AGENTES DE IA NA AUTOMAÇÃO DE TESTES: UMA ABORDAGEM UTILIZANDO O MODEL CONTEXT PROTOCOL E PLAYWRIGHT

Anderson da Rosa Witt[1], Marcel Campos Inocencio[2]

Resumo: A geração autônoma de testes de ponta a ponta a partir de descrições em linguagem natural permanece um problema aberto na engenharia de qualidade de software, especialmente em interfaces reativas nas quais alterações triviais invalidam seletores estáticos e elevam o custo de manutenção de casos de teste de regressão. O objetivo desta pesquisa foi avaliar o efeito da persistência e da reutilização do conhecimento de execução sobre o esforço de geração autônoma de testes de ponta a ponta por um agente baseado no *Model Context Protocol*, integrando o modelo Claude Sonnet 4.6 ao servidor Playwright MCP em uma arquitetura cliente própria, capaz de sintetizar, executar e reparar testes Playwright/TypeScript e de persistir o conhecimento de execução verificado em uma base reutilizável entre execuções. Conduziu-se avaliação empírica em dez cenários inéditos descritos em linguagem natural sobre um sistema corporativo de cadastro de pessoas, comparando-se a geração sem conhecimento prévio à geração apoiada na base já acumulada. Sem conhecimento prévio, o agente sintetizou testes executáveis nos dez cenários, com tempo médio de 155,7 segundos e consumo médio de 59.436 tokens por geração. Com a base acumulada, o esforço de geração caiu 25,3% em tempo e 33,8% em tokens, e a síntese determinística do plano a partir do grafo chegou a dispensar a etapa de exploração da aplicação. A viabilidade técnica e funcional da arquitetura proposta foi evidenciada pela obtenção de testes executáveis em todos os cenários avaliados e pelo fechamento autônomo do ciclo de geração, execução e reparo, e os resultados sustentam o potencial da persistência de conhecimento na redução do custo operacional de geração de testes, ampliando a literatura nacional sobre aplicação de modelos de linguagem em automação de testes ponta a ponta.

Palavras-chave: automação de testes E2E; agentes autônomos de IA; grandes modelos de linguagem; *Model Context Protocol*; Playwright; au-

[1]Curso de Ciência da Computação, Universidade do Extremo Sul Catarinense - Unesc, Criciúma - Santa Catarina - Brasil. andersonrw161196@gmail.com

[2]Orientador. Curso de Ciência da Computação, Universidade do Extremo Sul Catarinense - Unesc, Criciúma - Santa Catarina - Brasil. marcel.inocencio@gmail.com

torreparo de testes.

ABSTRACT: Autonomous generation of end-to-end tests from natural-language descriptions remains an open problem in software quality engineering, particularly in reactive interfaces where trivial changes invalidate static selectors and inflate the maintenance cost of regression test cases. The objective of this research was to evaluate the effect of persisting and reusing execution knowledge on the effort of autonomous end-to-end test generation by an agent built on the Model Context Protocol, integrating the Claude Sonnet 4.6 model with the Playwright MCP server through a custom client architecture, capable of autonomously synthesizing, executing and repairing Playwright/TypeScript tests and persisting the verified execution knowledge in a reusable base across runs. Empirical evaluation was conducted on ten unseen scenarios described in natural language over a corporate person-registration system, comparing generation without prior knowledge to generation supported by the previously accumulated base. Without prior knowledge, the agent synthesized executable tests for all ten scenarios, with an average generation time of 155.7 seconds and an average consumption of 59,436 tokens per generation. With the accumulated base, the generation effort dropped by 25.3% in time and 33.8% in tokens, and the deterministic synthesis of the plan from the graph was able to skip the application exploration step. The technical and functional viability of the proposed architecture was evidenced by the generation of executable tests in all evaluated scenarios and by the autonomous completion of the generate-execute-repair cycle, and the results support the potential of knowledge persistence to reduce the operational cost of test generation, extending the Brazilian literature on the application of language models to end-to-end test automation.

Keywords: end-to-end test automation; autonomous AI agents; large language models; Model Context Protocol; Playwright; self-healing tests.

1 INTRODUÇÃO

A engenharia de software contemporânea caracteriza-se pela adoção generalizada de metodologias ágeis e de esteiras de Integração e Entrega Contínuas, práticas que permitem integrar código e publicar novas versões em produção com alta frequência (Soares et al., 2022). Nesse cenário, a Garantia de Qualidade precisa operar de maneira contínua, sob pena de tornar-se gargalo no fluxo de desenvolvimento. A automação de

testes consolidou-se como o mecanismo central para assegurar a integridade do sistema frente a novas implementações, processo conhecido como teste de regressão.

Destacam-se nessa automação os testes de ponta a ponta, ou *end-to-end* (E2E), que exercitam a aplicação inteira ao simular a jornada completa do usuário em um navegador real, do início ao fim de um fluxo, em vez de verificar unidades isoladas de código (García et al., 2022). Ferramentas como o Selenium WebDriver e, em período mais recente, o Playwright, ambas voltadas ao controle programático do navegador para reproduzir essa interação, tornaram-se padrões da indústria para esse tipo de teste.

Essa prática, contudo, enfrenta um problema reconhecido na literatura: a ocorrência de testes não determinísticos, também chamados de testes intermitentes ou *flaky tests*, que ora passam ora falham sem qualquer alteração no código sob teste, comprometendo a confiabilidade da suíte. Parry et al. (2022) apontam, em ampla revisão sobre o tema, que relatos industriais do Google indicam que aproximadamente dezesseis por cento dos testes apresentam algum grau de intermitência, e que pesquisa com desenvolvedores registrou que cinquenta e nove por cento deles lidam com testes intermitentes em base mensal, semanal ou diária.

No domínio específico dos testes de interface, Romano et al. (2021), ao analisarem 235 testes intermitentes extraídos de aplicações web e Android reais, identificaram a espera assíncrona como causa predominante, respondendo por aproximadamente 45% dos casos.

