Revista Ibero-Americana de Humanidades, Ciências e Educação — REASE

OPEN ACCESS

doi.org/10.51891/rease.v12i11.31592

PARADIGMAS DE GERAÇÃO DE CÓDIGO POR INTELIGÊNCIA ARTIFICIAL, DO PROMPT ÚNICO AO MULTIAGENTE: UM ESTUDO EXPERIMENTAL COM IMPLICAÇÕES PARA A EDUCAÇÃO EM ENGENHARIA DE SOFTWARE

AI CODE-GENERATION PARADIGMS, FROM SINGLE PROMPT TO MULTI-AGENT: AN EXPERIMENTAL STUDY WITH IMPLICATIONS FOR SOFTWARE ENGINEERING EDUCATION

Pedro Vitor Chastalo Santos[1]
Thiago França Naves[2]

RESUMO: A engenharia de software incorporou, entre 2024 e 2026, assistentes de inteligência artificial capazes de construir aplicações inteiras de forma autônoma, transição que afeta a prática profissional e o ensino de computação e ainda carece de avaliação controlada. Este estudo de caso exploratório compara quatro paradigmas de geração de código por IA, dispostos como uma escada de orquestração: prompt único (P0), harness agêntico (P1), sistema multiagente (P2) e sistema multiagente com um agente dedicado de segurança (P3). Fixando modelo, amostragem e limites de parada, e variando apenas o paradigma, os quatro braços recebem a mesma especificação congelada e reconstroem, do zero, uma plataforma web full-stack de gestão de laboratório. A avaliação cobre fidelidade estrutural de esquema e de contrato, correção funcional por testes de aceitação de ponta a ponta, qualidade, segurança mapeada a categorias CWE e OWASP, e custo. Em dezesseis execuções iniciadas, quatro por paradigma, não foi detectada diferença de correção entre os três braços agênticos, com medianas de 34,5 a 35 critérios de aceitação aprovados sobre 48 (Kruskal-Wallis entre P1, P2 e P3 com valor-p de 0,73), ao passo que o prompt único foi frágil: de quatro execuções, uma gerou aplicação executável, duas geraram artefatos não executáveis e uma não produziu artefato avaliável. A correção não escalou com o esforço de geração e, dado o pequeno número de réplicas, toda a leitura é descritiva. A confiabilidade, e não a correção, distingue os paradigmas. Em dólares faturados, o harness de sessão única (P1) é o mais caro, por reprocessar grande volume de contexto cacheado. A segurança de aplicação é nula em todos os braços, sendo o único achado de alta severidade recorrente a execução do contêiner como root. Discute-se que pedir "código seguro" no enunciado não reduz por si só a vulnerabilidade, o que reposiciona o papel da verificação no ensino de engenharia de software.

Palavras-chave: Geração de código por inteligência artificial. Educação em computação. Engenharia de software. Sistemas multiagente. Segurança de software.

[1]Graduando no Curso em Bacharelado Ciência da Computação Universidade Tecnológica Federal do Paraná (UTFPR), Campus Santa Helena.
[2] Professor. Orientador. Doutor em Ciência da Computação, Universidade Tecnológica Federal do Paraná (UTFPR), Campus Santa Helena.

ABSTRACT: Between 2024 and 2026, software engineering absorbed artificial intelligence assistants able to build entire applications autonomously, a transition that reshapes professional practice and computing education and still lacks controlled evaluation. This exploratory case study compares four AI code-generation paradigms arranged as an orchestration ladder: single prompt (P0), agentic harness (P1), multi-agent system (P2), and a multi-agent system with a dedicated security agent (P3). Holding model, sampling and stopping limits fixed, and varying only the paradigm, the four arms receive the same frozen specification and rebuild, from scratch, a full-stack web platform for research-laboratory management. Evaluation covers structural faithfulness of schema and contract, functional correctness through end-to-end acceptance tests, quality, security mapped to CWE and OWASP categories, and cost. Across sixteen initiated runs, four per paradigm, no difference in correctness was detected among the three agentic arms, with medians from 34.5 to 35 acceptance criteria passed out of 48 (Kruskal-Wallis among P1, P2 and P3 with a p-value of 0.73), whereas the single prompt was fragile: of four initiated runs, one produced an executable application, two produced non-executable artifacts and one produced no assessable artifact. Correctness did not scale with generation effort and, given the small number of replicates, the entire reading is descriptive. Reliability, not correctness, separates the paradigms. In billed dollars the single-session harness (P1) is the most expensive, due to reprocessing a large volume of cached context. Application security is null across all arms, the only recurring high-severity finding being running the container as root. The study argues that asking for "secure code" in the prompt does not by itself reduce vulnerability, which repositions the role of verification in software engineering education.

Keywords: AI code generation. Computing education. Software engineering. Multi-agent systems. Software security.

RESUMEN: La ingeniería de software incorporó, entre 2024 y 2026, asistentes de inteligencia artificial capaces de construir aplicaciones enteras de forma autónoma, una transición que afecta la práctica profesional y la enseñanza de computación y aún carece de evaluación controlada. Este estudio de caso exploratorio compara cuatro paradigmas de generación de código por IA, dispuestos como una escalera de orquestación: prompt único (P0), harness agéntico (P1), sistema multiagente (P2) y sistema multiagente con un agente dedicado de seguridad (P3). Fijando modelo, muestreo y límites de parada, y variando solo el paradigma, los cuatro brazos reciben la misma especificación congelada y reconstruyen, desde cero, una plataforma web full-stack de gestión de laboratorio. La evaluación abarca fidelidad estructural de esquema y de contrato, corrección funcional mediante pruebas de aceptación de extremo a extremo, calidad, seguridad mapeada a categorías CWE y OWASP, y costo. En dieciséis ejecuciones iniciadas, cuatro por paradigma, no se detectó diferencia de corrección entre los tres brazos agénticos, con medianas de 34,5 a 35 criterios de aceptación aprobados sobre 48 (Kruskal-Wallis entre P1, P2 y P3 con valor-p de 0,73), mientras que el prompt único fue frágil: de cuatro ejecuciones, una generó una aplicación ejecutable, dos generaron artefactos no ejecutables y una no produjo artefacto evaluable. La corrección no escaló con el esfuerzo de generación y, dado el pequeño número de réplicas, toda la lectura es descriptiva. La confiabilidad, y no la corrección, distingue los paradigmas. En dólares facturados, el harness de sesión única (P1) es el más caro, por reprocesar gran volumen de contexto en caché. La seguridad de aplicación es nula en todos los brazos, siendo el único hallazgo de alta severidad recurrente la ejecución del contenedor como root. Se discute que pedir "código seguro" en el enunciado no reduce por sí

solo la vulnerabilidad, lo que reposiciona el papel de la verificación en la enseñanza de la ingeniería de software.

Palabras clave: Generación de código por inteligencia artificial. Educación en computación. Ingeniería de software. Sistemas multiagente. Seguridad de software.

INTRODUÇÃO

Programar com auxílio de modelos de linguagem de grande porte deixou de ser autocompletar linhas; hoje é a delegação de aplicações inteiras a agentes que planejam, escrevem, executam e corrigem código de forma quase autônoma. Entre 2024 e 2026, a avaliação desses sistemas migrou de problemas isolados em nível de função, medidos por suítes como HumanEval (CHEN et al., 2021) e MBPP (AUSTIN et al., 2021), para a construção de artefatos completos por agentes que operam em repositórios reais (VATS; GOLEV, 2026; GORINOVA et al., 2026). A mudança altera o que significa saber programar e o que precisa ser ensinado nos cursos de computação, tema sinalizado desde os primeiros assistentes de geração de código (BECKER et al., 2023; PRATHER et al., 2023). Quando o texto que o estudante escreve é um enunciado em linguagem natural, e não o corpo de uma função, as competências centrais se deslocam: da escrita de sintaxe para a especificação de requisitos, da depuração linha a linha para a verificação de comportamento, e da confiança na própria autoria para a desconfiança metódica diante de código que parece correto sem sê-lo.

Esse deslocamento ocorre sem um retrato controlado de como os diferentes modos de usar a IA se comportam diante de uma mesma tarefa realista. A prática profissional já convive com um espectro que vai do uso ingênuo, em que se aceita a primeira resposta de um único prompt, a arranjos de múltiplos agentes especializados que dividem os papéis de uma equipe de engenharia. Para o ensino, importa saber se mais orquestração significa software melhor e mais seguro, ou se apenas encarece o processo; sem essa evidência, corre-se o risco de ensinar como boa prática o que é apenas moda de ferramenta.

A literatura recente, já sistematizada em revisões abrangentes (HOU et al., 2024), oferece pistas convergentes, porém fragmentadas. Trocar o aparato que envolve o modelo, o chamado harness ou scaffold, pode alterar o consumo de tokens por tarefa resolvida em cerca de quarenta vezes, enquanto a taxa de sucesso se move muito pouco, o que sugere custo dominado pela orquestração, não pelo modelo isolado (VATS; GOLEV, 2026). Trabalhos sobre projetos completos relatam correlação negativa forte entre interação e qualidade final (LU, P. et al., 2026), e estudos de arquitetura multiagente advertem que acrescentar papéis e camadas

pode não melhorar a correção e ainda inflar a complexidade do código (ASHRAFI, 2026). Na segurança, instruir o modelo a "ser seguro" no enunciado não reduz de modo significativo a taxa de vulnerabilidades, apenas altera a distribuição das categorias de falha (KHARMA et al., 2026), coerente com a constatação de que cerca de 40% dos programas gerados pelo GitHub Copilot, em cenários construídos para provocar vulnerabilidades, continham falhas (PEARCE et al., 2022), e com a de que desenvolvedores assistidos por IA produzem código menos seguro sentindo-se mais confiantes (PERRY et al., 2023). Falta um estudo que reúna essas peças sobre um único produto realista, construído do zero, comparando de forma controlada os modos de gerar e medindo, lado a lado, fidelidade, qualidade, segurança e custo.

