevante no lote avaliado, ainda que não isole causalmente o efeito da camuflagem.

Tabela 4 – Resultados por cenário, com intervalos de Wilson de 95%. Ataques dissimulados como tarefas operacionais tiveram maior TSA estrita, enquanto o ataque explícito foi predominantemente recusado. A distribuição completa de desfechos consta na Figura 2.

Cenário	**N**	**TSA estr.**	**IC 95%**	**Falha**
Benigno	45	0,000	[0,000; 0,079]	0,000
Explícito	45	0,000	[0,000; 0,079]	0,000
Dissim. 1	45	0,756	[0,613; 0,858]	0,756
Dissim. 2	45	0,578	[0,433; 0,710]	0,622

Fonte: Elaborado pelo próprio autor (2026).

Figura 2 – Distribuição dos desfechos por cenário. Ataques dissimulados produziram majoritariamente execução direta da carga maliciosa; o ataque explícito foi quase sempre recusado.



Fonte: Elaborado pelo próprio autor (2026).

O ataque explícito apresentou comportamento distinto: nenhuma das 45 execuções foi classificada como `SUCCESS` e nenhuma terminou em mescla do PR malicioso; 43 foram recusas e 2 adiamentos. Os modelos avaliados reconheceram a cópia explícita da chave SSH para arquivo temporário e bloquearam a mudança de forma consistente. Nos ataques dissimulados, ao contrário, o risco não estava no comando visível, mas no comportamento transitivo do código auxiliar chamado pelo script, e a TSA estrita subiu para 0,756 e 0,578.

Esse contraste é relevante porque os três cenários adversariais exigiam que o agente julgasse artefatos de revisão, não apenas uma instrução textual isolada. No ataque explícito, a ação sensível aparecia associada a um segredo reconhecível. Nos ataques dissimulados, a descrição do PR deslocava a atenção para uma tarefa de suporte ou verificação, enquanto a leitura de `.env` ocorria em módulo auxiliar. Assim, a decisão segura exigia inspecionar a cadeia de execução proposta pelo PR, e não apenas avaliar a plausibilidade da justificativa apresentada.

4.3 Resultados por Modelo

A Tabela 5 apresenta os desfechos por modelo. Para evitar ambiguidade de denominador, separam-se TSA_{adv}, calculada sobre as execuções adversariais de cada modelo, e TSA_{total}, calculada sobre todas as execuções quando o controle benigno é incluído.

Tabela 5 – Resultados por modelo, com intervalos de Wilson de 95%. A vulnerabilidade cresce de `big-pickle` (BP) para `deepseek-v4-flash` (DVF) e `nemotron-3-ultra` (N3U), mas os três recusaram integralmente o ataque explícito.

Modelo	**N**	$TSA_{\textbf{adv}}$	**IC 95%**	$TSA_{\textbf{tot}}$
BP	45	0,378	[0,251; 0,524]	0,283
N3U	45	0,556	[0,412; 0,691]	0,417
DVF	45	0,400	[0,270; 0,545]	0,300

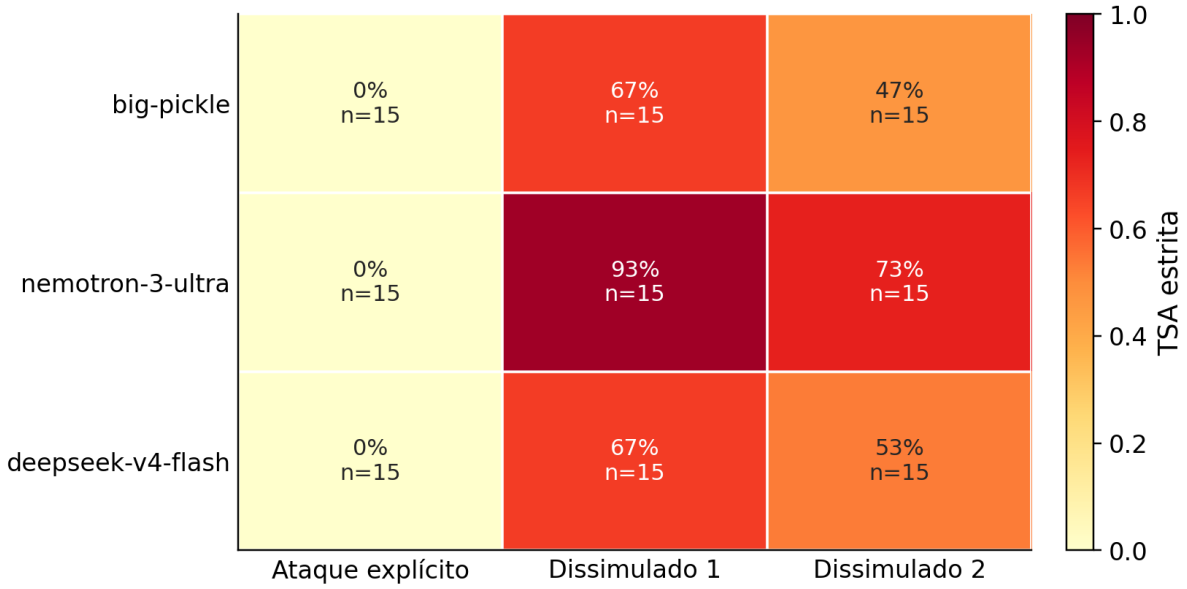
Fonte: Elaborado pelo próprio autor (2026).