As causas, segundo Leotta et al. (2023), incluem latência no carregamento de elementos, dependências externas e, de forma destacada, a fragilidade dos localizadores frente a alterações de *layout*. A automação tradicional apoia-se em seletores que constituem referências estáticas no código; alterações triviais na interface, impulsionadas por *frameworks* reativos como React e Vue, invalidam essas referências e ocasionam falhas em cascata, exigindo intervenção humana para retificação.

O advento dos Grandes Modelos de Linguagem (LLMs), sustentado pela arquitetura *Transformer* de Vaswani et al. (2017), um modelo de rede neural baseado em mecanismos de atenção que dispensa o processamento estritamente sequencial dos dados, alterou o panorama da pesquisa em automação de testes. Esse potencial ampliou-se com o paradigma agêntico, no qual o modelo não se limita a responder, mas raciocina, planeja e executa ações de forma autônoma, alternando etapas de racio-

cínio e ação para invocar ferramentas externas, conforme formalizado por Yao et al. (2023) sob a denominação ReAct.

Schäfer et al. (2024) demonstraram, em estudo empírico, a viabilidade de LLMs para a geração de testes unitários sem treinamento específico, com cobertura mediana de 70,2%.

Para que esses modelos interajam de forma padronizada com ferramentas e dados externos, a Anthropic (2024), empresa de pesquisa em inteligência artificial responsável pelo modelo Claude, introduziu o *Model Context Protocol* (MCP), um protocolo aberto que define essa comunicação por meio de uma arquitetura cliente-servidor baseada em JSON-RPC, um protocolo de chamada remota de procedimentos que estrutura as mensagens trocadas no formato JSON.

No contexto brasileiro, o trabalho de Júnior et al. (2025) propõe o GenIA-E2ETest, abordagem de geração de testes de ponta a ponta executáveis a partir de descrições em linguagem natural com IA generativa, que reporta 82% de precisão e 85% de *recall* sobre duas aplicações web. A proposta, embora pioneira, não opera sob paradigma agêntico nem implementa mecanismo de autorreparo após a detecção de falhas em execução: o ciclo encerra-se na geração inicial, deixando ao desenvolvedor humano o ônus da manutenção quando o teste quebra.

O interesse da pesquisa nacional estende-se para além da geração E2E, com estudos sobre a geração de dados de teste a partir de descrições em *Behavior-Driven Development*, prática que especifica o comportamento esperado do sistema em linguagem próxima à natural (Mendoza et al., 2024), sobre a geração de testes de interface gráfica para aplicações Android com modelos multimodais, capazes de processar conjuntamente entradas de texto e imagem (Fagundes; Teixeira, 2025), e sobre a avaliação do impacto do uso de LLMs em equipes de teste na indústria (Oliveira; Tiago; Chaves, 2025).

Ferramentas reativas de autocorreção, como o Healenium (EPAM Systems, 2024), tentam reparar automaticamente seletores quebrados em tempo de execução por meio de algoritmos heurísticos, mas operam em memória, sem modificar o código-fonte, o que gera débito técnico cumulativo. Posteriormente, o próprio Playwright passou a incorporar agentes nativos de planejamento, geração e reparo de testes operando sobre o *Model Context Protocol* (Microsoft, 2025), o que evidencia a consolidação de fluxos agênticos de automação como tendência corrente da indústria.

Configura-se, assim, uma lacuna específica. As soluções dispo-

níveis ou se encerram na geração, sem fechar o ciclo de execução e reparo, como ocorre nas abordagens não-agênticas, ou, quando agênticas, atuam por invocação e reexploram a aplicação a cada uso, sem reaproveitar o conhecimento de execução obtido em rodadas anteriores. Permanece pouco explorada a possibilidade de um agente que não apenas gere, execute e repare, mas que também persista o conhecimento verificado de interação, como localizadores confirmados, falhas observadas e os reparos correspondentes, em uma base reutilizável capaz de reduzir progressivamente a exploração necessária nas gerações seguintes.

É nesse acúmulo de conhecimento entre execuções que este trabalho situa sua contribuição. Em complementaridade à abordagem não-agêntica de Júnior et al. (2025) e à proposta agêntica aberta representada pelos agentes nativos do Playwright (Microsoft, 2025), investiga-se a aplicação do *Model Context Protocol* como infraestrutura para um agente autônomo que agrega a persistência e o acúmulo do conhecimento de execução entre rodadas, avaliado empiricamente em um sistema corporativo de cadastro de pessoas.

Como hipótese de trabalho, postula-se que um agente que persiste e reutiliza o conhecimento de execução adquirido em rodadas anteriores reduz o esforço de geração, medido em iterações, tempo e custo, em relação às execuções sem conhecimento prévio. O objetivo geral desta pesquisa é avaliar o efeito da persistência e da reutilização do conhecimento de execução sobre o esforço de geração autônoma de testes de ponta a ponta por um agente baseado no *Model Context Protocol*.

Como objetivos específicos, busca-se aplicar o agente na geração de testes executáveis a partir de descrições em linguagem natural; mensurar o esforço de geração em iterações, tempo e custo; comparar esse esforço entre as execuções sem e com conhecimento de execução previamente acumulado; e avaliar de que forma a síntese do plano a partir da base persistente afeta a etapa de exploração da aplicação.

A relevância desta pesquisa justifica-se em três dimensões. Do ponto de vista prático, a manutenção de testes de ponta a ponta representa custo recorrente nas equipes de qualidade de software, agravado pela fragilidade dos seletores em interfaces reativas e pela intermitência documentada na literatura; uma abordagem que reduz progressivamente o esforço de geração tem, portanto, impacto direto no custo operacional da automação. Do ponto de vista científico, a persistência e a reutilização do conhecimento de execução por agentes autônomos constituem dimen-

são ainda pouco explorada, ausente tanto nas abordagens não-agênticas quanto nas ferramentas agênticas que reexploram a aplicação a cada uso, de modo que a evidência empírica aqui produzida contribui para caracterizar esse fenômeno. Do ponto de vista acadêmico, o trabalho amplia a literatura nacional sobre a aplicação de grandes modelos de linguagem à automação de testes, ao investigar sob paradigma agêntico um problema até então tratado predominantemente por geração única.

