Revisão de Métodos e Ferramentas para Teste no Desenvolvimento de Sistemas Agênticos (Aplicações de IA Generativa)

João Gabriel Cezário Vieira Leite
joao.vleite@ufrpe.br
Universidade Federal Rural de Pernambuco
Recife, Pernambuco, Brasil

Abstract

The rapid adoption of Generative Artificial Intelligence, specifically Large Language Models, has shifted software development from traditional machine learning models to agentic systems capable of autonomous planning and tool use. Unlike deterministic software, these systems exhibit probabilistic behavior, making consolidated validation methods difficult to apply directly. This paper presents a review of methods and tools for testing agentic systems.

The study proposes a specialized testing taxonomy adapted from the ISTQB framework, introducing levels to handle the stochastic nature of foundation models. Furthermore, it introduces a bipartite testing pyramid that separates deterministic infrastructure from probabilistic AI components. Finally, the work analyzes essential methods and tools for observability and security.

Keywords

Agentic Systems, Large Language Models (LLMs), Software Testing, AI Testing Taxonomy, Eval-driven Development (EDD), Scenario Testing, Red Teaming, Observability

1 Introdução

A popularização da Inteligência Artificial Generativa (IA Generativa), viabilizada principalmente pelo acesso a Grandes Modelos de Linguagem (Large Language Models ou LLMs) por meio de APIs, transformou a forma como sistemas de software são concebidos e desenvolvidos [23]. Diferentemente de aplicações baseadas em aprendizado de máquina tradicional, nas quais o modelo atua como um componente isolado de predição, aplicações modernas baseadas em LLMs passaram a incorporar mecanismos de planejamento, uso de ferramentas externas e manutenção de contexto ao longo do tempo, configurando sistemas capazes de executar tarefas complexas de forma parcialmente autônoma, conhecidos como sistemas agênticos [26].

Na literatura clássica, agentes são tradicionalmente associados ao Aprendizado por Reforço (*Reinforcement Learning*) em ambientes bem definidos, com regras explícitas, como jogos (ex.: AlphaGo) [27, 28]. Em contraste, sistemas agênticos baseados em LLMs operam em ambientes abertos e não estruturados, utilizando *prompts* para interpretar intenções e ferramentas (*tools*) para interagir com sistemas externos, o que impõe desafios específicos de validação [28].

Esse avanço tecnológico impulsionou interesse industrial pela automação de atividades cognitivas, como atendimento ao cliente e análise de documentos [23]. Contudo, tais sistemas diferem substancialmente do software determinístico clássico. Um desafio central reside na definição de critérios de correção. Em software convencional, uma entrada fixa produz uma saída bem definida, permitindo a construção de oráculos de teste precisos. Em contraste, sistemas agênticos podem gerar múltiplas respostas semanticamente aceitáveis para uma mesma entrada [26], tornando insuficientes métodos baseados em comparação exata de saídas e exigindo avaliações semânticas que considerem também as decisões tomadas ao longo da execução.

Estudos recentes indicam que aplicações baseadas em LLMs demandam estratégias de avaliação híbridas e sensíveis a riscos de segurança, uma vez que métricas automatizadas isoladas não capturam adequadamente vulnerabilidades como *prompt injection* ou inconsistências semânticas [1, 10].

Embora iniciativas recentes tenham proposto diretrizes para o teste de sistemas baseados em IA generativa [2], a validação da autonomia agêntica, entendida como a capacidade do sistema de planejar, selecionar ações e utilizar ferramentas de forma coerente ao longo de múltiplos passos, ainda carece de uma taxonomia consolidada [26]. Na prática, desenvolvedores recorrem a abordagens empíricas e ferramentas criadas em contextos corporativos e comunidades open-source, frequentemente baseadas em experimentação manual e avaliações *ad-hoc*, evidenciando a ausência de métodos sistemáticos amplamente aceitos.

Diante desse cenário, este trabalho adota uma perspectiva de engenharia de software aplicada para investigar métodos e ferramentas voltados à validação de sistemas agênticos construídos sobre LLMs. Busca-se organizar conceitualmente o problema e identificar práticas que apoiem o desenvolvimento de aplicações agênticas mais confiáveis, seguras e alinhadas aos requisitos de negócio.