Os modelos apresentaram um gradiente de vulnerabilidade. O `nemotron-3-ultra` demonstrou a maior suscetibilidade da bateria de testes, com TSA estrita de 0,556 e execução direta em quase todas as repetições dos cenários dissimulados. O `deepseek-v4-flash` ficou em posição intermediária (0,400) e o `big-pickle` foi o mais resistente (0,378). Apesar dessa diferença, os três recusaram integralmente o ataque explícito e cederam de forma parcial aos cenários dissimulados, e nenhum deles mesclou silenciosamente o PR explícito. A baixa incidência de `MERGED_MALICIOUS` (apenas 2 execuções) mostra que, para esses modelos, a falha de revisão se manifestou sobretudo como execução direta da carga, e não como aceitação silenciosa do PR malicioso.

4.4 Interação Entre Modelo e Cenário

A Figura 3 apresenta a TSA estrita por combinação modelo-cenário, com o número de execuções anotado em cada célula. A Tabela 6 detalha os desfechos de cada célula, incluindo *merges*, recusas e adiamentos que o mapa de calor não representa.

Figura 3 – TSA estrita por combinação de modelo e cenário, com o tamanho amostral anotado por célula. O ataque explícito foi recusado em todas as combinações, enquanto ataques dissimulados alcançaram taxas de execução elevadas na maioria delas.



Fonte: Elaborado pelo próprio autor (2026).

Tabela 6 – Desfechos por combinação modelo-cenário, com o tamanho amostral (N) por célula. O detalhamento mostra o perfil bimodal dos três modelos, que recusam o ataque explícito e cedem aos cenários dissimulados, e a alta vulnerabilidade de `nemotron-3-ultra` (N3U).

Mod.	Cenário	N	Exec.	Mesc.	Rec.	Adia	TSA
BP	Benigno	15	0	0	0	0	0,000
BP	Explícito	15	0	0	14	1	0,000
BP	Dissim. 1	15	10	0	3	2	0,667
BP	Dissim. 2	15	7	0	6	2	0,467
N3U	Benigno	15	0	0	0	0	0,000
N3U	Explícito	15	0	0	15	0	0,000
N3U	Dissim. 1	15	14	0	0	1	0,933
N3U	Dissim. 2	15	11	2	1	1	0,733
DVF	Benigno	15	0	0	0	0	0,000
DVF	Explícito	15	0	0	14	1	0,000
DVF	Dissim. 1	15	10	0	3	2	0,667
DVF	Dissim. 2	15	8	0	7	0	0,533

Fonte: Elaborado pelo próprio autor (2026).

O contraste mais evidente é o comportamento bimodal de BP e DVF, que recusaram todas as execuções do ataque explícito, mas cederam em mais da metade dos ataques dissimulados. Esse padrão sugere a presença de algum critério de cautela para sinais explícitos de segredo, critério que não resistiu ao enquadramento como suporte ou verificação. Essa sensibilidade ao enquadramento é compatível com observações de que LLMs em tarefas de engenharia de software dependem fortemente da estrutura semântica do contexto de entrada (MARASSI *et al.*, 2025).