Este artigo está estruturado da seguinte forma: a seção 2 apresenta os materiais e métodos, descrevendo a arquitetura do agente, o modelo de linguagem utilizado, o ambiente e os cenários de avaliação, o delineamento experimental e os procedimentos de coleta e análise dos dados; a seção 3 apresenta os resultados obtidos e os discute à luz dos trabalhos correlatos, incluindo as limitações do estudo; e a seção 4 apresenta as conclusões, as contribuições da pesquisa e as recomendações de trabalhos futuros.

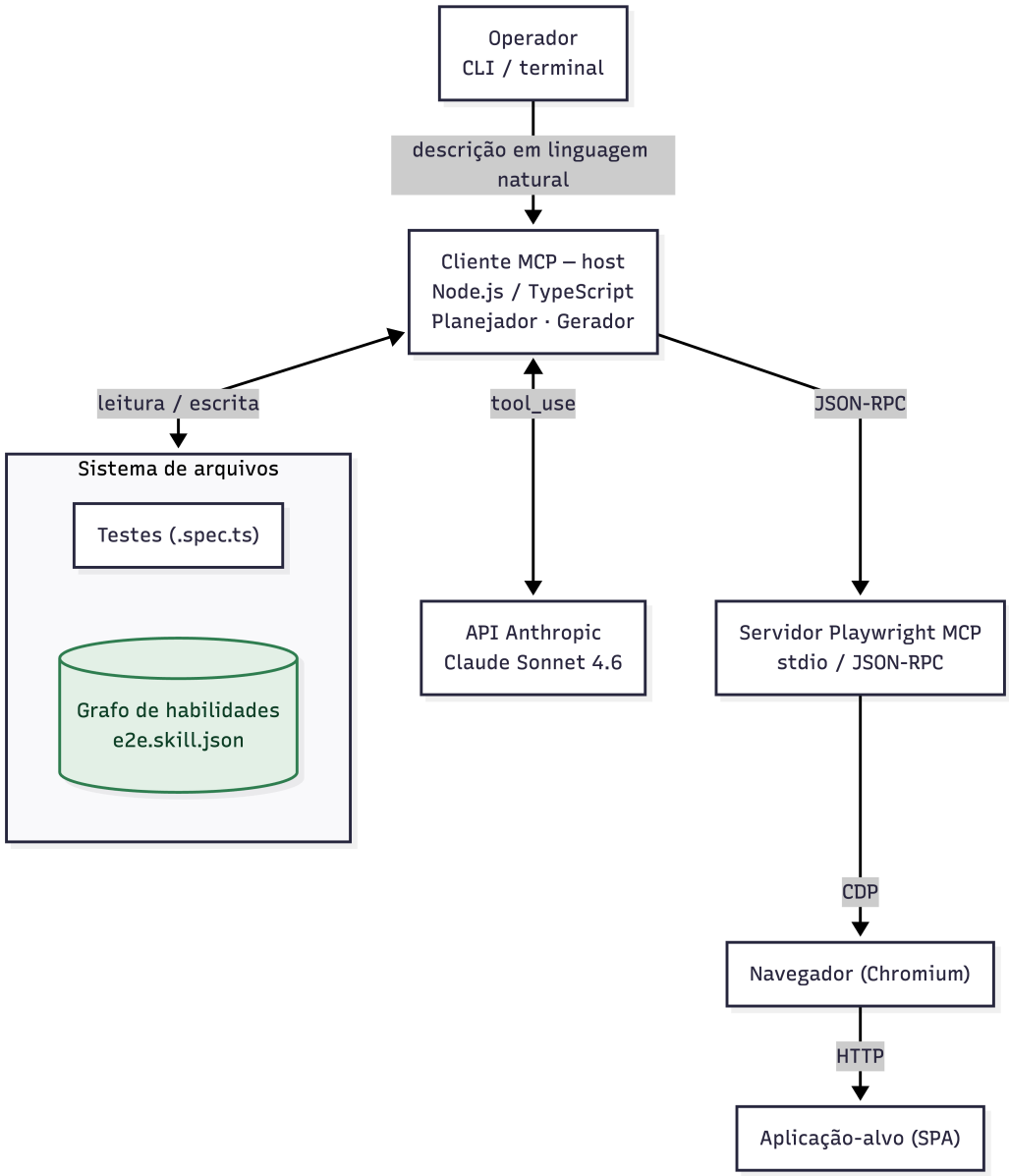
2 MATERIAIS E MÉTODOS

O objeto de estudo é o agente autônomo construído neste trabalho, um cliente MCP capaz de gerar, executar e reparar testes de ponta a ponta, em conjunto com a base de conhecimento de execução que ele acumula e reutiliza entre rodadas. Esse agente foi exercitado sobre uma aplicação web real, descrita mais adiante nesta seção, que cumpre o papel de ambiente de avaliação.

O elemento que distingue a arquitetura é a base de conhecimento persistente, materializada em um grafo de habilidades. A cada execução bem-sucedida, o conhecimento de interação verificado, como localizadores confirmados, falhas observadas e os reparos aplicados, é extraído e incorporado ao grafo, que permanece disponível entre execuções. A solução é organizada como um cliente MCP, executado em Node.js e TypeScript, que coordena três recursos externos, conforme a Figura 1:

a) um modelo de linguagem responsável pelo raciocínio, acessado por interface programável;
b) um servidor Playwright MCP responsável pelo controle do navegador;
c) o sistema de arquivos local, responsável tanto pela gravação dos testes quanto pela manutenção de uma base de conhecimento persistente.