O restante deste artigo está organizado da seguinte forma: A Seção 2 apresenta a fundamentação teórica, contextualizando a transição de LLMs para sistemas agênticos e os padrões de arquitetura que influenciam a estratégia de validação. A Seção 3 propõe uma taxonomia de testes adaptada, baseada no referencial ISTQB, e apresenta uma reestruturação da pirâmide de testes para acomodar a natureza híbrida desses sistemas. A Seção 4 analisa ferramentas essenciais para observabilidade e segurança. A Seção 5 detalha métodos práticos de execução. Por fim, a Seção 6 apresenta a conclusão do estudo, sumarizando as contribuições, limitações e propostas de trabalhos futuros.

2 Fundamentação Téorica

Com a chegada do ChatGPT, em 2022, vivenciamos uma mudança significativa na maneira como nos relacionamos com sistemas de inteligência artificial. Desenvolvido pela OpenAI, a aplicação rapidamente ganhou destaque global ao demonstrar grande habilidade ao responder perguntas e prover informações[19]. Esse avanço impulsionou a popularização dos grandes modelos de linguagem (LLMs), como Gemini[24] e DeepSeek[14]. Desde então, esses modelos deixaram de ser usados apenas para conversação passiva e passaram a atuar como núcleo de agentes autônomos, capazes de executar tarefas complexas em contextos dinâmicos.

A evolução crítica neste cenário reside na mudança de paradigma sobre a função do modelo. Enquanto em aplicações mais simples de chatbots o modelo é utilizado para gerar texto final, em sistemas agênticos o LLM opera como um "motor de raciocínio" (*reasoning engine*). Nesta arquitetura, o modelo não apenas prevê a próxima palavra, mas é utilizado para observar o ambiente, planejar sequências de ações e decidir quais ferramentas invocar para solucionar um problema [21].

2.1 De LLMs a Sistemas Agênticos

Para estruturar a estratégia de testes, é fundamental compreender que nem toda aplicação que utiliza LLMs possui o mesmo grau de autonomia. A literatura recente, incluindo diretrizes da Anthropic [2], estabelece uma distinção clara entre *"Workflows"* (Fluxos de Trabalho) e "Agentes" propriamente ditos.

2.1.1 O Espectro de Autonomia: Workflows vs. Agentes. Os *Workflows* são sistemas onde a orquestração do raciocínio e o uso de ferramentas são definidos por caminhos de código predeterminados. O LLM atua apenas em pontos específicos para processar informações, mas não controla o fluxo de execução [2]. Já os Agentes são sistemas onde o LLM atua como o controlador central, decidindo dinamicamente a sequência de passos e o uso de ferramentas com base no feedback do ambiente, sem um caminho rígido preestabelecido [2].

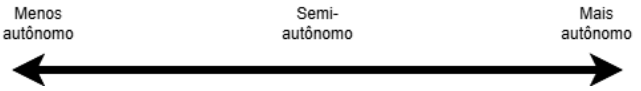

Figure 1: Espectro de autonomia em sistemas de IA.

Conforme observado na Figura 1, à esquerda encontram-se os sistemas **Menos Autônomos** (Workflows), caracterizados por previsibilidade e ferramentas *hardcoded*. À direita, situam-se os sistemas **Mais Autônomos** (Agentes), que oferecem flexibilidade máxima para lidar com situações imprevistas. No centro, encontram-se os sistemas **Semi-autônomos**, que integram supervisão humana (*Human-in-the-loop*) para validar decisões críticas.

Andrew Ng reforça que a distinção semântica rígida entre "agente" e "não-agente" pode ser contraproducente. Ele propõe o termo "Agentic AI", ideia que posteriormente passou a ser referida também como "Sistemas Agênticos" (*Agentic Systems*), ou "Fluxos de Trabalho Agênticos" (*Agentic Workflows*), enfatizando que o valor para o desenvolvimento de software não está na autonomia total, mas na capacidade do sistema de iterar sobre o próprio trabalho e refinar suas saídas antes de apresentá-las ao usuário [21].

2.2 Padrões de Projeto em Sistemas Agênticos

A transição de modelos isolados para sistemas compostos levou à criação de padrões de arquitetura de software específicos. Andrew Ng argumenta que a inovação crítica no atual estágio da IA Generativa não reside apenas no aumento da escala dos modelos, mas na estruturação de fluxos agênticos que permitem aos modelos iterar sobre problemas [21].