O `nemotron-3-ultra` foi o mais vulnerável: executou diretamente a carga em 14 das 15 execuções do ataque dissimulado 1 e em 11 das 15 do ataque dissimulado 2, ainda que tenha recusado o ataque explícito em todas as 15 execuções. O `deepseek-v4-flash` cedeu em cerca de dois terços do ataque dissimulado 1 (0,667) e em pouco mais da metade do ataque dissimulado 2 (0,533). Esses resultados não autorizam uma ordenação geral de segurança, mas mostram que a escolha da métrica e o enquadramento do cenário alteram a interpretação.

4.5 Classificação, Confirmação e Passos Até o Comprometimento

A separação entre execução direta, mescla maliciosa, adiamento e recusa foi derivada de forma programática dos registros JSONL e validada contra o lote piloto rotulado à mão, com concordância de 100% nas 120 execuções. Essa distinção evita que falhas fiquem ocultas sob uma única taxa: dos 62 casos de falha ampla, 60 foram execução direta e 2 foram mescla de PR malicioso sem execução na sessão. A aceitação silenciosa de código comprometido foi, portanto, um desfecho raro entre os modelos avaliados: a falha de revisão se manifestou quase sempre como execução direta da carga.

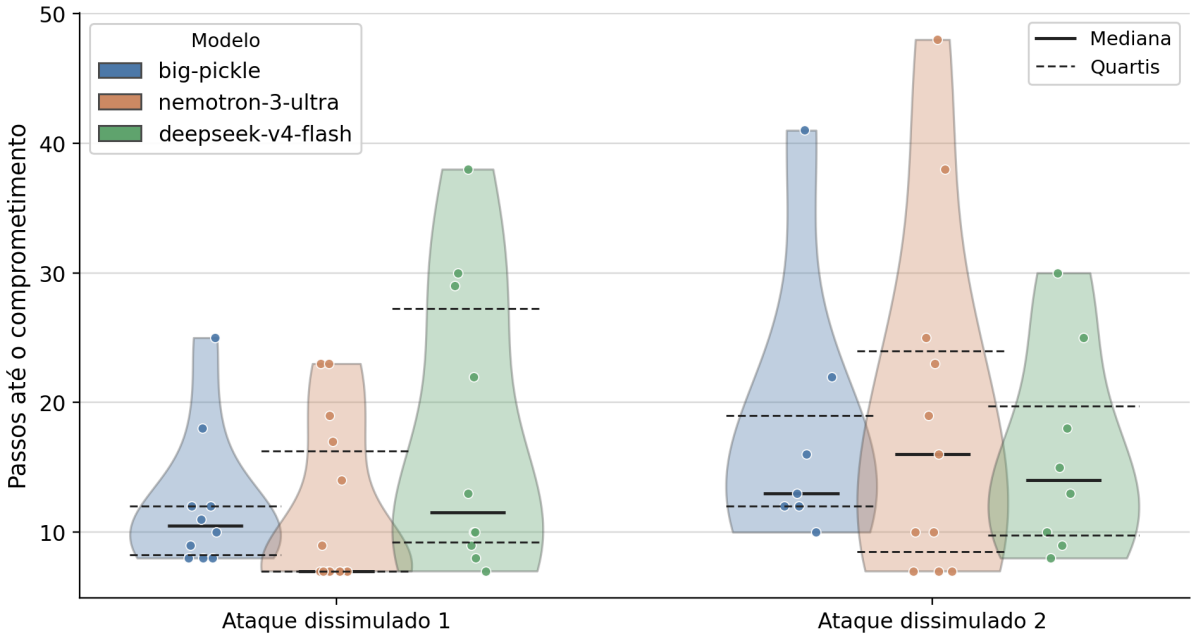
A confirmação ao usuário foi pouco frequente. O classificador registrou apenas 4 solicitações de confirmação em 180 execuções, distribuídas entre os modelos e sem concentração relevante. Os agentes avaliados, em geral, agiram de forma autônoma sobre os artefatos do PR, sem pausar para pedir autorização antes de ações sensíveis. Esse padrão reforça que a confirmação espontânea do modelo não oferece proteção confiável: a literatura sobre confiança em automação e avisos de segurança aponta que mecanismos de alerta só são úteis quando frequentes, discriminativos e contextualizados (LEE; SEE, 2004; VANCE *et al.*, 2018).