Este trabalho ocupa essa lacuna e a conecta à educação em computação. O objetivo é comparar quatro paradigmas tratados como uma escada ordinal de orquestração: o prompt único (P_0), geração isolada em que se aceita a primeira resposta, sem execução nem correção; o harness agêntico (P_1), um único agente que planeja, age e observa em um laço de execução e testes; o sistema multiagente (P_2), que distribui papéis especializados de uma equipe de engenharia, como arquiteto, dados, backend, frontend e garantia de qualidade, com artefatos verificáveis encadeados; e o multiagente com segurança (P_3), idêntico ao anterior mais um papel dedicado que executa análise estática e exige remediação antes de concluir. Os quatro braços recebem a mesma especificação congelada e a mesma tarefa: reconstruir do zero uma plataforma web de gestão de um laboratório de pesquisa, com frontend, backend e banco de dados.

A pergunta de pesquisa é empírica: quão fielmente e com que qualidade, segurança e custo cada paradigma reproduz um produto especificado de antemão. A ela soma-se uma questão formativa que o desenho não mede diretamente, mas que a discussão examina por inferência: o que esses achados sugerem sobre as competências que a educação em engenharia de software precisa priorizar quando a escrita de código passa a ser mediada por agentes. A distinção é deliberada, pois o estudo não inclui estudantes, currículo ou desfecho de aprendizagem, e a leitura formativa é interpretativa, não experimental. A contribuição central é empírica, na interseção de engenharia de software e inteligência artificial, mas o artigo assume o recorte da educação em computação, por entender que a reconfiguração das competências profissionais provocada pela geração de código por IA é também um problema de currículo e de ensino (BECKER et al., 2023; DENNY et al., 2024); não se propõe componente curricular nem intervenção pedagógica, e sim evidência sobre a qual decisões de ensino podem

se apoiar. Dois esclarecimentos delimitam o alcance. O achado de que solicitar código seguro no enunciado não reduz a vulnerabilidade já está estabelecido na literatura (KHARMA et al., 2026; PEARCE et al., 2022; PERRY et al., 2023), adotado como motivação e hipótese a reexaminar no braço P3, não descoberta própria. E trata-se de um estudo de caso sobre um único domínio, a gestão de um laboratório, cuja generalização é discutida entre as ameaças à validade.

REVISÃO DA LITERATURA

A avaliação de código gerado por modelos de linguagem amadureceu depressa entre 2024 e 2026, deslocando o foco de funções isoladas para aplicações inteiras construídas por agentes em repositórios reais. Esta seção percorre três frentes, a fidelidade a uma especificação e o desenvolvimento full-stack, a qualidade e a segurança, e o efeito do aparato com os sistemas multiagente, e posiciona o trabalho nas lacunas restantes.

FIDELIDADE A ESPECIFICAÇÃO, APLICAÇÕES COMPLETAS E INTERFACE

O trabalho mais próximo em objeto mede a perda de fidelidade de implementação quando a especificação é revelada aos poucos ao agente, em vez de entregue de uma vez (YAN; CHEN; ZHANG, 2026). Reimplementando vinte artigos de aprendizado de máquina decompostos em componentes atômicos, os autores mostram que a fidelidade cai da condição de entrega única para a emergente, com o índice agregado do agente Claude Code recuando de 0,530 para 0,414 ($p = 0{,}0003$), e que a perda se concentra na integração entre módulos, o que motiva medir camadas conectadas. Aquele estudo reimplementa código de pesquisa em aparatos comerciais; aqui gera-se uma aplicação de negócio full-stack do zero, comparando quatro paradigmas e acrescentando qualidade, segurança e custo.

Na mesma linha, um benchmark de desenvolvimento de ponta a ponta reporta correlação negativa forte entre o consumo de tokens e a nota final ($\rho = -0{,}734$; $p = 0{,}0002$), evidência de que mais interação não implica melhor resultado (LU, P. et al., 2026); por usar problemas em estilo de juiz online, a diferenciação deste está em medir um produto real. Um sistema multiagente full-stack distingue a acurácia comum da acurácia com banco válido, contabilizada só quando os registros confirmam a persistência real, eliminando a aplicação de fachada (LU, Z. et al., 2026); em ablação, remover a ferramenta de depuração do backend derruba o desempenho da camada de 77,8% para 57,9%, indicando que o ganho vem do laço de depuração por camada, não da divisão de papéis, o que fundamenta a avaliação por camada e o

critério de banco válido aqui. Um benchmark de geração web a partir de requisitos reais observa ainda que os modelos vão melhor com requisitos vagos do que claros (LIU, C. et al., 2026), o que sugere que uma especificação estrita, como a deste estudo, tende a ser mais discriminante.

Um conjunto de benchmarks define o padrão de oráculo para aplicações web. Um avalia a criação de sites a partir de instrução em linguagem natural com um agente de navegação como oráculo funcional, com melhor acurácia de base em torno de 27,8%, restrita ao frontend (LU, Z. et al., 2025). Outro propõe cinquenta projetos com tarefas sequenciais dependentes e adota testes de ponta a ponta em Playwright como oráculo determinístico, com estado da arte de apenas 25,1% de aprovação na primeira tentativa (XU et al., 2025). Um terceiro renderiza o artefato em ambiente isolado, captura telas em três momentos e as submete a um juiz multimodal validado, cuja meta-avaliação alcança 90,95% de concordância com humanos e cai para 79,06% sem a evidência visual (ZHANG, C. et al., 2025). Desse trio deriva o núcleo do oráculo aqui: testes de ponta a ponta para a correção, capturas temporais com juiz multimodal para a fidelidade visual e a separação entre acurácia e aparência. A conversão de interface em código complementa o quadro: o trabalho fundacional de captura de tela para código alerta que a similaridade textual teve correlação negativa com o julgamento humano (SI et al., 2025); estudos posteriores mostram que o arcabouço de frontend afeta muito o desempenho (XIAO et al., 2025) e que a avaliação executável em ambiente controlado alcança alta concordância com humanos (ZHU et al., 2025). O mais alinhado a este estudo avalia a aplicação entregue a partir de uma especificação congelada e encontra um vão expressivo entre a taxa de aplicações apenas utilizáveis, de 76,9%, e a de excelentes, de apenas 20,2% (ZHANG, W. et al., 2026). Um trabalho sobre interatividade em aplicações científicas é citado apenas pela ideia de separar o que compila do que funciona quando o usuário interage, pois seus números têm conflito interno não resolvido (DAI et al., 2026).

QUALIDADE E SEGURANÇA DO CÓDIGO GERADO

Dois estudos revisados por pares ancoram a dimensão de segurança e dão ao trabalho seu principal fio formativo. O fundacional avaliou as sugestões do GitHub Copilot em 89 cenários voltados às vulnerabilidades mais comuns e encontrou falhas em cerca de 40% dos programas, por análise estática mapeada a categorias de fraqueza (PEARCE et al., 2022). O segundo, controlado com participantes humanos, observou que o código assistido por IA foi menos seguro em quatro das cinco tarefas, enquanto os participantes acreditavam ter escrito

código mais seguro, um caso de falsa confiança (PERRY et al., 2023). A esses junta-se a evidência de que a engenharia de prompt não reduz de modo significativo o total de vulnerabilidades, apenas altera a distribuição das categorias de falha (KHARMA et al., 2026), a justificativa mais forte para tratar a segurança por um agente que executa a ferramenta e força a remediação, não por uma instrução no enunciado. Na qualidade, um estudo mediu aumento médio da ordem de 63% em maus cheiros de código gerado por modelos em relação a soluções humanas de referência (PAUL; ZHU; BAYLEY, 2025), o que reforça medir manutenibilidade, não apenas correção.

O APARATO E OS SISTEMAS MULTIAGENTE

Três trabalhos recentes sustentam a decisão de tratar o paradigma de geração, e não o modelo isolado, como variável de primeira ordem. Um experimento fatorial mostra que trocar o aparato altera o consumo de tokens por tarefa resolvida em cerca de quarenta vezes, enquanto a taxa de sucesso varia muito pouco, ao passo que atualizar o modelo mexe mais no sucesso e quase nada nos tokens (VATS; GOLEV, 2026). Um artigo de posição argumenta que o agente em produção é um sistema, composto de tarefas, aparato, ambiente, contexto e retorno, não um modelo, e defende verificadores comportamentais em vez da comparação contra uma solução de referência única (GORINOVA et al., 2026). Um terceiro isola o efeito do aparato em larga escala e propõe um escore em que a segurança atua como fator multiplicativo, de modo que uma violação zera o resultado (YAO et al., 2026). Os três convergem para os controles aqui: fixar modelo e amostragem, equalizar os tetos de parada e reportar o custo por tarefa resolvida ao lado do sucesso.

Os arcabouços multiagente que inspiram os braços mais orquestrados codificam procedimentos operacionais padrão de equipes de software. Um organiza uma linha de montagem de cinco papéis, cada um produzindo um artefato verificável, o que reduz a cascata de alucinações (HONG et al., 2024); outro modela uma empresa de software virtual por uma cadeia de diálogos entre papéis e propõe uma métrica composta de qualidade (QIAN et al., 2024). Nenhum possui papel de segurança ou métricas de vulnerabilidade sobre o artefato, lacuna que o quarto paradigma ocupa. Transposições da orquestração multiagente para a pesquisa científica advertem contra o juiz automático ingênuo, que superestima a qualidade quando não é calibrado (SCHMIDGALL et al., 2025; LU, C. et al., 2024). Como ameaça honesta à hipótese de que mais orquestração é sempre melhor, um estudo pareado que variou seis

arquiteturas concluiu que acrescentar papéis e camadas não melhorou a correção e inflou a complexidade do código entre 50% e 130% (ASHRAFI, 2026), o que motiva medir complexidade, fidelidade e segurança, não apenas o sucesso funcional.