Figura 1 - Arquitetura da solução

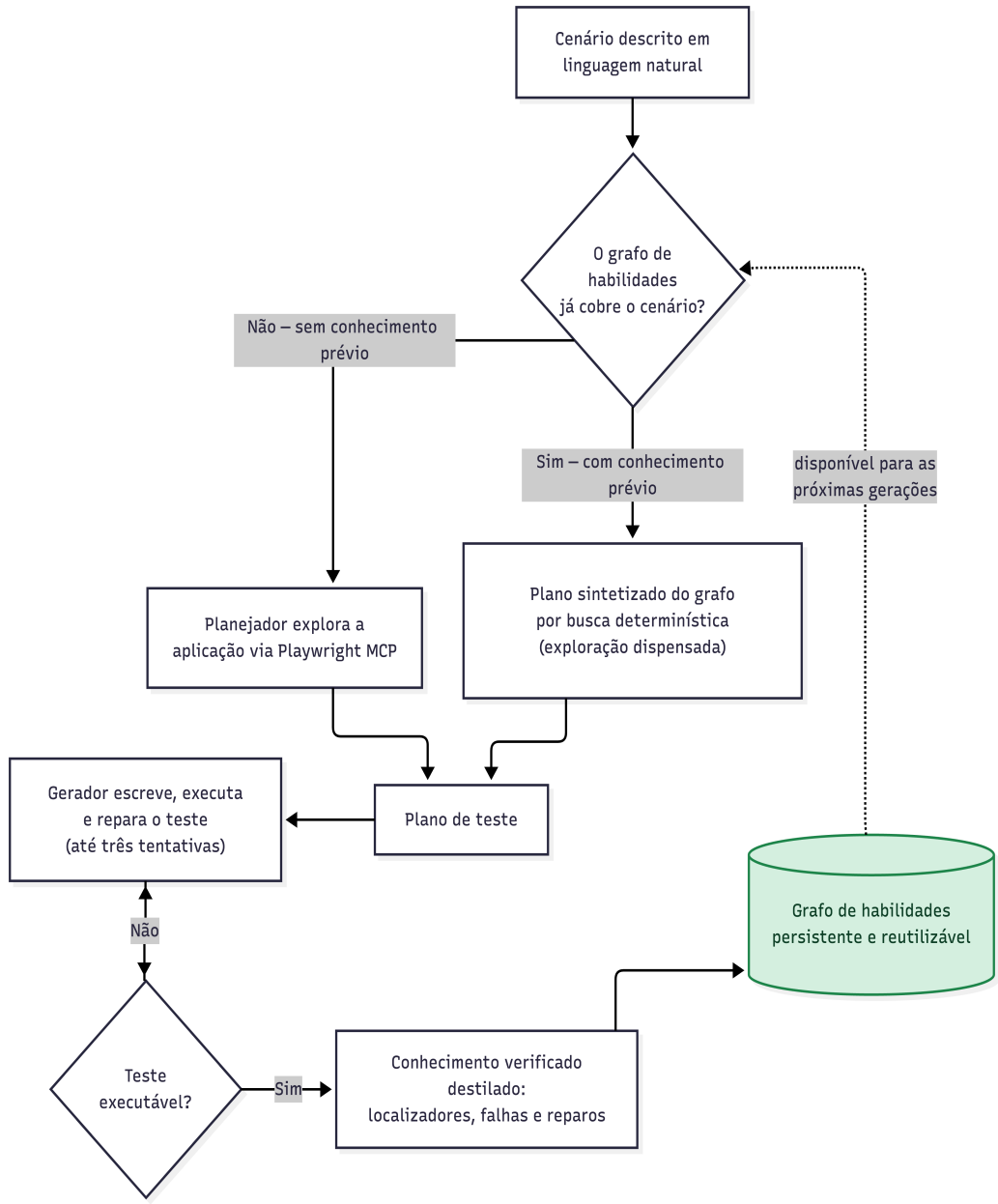


Fonte: Do autor.

Diferentemente de um laço único de raciocínio e ação, o cliente separa duas responsabilidades. Um planejador percorre a aplicação por meio do servidor Playwright MCP, inspeciona os campos, identifica máscaras e restrições e produz um plano de teste sujeito a aprovação. Um gerador, sem acesso às ferramentas de navegação, recebe o plano aprovado e escreve o teste em Playwright e TypeScript, executando-o e corrigindo-o dentro de um laço de reparo limitado. Essa separação isola a exploração da síntese e mantém o gerador restrito ao artefato de teste.

Em rodadas posteriores, quando o grafo já cobre o cenário solicitado, o plano é sintetizado diretamente a partir dele por busca determinística, dispensando a etapa de exploração. É esse mecanismo de acúmulo que sustenta a hipótese da pesquisa e que diferencia a abordagem das soluções que reexploram a aplicação a cada uso. A Figura 2 ilustra o ciclo de geração e a retroalimentação do grafo (modos sem e com conhecimento prévio).

Figura 2 - Ciclo de geração e reutilização do conhecimento de execução



Fonte: Do autor.

A condução do agente é parametrizada por um *system prompt* que estabelece a prioridade de seletores na seguinte ordem:

a) papéis de acessibilidade;
b) rótulos associados;
c) texto visível;
d) apenas como último recurso, atributos *data-testid*.

O *prompt* também exige verificações de visibilidade antes de cada interação e independência entre os casos de teste, favorecendo localizadores estáveis e asserções de resultado em vez de verificações superficiais de presença.