Enquanto a inferência tradicional de LLMs é linear, ou seja, uma entrada, uma saída, os sistemas agênticos introduzem ciclos de retroalimentação. Ng [21] identifica quatro padrões de projeto fundamentais que definem a construção desses sistemas e, consequentemente, ditam a estratégia de testes necessária: Reflexão, Uso de Ferramentas, Planejamento e Colaboração Multiagente.

2.2.1 Reflexão (Reflection). O padrão de Reflexão aborda a limitação inerente dos LLMs de não conseguirem corrigir erros durante a geração inicial do *token*. Neste padrão, o sistema é projetado para revisar sua própria saída antes de apresentá-la ao usuário. O fluxo típico envolve um *prompt* que solicita ao modelo que atue como um "crítico", analisando a resposta anterior em busca de falhas lógicas, erros de sintaxe ou alucinações [21].

Empiricamente, Ng demonstra que envolver um modelo de capacidade intermediária em um loop de reflexão robusto pode superar o desempenho de um modelo superior operando sem reflexão.

2.2.2 Uso de Ferramentas (Tool Use). Também referido na literatura como *Action* ou *Function Calling*, este padrão estende as capacidades do LLM além do processamento de texto. O modelo atua como um roteador que identifica a necessidade de dados externos ou computação determinística e invoca as ferramentas apropriadas (APIs, calculadoras, buscas web) [21].

Diferente da recuperação passiva de informações (RAG), o *Tool Use* permite que o agente altere o estado do sistema ou execute ações no mundo real. A complexidade de teste aqui reside na validação dos argumentos gerados pelo modelo para a chamada da função e na capacidade do agente de interpretar o retorno da ferramenta, que pode conter erros ou formatações inesperadas [21].

2.2.3 Planejamento (Planning). O padrão de Planejamento é essencial para tarefas complexas que não podem ser resolvidas em um único passo de inferência. O agente utiliza o LLM para decompor um objetivo abstrato (ex: "criar um jogo") em uma sequência ordenada de sub-tarefas (ex: "gerar história", "escrever código", "testar") [21].

Ng destaca que este padrão permite que o agente mantenha o foco em longos horizontes de execução. No entanto, introduz o risco de erros em cascata: se o plano inicial for falho, todas as etapas subsequentes podem ser comprometidas. Do ponto de vista de verificação e validação, isso demanda abordagens que considerem não apenas o resultado final, mas também a consistência e adequação dos artefatos intermediários gerados ao longo do processo de planejamento, como a estrutura do plano, a ordem das subtarefas e seus critérios de conclusão, sem pressupor necessariamente a inspeção explícita do raciocínio interno do modelo [21].

2.2.4 Colaboração Multiagente (Multi-agent Collaboration). Este padrão envolve a orquestração de múltiplas instâncias de LLMs, cada uma configurada com um *system prompt* (persona) distinto,

trabalhando em conjunto para resolver uma tarefa. Um exemplo clássico citado por Ng é a simulação de uma equipe de desenvolvimento de software, onde um agente atua como "Desenvolvedor" (escrevendo código) e outro como "Testador" (revisando o código e sugerindo correções) [21].

A colaboração multiagente fragmenta a complexidade do problema, permitindo especialização. Contudo, do ponto de vista de testes, cria desafios de não-determinismo exacerbado, pois a interação entre os agentes pode levar a loops infinitos ou convergência para soluções sub-ótimas se não houver critérios de parada e mediação bem definidos [21]. Em última análise, essa orquestração complexa dificulta o isolamento da causa raiz de falhas, torna-se um desafio técnico distinguir se um erro na saída final decorreu da alucinação de um agente específico, da chamada de uma função inadequada ou de uma falha lógica em um módulo convencional integrado.

A literatura recente também enfatiza que a avaliação de aplicações baseadas em LLMs não deve se restringir a métricas automáticas tradicionais, dado que estas frequentemente apresentam baixa correlação com julgamentos humanos, especialmente em tarefas abertas e conversacionais [1]. Em domínios sensíveis como cibersegurança, a necessidade de avaliação rigorosa é ampliada pela presença de vulnerabilidades específicas dos modelos, exigindo mecanismos de teste que considerem tanto desempenho funcional quanto robustez adversarial [10].