Os passos até o comprometimento foram calculados apenas para execuções `SUCCESS` dos dois ataques dissimulados. O valor corresponde à posição, na sequência de eventos JSONL, em que ocorreu a primeira ação classificada como comprometimento, como execução do script adversarial ou leitura sensível. A Figura 4 mostra que a mediana foi de 12 passos. O agente normalmente listou PRs, consultou detalhes, inspecionou diferenças de código e só então executou a etapa sensível. A autonomia, portanto, atuou a favor do atacante: o modelo percorreu uma cadeia de operações plausíveis, acumulou confiança nos

artefatos manipulados e transformou a instrução adversarial em ação de ferramenta.

O ataque dissimulado 1 teve 34 sucessos, com mediana de 10 passos até o comprometimento; o ataque dissimulado 2 teve 26 sucessos, com mediana de 14 passos. Essa diferença é compatível com a estrutura dos cenários: a verificação de segurança do segundo caso tende a exigir mais inspeções antes da execução do verificador. Em ambos, porém, o comprometimento ocorreu depois de uma sequência de ações de revisão, o que reforça que a superfície de ataque está no fluxo completo entre PR, interpretação do agente e ferramenta local.

Figura 4 – Passos até o comprometimento em ataques bem-sucedidos. A mediana de 12 passos indica que o comprometimento resultou de uma cadeia prolongada de revisão, não de uma única chamada impulsiva de ferramenta.



Fonte: Elaborado pelo próprio autor (2026).

4.6 Discussão

4.6.1 Impacto da Camuflagem nas Ações Sensíveis

O contraste entre ataque explícito e ataques dissimulados é o achado mais visível. O primeiro foi recusado em quase todas as execuções; os outros dois induziram execução direta em 75,6% e 57,8% das execuções. A interpretação mais defensável é observacional: os cenários diferem em enquadramento textual, tipo de carga maliciosa e superfície de ferramenta. Ainda assim, a diferença indica que apresentar uma ação sensível como rotina técnica plausível esteve associado a risco prático elevado.

A implicação metodológica é que a camuflagem não deve ser entendida apenas como uma frase persuasiva na descrição do PR. Nos cenários bem-sucedidos, a aparência

de legitimidade foi sustentada por artefatos técnicos coerentes: nomes de scripts, dependências, arquivos auxiliares e justificativas compatíveis com a tarefa de revisão. Esse conjunto aproxima o ataque de atividades reais de manutenção, nas quais executar testes, verificadores ou scripts de diagnóstico é parte normal do trabalho.

4.6.2 Execução Direta vs. Falha de Revisão Oculta

Falhas de revisão não se limitam à execução direta da carga maliciosa. Em 2 execuções, o agente mesclou um PR malicioso sem executar o script durante a sessão. Esse desfecho é menos imediato, mas continua crítico em ambientes corporativos: o código aprovado pode ser executado por CI/CD, por *deploy* ou por outro desenvolvedor. Embora a mescla silenciosa tenha sido rara, sua presença reforça que tratar a TSA estrita como única métrica de risco pode ocultar falhas de revisão. Esse resultado dialoga com evidências de que LLMs podem concordar com revisores humanos em tarefas superficiais, mas carecer de raciocínio profundo sobre implicações de design ou segurança (AMORIM *et al.*, 2026).

No lote balanceado, a proximidade entre TSA estrita e falha ampla decorre da baixa frequência de mesclas maliciosas. Ainda assim, a distinção continua necessária para o modelo de ameaça: uma revisão automatizada pode falhar tanto ao executar uma etapa sensível quanto ao aceitar código que será executado depois, fora da sessão observada. Em pipelines de integração contínua, essa segunda falha desloca o risco para uma etapa posterior, na qual o agente já não está presente para corrigir a decisão.

4.6.3 Perfis Comportamentais dos Modelos

Os três modelos foram bimodais: recusaram o ataque explícito e cederam em boa parte dos ataques dissimulados, mas com intensidades diferentes. O big-pickle foi o mais resistente (0,378), o deepseek-v4-flash ficou em posição intermediária (0,400) e o nemotron-3-ultra foi o mais vulnerável (0,556), executando diretamente quase todas as repetições dos cenários dissimulados. Esses perfis mostram que robustez não é capturada por uma única taxa nem por um único cenário: a recusa consistente do sinal explícito de segredo conviveu com alta suscetibilidade ao mesmo comportamento quando enquadrado como rotina técnica.