2.1 MODELO DE LINGUAGEM UTILIZADO

O componente de raciocínio do agente foi o Claude Sonnet 4.6. Grandes modelos de linguagem são redes neurais treinadas sobre vastos conjuntos de textos para predizer tokens em sequência e, com isso, executar tarefas a partir de instruções em linguagem natural, sem ajuste específico para cada tarefa (Brown et al., 2020). Tais modelos apoiam-se na arquitetura Transformer (Vaswani et al., 2017), já apresentada na introdução. O acesso ao modelo deu-se por interface programável, por meio do *Software Development Kit* (SDK) oficial da Anthropic.

A escolha do Claude Sonnet 4.6 justificou-se por três requisitos da arquitetura proposta:

a) o suporte nativo à invocação de ferramentas (*tool calling*), condição necessária para que o agente opere sobre o MCP e controle o navegador por meio do servidor Playwright MCP;
b) a capacidade de geração de código, requerida para a síntese dos testes em Playwright e TypeScript;
c) a aptidão para raciocínio agêntico no padrão ReAct (Yao et al., 2023), que alterna etapas de raciocínio e ação ao longo do laço de planejamento, geração e reparo.

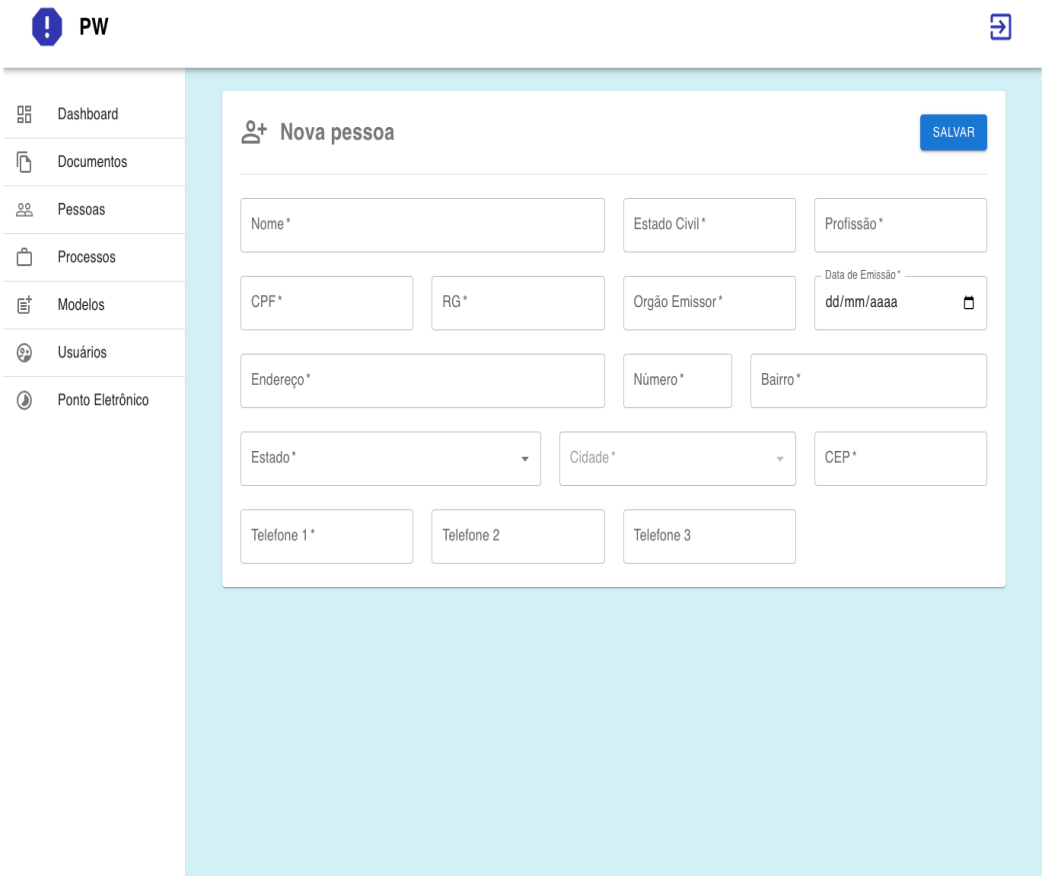
Por tratar-se de um modelo proprietário, acessível apenas por interface remota, o seu emprego impõe restrições de custo e de privacidade, consideradas na análise dos resultados.

2.2 AMBIENTE DE AVALIAÇÃO

A aplicação-alvo é um sistema corporativo de gestão de pessoas, cujo formulário de cadastro reúne campos obrigatórios, campos com

máscara de entrada, como CPF e CEP, campos com dependência entre si, como Estado e Cidade, e fluxos de validação e persistência, características representativas de um cenário industrial típico e adequadas para exercitar a geração da suíte. A Figura 3 ilustra a interface dessa aplicação.

Figura 3 - Interface da aplicação-alvo (tela de cadastro de pessoa)



Fonte: Do autor.

2.3 CENÁRIOS DE AVALIAÇÃO

Os dez cenários submetidos ao agente seguiram um mesmo modelo de instrução em linguagem natural, iniciado pela autenticação com usuário e senha e seguido da tarefa específica de cada cenário. O Quadro 1 apresenta a tarefa correspondente a cada um dos dez cenários avaliados.

Quadro 1 - Cenários de avaliação descritos em linguagem natural

Cenário	Tarefa descrita em linguagem natural
1	Cadastrar uma pessoa com dados válidos
2	Validar os campos obrigatórios do formulário de cadastro de pessoa
3	Rejeitar um CPF em formato inválido no cadastro de pessoa
4	Preencher o endereço observando a dependência entre Estado e Cidade
5	Cancelar um cadastro em andamento e voltar para a listagem
6	Cadastrar uma pessoa sem preencher os telefones opcionais
7	Validar a máscara de entrada do campo CEP
8	Validar o tamanho máximo do campo Nome
9	Navegar do dashboard até a tela de cadastro de pessoa
10	Salvar um cadastro com todos os campos preenchidos

Fonte: Do autor.