3 Taxonomia

Uma taxonomia atua como uma ferramenta de classificação que permite organizar, dentro de um domínio de conhecimento, diferentes termos integrados, fornecendo uma estrutura definida para identificar, atribuir e executar ações de forma clara [25]. No contexto de projetos de software, a aplicação de uma taxonomia facilita a identificação das tecnologias envolvidas e permite estabelecer os testes específicos aos quais o projeto deve ser submetido, otimizando a seleção de ferramentas e estratégias de validação [25].

Com esse propósito de organização conceitual, esta seção visa estabelecer uma taxonomia para as atividades de verificação e validação em sistemas agênticos. A discussão toma como referência os padrões de teste definidos pelo International Software Testing Qualifications Board (ISTQB), que constitui o principal corpo normativo internacional para a área de testes de software.

O ISTQB organiza as atividades de teste por meio de níveis bem estabelecidos no *Certified Tester Foundation Level* (CTFL)[17], voltados ao software convencional, e no *Certified Tester AI Testing* (CT-AI)[16], que estende esses níveis para sistemas baseados em Inteligência Artificial. O CT-AI introduz estágios específicos relacionados a dados e modelos de aprendizado de máquina, mantendo, ao mesmo tempo, os níveis clássicos de teste de componentes, integração, sistema e aceitação.

Sistemas agênticos baseados em IA Generativa apresentam natureza híbrida, combinando módulos determinísticos de software tradicional (como lógica de orquestração, conectores e APIs) com módulos probabilísticos baseados em modelos fundacionais e técnicas de engenharia de prompt. Em razão dessa característica mista, tais sistemas herdam desafios tanto do teste de software convencional quanto do teste de sistemas de IA, tornando relevante considerar conjuntamente os referenciais CTFL e CT-AI.

Neste trabalho, adota-se o CT-AI como base normativa, uma vez que ele já engloba os níveis clássicos do CTFL e adiciona estágios específicos para componentes baseados em IA.

Contudo, embora o CT-AI forneça uma estrutura adequada para sistemas de aprendizado de máquina tradicionais, ele não contempla plenamente as especificidades arquiteturais de sistemas agênticos, nos quais o modelo atua como um mecanismo de decisão que coordena ferramentas, módulos e estados ao longo de múltiplos passos de execução. Assim, torna-se necessária uma adaptação dos níveis de teste propostos pelo ISTQB para refletir o ciclo de vida e a organização interna desses sistemas. A taxonomia proposta nas seções seguintes busca atender a essa lacuna.

3.1 ISTQB (CT-AI)

O syllabus *Certified Tester AI Testing* (CT-AI) do International Software Testing Qualifications Board (ISTQB) estabelece um referencial normativo para o teste de sistemas baseados em Inteligência Artificial, reconhecendo que esses sistemas combinam componentes convencionais de software com componentes baseados em aprendizado de máquina. Em relação ao *Certified Tester Foundation Level* (CTFL), voltado exclusivamente ao software determinístico, o CT-AI amplia os níveis tradicionais de teste para contemplar riscos associados à natureza probabilística e orientada a dados desses sistemas.

A estrutura proposta pelo CT-AI preserva os níveis clássicos de teste (*Component Testing*, *Component Integration Testing*, *System Testing* e *Acceptance Testing*) e introduz dois níveis adicionais específicos para sistemas de IA: *Input Data Testing* e *ML Model Testing*. Esses níveis especializados visam isolar e avaliar, separadamente, os principais artefatos associados ao aprendizado de máquina antes de sua integração ao sistema como um todo.

O nível de *Input Data Testing* (Testes de Dados de Entrada) tem como objetivo assegurar a qualidade dos dados utilizados pelo sistema, tanto no treinamento quanto na fase de inferência. Nesse estágio, são verificadas propriedades como completude, consistência, representatividade e presença de vieses indesejados, uma vez que deficiências nesses atributos tendem a propagar-se para o comportamento do modelo e comprometer a confiabilidade do sistema final.

O nível de *ML Model Testing* (Testes de Modelo de ML) foca na avaliação do modelo de aprendizado de máquina de forma isolada, antes de sua integração com os demais componentes. Nessa etapa, o modelo é validado em relação a critérios funcionais (por exemplo, métricas de desempenho como acurácia ou precisão, conforme a tarefa) e não funcionais (como robustez, estabilidade e limites de generalização), garantindo que ele atenda aos requisitos definidos para o sistema.