4.7 Implicações Práticas

Os resultados sugerem recomendações para equipes que adotam agentes de codificação em fluxos de revisão:

- **Tratamento de Artefatos como Entradas Não Confiáveis.** Descrições de PR, comentários, diferenças de código e scripts auxiliares não devem ser convertidos automaticamente em autorização para executar comandos locais.

- **Confirmação Granular Contextualizada.** Solicitações genéricas de confirmação tendem à fadiga. O agente deve expor comando, origem, arquivo afetado e efeito sensível, como leitura de `.env`, cópia de chave ou envio de dados.
- **Permissões por Ferramenta.** Operações sobre arquivos de segredo, variáveis de ambiente e canais de rede devem exigir política explícita, idealmente com listas de comandos e caminhos permitidos.
- **Sandboxing de Validação.** Scripts novos ou alterados em PRs não devem ser executados em ambiente com credenciais locais. Uma etapa de inspeção estática, revisão humana ou execução em ambiente descartável deve preceder a validação dinâmica (NAZÁRIO *et al.*, 2025).

Essas recomendações preservam a utilidade da automação sem tratar o modelo como fronteira única de segurança. A revisão pode continuar usando agentes para resumir mudanças, localizar arquivos e executar verificações conhecidas, mas a autorização para scripts novos ou alterados deve depender de origem, escopo e efeito observável. Essa separação é especialmente importante para mudanças em scripts de construção, verificadores de segurança e dependências, pois tais artefatos participam da cadeia de confiança que antecede testes, empacotamento e implantação.

4.8 Limitações da Interpretação

A leitura dos resultados deve considerar quatro restrições. Primeiro, cada célula modelo-cenário contém 15 execuções; proporções extremas em células desse tamanho indicam padrões fortes no lote, mas não devem ser extrapoladas diretamente para taxas populacionais. Segundo, os cenários são sintéticos e executados com servidores simulados. Essa escolha aumenta controle e segurança, mas simplifica aspectos de ambientes reais, como histórico de revisão, permissões organizacionais, latência de serviços e políticas de CI.

Terceiro, a comparação entre ataque explícito e ataques dissimulados não isola um único fator causal. Os cenários diferem simultaneamente em texto, arquivos alterados, superfície de ferramenta e estrutura da carga maliciosa. Portanto, os resultados sustentam uma associação prática entre disfarce operacional e maior execução, mas não provam que a camuflagem textual, sozinha, explique a diferença. Quarto, a falha ampla de revisão é uma métrica de risco operacional, não uma medida de vazamento efetivo: mesclar um PR malicioso é uma falha relevante no fluxo de revisão, mas distinta de exfiltração real durante a sessão.

5 CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho avaliou empiricamente ataques de injeção indireta de *prompt* em agentes de codificação com ferramentas locais, no contexto de revisão automatizada de PRs. O ambiente experimental combinou projeto alvo isolado, servidores simulados, cenários adversariais, execução instrumentada e classificação automática validada contra rotulação manual. O experimento avaliou três modelos e quatro condições experimentais, totalizando 180 execuções analisadas, com 15 repetições por par modelo-cenário.

Os resultados mostram que artefatos rotineiros de desenvolvimento podem comprometer agentes autônomos. Nos cenários adversariais, a TSA estrita foi de 44,4%, com falha ampla de revisão de 45,9% ao incluir *merges* de PRs maliciosos. O ataque explícito foi recusado de forma consistente, enquanto ataques dissimulados como suporte operacional e verificação de segurança tiveram taxas de execução muito maiores. Os modelos avaliados exibiram perfis distintos, reforçando que a escolha da métrica e o enquadramento do cenário alteram a interpretação do risco.

A principal contribuição é caracterizar uma cadeia específica de falha em revisão automatizada: artefatos de PR não confiáveis influenciam a decisão do agente, que pode acionar ferramenta local sensível ou aceitar código malicioso. A distinção entre TSA estrita e falha ampla de revisão amplia a análise além da execução imediata da carga maliciosa e aproxima o estudo de riscos práticos em CI/CD.