POSICIONAMENTO: LACUNAS E O FIO EDUCACIONAL

A leitura em conjunto revela lacunas que delimitam a contribuição deste estudo. Nenhum trabalho cruza os quatro paradigmas sobre a mesma aplicação, gerando o mesmo produto sob tarefa fixa; nenhuma avaliação de fidelidade a especificação usa um produto real como oráculo, recorrendo a requisitos curtos, código de pesquisa ou problemas de juiz online; e fidelidade, qualidade e segurança raramente são medidas em conjunto sobre o mesmo artefato construído do zero. Falta um estudo que troque a amplitude de muitas tarefas com poucas dimensões por um produto único avaliado em muitas dimensões. Os arcabouços consolidados também não incluem papel de segurança nem métricas de vulnerabilidade sobre o software entregue, o que faz do quarto paradigma uma contribuição original. Essas lacunas conversam com o fio educacional: se a fidelidade se perde na integração entre módulos (YAN; CHEN; ZHANG, 2026), a competência a ensinar migra da escrita de trechos corretos para a especificação e a verificação do todo; se mais orquestração pode não compensar (LU, P. et al., 2026; ASHRAFI, 2026), o julgamento sobre quando usar cada paradigma torna-se conteúdo formativo; e se pedir código seguro no enunciado não reduz a vulnerabilidade (KHARMA et al., 2026), enquanto quem programa com IA tende a confiar em excesso (PERRY et al., 2023), a verificação de segurança passa a ser habilidade central da formação.

MÉTODOS

DESENHO DO ESTUDO

O estudo compara quatro configurações completas de geração sob a mesma especificação e os mesmos tetos de recursos, ordenadas como graus crescentes de orquestração (P_0, P_1, P_2 e P_3) e detalhadas adiante. Os níveis variam como pacotes de procedimentos, não apenas pelo número de agentes: P_0 não executa nem corrige; P_1 opera uma única sessão longa; P_2 reparte o trabalho entre papéis; P_3 acrescenta análise estática com remediação. O desenho, portanto, não isola o efeito individual de cada componente da orquestração. As demais condições permanecem constantes: um único modelo gerador, a mesma amostragem, o mesmo conjunto de ferramentas, os mesmos limites de parada e a mesma especificação. A opção segue

a evidência de que o aparato é uma variável oculta capaz de dominar o custo de geração (VATS; GOLEV, 2026; GORINOVA et al., 2026); mantido constante tudo o que não é a configuração de geração, as diferenças observadas são associadas ao pacote de orquestração como um todo, não a um componente isolado nem ao substrato.

A unidade da comparação estatística é o build, a aplicação completa gerada em uma execução independente, com quatro réplicas por paradigma. Os 48 critérios de aceitação bloqueantes são medidas tomadas dentro de cada build e agregadas em um escore funcional de o a 48; não constituem 48 observações independentes, pois compartilham especificação, estado do modelo e contexto, razão pela qual a análise apoia-se em estatística descritiva robusta sobre os builds e em testes não paramétricos. A integração da aplicação completa entra como métrica de nível superior, pois é na costura entre módulos que a fidelidade tende a cair (YAN; CHEN; ZHANG, 2026). O estudo não inclui participantes humanos no laço de codificação; a comparação é entre configurações de geração automática, avaliadas por oráculos objetivos. O modelo gerador é único e registrado a cada execução pelo identificador público claude-sonnet-5, um alias do provedor que não expõe data de versão; a coleta ocorreu entre 27 de setembro e 1º de outubro de 2026, o identificador observado nos logs é o mesmo nos dezesseis builds, e a amostragem é a padrão da interface de linha de comando, não configurada e idêntica entre os braços. Não há como comprovar a ausência de atualização silenciosa do modelo pelo provedor dentro dessa janela, limitação declarada nas ameaças à validade. O planejamento previa um modelo que deixou de estar disponível antes da coleta, e a substituição pelo sucessor foi registrada como decisão datada e aplicada de modo uniforme aos quatro braços.

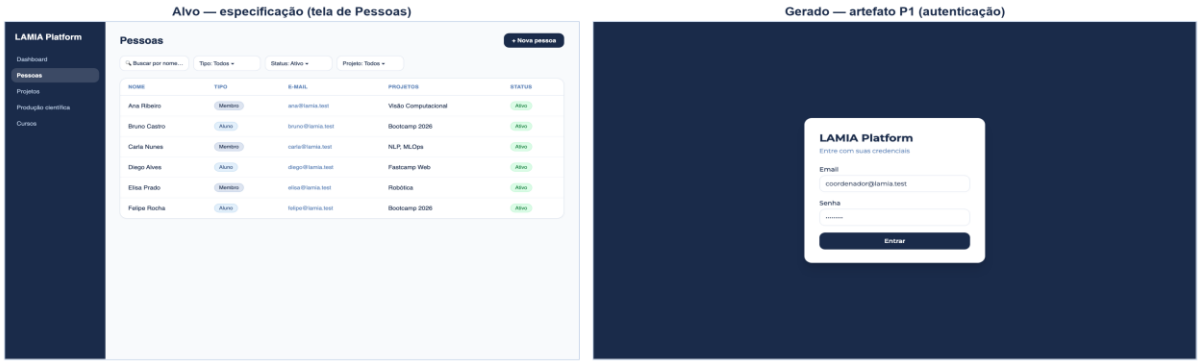
A ESPECIFICAÇÃO CONGELADA E O ORÁCULO

A fidelidade a uma especificação só é mensurável se esta existir e estiver fixada antes de qualquer geração. O estudo parte de uma plataforma de gestão de laboratório já construída, que cumpre dois papéis. Como fonte, serve para derivar, uma única vez e de forma supervisionada, um pacote de especificação congelado e versionado, reunindo a visão geral do produto, o modelo de dados com diagrama de entidades e definições de tabelas, o sistema de design com tokens e componentes, o contrato de interface de programação de aplicações, os critérios de aceitação testáveis e as regras de negócio. Como padrão de referência, serve apenas na avaliação, para medir a distância entre o gerado e o produto esperado. O pacote congelado é identificado pela etiqueta de versão spec-v1.0 no controle de versão do estudo e reúne seis

documentos: visão geral, modelo de dados, sistema de design, contrato de interface, critérios de aceitação e regras de negócio. O oráculo funcional aplica 60 critérios de aceitação, dos quais 48 bloqueantes, por suíte determinística de ponta a ponta com dados de semente fixos; os critérios foram redigidos pelos autores a partir da especificação e versionados em setembro de 2026, antes de qualquer geração medida, portanto sem acesso às soluções geradas, e a implementação de referência serviu apenas para validar que a suíte executa e aprova. O pacote de replicação, com a especificação, os critérios, os protocolos, os dados por réplica e os scripts de análise, é disponibilizado pelos autores mediante solicitação.

A Figura 3 contrasta a tela principal da especificação-alvo com um artefato gerado por um dos paradigmas.

Figura 3 – À esquerda, a tela principal da especificação-alvo (cadastro de Pessoas com filtros combináveis, no design system definido na especificação); à direita, a tela de autenticação do artefato gerado pelo paradigma P1, que aplica o mesmo design system.



Fonte: Autoria própria (2026).

O controle de vazamento é central: os agentes partem do zero e recebem apenas o pacote congelado, mais um contrato mínimo de acoplamento com o ambiente, sem jamais acessar o código-fonte da plataforma de referência. Garante-se, assim, que a IA constrói a partir da especificação, e não por cópia, o que reproduz o cenário real de desenvolvimento e protege o estudo da crítica de contaminação. O produto de referência é uma plataforma web full-stack, com frontend em Next.js e React, backend em Node.js e Express e banco PostgreSQL, que cobre o cadastro e a consulta de pessoas, projetos e produção científica, além de um bloco novo de cursos de formação, matrículas, automação de prazos, certificados, relatórios e integrações externas. O oráculo opera em dois níveis: para as funcionalidades já existentes, a fidelidade é medida contra a implementação de referência, capaz de comparar design e esquema de dados;

para as novas, como aderência aos critérios de aceitação previamente redigidos. Os dois níveis são reportados separadamente.

A VARIÁVEL INDEPENDENTE: A ESCADA DE ORQUESTRAÇÃO

A variável independente é uma progressão de graus de orquestração, do P0 ao P3, definidos na introdução e resumidos na Figura 1. A ordenação é parcial: a passagem de P2 para P3 é aninhada, pois P3 é P2 acrescido de um papel de segurança, ao passo que a de P1 para P2 muda a arquitetura, em que um agente único dá lugar a uma decomposição em papéis; por isso os níveis são tratados como comparações planejadas, sem assumir monotonicidade. Nos três braços agênticos as ferramentas de leitura, escrita, execução e teste são as mesmas; o P1 opera em um único laço de raciocínio, ação e observação, enquanto P2 e P3 distribuem a geração entre papéis que encadeiam artefatos verificáveis, e o papel de segurança de P3 executa análise estática, mapeia os achados a categorias de vulnerabilidade e exige remediação antes de concluir.

Figura 1 – A escada ordinal de orquestração. Cada degrau adiciona uma camada de aparato sobre o anterior, mantendo constantes o modelo, a amostragem, o conjunto de ferramentas e os limites de parada.



Fonte: Autoria própria (2026).

Cada degrau se ancora na literatura. O laço de raciocínio e ação que define P1 corresponde ao padrão ReAct (YAO et al., 2023), e o refino a partir do resultado de execução, ao mecanismo de reflexão sobre feedback (SHINN et al., 2023). A decomposição em papéis com artefatos verificáveis que define P2 apoia-se em MetaGPT (HONG et al., 2024) e ChatDev (QIAN et al., 2024), e dialoga com a autocolaboração por papéis em um único modelo (DONG et al., 2024). O papel de segurança de P3 responde a uma lacuna concreta, pois nem MetaGPT nem ChatDev têm papel de segurança ou métricas de vulnerabilidade sobre o artefato; sua inclusão como papel, e não como instrução de enunciado, justifica-se pela evidência de que a instrução isolada não reduz a vulnerabilidade (KHARMA et al., 2026) e de que a maior parte das falhas geradas é detectável por análise estática após a geração (FENG et al., 2026). Para preservar o isolamento do fator, tudo o que não seja a orquestração é mantido idêntico: modelo e amostragem; conjunto de ferramentas, exceto a análise estática, exclusiva do papel de segurança de P3; a especificação congelada; e o ambiente de execução. A

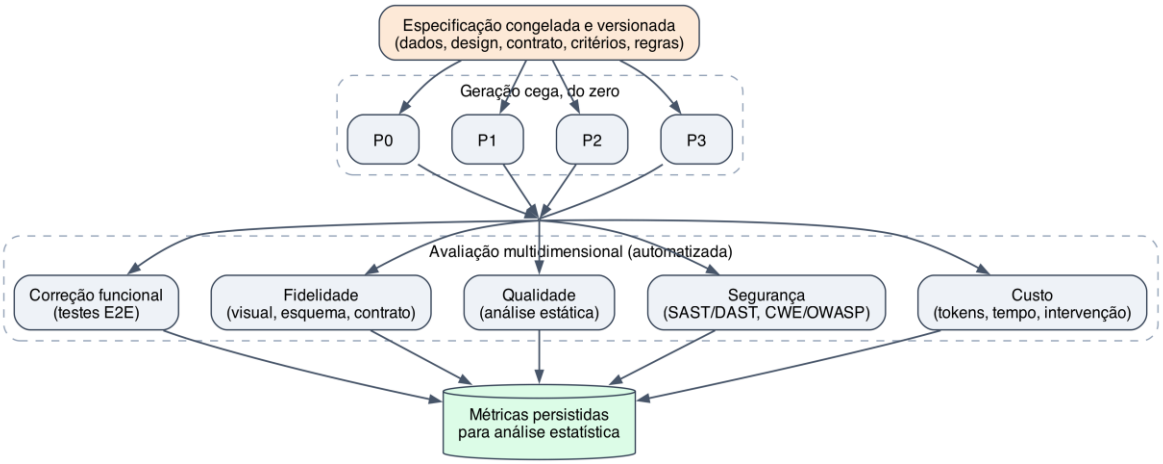
coordenação em P2 e P3 é determinística, por código, não por um agente conversacional adicional, decisão que evita gastar orçamento com coordenação e mantém a comparação limpa (ASHRAFI, 2026).