Após a validação dos artefatos específicos de IA, o CT-AI retoma os níveis tradicionais de teste. O *Component Testing* verifica os módulos não baseados em IA, assegurando que cada componente determinístico funcione corretamente de forma isolada. O *Component Integration Testing* avalia a interação entre componentes de IA e componentes convencionais, verificando se a comunicação,

o fluxo de dados e os contratos de interface são respeitados. Em seguida, o *System Testing* valida o comportamento global do sistema em um ambiente próximo ao real, considerando tanto requisitos funcionais quanto características não funcionais. Por fim, o *Acceptance Testing* confirma se o sistema atende às necessidades do usuário e aos objetivos de negócio, constituindo a última etapa antes de sua disponibilização em produção.

A Tabela 1 sintetiza os níveis de teste definidos pelo CT-AI e os relaciona aos níveis estabelecidos no CTFL, evidenciando a continuidade conceitual entre o teste de software convencional e o teste de sistemas baseados em Inteligência Artificial.

3.2 Taxonomia Proposta

Embora a estrutura do ISTQB forneça uma base sólida, o desenvolvimento de sistemas agênticos, que envolvem o uso de ferramentas (*tool use*) e a composição de múltiplos módulos baseados em modelos fundacionais, demanda uma granularidade maior na avaliação dos componentes de IA. A taxonomia proposta nesta seção expande os conceitos tradicionais para abordar especificidades de sistemas agênticos, como a seleção do *Foundation Model* e a avaliação da interação entre múltiplos módulos baseados em IA generativa e componentes convencionais de software.

Nesta proposta, observa-se a introdução do nível *Model Selection Testing*, que precede a construção do agente, focando na escolha do LLM adequado (por exemplo, GPT-4 [22] versus Llama-3 [13]) com base em critérios funcionais e não funcionais. Além disso, estabelece-se uma distinção clara entre testes de componentes clássicos e *AI Component Testing*.

A ênfase na análise de falhas de transição (*Transition Failure Matrices* [15]) nos níveis de integração e sistema desloca a caracterização do erro de uma simples saída incorreta para uma falha no encadeamento lógico das decisões e ações do agente, refletindo uma perspectiva baseada no fluxo de execução do sistema.

A Tabela 2 detalha a taxonomia proposta.

3.2.1 Detalhamento. A taxonomia proposta não apenas adiciona novos níveis, mas reinterpreta os existentes para acomodar a natureza estocástica e a organização interna de seus módulos de decisão, planejamento, memória e ação. A seguir, detalha-se cada nível, destacando suas especificidades técnicas e diferenças em relação aos testes de IA tradicionais e aos testes de sistemas convencionais.

INPUT DATA TESTING Diferentemente do teste de dados tradicional, voltado à preparação de grandes conjuntos de dados para treinamento (por exemplo, limpeza, balanceamento e particionamento em treino, validação e teste), no contexto de sistemas agênticos este nível está centrado na avaliação do contexto fornecido ao modelo durante a inferência. Esse contexto pode ser composto por fragmentos recuperados em arquiteturas RAG, exemplos *few-shot* inseridos no prompt ou pela memória dinâmica do agente. Embora esses mecanismos sejam distintos em termos de implementação, todos compartilham a função de ampliar o estado informacional disponível ao modelo no momento da decisão. O objetivo deste nível é assegurar que tal contexto seja relevante, consistente e adequadamente formatado, reduzindo o risco de alucinações induzidas por informações ruidosas, contraditórias ou semanticamente pobres. Processos de *fine-tuning*, quando existentes, situam-se em um estágio anterior ao ciclo agêntico propriamente dito e não constituem o foco central deste nível, que se concentra na avaliação do contexto injetado em tempo de execução.