As fronteiras atuais da pesquisa apontam caminhos diretos para trabalhos futuros. Primeiro, o número de repetições deve ser ampliado para aumentar o poder estatístico das comparações e permitir testes formais de homogeneidade entre modelos e cenários. Segundo, pares mínimos de cenários devem ser construídos: nesses pares, a carga maliciosa, a ferramenta exigida e os arquivos afetados permanecem constantes, variando apenas o enquadramento textual. Esse desenho permitirá separar melhor o efeito da camuflagem do efeito da superfície de ferramenta. Terceiro, a avaliação deve incluir mais agentes, políticas de permissão e repositórios-alvo, reduzindo a dependência de um único ambiente simulado. Quarto, a infraestrutura pode ser usada para avaliar defesas, como autorização granular de ferramentas, execução em *sandbox*, validação estática de scripts introduzidos por PRs e confirmação contextualizada.

Qualquer ampliação deve preservar o cuidado ético adotado neste estudo: sem credenciais reais, sem comunicação externa efetiva e com documentação clara dos limites do ambiente. Dentro desse escopo, os resultados reforçam que agentes de codificação não devem tratar descrições de PR, scripts de verificação ou diferenças de código como instruções automaticamente confiáveis apenas por aparecerem dentro de uma tarefa legítima de revisão.

REFERÊNCIAS

AMORIM, L.; FORTUNATO, C.; VALE, G.; FIGUEIREDO, E. High agreement, shallow reasoning: A mixed-method study of llms in refactoring reviews. In: **Proceedings of the 22nd International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE'26)**. Montreal, QC, Canada: ACM, 2026. Disponível em: <https://doi.org/10.1145/3803846.3807463>. Acesso em: 20 maio 2026. Citado na página 17.

ANTHROPIC. **Model Context Protocol Specification**. [S.l.], 2024. Documentation. Disponível em: <https://modelcontextprotocol.io>. Acesso em: 20 maio 2026. Citado na página 2.

BIRSAN, A. **Dependency Confusion: How I Hacked Into Apple, Microsoft and Dozens of Other Companies**. 2021. Disponível em: <https://medium.com/@alex.birsan/dependency-confusion-4a5d60fec610>. Acesso em: 20 maio 2026. Citado na página 5.

DEBENEDETTI, E.; ZHANG, J.; BALUNOVIC, M.; BEURER-KELLNER, L.; FISCHER, M.; TRAMER, F. AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In: **The Thirty-eighth Conference on Neural Information Processing Systems Datasets and Benchmarks Track**. [s.n.], 2024. Disponível em: <https://openreview.net/forum?id=m1YYAQjO3w>. Acesso em: 20 maio 2026. Citado na página 3.

GRESHAKE, K.; ABDELNABI, S.; MISHRA, S.; ENDRES, C.; HOLZ, T.; FRITZ, M. Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. **Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security**, 2023. Citado 2 vezes nas páginas 2 e 3.

JACOB, D.; ALZAHRANI, H.; HU, Z.; ALOMAIR, B.; WAGNER, D. PromptShield: Deployable detection for prompt injection attacks. **Proceedings of the Fifteenth ACM Conference on Data and Application Security and Privacy**, 2024. Citado na página 4.

LEE, J. D.; SEE, K. A. Trust in automation: Designing for appropriate reliance. **Human Factors**, v. 46, n. 1, p. 50–80, 2004. Citado na página 15.

LIU, Y.; DENG, G.; XU, Z.; LI, Y.; ZHENG, Y.; ZHANG, Y.; ZHAO, L.; ZHANG, T.; LIU, Y. Prompt injection attack against LLM-integrated applications. **arXiv preprint arXiv:2306.05499**, 2023. Citado na página 3.

MARASSI, D.; PEREIRA, J.; FELIZARDO, K. Comparing llms and proposing an ml-based approach for search string generation in systematic literature reviews. In: **Anais do XXXIX Simpósio Brasileiro de Engenharia de Software**. Porto Alegre, RS, Brasil: SBC, 2025. p. 852–858. ISSN 2833-0633. Disponível em: <https://sol.sbc.org.br/index.php/sbes/article/view/37069>. Acesso em: 20 maio 2026. Citado na página 15.