EQUALIZAÇÃO DE ORÇAMENTO E FLUXO DO EXPERIMENTO

Uma ameaça previsível é supor que os paradigmas mais orquestrados vencem apenas por dispor de mais recursos. O desenho a neutraliza tratando os limites de parada como tetos de braço, não de agente: a soma dos recursos de todos os agentes de um build, incluindo rodadas de remediação em P3, é comparada aos mesmos tetos de tokens, turnos e tempo de parede de Po e P1; em P2 e P3, um orçamento fixo é repartido entre os papéis, nunca ampliado. Atingir o teto é dado do experimento, um build que não concluiu dentro do orçamento, não motivo para reexecução (VATS; GOLEV, 2026). Os valores são idênticos nos quatro braços: 45 minutos de tempo de parede por build, único limite que interrompe a execução, por encerramento forçado do processo; e tetos registrados a posteriori, sem interrupção, de 20 dólares de custo faturado e 10 milhões de tokens não cacheados por build. O número de turnos é registrado como métrica de consumo, não como regra de parada. Nenhuma réplica atingiu teto algum: os máximos observados foram 36,6 minutos de parede, 15,91 dólares e 1,05 milhão de tokens não cacheados, e nenhum build foi interrompido ou reexecutado por orçamento; o consumo de cada réplica é reportado na seção de custo.

O procedimento segue três etapas (Figura 2). Primeiro, a especificação é congelada e versionada, tornando-se o único insumo dos geradores. Segundo, cada condição executa a geração do zero, sob protocolo padronizado e integralmente registrado, com prompts, logs, marcas de tempo e tokens, cego ao código de referência. Terceiro, um pipeline automatizado avalia cada artefato sem intervenção humana no julgamento objetivo: compilação, implantação em ambiente isolado, suíte de aceitação de ponta a ponta, qualidade e segurança estática, análise dinâmica sobre a aplicação em execução, fidelidade de esquema e de contrato, além da camada visual prevista no desenho e depois excluída por limitação de instrumento, e coleta de custo, com persistência de todas as métricas.

Figura 2 – Fluxo do experimento. A especificação congelada é o único insumo da geração cega; cada artefato passa por avaliação multidimensional automatizada, cujas métricas alimentam a análise estatística. A etapa de fidelidade visual, presente no fluxo, não integrou a comparação final (ver Métricas e controles de validade).



Fonte: Autoria própria (2026).

Cada build registra metadados de reprodutibilidade: o identificador do modelo, a revisão do aparato e da especificação, a identificação do ambiente e a marca temporal. Os recursos consumidos são registrados por camada e por papel, de modo que o custo seja reportado por componente, e não apenas como número agregado de ponta a ponta (GORINOVA et al., 2026), o que permite localizar em que etapa a fidelidade se perde.

MÉTRICAS E CONTROLES DE VALIDADE

A avaliação é multidimensional porque nenhuma dimensão isolada distingue os paradigmas hoje. O desenho previa medir a fidelidade à especificação em quatro frentes: visual, de esquema de dados, de contrato de interface e de completude dos critérios de aceitação, com rubrica multinível e auditoria que cobra apenas o que estava recuperável do insumo, no desenho de estudos de fidelidade de implementação (YAN; CHEN; ZHANG, 2026). Foram efetivamente medidas três frentes: a fidelidade do modelo de dados, por comparação estrutural do esquema; a do contrato de interface, por conformidade com a especificação; e a completude, como proporção dos critérios de aceitação atendidos. A fidelidade visual foi explorada com hash perceptual, mas o instrumento não se mostrou discriminante e por isso não integra a comparação de resultados; não houve avaliação por juiz multimodal nesta coleta. O juiz multimodal de família distinta da do gerador, calibrado contra amostra anotada manualmente com concordância por coeficiente kappa (ZHANG, C. et al., 2025), permanece como trabalho futuro.

A correção funcional é aferida por conformidade a uma suíte de aceitação de ponta a ponta, executada em caixa-preta contra a aplicação em funcionamento, o que substitui os testes unitários clássicos por um verificador comportamental (GORINOVA et al., 2026). Uma funcionalidade só é contabilizada quando a suíte confirma o comportamento real, incluindo a persistência efetiva no banco, critério que impede creditar uma aplicação de fachada (LU, Z. et al., 2026). Qualidade e segurança aferem, respectivamente, complexidade, duplicação e manutenibilidade por análise estática, e vulnerabilidades por análise estática e dinâmica mapeadas a categorias CWE e OWASP, reportadas por categoria e não só de forma agregada (PEARCE et al., 2022). O custo reúne tokens consumidos, tempo de parede, número de intervenções humanas e de iterações de autocorreção, e o registro de se o build atingiu estado executável dentro do orçamento. Intervenção humana, neste estudo sem participantes no laço, é apenas a ação pontual de um operador para destravar um agente parado por erro de ambiente, sempre contabilizada como custo, jamais a edição do código. Os indicadores de esforço, iterações até um estado utilizável, tempo até esse estado e número de destravamentos, seguem o vocabulário operacional proposto para a geração dirigida por especificação (ROSA et al., 2026).

Quatro controles de validade atravessam todas as medidas. O primeiro fixa modelo, versão e amostragem idênticos em todos os braços, para que só o paradigma varie. O segundo equaliza os tetos de parada, para que a orquestração mais pesada não seja favorecida nem penalizada. O terceiro, quando houver juízo automático de fidelidade por modelo, impede que o modelo juiz seja o que gerou o código, evitando viés de autopreferência; o mesmo princípio orienta a segurança, em que qualquer vantagem medida pela ferramenta que o próprio agente executa é checada contra um motor independente. O quarto exige verificação de persistência e comportamento real antes de contabilizar qualquer funcionalidade. Dado o número pequeno de réplicas, a leitura é predominantemente descritiva, apoiada em mediana, intervalo interquartílico, amplitude e tamanho de efeito; pelo caráter exploratório, não se adota correção formal para múltiplas comparações, e nenhuma diferença é tomada como estabelecida por inferência (PERRY et al., 2023).

RESULTADOS

A coleta iniciou dezesseis execuções, quatro repetições independentes de cada paradigma, todas reconstruindo a mesma plataforma a partir da especificação congelada, com

modelo, amostragem e limites de parada idênticos; quinze produziram artefato. Foram iniciadas quatro execuções de Po: uma gerou aplicação executável, duas geraram artefatos não executáveis e uma não produziu artefato avaliável, de modo que, na correção, esse braço contribui com uma única observação. Com quatro repetições por braço, qualquer teste de hipótese tem poder muito baixo, e a leitura se apoia em estatística descritiva robusta, tamanho de efeito e padrão qualitativo. A unidade de análise de custo é o build; a de correção, o critério de aceitação bloqueante, sobre um total de 48.

EXECUTABILIDADE DOS BUILDS

A primeira medida é se o artefato sobe e responde, condição para aplicar o oráculo funcional; é nessa dimensão, não na correção, que o prompt único se separa. Foram iniciadas quatro execuções de Po: uma gerou aplicação executável e saudável, duas geraram artefatos não executáveis, cujo container não atingiu estado saudável, e uma não produziu artefato avaliável, caso discutido entre as ameaças à validade. Os três paradigmas agênticos subiram e responderam nos quatro builds cada. Dois indicadores resumem a dimensão. Na entrega de ponta a ponta do protocolo completo, por tentativa iniciada, a executabilidade de Po é de uma em quatro tentativas, contra quatro em quatro em cada braço agêntico; condicionada aos artefatos efetivamente gerados, é de um em três artefatos. O contraste é esperado: o braço de orquestração nula, sem laço de execução nem autocorreção, com frequência entrega uma aplicação que não roda. Os casos não executáveis não foram imputados nem regenerados; apenas saíram das métricas que dependem da aplicação no ar, a correção funcional, a fidelidade de contrato e a análise dinâmica. Em uma leitura de ponta a ponta em que a aplicação que não sobe recebe escore funcional zero, as quatro execuções de Po valeriam 0, 0, 0 e 11, com mediana 0, contra medianas de 34,5 a 35 nos braços agênticos; essa leitura complementa, sem substituir, a análise condicionada à executabilidade da Tabela 1.

CORREÇÃO FUNCIONAL (ESCORE DE ACEITAÇÃO SOBRE 48)

O oráculo funcional aplica 48 critérios de aceitação bloqueantes contra a aplicação em execução, e a correção é o escore funcional: a contagem de critérios de aceitação aprovados na primeira e única geração de cada build, de 0 a 48 (Tabela 1). Evita-se o rótulo pass@1, cuja definição convencional é a probabilidade de sucesso por tarefa estimada a partir de amostras geradas (CHEN et al., 2021) e não descreve esta métrica.