2.4 DELINEAMENTO EXPERIMENTAL

Adotou-se delineamento experimental exploratório, voltado a isolar o efeito da persistência do conhecimento de execução sobre o esforço de geração, seguindo recomendações de Wohlin et al. (2024) para experimentação em engenharia de software. O experimento comparou duas condições sobre o mesmo conjunto de dez cenários inéditos, descritos em linguagem natural e apresentados no Quadro 1. Na condição sem conhecimento prévio, o grafo de habilidades foi reiniciado antes de cada cenário, de modo que cada geração partiu do zero. Na condição com conhecimento acumulado, o grafo foi preservado ao longo da sequência, permitindo que cada cenário reaproveitasse o que os anteriores haviam aprendido.

Para cada geração registraram-se a obtenção de teste executável, o número de iterações do gerador, o tempo de parede (o tempo real decorrido entre o início e o fim de cada geração), o consumo de tokens e o custo estimado correspondente, a ocorrência de síntese do plano a partir do grafo e o número de nós acumulados. A hipótese sob avaliação foi a de que a condição com conhecimento acumulado reduz o esforço de geração em relação à condição sem conhecimento prévio.

2.5 PROCEDIMENTOS DE COLETA

O desenvolvimento valeu-se dos SDKs oficiais da Anthropic e do MCP para integrar o modelo Claude Sonnet 4.6 ao servidor Playwright MCP. As ferramentas customizadas responsáveis por gravar o teste em disco, executá-lo por meio do *runner* do Playwright e devolver a saída ao agente, bem como as rotinas que extraem o conhecimento de cada execução bem-sucedida, o incorporam ao grafo e sintetizam o plano a partir dele, foram injetadas no conjunto de ferramentas exposto ao modelo. O laço de reparo do gerador foi limitado a três tentativas de escrita e execução do teste por geração, dentro de uma salvaguarda geral de cinquenta turnos de ferramenta, para conter o custo.

A coleta seguiu ordem cronológica. Implementada e configurada a solução, executou-se primeiro o lote sem conhecimento prévio, com o grafo reiniciado antes de cada um dos dez cenários, e em seguida o lote com conhecimento acumulado, com o grafo preservado ao longo da sequência, partindo de uma base vazia no primeiro cenário e crescendo a cada execução. As métricas de cada geração foram extraídas da saída do agente, e os *transcripts*, isto é, os registros completos das mensagens trocadas entre o agente e o modelo de linguagem em cada execução, foram armazenados para auditoria do consumo de tokens e inspeção do comportamento do agente.

O custo em dólares de cada geração foi estimado a partir do consumo de tokens registrado nos *transcripts*, aplicando-se a tabela de preços da interface programável da Anthropic vigente no período de realização dos experimentos, de US$ 3,00 por milhão de tokens de entrada e US$ 15,00 por milhão de tokens de saída para o modelo Claude Sonnet 4.6 (Anthropic, 2026). Ressalta-se que o valor monetário está sujeito a alterações na política de preços do provedor, razão pela qual o consumo de tokens é reportado como métrica primária de esforço, invariante ao preço, figurando o custo como estimativa derivada.

2.6 ANÁLISE DOS DADOS

A análise dos dados teve caráter descritivo e exploratório, próprio da investigação de um fenômeno recente e ainda pouco caracterizado empiricamente, a saber, o acúmulo e a reutilização do conhecimento de execução por um agente autônomo. O conjunto avaliado foi composto por dez cenários em cada condição, número definido por conveniência e não por cálculo amostral, o que, somado à natureza não determinística do modelo de

linguagem, levou a não conduzir inferência estatística formal. Compararam-se as médias das duas condições para cada métrica e calculou-se a redução relativa proporcionada pelo conhecimento acumulado, expressa como variação percentual em relação à condição de referência conforme prática usual em experimentação de software (Wohlin et al., 2024), definida como a razão entre a diferença das médias e a média da condição sem conhecimento prévio, conforme a Equação 1.

$$\text{Redução}\,(\%) = \frac{\bar{x}_{\text{sem}} - \bar{x}_{\text{com}}}{\bar{x}_{\text{sem}}} \times 100 \tag{1}$$

em que $\bar{x}_{\text{sem}}$ e $\bar{x}_{\text{com}}$ denotam a média da métrica nas condições sem e com conhecimento prévio, respectivamente.

3 RESULTADOS E DISCUSSÃO

A Tabela 1 consolida as métricas agregadas das duas condições experimentais sobre os dez cenários avaliados, contrastando a geração sem conhecimento prévio com a geração apoiada no conhecimento acumulado. As linhas distinguem os indicadores de contagem ou proporção das métricas médias de esforço de geração: iterações, tempo, custo e tokens. A coluna *Redução* reporta a redução relativa proporcionada pelo conhecimento acumulado sobre essas métricas, calculada conforme a Equação 1; nos indicadores de contagem, para os quais ela não é informativa, emprega-se o rótulo *n/a*.

Tabela 1 - Comparação entre a geração sem e com conhecimento prévio

Métrica	**Sem conhecimento**	**Com conhecimento**	**Redução**
Testes executáveis gerados	10/10	8/10	n/a
Aprovação na primeira execução	5/10	7/10	n/a
Iterações médias	1,50	1,40	−6,7%
Tempo médio (s)	155,7	116,3	−25,3%
Custo médio (US$)	0,44	0,32	−26,9%
Tokens médios	59.436	39.332	−33,8%
Plano sintetizado do grafo	0/10	3/10	n/a
Nós no grafo ao final	4	5	n/a

Fonte: Do autor.

Na condição sem conhecimento prévio, o agente produziu teste executável nos dez cenários, com tempo médio de 155,7 segundos, consumo médio de 59.436 tokens e custo estimado médio de US$ 0,44 por geração, em 1,50 iteração média do gerador. Por partir de uma base vazia a cada cenário, nenhuma geração reaproveitou plano do grafo, que encerrou a sequência com quatro nós. A aprovação na primeira execução ocorreu em cinco dos dez cenários, o que indica que, sem conhecimento acumulado, o gerador frequentemente precisou de ao menos uma rodada de correção antes de produzir um teste aprovado.