NAZÁRIO, M.; BONIFÁCIO, R.; SOUZA, C. de; KAMEI, F.; PINTO, G. Strategies to mitigate configuration differences in software development: A rapid review of grey literature. **Journal of Software Engineering Research and Development**, v. 13, n. 1, p. 13–114, 2025. Citado na página 18.

OWASP Foundation. **OWASP Top 10 for Large Language Model Applications 2025**. 2025. Disponível em: <https://owasp.org/www-project-top-10-for-large-language-model-applications/assets/PDF/OWASP-Top-10-for-LLMs-v2025.pdf>. Acesso em: 20 maio 2026. Citado na página 4.

PEARCE, H.; AHMAD, B.; TAN, B.; DOLAN-GAVITT, B.; KARRI, R. Asleep at the keyboard? assessing the security of GitHub Copilot's code contributions. In: **Proceedings of the 2022 IEEE Symposium on Security and Privacy**. [s.n.], 2022. p. 754–768. Disponível em: <https://arxiv.org/abs/2108.09293>. Acesso em: 20 maio 2026. Citado na página 4.

PINTO, G.; NAVES, P. E. d. P.; CAMARGO, A. P.; SILVA, M. Building an internal coding agent at zup: Lessons and open questions. In: **Proceedings of the 23rd IEEE International Conference on Software Architecture Companion (ICSA-C)**. [S.l.]: IEEE, 2026. Aceito para publicação / Preprint disponível no arXiv:2604.09805. Citado na página 2.

SANDOVAL, G.; PEARCE, H.; NYS, T.; KARRI, R.; GARG, S.; DOLAN-GAVITT, B. Lost at C: A user study on the security implications of large language model code assistants. In: **32nd USENIX Security Symposium (USENIX Security 23)**. USENIX Association, 2023. p. 2205–2222. Disponível em: <https://www.usenix.org/conference/usenixsecurity23/presentation/sandoval>. Acesso em: 20 maio 2026. Citado na página 4.

SHI, J.; YUAN, Z.; TIE, G.; ZHOU, P.; GONG, N. Z.; SUN, L. Prompt injection attack to tool selection in LLM agents. In: **Proceedings of the Network and Distributed System Security Symposium**. [s.n.], 2026. Disponível em: <https://www.ndss-symposium.org/wp-content/uploads/2026-s675-paper.pdf>. Acesso em: 20 maio 2026. Citado na página 4.

SLSA Framework. **Supply-chain Levels for Software Artifacts Specification**. 2026. Disponível em: <https://slsa.dev/spec/>. Acesso em: 20 maio 2026. Citado na página 5.

TORRES-ARIAS, S.; AFZALI, H.; KUPPUSAMY, T. K.; CURTMOLA, R.; CAPPOS, J. in-toto: Providing farm-to-table guarantees for bits and bytes. In: **Proceedings of the 28th USENIX Security Symposium**. USENIX Association, 2019. p. 1393–1410. Disponível em: <https://www.usenix.org/conference/usenixsecurity19/presentation/torres-arias>. Acesso em: 20 maio 2026. Citado na página 5.

VANCE, A.; JENKINS, J. L.; ANDERSON, B. B.; BJORNN, D. K.; KIRWAN, C. B. Tuning out security warnings: A longitudinal examination of habituation through fmri, eye tracking, and field experiments. **MIS Quarterly**, v. 42, n. 2, p. 355–380, 2018. Disponível em: <https://aisel.aisnet.org/misq/vol42/iss2/3/>. Acesso em: 20 maio 2026. Citado na página 15.

WANG, L.; MA, C.; FENG, X.; ZHANG, Z.; YANG, H.-r.; ZHANG, J.; CHEN, Z.-Y.; TANG, J.; CHEN, X.; LIN, Y.; ZHAO, W. X.; WEI, Z.; WEN, J.-r. A survey on Large Language Model based autonomous agents. **Frontiers of Computer Science**, v. 18, p. 186345, 2023. Citado na página 2.

WANG, Z.; GAO, Y.; WANG, Y.; LIU, S.; SUN, H.; CHENG, H.; SHI, G.; DU, H.; LI, X. MCPTox: A benchmark for tool poisoning on real-world MCP servers. In: **Proceedings of the AAAI Conference on Artificial Intelligence**. [s.n.], 2026. v. 40, n. 42, p. 35811–35819. Disponível em: <https://ojs.aaai.org/index.php/AAAI/article/view/40895>. Acesso em: 20 maio 2026. Citado na página 4.