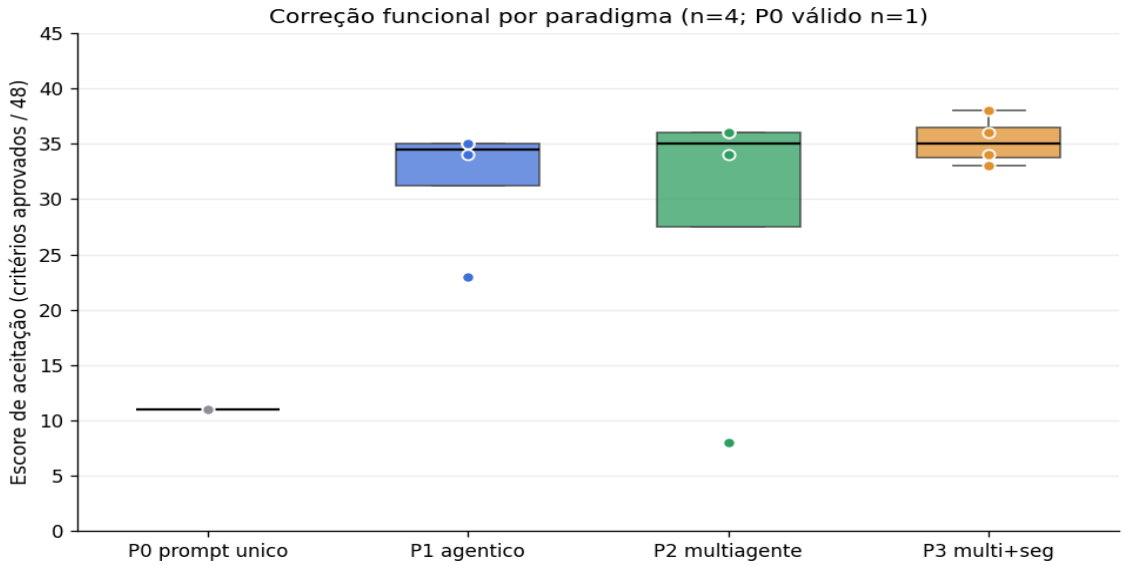
Tabela 1 – Correção funcional por paradigma (critérios de aceitação bloqueantes aprovados, sobre 48). O n de P0 é um porque apenas uma réplica foi executável.

Paradigma	n	Mediana	IQR	Mín. a Máx.	Réplicas
P0 (prompt único)	1	11	n/a	11 a 11	11
P1 (agêntico)	4	34,5	25,8 a 35,0	23 a 35	23, 34, 35, 35
P2 (multiagente)	4	35,0	14,5 a 36,0	8 a 36	8, 34, 36, 36
P3 (multi + segurança)	4	35,0	33,2 a 37,5	33 a 38	33, 34, 36, 38

Fonte: Autoria própria (2026). Nota: quartis calculados por interpolação linear na posição (n+1)p, método tipo 6, equivalente à função QUARTILE.EXC; com quatro observações por braço, o intervalo interquartílico é sensível à convenção adotada. Não aplicável a P0 (n=1).

As medianas dos três braços agênticos ficam praticamente coincidentes, entre 34,5 e 35 sobre 48, e nenhum se destaca em correção funcional pura (Figura 4). O prompt único fica muito abaixo, com uma única observação válida de 11 sobre 48, que descreve mais a fragilidade do braço do que um patamar típico. O dado individual que merece nota é um colapso estocástico: a réplica P2-repo3 marcou apenas 8 de 48, enquanto as outras três do multiagente marcaram 34, 36 e 36. Esse ponto foi mantido de propósito, como política antisseleção, e é ele que alarga o intervalo interquartílico do P2 sem mover a mediana. O braço que colapsa não é fixo: nesta coleta o ponto atípico caiu no P2; na coleta anterior, documentada nas ameaças à validade, caiu sobre uma réplica do P3. Essa migração é, ela própria, um achado sobre confiabilidade, e desaconselha ler a menor dispersão de um braço como estabilidade maior.

Figura 4 – Correção funcional (escore de aceitação: critérios aprovados sobre 48) por paradigma, com mediana, caixa interquartílica e pontos das réplicas. O n de P0 é um, pois apenas uma réplica foi executável. A réplica de colapso P2-repo3, com 8 de 48, foi mantida por integridade metodológica.



Fonte: Autoria própria (2026).

O teste de Kruskal-Wallis entre os três braços agênticos, com correção de empates e a ressalva de poder já enunciada, resulta em H igual a 0,635 e valor-p igual a 0,728; o epsilon-quadrado, calculado como $H(n+1)/(n^2-1)$ com n igual a 12, é de 0,058, um efeito pequeno. O código do cálculo acompanha o pacote de replicação. Não foi detectada diferença de correção funcional entre os três braços agênticos; com quatro réplicas por braço, essa ausência de diferença detectável não equivale a demonstrar igualdade, e nenhum ranking é proposto.

SEGURANÇA

A segurança foi a dimensão medida com mais profundidade, e a segurança de aplicação é nula em todos os braços. A análise estática foi conduzida com o Semgrep sobre conjuntos de regras ampliados, incluindo o OWASP Top Ten, regras para TypeScript, uma auditoria de segurança geral e a detecção de segredos. P0, P1 e P2 apresentaram mediana de 2 achados de alta severidade, com todos os builds com artefato de cada braço marcando 2 (três em P0, quatro em P1 e em P2), enquanto o P3 teve mediana 0, com réplicas 2, 0, 0 e 0.

O ponto decisivo é o conteúdo desses achados, não a contagem. O único achado de alta severidade recorrente, mesmo com os rulesets ampliados, é o contêiner que roda como usuário root, da categoria CWE-250, execução com privilégios desnecessários. Não há, em nenhum dos dezesseis builds, injeção de SQL, cross-site scripting, falha de autorização ou segredo embutido, o que foi confirmado pelo Semgrep e por inspeção manual, que não encontrou renderização de HTML cru no frontend nem resposta de HTML não tratada no backend, e verificou a presença de middleware de autenticação nas rotas. A alta severidade, portanto, não é vulnerabilidade de lógica, mas higiene de configuração do contêiner. O P3 remove esse achado em três de suas quatro réplicas, adicionando a diretiva de usuário não privilegiado ao Dockerfile, enquanto a P3-repoo ainda o carrega. Três ressalvas impedem ler isso como vantagem de segurança: a natureza do achado, de configuração e não de lógica; a circularidade, pois o agente de P3 executa internamente o mesmo Semgrep do avaliador; e a ausência de confirmação por motor independente.

A triangulação com outros motores confirma a ausência de vantagem de aplicação. No SonarQube, a classificação de segurança por réplica foi de [3, 3, 3, 3] para o P2 e de [3, 3, 3, 4] para o P3, em uma escala em que valores menores são melhores, ou seja, o multiagente sem segurança empata ou supera o multiagente com segurança. Na análise dinâmica, com a varredura baseline do ZAP, não há vulnerabilidades de alta severidade em nenhum braço; os

achados de risco médio são de higiene de cabeçalhos HTTP, como ausência de política de segurança de conteúdo e configuração permissiva de compartilhamento entre origens, uniformes entre os braços. Sendo a varredura passiva e não autenticada, e tendo excedido o tempo em uma réplica, não tem a profundidade da estática. Ainda assim, o quadro é coerente entre três motores e a inspeção manual.

CUSTO E CONSUMO DE RECURSOS

O custo precisa ser lido por duas lentes. A primeira é o esforço de geração, em tokens não cacheados, isto é, a soma de entrada, saída e criação de cache, que aproxima o trabalho novo de cada build. A segunda é o custo monetário efetivo, em dólares faturados, que inclui também a releitura de contexto já cacheado, cobrada com desconto mas presente em grande volume nos braços que repassam o mesmo contexto muitas vezes (Tabela 2, Figura 5).

Tabela 2 – Consumo de recursos por paradigma (medianas por build e custo por réplica). Tokens não cacheados em milhares; releitura de cache em milhões. A execução de P0 sem artefato não registrou consumo completo e não entra nas medianas.

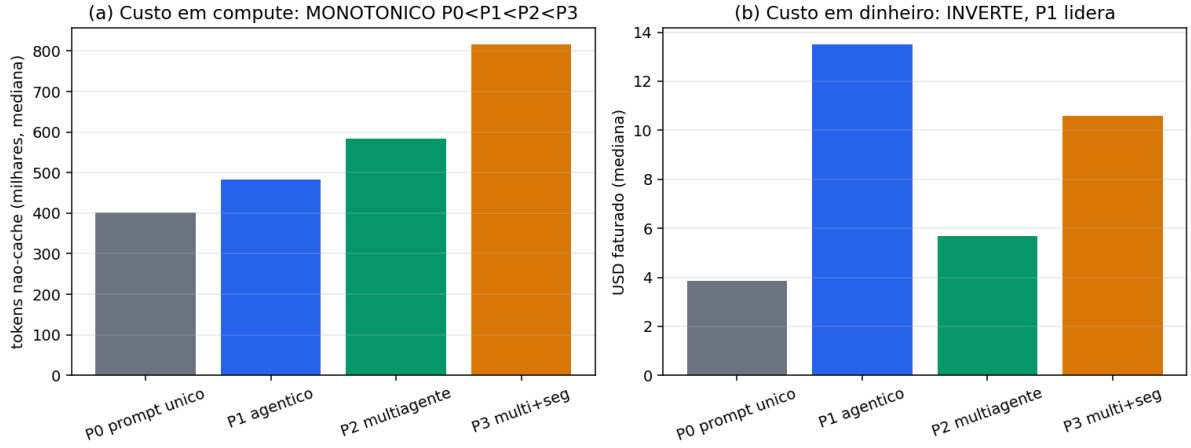
Paradigma	Tokens não-cache (mil)	Saída (mil)	Custo US$	Turnos	Releitura de cache (mi)	Custo por réplica (US$)
P0 (prompt único)	401	138	3,86	8	0,7	2,35; 3,86; 4,61
P1 (agêntico)	482	147	13,50	174	30,7	7,33; 13,22; 13,77; 13,93
P2 (multiagente)	583	46	5,69	85	6,0	4,01; 4,15; 7,23; 8,12
P3 (multi + segurança)	816	104	10,59	157	15,0	6,12; 6,44; 14,73; 15,91

Fonte: Autoria própria (2026).