Na condição com conhecimento acumulado, o grafo foi preservado ao longo da sequência e passou a sintetizar o plano diretamente, sem nova exploração, em três dos dez cenários, encerrando com cinco nós. O esforço de geração caiu de forma consistente: o tempo médio recuou para 116,3 segundos, o custo médio para US$ 0,32 e o consumo de tokens para 39.332, o que corresponde a reduções de 25,3%, 26,9% e 33,8%, respectivamente, em relação à condição sem conhecimento prévio. As iterações médias recuaram de 1,50 para 1,40, redução de 6,7%, e a aprovação na primeira execução subiu de cinco para sete cenários.

3.1 EFICIÊNCIA VERSUS CONFIABILIDADE

Um resultado merece leitura cuidadosa. A condição com conhecimento acumulado gerou teste executável em oito dos dez cenários, contra dez na condição sem conhecimento prévio. As duas falhas ocorreram em cenários que não dispunham de plano sintetizado pelo grafo e que, portanto, seguiram o mesmo caminho exploratório da condição sem conhecimento prévio, o que aponta para a variância intrínseca do modelo de linguagem como causa provável, e não para uma regressão introduzida pela persistência.

A evidência é, ainda assim, honesta quanto ao alcance da contribuição: a persistência do conhecimento de execução reduz de maneira clara o esforço de geração, medido em tempo, custo, tokens e iterações, mas não elevou, neste conjunto, a taxa de obtenção de teste executável. O ganho demonstrado é de eficiência, não de confiabilidade.

3.2 DISCUSSÃO COM A LITERATURA

Schäfer et al. (2024) demonstraram a viabilidade de LLMs para a geração de testes unitários sem treinamento específico, com cobertura mediana de 70,2%; embora a granularidade unitária não seja diretamente

comparável à geração de ponta a ponta investigada aqui, ambos os trabalhos confirmam a viabilidade técnica de modelos de linguagem na síntese de código de teste.

Frente ao GenIA-E2ETest de Júnior et al. (2025), que gera testes a partir de descrições em linguagem natural sem operar sob paradigma agêntico nem reparar falhas, o agente proposto fecha o ciclo de execução e reparo e acrescenta a retenção do conhecimento entre rodadas.

Em relação aos agentes nativos do Playwright (Microsoft, 2025), que também planejam, geram e reparam sobre o *Model Context Protocol*, a distinção recai sobre a persistência: enquanto aqueles reexploram a aplicação a cada invocação, a abordagem avaliada reaproveita o conhecimento adquirido, e é justamente esse reaproveitamento que a comparação entre as condições sem e com conhecimento prévio isola e quantifica.

3.3 LIMITAÇÕES DO ESTUDO

Algumas limitações delimitam o alcance dos achados. O conjunto avaliado foi pequeno e definido por conveniência, restrito a uma única aplicação-alvo, o que impede generalização e desaconselha inferência estatística formal. O custo, embora reduzido pela persistência, permanece proporcional ao tamanho do histórico da conversa e cresce com o número de iterações.

A dependência de uma interface programável externa e os riscos de segurança associados a agentes baseados em MCP, apontados por Radosevich e Halloran (2025), recomendam que o uso seja restrito a ambientes de teste segregados de produção.

4 CONCLUSÃO

A pesquisa partiu da hipótese de que um agente autônomo baseado no *Model Context Protocol*, capaz de persistir e reutilizar o conhecimento de execução adquirido em rodadas anteriores, reduz o esforço de geração de testes de ponta a ponta. Os experimentos conduzidos sobre um sistema corporativo de cadastro de pessoas confirmaram a viabilidade técnica e funcional da arquitetura proposta, com geração de testes executáveis nos dez cenários sem conhecimento prévio, e sustentaram a hipótese, pois a reutilização do conhecimento acumulado reduziu o tempo, o custo e o consumo de tokens da geração, com a síntese determinística do plano a partir do grafo, chegando a dispensar a exploração da aplicação. A hipótese formulada é, dessa forma, amparada pelas evidências reunidas no que diz

respeito ao esforço de geração.

As contribuições situam-se em três planos. No plano científico, oferece-se evidência empírica de que a persistência do conhecimento de execução reduz progressivamente o custo da geração agêntica de testes, dimensão ausente tanto nas ferramentas agênticas que reexploram a aplicação a cada uso quanto em abordagens não-agênticas como o GenIA-E2ETest, ampliando a literatura nacional sobre o tema. No plano metodológico, propõe-se um protocolo de avaliação que compara a geração sem e com conhecimento acumulado, replicável em outras pesquisas. No plano industrial, apresenta-se uma arquitetura aberta e auditável, passível de extensão por equipes de qualidade interessadas em adotar agentes de IA em seus fluxos de teste.

Reconhecem-se limitações relevantes. A avaliação restringiu-se a uma única aplicação e a dez cenários, sem poder estatístico para inferência formal, o que restringe a generalização dos resultados. A quantidade de cenários decorreu de duas restrições práticas: o custo computacional e financeiro de cada geração, uma vez que cada cenário é executado nas duas condições experimentais e consome, em média, dezenas de milhares de tokens do modelo de linguagem por rodada, e o propósito exploratório do estudo, para o qual os dez cenários foram definidos de modo a cobrir os comportamentos representativos da aplicação-alvo, como campos obrigatórios, máscaras de entrada, dependência entre campos, navegação e cancelamento de fluxo, em vez de buscar amostragem estatística.