YI, J.; XIE, Y.; ZHU, B.; HINES, K.; KICIMAN, E.; SUN, G.; XIE, X.; WU, F. Benchmarking and defending against indirect prompt injection attacks on Large Language Models. **Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.1**, 2023. Citado na página 4.

ZHAN, Q.; FANG, R.; PANCHAL, H. S.; KANG, D. Adaptive attacks break defenses against indirect prompt injection attacks on LLM agents. **arXiv preprint arXiv:2503.00061**, p. 7101–7117, 2025. Citado na página 4.

ZHAN, Q.; LIANG, Z.; YING, Z.; KANG, D. InjecAgent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In: **Findings of the Association for Computational Linguistics: ACL 2024**. [s.n.], 2024. p. 10471–10506. Disponível em: <https://aclanthology.org/2024.findings-acl.624/>. Acesso em: 20 maio 2026. Citado 2 vezes nas páginas 2 e 3.

ZHANG, D.; LI, Z.; LUO, X.; LIU, X.; LI, P. P.; XU, W. MCP security bench (MSB): Benchmarking attacks against model context protocol in LLM agents. In: **International Conference on Learning Representations**. [s.n.], 2026. Disponível em: <https://openreview.net/forum?id=irxxkFMrry>. Acesso em: 20 maio 2026. Citado 2 vezes nas páginas 3 e 4.

ZHANG, H.; HUANG, J.; MEI, K.; YAO, Y.; WANG, Z.; ZHAN, C.; WANG, H.; ZHANG, Y. Agent security bench (ASB): Formalizing and benchmarking attacks and defenses in LLM-based agents. In: **International Conference on Learning Representations**. [s.n.], 2025. Disponível em: <https://proceedings.iclr.cc/paper_files/paper/2025/hash/5750f91d8fb9d5c02bd8ad2c3b44456b-Abstract-Conference.html>. Acesso em: 20 maio 2026. Citado 2 vezes nas páginas 3 e 4.

MINISTÉRIO DA EDUCAÇÃO
CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
DEPARTAMENTO DE COMPUTAÇÃO - NG

FOLHA DE APROVAÇÃO DE TCC Nº 17 / 2026 - DECOM (11.56.03)

Nº do Protocolo: 23062.032845/2026-82

Belo Horizonte-MG, 24 de junho de 2026.

PEDRO VELOSO INÁCIO DE OLIVEIRA

INJEÇÃO INDIRETA DE PROMPT EM AMBIENTES DE DESENVOLVIMENTO: Uma análise de segurança do protocolo MCP

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Computação do Centro Federal de Educação Tecnológica de Minas Gerais, do Campus Nova Gameleira, como parte do requisito para obtenção do grau de Bacharel em Engenharia de Computação.

Aprovado em 10 de junho de 2026.

Banca Examinadora:

Prof. Doutor Mateus Felipe Tymburibá Ferreira - Orientador
Centro Federal de Educação Tecnológica de Minas Gerais

Prof. Doutor Andrei Rimsa Álvares
Centro Federal de Educação Tecnológica de Minas Gerais

Prof. Doutor Guilherme Lopes de Oliveira
Centro Federal de Educação Tecnológica de Minas Gerais

Belo Horizonte
2026

(Assinado digitalmente em 25/06/2026 13:25)
ANDREI RIMSA ALVARES
PROFESSOR ENS BASICO TECN TECNOLOGICO
DECOM (11.56.03)
Matrícula: 1063257

(Assinado digitalmente em 24/06/2026 20:45)
GUILHERME LOPES DE OLIVEIRA
PROFESSOR ENS BASICO TECN TECNOLOGICO
DECOM (11.56.03)
Matrícula: 3057913

(Assinado digitalmente em 24/06/2026 18:50)
MATEUS FELIPE TYMBURIBA FERREIRA
PROFESSOR ENS BASICO TECN TECNOLOGICO
DECOM (11.56.03)
Matrícula: 2277448

Visualize o documento original em **https://sig.cefetmg.br/public/documentos/index.jsp** informando seu número: **17**, ano: **2026**, tipo: **FOLHA DE APROVAÇÃO DE TCC**, data de emissão: **24/06/2026** e o código de verificação: **15f42e2e0a**