Registre-se o significado dos turnos reportados: a interface contabiliza como turno cada mensagem do assistente no fluxo de resposta; a geração de P0 é uma única requisição ao modelo, sem ferramentas, verificada nos metadados como uma chamada por build, cuja resposta longa é emitida em múltiplos blocos contados como turnos, daí a mediana de oito em um braço de prompt único. Na lente do esforço de geração, o custo cresce de forma monotônica ao longo da escada, de cerca de 401 mil tokens não cacheados no P0 a 816 mil no P3, uma duplicação aproximada. Esse crescimento não vem de mais saída, pois a do P2, cerca de 46 mil tokens, é até menor que a do P0, de 138 mil; vem da criação de cache associada à fragmentação do trabalho

em múltiplas sessões. Na lente monetária a ordem se inverte: o braço mais caro em dólares não é o mais orquestrado, e sim o harness agêntico de sessão única (P1), com mediana de 13,50 dólares, mais que o dobro do P2 e acima do P3. A razão está na releitura de cache: o P1, por operar em uma única sessão longa de cerca de 174 turnos, reprocessa o mesmo contexto repetidas vezes, acumulando cerca de 30,7 milhões de tokens de releitura, contra 6 milhões do P2 e 15 milhões do P3. Adota-se, por isso, o token não cacheado como lente primária, por medir o trabalho novo e não a releitura descontada; em dinheiro faturado, porém, o agêntico de sessão única é o mais caro, resultado que contraria a intuição de que mais agentes custam mais.

Figura 5 – Custo por paradigma em duas lentes. À esquerda, tokens não cacheados (esforço de geração), monotônico de P0 a P3; à direita, custo em dólares faturados, não monotônico, em que o harness agêntico de sessão única (P1) é o mais caro por reprocessar grande volume de contexto cacheado.



Fonte: Autoria própria (2026).

Os braços diferem muito em consumo de recursos, como esperado por construção, mas esse maior consumo não se traduz em maior correção.

TRADE-OFFS ENTRE CUSTO, ORQUESTRAÇÃO, CORREÇÃO E SEGURANÇA

O cruzamento das dimensões responde à pergunta prática do estudo. As associações foram examinadas por correlação de postos de Spearman, apropriada a amostras pequenas, com Pearson em paralelo; os valores por réplica que alimentam os cálculos estão nas Tabelas 1 e 2. Com treze builds executáveis, nenhuma correlação com a correção atinge significância. Na análise agrupada, que mistura os quatro braços, a associação entre custo e correção é praticamente nula, levemente positiva, com rho de 0,29 (valor-p 0,34) para o esforço em tokens não cacheados e de 0,15 (valor-p 0,62) para o custo em dólares; turnos contra o escore funcional teve rho de 0,34 (valor-p 0,25), número de papéis rho de 0,33 (valor-p 0,28) e linhas de código

rho de 0,38 (valor-p 0,20). A análise agrupada, porém, faz custo e correção variarem junto com o próprio tratamento, sobretudo pela presença do P0, ao mesmo tempo o mais barato e o único a falhar; por isso descreve associação, não efeito causal. A análise de sensibilidade restrita aos doze builds agênticos, em que esse acoplamento se desfaz, enfraquece ainda mais as associações: rho de 0,17 (valor-p 0,60) para tokens não cacheados, de 0,06 (valor-p 0,86) para dólares e de 0,20 (valor-p 0,53) para turnos; excluído também o colapso P2-repo3, rho de 0,39 (valor-p 0,24) para tokens não cacheados e de 0,08 (valor-p 0,81) para dólares. Na análise agrupada sem o colapso, o rho sobe para 0,52 (valor-p 0,08), o que mostra que a ausência de associação não depende do outlier, um build caro e ruim que apenas reforçaria uma correlação positiva. Em nenhuma das análises gastar mais tokens, mais turnos ou acrescentar papéis comprou mais critérios aprovados.

Em contraste, o número de papéis tem associação negativa com as contagens de alta severidade, com rho de -0,69 (valor-p 0,005, n igual a 15). Essa relação, porém, é inteiramente explicada pelo braço P3, que remove o único achado recorrente de configuração de contêiner, e não por um gradiente em que cada papel reduz vulnerabilidade; desaparece se esse braço é retirado e não deve ser lida como evidência de segurança de aplicação.

Resta a fidelidade, que não discrimina os paradigmas e, por isso, é tratada como verificação de conformidade estrutural. A fidelidade de contrato é uniformemente perfeita entre os builds executáveis, com as 20 rotas de referência presentes em todos, e a de esquema é integral onde há arquivo a inspecionar, com as 16 tabelas de referência presentes em todos os builds agênticos; no P0, a maioria das réplicas sequer chega a estado avaliável. Os dois indicadores estão saturados no teto: quando a aplicação sobe, a estrutura de dados e a superfície de interface contratadas são reproduzidas independentemente do paradigma, um piso comum, não um diferencial. A camada visual exige ressalva: a similaridade perceptual por hash de imagem contra a referência ficou em torno de 0,5 em todos os builds, próximo do esperado para hashes descorrelacionados, de modo que o instrumento não discrimina e não havia referência estável para medida mais fina; por isso a fidelidade visual automática é declarada fora do escopo. A interface, porém, é exercitada funcionalmente pela suíte de aceitação de ponta a ponta, que navega as telas e confere o conteúdo renderizado; fica de fora apenas a parecença estética.

QUALIDADE DE CÓDIGO

A qualidade foi aferida por um proxy estático determinístico sobre os quinze builds com código avaliável, e não pelo servidor de análise originalmente previsto, cuja execução foi deferida e retomada entre as ameaças à validade. O proxy cobre o tamanho do código em linhas, excluídas as dependências; a complexidade ciclomática média por função; a duplicação de linhas, por casamento grosseiro; e uma contagem de maus cheiros na forma de funções de complexidade alta (Tabela 3).

Tabela 3 – Indicadores de qualidade de código por paradigma (medianas por build), medidos por proxy estático determinístico. O rating de manutenibilidade é um proxy de estilo A a E, em que letras mais avançadas indicam pior manutenibilidade.

Paradigma	LOC do app	Complexidade média por função	Duplicação	Rating (proxy)
P0 (prompt único)	2851	~1,9	~29%	D
P1 (agêntico)	4167	~1,9	39% a 41%	E
P2 (multiagente)	5695	~1,9	39% a 41%	E
P3 (multi + segurança)	6087	~1,9	39% a 41%	E

Fonte: Autoria própria (2026). Nota: linhas de código por réplica, conferidas nos dados brutos: P0 2061, 2851 e 2918; P1 3049, 4048, 4286 e 4727; P2 5046, 5635, 5754 e 5918; P3 5191, 5553, 6621 e 6631; as medianas de braços com quatro réplicas interpolam os dois valores centrais.

O achado mais claro é o do tamanho. Para a mesma especificação, o volume de código cresceu de forma monotônica ao longo da escada, de cerca de 2851 linhas na mediana do P0 a 6087 no P3, de duas a duas vírgula três vezes mais código para o mesmo produto. A complexidade ciclomática média por função ficou em torno de 1,9 em todos os braços, sem discriminar paradigmas, o que indica que o código adicional dos braços mais orquestrados não é feito de funções mais intrincadas, mas de mais funções e estrutura para o mesmo comportamento; coerente com isso, a contagem de maus cheiros foi quase nula. Duplicação e manutenibilidade acompanham o tamanho, com P0 em torno de 29% de duplicação e rating proxy D, contra 39% a 41% e rating E nos demais, mas dependem de casamento de linhas grosseiro e mapeamento simplificado, e seu valor absoluto deve ser lido com reserva. O sinal firme é duplo: a complexidade por função é uniforme entre os paradigmas, e o tamanho cresce com a orquestração sem que a correção acompanhe.

DISCUSSÃO

Os resultados desenham uma figura, em parte, contraintuitiva: entre os três braços agênticos a correção funcional permaneceu praticamente constante enquanto o esforço de

geração cresceu de forma monotônica, o custo em dólares não acompanhou essa ordem, e a orquestração não comprou ganho de correção nem de segurança de aplicação. O que de fato variou foi a confiabilidade e o custo.

POR QUE MAIS CUSTO E MAIS AGENTES NÃO COMPRARAM CORREÇÃO

A ausência de relação útil entre custo e correção sustenta a tese, já sinalizada na literatura, de que mais orquestração nem sempre compensa; a leve inclinação positiva é puxada pelo prompt único, barato e falho ao mesmo tempo. Isso conversa com a evidência de que atualizar o aparato altera o consumo de tokens em ordens de grandeza sem mover proporcionalmente a taxa de sucesso (VATS; GOLEV, 2026) e com o estudo pareado que variou seis arquiteturas e concluiu que acrescentar papéis e camadas não melhorou a correção, apenas inflou a complexidade (ASHRAFI, 2026). O colapso ocasional não é fixo em um braço, o que sugere que a decomposição em papéis reduz, mas não elimina, a cascata de alucinações que esses arranjos buscam conter (HONG et al., 2024; QIAN et al., 2024). O padrão é consistente: nenhum braço agêntico superou os demais.

A qualidade de código reforça a leitura: para a mesma especificação, o volume de código cresceu de forma monotônica enquanto a correção não acompanhou e a complexidade média por função ficou uniforme em torno de 1,9. Mais orquestração não produziu código mais correto, e sim mais código a manter. Isso replica, sobre um produto full-stack construído do zero, a observação de que acrescentar papéis e camadas infla o tamanho do código sem melhorar a correção (ASHRAFI, 2026): o custo oculto da orquestração se paga não só em geração, mas em superfície de código a sustentar depois.

POR QUE A SEGURANÇA, MEDIDA A FUNDO, NÃO DISTINGUIU OS PARADIGMAS

O exame da segurança frustra a hipótese de ganho limpo do agente dedicado. A segurança de aplicação é nula em todos os braços, e o único achado de alta severidade recorrente, a execução do contêiner como root (CWE-250), é higiene de configuração, não vulnerabilidade de lógica. O P3 o remove na maioria das réplicas, mas três fatores impedem tratar isso como vantagem de segurança do paradigma: o achado é de configuração, não de código; há circularidade, pois o agente de segurança do P3 executa internamente a mesma ferramenta que o avaliador usa para julgar, viés que a literatura adverte contra o juiz não independente do

gerador (ZHANG, C. et al., 2025); e, decisivo, no SonarQube a classificação do multiagente sem segurança é igual ou melhor que a do multiagente com segurança, enquanto a análise dinâmica não encontra alta severidade em braço algum.