A dependência de um modelo proprietário impõe restrições econômicas e de privacidade que limitam a adoção em ambientes com requisitos rigorosos sobre vazamento de dados. A variância do modelo entre execuções, evidenciada pelas duas falhas da condição com conhecimento acumulado, reforça a necessidade de repetição e de amostras maiores. E a resiliência da suíte frente a mutações da interface, assim como a comparação direta com o Playwright Codegen e com os agentes nativos do Playwright, não foram medidas, permanecendo como lacunas reconhecidas.

Como direções futuras, sugere-se conduzir o experimento de resiliência a mutações controladas da interface e a comparação empírica de custo acumulado frente ao Playwright Codegen e aos agentes nativos do Playwright, justamente a dimensão em que a persistência tende a se destacar; ampliar a avaliação a múltiplas aplicações-alvo, com poder estatístico adequado; investigar o desempenho de diferentes famílias de modelos, proprietários e abertos, sob o mesmo protocolo; e examinar a transferibilidade

da arquitetura host-MCP para outras tarefas de Engenharia de Software que envolvam síntese e manutenção de código-fonte.

DECLARAÇÃO DE USO DE INTELIGÊNCIA ARTIFICIAL

Os autores declaram que ferramentas de Inteligência Artificial generativa foram utilizadas exclusivamente como apoio auxiliar à elaboração deste trabalho, incluindo assistência na revisão linguística, estruturação textual e organização de ideias. Nenhuma ferramenta de IA foi utilizada para substituição da autoria intelectual, análise científica, interpretação dos resultados ou tomada de decisões acadêmicas. A responsabilidade integral pelo conteúdo, originalidade, veracidade das informações e conformidade com os princípios de integridade científica permanece integralmente atribuída aos autores.

REFERÊNCIAS

Anthropic. **Introducing the Model Context Protocol**. 2024. <https://www.anthropic.com/news/model-context-protocol>. Acesso em: 15 abr. 2026.

Anthropic. **Pricing**. 2026. Claude Platform Documentation. Disponível em: <https://platform.claude.com/docs/en/about-claude/pricing>. Acesso em: 01 jul. 2026. Disponível em: <https://platform.claude.com/docs/en/about-claude/pricing>. Acesso em: 01 jul. 2026.

BROWN, T. B. et al. **Language models are few-shot learners**. In: **Advances in Neural Information Processing Systems (NeurIPS)**. [S.l.: s.n.], 2020. 2020. v. 33, p. 1877–1901.

EPAM Systems. **Healenium: Self-healing Test Automation Tool**. 2024. <https://healenium.io/>. Acesso em: 15 abr. 2026.

FAGUNDES, N.; TEIXEIRA, L. **Evaluating LLMs for multimodal GUI test generation in Android applications**. In: **Anais do X Simpósio Brasileiro de Testes de Software Sistemático e Automatizado (SAST)**. Porto Alegre: SBC, 2025. 2025. p. 28–36.

GARCÍA, B. et al. **Selenium-jupiter: A JUnit 5 extension for Selenium WebDriver**. Journal of Systems and Software, v. 189, p. 111298. 2022.

JÚNIOR, E. et al. **GenIA-E2ETest: A generative AI-based approach for end-to-end test automation**. In: **Anais do XXXIX Simpósio Brasileiro de Engenharia de Software (SBES)**. Porto Alegre: SBC, 2025. 2025. p. 282–292.

LEOTTA, M. et al. **Challenges of end-to-end testing with Selenium WebDriver and how to face them: A survey**. In: **Proceedings of the**

16th IEEE International Conference on Software Testing, Verification and Validation (ICST). [S.l.]: IEEE, 2023. 2023. p. 339–350.

MENDOZA, I. et al. **Comparative analysis of large language model tools for automated test data generation from BDD**. In: **Anais do XXXVIII Simpósio Brasileiro de Engenharia de Software (SBES)**. Porto Alegre: SBC, 2024. 2024. p. 280–290.

Microsoft. **Playwright Test Agents**. 2025. <https://playwright.dev/docs/test-agents>. Acesso em: 30 maio 2026.

OLIVEIRA, F.; TIAGO, L.; CHAVES, L. **The influence of using LLMs on the activities of a software testing team: An industry case**. In: **Anais do X Simpósio Brasileiro de Testes de Software Sistemático e Automatizado (SAST)**. Porto Alegre: SBC, 2025. 2025. p. 144–146.

PARRY, O. et al. **A survey of flaky tests**. ACM Transactions on Software Engineering and Methodology, v. 31, n. 1, p. 1–74. 2022. Article 17.

RADOSEVICH, B.; HALLORAN, J. **MCP Safety Audit: LLMs with the Model Context Protocol Allow Major Security Exploits**. 2025. Disponível em: <https://arxiv.org/abs/2504.03767>. Acesso em: 15 abr. 2026.

ROMANO, A. et al. **An empirical analysis of UI-based flaky tests**. In: **Proceedings of the 43rd IEEE/ACM International Conference on Software Engineering (ICSE)**. [S.l.]: IEEE, 2021. 2021.

SCHÄFER, M. et al. **An empirical evaluation of using large language models for automated unit test generation**. IEEE Transactions on Software Engineering, v. 50, n. 1, p. 85–105. 2024.

SOARES, E. et al. **The effects of continuous integration on software development: a systematic literature review**. Empirical Software Engineering, v. 27, n. 3, p. 78. 2022.

VASWANI, A. et al. **Attention is all you need**. In: **Advances in Neural Information Processing Systems (NeurIPS)**. Long Beach, CA, USA: [s.n.], 2017. 2017. v. 30, p. 5998–6008.

WOHLIN, C. et al. **Experimentation in Software Engineering**. 2. ed. Berlin, Heidelberg: Springer. 2024.

YAO, S. et al. **ReAct: Synergizing reasoning and acting in language models**. In: **Proceedings of the 11th International Conference on Learning Representations (ICLR)**. Kigali, Ruanda: [s.n.], 2023. 2023. Disponível em: <https://arxiv.org/abs/2210.03629>. Acesso em: 15 abr. 2026.