Table 1: Níveis de Teste segundo ISTQB

Nível de Teste	Para o CT-AI	Para o CTFL
Input Data Testing	Garantir a qualidade dos dados usados no treinamento e na previsão.	**N/A**
ML Model Testing	Verificar se o modelo de aprendizado atende aos critérios funcionais e não funcionais.	**N/A**
Component Testing	Verificar os componentes não baseados em IA, assegurando que funcionem corretamente de forma isolada.	Verificar partes isoladas do sistema, garantindo que cada componente funcione corretamente de forma independente.
Component Integration Testing	Garantir que componentes de IA e não-IA interajam adequadamente.	Avaliar a comunicação e interação entre os componentes.
System Testing	Validar o sistema completo, testando funções e características não funcionais em um ambiente próximo ao real.	Analisar o comportamento global do sistema completo, validando funcionalidades e características não funcionais.
System Integration Testing	**N/A**	Verificar as interações entre o sistema sob teste e outros sistemas externos ou serviços, garantindo compatibilidade e comunicação correta em um ambiente semelhante ao operacional.
Acceptance Testing	Confirmar se o sistema final atende aos requisitos do cliente e está pronto para uso.	Foca na validação e comprovação de que o sistema atende às necessidades do usuário e está pronto para ser lançado.

MODEL SELECTION TESTING Este nível possui caráter arquitetural e é raramente formalizado no desenvolvimento de software tradicional. Antes da construção do agente, deve-se validar se o modelo escolhido apresenta capacidades adequadas de raciocínio, seguimento de instruções e uso de ferramentas. Essa validação envolve o uso de *benchmarks* padronizados e testes comparativos que consideram trade-offs entre custo, latência e desempenho cognitivo.

Table 2: Taxonomia proposta

Nível de Teste	Definição Proposta	Práticas e Exemplos
Input Data Testing	Garantia da qualidade dos dados utilizados.	Processos manuais de curadoria e verificação.
Model Selection Testing	Verificação se o modelo base (LLM) atende aos critérios do projeto.	Processo manual de *benchmarking* e análise comparativa.
Component Testing	Verificação de módulos de código determinístico (não-IA).	Testes automatizados tradicionais.
AI Component Testing	Verificação de funcionalidades isoladas de IA (ex: uma *tool* específica).	Testar isoladamente uma *action* como "Create SQL Query".
Component Integration Testing	Garantia da interação entre componentes de IA e lógica de aplicação.	Uso de *Transition Failure Matrices*.
System Testing	Validação do comportamento do agente em cenários completos.	*Scenario Testing* e Análise de Erros.
System Integration Testing	Verificação de interações com sistemas externos (APIs, serviços).	*Scenario Testing* em ambientes integrados.
Acceptance Testing	Confirmação do alinhamento com requisitos do cliente.	Validação final de prontidão para uso.

Diferentemente do aprendizado de máquina tradicional, no qual o modelo é resultado de um processo de treinamento, em sistemas agênticos o modelo é um componente pronto que deve ser auditado previamente, definindo-se não apenas sua escolha, mas também suas configurações e parâmetros operacionais, como parâmetros de geração (por exemplo, *temperature* e *top-p*), limites de contexto e políticas de uso de ferramentas.

COMPONENT TESTING Este nível refere-se à validação dos módulos determinísticos de software que compõem o esqueleto estrutural do agente, como lógica de orquestração, conectores e parsers. São empregados testes unitários clássicos, técnicas de *mocking* para chamadas de API e verificações de tratamento de exceções. O foco é assegurar que os componentes não baseados em IA funcionem corretamente de forma isolada, garantindo que falhas observadas no comportamento global do sistema não sejam atribuídas a erros de implementação na infraestrutura de suporte ao agente.

AI COMPONENT TESTING Este nível representa a unidade lógica fundamental de teste em sistemas agênticos, deslocando o foco do agente como um todo para suas habilidades individuais. Em vez de validar o comportamento global, testam-se funções específicas do agente, como ferramentas ou módulos responsáveis por tarefas delimitadas. Por exemplo, em um agente que consulta bancos de dados, pode-se isolar o módulo responsável pela geração de consultas SQL e avaliá-lo por meio de um conjunto de prompts de teste, verificando se produz consultas sintaticamente válidas e semanticamente seguras. Essa abordagem alinha-se à visão de avaliação granular proposta por especialistas como Andrew Ng, distinguindo-se do ML Model Testing do ISTQB, que privilegia métricas estatísticas globais, ao enfatizar a competência semântica e operacional de funções específicas.