Esse desfecho confirma a lição que motivou o desenho do braço P3: instruir o modelo a produzir código seguro no enunciado não reduz de modo significativo a vulnerabilidade, apenas redistribui as categorias de falha (KHARMA et al., 2026), coerente com os cerca de 40% de programas falhos em cenários de risco (PEARCE et al., 2022) e com a tendência de quem programa com IA a produzir código menos seguro sentindo-se mais confiante (PERRY et al., 2023). Acrescenta-se uma ressalva metodológica: quando o verificador de segurança executa a mesma ferramenta pela qual será avaliado, qualquer ganho aparente precisa ser confirmado por um motor independente, sob pena de se medir apenas a capacidade do agente de gabaritar o próprio teste.

O CUSTO DA ORQUESTRAÇÃO TEM DUAS LENTES

O custo ensina menos sobre qual paradigma é barato e mais sobre como medi-lo quando há cache de contexto no laço. Em esforço de geração, por tokens não cacheados, o custo cresce de forma monotônica com a orquestração; em dinheiro faturado a ordem se inverte, pois o harness agêntico de sessão única reprocessa o mesmo contexto ao longo de uma sessão longa e acumula muito mais releitura de cache que os arranjos multiagente, cuja divisão em sessões curtas a limita. A lição é dupla: afirmar que um paradigma é barato ou caro depende da lente, e relatórios que citam só o dólar faturado penalizam um agêntico de sessão única por um custo que é, em grande parte, releitura descontada, razão de este estudo adotar o token não cacheado como medida primária e o dólar como complementar; e, em linha com a observação de que o aparato domina o custo (VATS; GOLEV, 2026), o determinante do gasto não é o número de agentes, mas a estrutura de sessões e o reaproveitamento de contexto, invisível se só o total agregado é reportado. Em nenhuma das lentes o maior gasto comprou maior correção, de modo que a decisão sobre quanto orquestrar deve pesar custo, confiabilidade e superfície de código a manter.

IMPLICAÇÕES PARA A EDUCAÇÃO EM ENGENHARIA DE SOFTWARE

Cabe reafirmar o recorte: o estudo não incluiu estudantes, currículo nem desfecho de aprendizagem, e não mede intervenção pedagógica; o que oferece ao ensino é evidência sobre a

qual decisões formativas podem se apoiar, em registro interpretativo. A primeira implicação é sobre o que se ensina como boa prática: se mais orquestração e mais gasto não compraram correção sobre um produto realista, o julgamento sobre quando cada paradigma vale a pena torna-se conteúdo formativo, pois ensinar pipelines multiagente elaborados como garantia de software melhor seria ensinar moda de ferramenta no lugar de critério de engenharia. A segunda é sobre a segurança: pedir segurança no enunciado não a produz, e o agente de segurança revela uma armadilha, pois um verificador no laço só tem valor se for independente do que se quer comprovar, já que executar a mesma ferramenta pela qual será julgado tende a gabaritar o próprio teste sem significar software mais seguro. A competência a formar não é inserir uma verificação automática, mas saber desconfiar dela, triangular motores e distinguir higiene de configuração de segurança de aplicação. Essa leitura dialoga com a lacuna dos arcabouços consolidados, que não incluem papel de segurança nem métricas de vulnerabilidade sobre o artefato entregue (HONG et al., 2024; QIAN et al., 2024), e com o problema de currículo que a educação em computação sinaliza desde a chegada dos assistentes de geração (BECKER et al., 2023; DENNY et al., 2024; PEREIRA FILHO et al., 2023). A falsa confiança de quem programa com IA (PERRY et al., 2023) fecha o argumento: a habilidade a formar não é apenas gerar código com IA nem apenas automatizar sua verificação, e sim desconfiar de ambos de forma metódica e instrumentada, com juízes independentes do gerador.

AMEAÇAS À VALIDADE

A honestidade sobre os limites é parte do método; esta seção reúne ameaças já sinalizadas nos resultados. A mais séria é o tamanho da amostra: quatro repetições por braço, e apenas uma observação válida no escore funcional de Po, tornam qualquer teste de significância de baixo poder. A ausência de significância não é evidência de ausência de efeito, e a ausência de diferença detectada é descritiva, não comprovada; toda conclusão deve ser lida como indício a confirmar em amostras maiores.

O colapso estocástico de correção é a segunda ameaça. A réplica P2-repo3 marcou apenas 8 de 48, muito abaixo das outras três do multiagente, e foi mantida por integridade metodológica. O braço onde o colapso ocorre não é estável entre corridas. A corrida anterior aqui referida foi a primeira coleta completa do estudo, realizada em 19 e 20 de setembro de 2026, com doze builds, três réplicas por braço, sob a mesma especificação, o mesmo modelo, a mesma revisão do aparato e os mesmos tetos; nela, o ponto atípico recaiu sobre uma réplica do

multiagente com segurança, com 11 critérios aprovados sobre 48 contra 36 e 36 das demais réplicas do braço, enquanto as três réplicas do P2 marcaram 35. Essa coleta foi integralmente substituída pela definitiva, de quatro réplicas por braço, após correção de instrumentação no rótulo do teto de turnos, sem alteração do comportamento de geração; seus dados brutos permanecem arquivados no pacote de replicação e não integram as tabelas por se referirem à coleta substituída. A migração do colapso entre coletas indica propriedade estocástica do processo de geração, não defeito de um paradigma específico, e desaconselha atribuir maior confiabilidade ao braço que por acaso não colapsou. A fragilidade do prompt único é a terceira: foram iniciadas quatro execuções de Po, uma gerou aplicação executável, duas geraram artefatos não executáveis e uma não produziu artefato avaliável, permanecendo como caso órfão que se declara em vez de imputar.

Há ameaças de instrumentação em cada dimensão. Na segurança estática, a análise de fluxo de dados do Semgrep não concluiu o ponto fixo em algumas regras de severidade média, como origens e cabeçalhos, em vários builds, de modo que a ausência desses achados não pode ser atribuída só à ferramenta; por isso essas classes foram verificadas manualmente, confirmando a mesma configuração permissiva de origens e a ausência de cabeçalhos de segurança em todos os builds. Um controle positivo, injetando injeção e cross-site scripting em um módulo real, confirmou que as regras de aplicação disparam quando o problema existe. A análise dinâmica tem cobertura limitada: a varredura do ZAP é passiva e não autenticada e excedeu o tempo em uma réplica. A circularidade do braço de segurança já foi tratada, com a vantagem aparente do P3 checada contra um motor independente, que não a confirmou. A fidelidade visual automática foi excluída por limitação de instrumento, com similaridade perceptual em torno de 0,5, próxima de hashes descorrelacionados. A qualidade de código, medida por proxy estático determinístico e não pelo servidor de análise previsto, tem duplicação e rating apoiados em casamento de linhas grosseiro e mapeamento simplificado, de modo que seus valores absolutos não equivalem ao que um analisador consolidado reportaria; apenas a direção monotônica e a complexidade média por função uniforme em torno de 1,9 são tomadas como firmes.

Três ameaças de medida, configuração e escopo completam o quadro. Adotar o token não cacheado como lente primária privilegia o esforço de geração sobre o dólar faturado, e quem considere apenas o custo monetário chegaria a outra ordenação, razão de as duas lentes serem apresentadas lado a lado. A amostragem, padrão da interface de linha de comando, é declarada

e idêntica entre os braços, mas os resultados absolutos poderiam mudar sob outra configuração. Por fim, é um estudo de caso sobre um único domínio, com uma única aplicação e um único juiz determinístico de aceitação; a profundidade em muitas dimensões foi obtida ao preço da amplitude, e a generalização para outros domínios permanece em aberto, a ser tratada por replicação, não por extrapolação. Registre-se, por fim, a confusão entre paradigma e modelo: o identificador do modelo gerador, claude-sonnet-5, foi verificado nos logs dos dezesseis builds e é o mesmo em todos, com amostragem padrão idêntica, e a coleta ocorreu entre 27 de setembro e 1º de outubro de 2026. Por ser um alias do provedor sem data de versão, porém, uma atualização silenciosa dentro dessa janela não seria detectável; a constância da versão é, portanto, suportada pelos registros disponíveis, não comprovada.

CONCLUSÃO

Este estudo comparou, de maneira controlada e sobre um mesmo produto de software reconstruído do zero, quatro paradigmas de geração de código por inteligência artificial dispostos como uma escada de orquestração, do prompt único ao multiagente com agente dedicado de segurança, mantendo fixos o modelo, a amostragem e os limites de parada e variando apenas o paradigma. A pergunta era empírica: com que fidelidade, qualidade, segurança e custo cada paradigma reproduz um produto especificado de antemão.

A resposta, dentro dos limites de um estudo de caso exploratório, é um mapa de trade-offs: não foi detectada diferença de correção funcional entre os três paradigmas agênticos, e a correção não escalou com o esforço de geração, de modo que o que distingue os paradigmas é a confiabilidade e o custo, não a correção. A confiabilidade separa o prompt único, que produziu aplicação executável em uma de quatro execuções iniciadas, dos agênticos, que subiram sempre, com o colapso de correção migrando de braço entre corridas. O custo depende da lente, sendo o harness agêntico de sessão única o mais caro em dólares por reprocessar contexto cacheado. E a segurança de aplicação é nula em todos os braços, sem ganho limpo do agente dedicado, cujo único efeito, remover o achado de contêiner como root, é circular e não confirmado em motor independente.