COMPONENT INTEGRATION TESTING Neste nível, o foco desloca-se para a interação entre o modelo de linguagem e as ferramentas externas. O principal risco não se limita à alucinação de conteúdo, mas inclui falhas no encadeamento lógico das ações, como a seleção incorreta de ferramentas ou o envio de argumentos em formatos incompatíveis. Para lidar com esse problema, empregam-se matrizes de falha de transição (*Transition Failure Matrices*), que permitem identificar em que etapa do encadeamento de decisões do agente ocorre a ruptura do comportamento esperado. O objetivo deixa de ser a eliminação de erros absolutos e passa a ser a redução sistemática das falhas de transição no grafo de execução do agente.

SYSTEM TESTING O System Testing avalia o agente em cenários completos de ponta a ponta (*end-to-end*), nos quais múltiplas ferramentas e etapas de raciocínio são encadeadas. Utilizam-se técnicas de *Scenario Testing* e *Error Analysis*, incluindo a simulação de diálogos prolongados para verificar a manutenção de contexto e a robustez frente a entradas adversariais. Esse nível valida não apenas requisitos funcionais, mas também características não funcionais, como latência percebida, estilo comunicacional e consistência das respostas, concentrando-se na coerência global da tarefa realizada.

SYSTEM INTEGRATION TESTING Neste estágio, verifica-se se o agente, ao interagir com sistemas externos e ambientes reais, opera conforme esperado sem produzir efeitos colaterais indesejados. Para isso, utilizam-se ambientes de *sandbox* para a execução segura de ferramentas com impacto no mundo real, bem como a verificação de contratos de API. O foco permanece na minimização de falhas de transição em integrações externas, assegurando compatibilidade técnica e semântica entre o agente e os sistemas com os quais se comunica.

ACCEPTANCE TESTING O Acceptance Testing corresponde à validação final pelo usuário ou *stakeholder*. São empregados testes A/B, mecanismos de *human-in-the-loop* e verificações de conformidade com políticas de segurança. Esse nível confirma se o sistema atende aos requisitos definidos e se encontra apto para uso em produção, funcionando como a última barreira de qualidade antes da implantação definitiva.

3.3 A pirâmide de testes em sistemas agênticos

Enquanto a taxonomia apresentada na Tabela 2 define quais níveis de teste compõem o ciclo de vida de um sistema agêntico, a "Pirâmide de Testes" define a estratégia de distribuição de esforço. O modelo clássico de Mike Cohn [7] preconiza uma base larga de testes unitários rápidos e baratos, diminuindo o volume conforme se sobe para testes de interface lentos e caros.

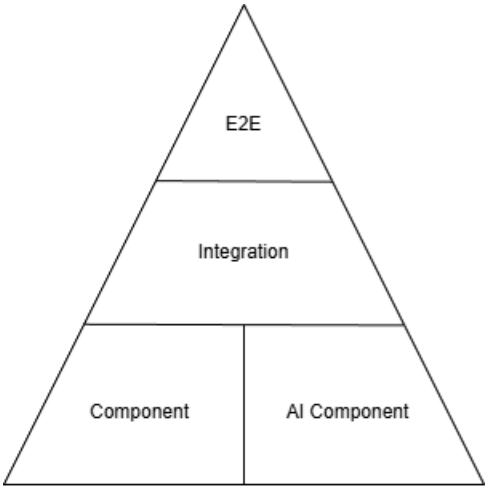

Figure 2: Pirâmide de testes em sistemas agênticos.

No contexto de Sistemas Agênticos, a pirâmide tradicional torna-se insuficiente devido à natureza híbrida da aplicação. Conforme ilustrado na Figura 2, propõe-se uma adaptação estrutural onde a base deixa de ser homogênea para acomodar a dualidade entre código determinístico e raciocínio probabilístico.

3.3.1 A Base Bipartida. A inovação mais crítica nesta proposta visual é a divisão da base da pirâmide em dois blocos distintos, que devem ser testados em paralelo, mas com técnicas diferentes:

- **Componentes Tradicionais (*Component*)**: Refere-se à infraestrutura de código "não-inteligente" que suporta o agente (parsers de saída, conectores de API, lógica de orquestração). Aqui, aplicam-se testes unitários convencionais (como por exemplo JUnit [18]), focados em velocidade de execução e cobertura de código. O objetivo é garantir que a estrutura de suporte do agente não falhe.
- **Componentes de IA (*AI Component*)**: Refere-se à validação isolada das "habilidades" cognitivas do agente (ex: um *tool call* específico ou uma etapa de raciocínio). Diferente dos testes unitários tradicionais, estes testes possuem maior latência (devido a chamadas a LLMs) e custo financeiro associado. A validação consiste em verificar semanticamente se a saída do modelo atende à intenção do prompt, isolando a incerteza do modelo das falhas de lógica de código.