A contribuição é dupla e, em parte, metodológica. No método, oferece um desenho reprodutível de comparação de paradigmas sobre um produto real, com oráculo multidimensional que separa correção, fidelidade estrutural de esquema e contrato, qualidade, segurança e custo, e expõe duas armadilhas de medida de valor geral: reportar o custo por duas

lentes quando há cache de contexto no laço, sob pena de inverter a ordenação dos paradigmas, e usar um juiz de segurança independente do gerador, sob pena de medir apenas a capacidade do agente de gabaritar o próprio teste. No conteúdo, produz evidência concreta, sobre software construído do zero, de que mais orquestração não compra correção e de que a segurança de aplicação não distingue os modos de gerar quando medida com profundidade, dando substância empírica a preocupações que a literatura formulava de modo fragmentado. Para o ensino de engenharia de software, a implicação, em registro interpretativo, é que o julgamento sobre quando orquestrar e o hábito de verificar segurança de forma instrumentada e independente precisam ocupar o centro da formação de quem vai programar mediado por agentes.

Os trabalhos futuros seguem das ameaças à validade. O passo mais urgente é ampliar repetições e domínios, para converter os padrões observados em estimativas com poder estatístico adequado, medir a taxa de colapso estocástico e testar se a ausência de diferença de correção e de segurança se mantém fora da gestão de laboratório. Convém recalibrar o instrumento de fidelidade visual e estender a análise de segurança com um avaliador independente da ferramenta usada pelo agente de P3 e com varredura dinâmica autenticada, para testar se há benefício de segurança atribuível ao paradigma uma vez removida a circularidade. Sobre essas extensões, não sobre a extrapolação da amostra presente, é que se poderá firmar o que aqui aparece como direção consistente.

DECLARAÇÃO SOBRE O USO DE INTELIGÊNCIA ARTIFICIAL

A redação e a edição deste manuscrito contaram com assistência de ferramentas de inteligência artificial. Todo o conteúdo, os dados, as análises e as conclusões foram revisados e são de inteira responsabilidade dos autores.

REFERÊNCIAS

ASHRAFI, N. **How Generation Architecture Shapes Code Complexity in Multi-Agent LLM Systems: A Paired Study on HumanEval**. arXiv:2606.00308, 2026.

AUSTIN, J. et al. **Program Synthesis with Large Language Models**. arXiv:2108.07732, 2021.

BECKER, B. A. et al. **Programming Is Hard, Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation**. In: PROCEEDINGS OF THE 54TH ACM TECHNICAL SYMPOSIUM ON COMPUTER SCIENCE EDUCATION (SIGCSE 2023). New York: ACM, 2023.

CHEN, M. et al. **Evaluating Large Language Models Trained on Code**. arXiv:2107.03374, 2021.

DAI, D.; WU, B.; FANG, M.; LI, S.; WANG, W. **I-WebGenBench: Evaluating Interactivity in LLM-Generated Scientific Web Applications**. arXiv:2606.00750, 2026.

DENNY, P. et al. **Computing Education in the Era of Generative AI**. Communications of the ACM, v. 67, n. 2, 2024.

DONG, Y.; JIANG, X.; JIN, Z.; LI, G. **Self-Collaboration Code Generation via ChatGPT**. ACM Transactions on Software Engineering and Methodology, v. 33, n. 7, 2024. DOI: 10.1145/3672459.

FENG, S.; CHEN, B.; MEYER, B. H.; MUSSBACHER, G. **LLM-Assisted Repository-Level Generation with Structured Spec-Driven Engineering**. In: COMPANION PROCEEDINGS OF THE ACM INTERNATIONAL CONFERENCE ON THE FOUNDATIONS OF SOFTWARE ENGINEERING (FSE Companion 2026). New York: ACM, 2026. DOI: 10.1145/3803437.3805567.

GORINOVA, M. I.; BAKER, M.; HEINEIKE, A.; SHAPOSHNIKOV, M.; WILLOUGHBY, R.; KNOX, D. **Position: Coding Benchmarks Are Misaligned with Agentic Software Engineering**. arXiv:2606.17799, 2026.

HONG, S.; ZHUGE, M.; CHEN, J. et al. **MetaGPT: Meta Programming for a Multi-Agent Collaborative Framework**. In: INTERNATIONAL CONFERENCE ON LEARNING REPRESENTATIONS (ICLR), 2024. arXiv:2308.00352.

HOU, X. et al. **Large Language Models for Software Engineering: A Systematic Literature Review**. ACM Transactions on Software Engineering and Methodology, v. 33, n. 8, 2024. DOI: 10.1145/3695988.

KHARMA, M.; SABBAH, A.; ALKHANAFSEH, M.; HAMMOUDEH, M.; MOHAISEN, D. **An Empirical Evaluation of LLM-Generated Code Security Across Prompting Methods**. arXiv:2605.24298, 2026.

LIU, C.; FU, Y.; YANG, W.; ZHANG, Y.; XIE, T. **WebCoderBench: Benchmarking Web Application Generation with Comprehensive and Interpretable Evaluation Metrics**. arXiv:2601.02430, 2026.

LU, C. et al. **The AI Scientist: Towards Fully Automated Open-Ended Scientific Discovery**. arXiv:2408.06292, 2024.

LU, P.; ZHANG, S.; HOU, Y. et al. **ProjDevBench: Benchmarking AI Coding Agents on End-to-End Project Development**. arXiv:2602.01655, 2026.

LU, Z.; YANG, Y.; REN, H. et al. **WebGen-Bench: Evaluating LLMs on Generating Interactive and Functional Websites from Scratch**. In: NEURIPS 2025 DATASETS AND BENCHMARKS TRACK, 2025. arXiv:2505.03733.

LU, Z.; REN, H.; YANG, Y.; WANG, K.; ZONG, Z.; ZHAN, M.; LI, H. **FullStack-Agent: Enhancing Agentic Full-Stack Web Coding via Development-Oriented Testing and Repository Back-Translation**. arXiv:2602.03798, 2026.

PAUL, D. G.; ZHU, H.; BAYLEY, I. **Investigating the Smells of LLM-Generated Code**. arXiv:2510.03029, 2025.

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: IEEE SYMPOSIUM ON SECURITY AND PRIVACY (S&P), 2022. arXiv:2108.09293.

PEREIRA FILHO, L. C.; SOUZA, T. de P. C. de; PAULA, L. B. de. **Análise das Respostas do ChatGPT em Relação ao Conteúdo de Programação para Iniciantes**. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO (SBIE), 2023. Porto Alegre: SBC, 2023. p. 1738-1748. DOI: 10.5753/sbie.2023.234870.

PERRY, N.; SRIVASTAVA, M.; KUMAR, D.; BONEH, D. **Do Users Write More Insecure Code with AI Assistants?**. In: ACM SIGSAC CONFERENCE ON COMPUTER AND COMMUNICATIONS SECURITY (CCS), 2023. p. 2785-2799. arXiv:2211.03622.

PRATHER, J. et al. **The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming**. In: AUSTRALASIAN COMPUTING EDUCATION CONFERENCE (ACE), 2023.

QIAN, C.; LIU, W. et al. **ChatDev: Communicative Agents for Software Development**. In: ANNUAL MEETING OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS (ACL), p. 15174-15186, 2024. DOI: 10.18653/v1/2024.acl-long.810.

ROSA, G.; MORENO-LUMBRERAS, D.; ROBLES, G.; GONZÁLEZ-BARAHONA, J. M. **Understanding Specification-Driven Code Generation with LLMs: An Empirical Study Design**. In: IEEE INTERNATIONAL CONFERENCE ON SOFTWARE ANALYSIS, EVOLUTION AND REENGINEERING (SANER), 2026. Registered Report (Stage 1). arXiv:2601.03878.

SCHMIDGALL, S.; SU, Y. et al. **Agent Laboratory: Using LLM Agents as Research Assistants**. In: FINDINGS OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS: EMNLP 2025, p. 5977-6043, 2025. DOI: 10.18653/v1/2025.findings-emnlp.320.

SHINN, N.; CASSANO, F.; GOPINATH, A.; NARASIMHAN, K.; YAO, S. **Reflexion: Language Agents with Verbal Reinforcement Learning**. In: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS (NeurIPS), 2023. arXiv:2303.11366.

SI, C.; ZHANG, Y.; LI, R.; YANG, Z.; LIU, R.; YANG, D. **Design2Code: Benchmarking Multimodal Code Generation for Automated Front-End Engineering**. In: PROCEEDINGS OF THE 2025 CONFERENCE OF THE NATIONS OF THE AMERICAS CHAPTER OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS (NAACL), p. 3956-3974, 2025. DOI: 10.18653/v1/2025.naacl-long.199.

VATS, N.; GOLEV, O. **The Scaffold Effect in Coding Agents: Harness Choice as a Hidden Variable in Coding-Agent Evaluation**. arXiv:2607.22585, 2026.

XIAO, J.; WANG, M.; LAM, M. H.; WAN, Y.; LIU, J.; HUO, Y.; LYU, M. R. **DesignBench: A Comprehensive Benchmark for MLLM-based Front-end Code Generation**. arXiv:2506.06251, 2025.

XU, K.; MAO, Y.; GUAN, X.; FENG, Z. **Web-Bench: A LLM Code Benchmark Based on Web Standards and Frameworks**. arXiv:2505.07473, 2025.

YAN, L.; CHEN, X.; ZHANG, X. **When the Specification Emerges: Benchmarking Faithfulness Loss in Long-Horizon Coding Agents**. arXiv:2603.17104, 2026.

YAO, S.; ZHAO, J.; YU, D.; DU, N.; SHAFRAN, I.; NARASIMHAN, K.; CAO, Y. **ReAct: Synergizing Reasoning and Acting in Language Models**. In: INTERNATIONAL CONFERENCE ON LEARNING REPRESENTATIONS (ICLR), 2023. arXiv:2210.03629.

YAO, Y.; TAN, X.; LIU, C.-H. et al. **Harness-Bench: Measuring Harness Effects across Models in Realistic Agent Workflows**. arXiv:2605.27922, 2026.

ZHANG, C.; LI, Y.; XU, C. et al. **ArtifactsBench: Bridging the Visual-Interactive Gap in LLM Code Generation Evaluation**. arXiv:2507.04952, 2025.

ZHANG, W.; YOU, G. et al. **WebGameBench: Requirement-to-Application Evaluation for Coding Agents via Browser-Native Games**. arXiv:2605.17637, 2026.

ZHU, H. et al. **FrontendBench: A Benchmark for Evaluating LLMs on Front-End Development via Automatic Evaluation**. arXiv:2506.13832, 2025.