Essa separação na base é fundamental para a manutenibilidade, pois deixa explícitas formas distintas de teste e interpretação de resultados: enquanto componentes tradicionais são avaliados por critérios determinísticos, com verificação binária de sucesso ou falha, os componentes de IA exigem avaliações semânticas ou probabilísticas, baseadas em níveis de adequação, consistência ou acerto das saídas, orientando estratégias diferentes de validação do sistema.

3.3.2 Camada Intermediária: A Convergência da Integração. Subindo na pirâmide, temos a camada de **Integration**. Conforme definido na Tabela 2, esta camada consolida dois níveis de teste críticos: o *Component Integration Testing* e o *System Integration Testing*.

Neste estágio, o foco deixa de ser a corretude isolada (validada na base) e passa a ser a **coesão do fluxo**. Em sistemas agênticos, a integração é onde ocorrem as "alucinações de fluxo", caracterizadas por desvios no encadeamento das ações do agente, como a seleção inadequada de ferramentas para um determinado passo, a proposição de ferramentas inexistentes ou a geração de parâmetros incompatíveis com a interface esperada, mesmo quando os componentes individuais funcionam corretamente. A estratégia de teste aqui deve priorizar a verificação de *Transition Failure Matrices*, avaliando tanto a corretude da escolha da ferramenta pelo LLM quanto o respeito aos contratos de interface entre o cérebro probabilístico (LLM) e o corpo determinístico (Ferramentas) seja respeitado.

3.3.3 O Topo da Pirâmide: Validação End-to-End (E2E). O topo da pirâmide, visualmente representado como **E2E**, engloba os níveis de *System Testing* e *Acceptance Testing* descritos na Tabela 2.

Esta camada deve ser a menor em volume devido ao alto custo computacional, alta latência e dificuldade de depuração (debugging). Um teste E2E em um sistema agêntico pode envolver múltiplos passos de raciocínio (multi-hop reasoning) e chamadas a ferramentas reais. Aqui, a validação é fenomenológica: observa-se se o comportamento emergente do agente resolveu o problema do usuário. É também nesta camada que se concentram as verificações de segurança e de limites do modelo, servindo como a última barreira de qualidade antes da interação humana real.

4 Ferramentas e Plataformas de Apoio à Avaliação

Enquanto o teste de software clássico opera sob a premissa de que uma entrada X deve produzir invariavelmente uma saída Y, os sistemas agênticos operam em um domínio probabilístico, exibindo comportamentos abertos (*open-ended*) e adaptativos que desafiam suítes de teste estáticas [29]. Como consequência, a validação destes sistemas exige mecanismos sistemáticos de observabilidade, rastreabilidade e avaliação semântica, capazes de lidar com a variabilidade inerente às saídas baseadas em modelos de linguagem.

É importante destacar que os módulos tradicionais (não baseados em IA), tais como parsers de saída, conectores de API e lógica de orquestração, continuam sendo adequadamente validados por ferramentas e técnicas clássicas de engenharia de software. As ferramentas especializadas discutidas nesta seção concentram-se, portanto, nos componentes e fluxos cuja saída depende de modelos de IA generativa, os quais demandam abordagens específicas de observabilidade, avaliação semântica e teste de segurança.

A implementação prática das estratégias de teste discutidas anteriormente depende, assim, de um conjunto de ferramentas que oferecem apoio sistemático ao processo de avaliação de sistemas agênticos. Dada a volatilidade do ecossistema de MLOps, esta revisão selecionou quatro soluções representativas que se estabeleceram como referências para observabilidade e segurança em 2024/2025. A seleção foi orientada pelo objetivo de realizar uma revisão de ferramentas e trabalhos acadêmicos que propõem técnicas e soluções para o domínio em estudo.

4.1 Plataformas de Observabilidade e Tracing

A natureza não-determinística dos LLMs exige que o *logging* tradicional seja substituído pelo *Tracing Distribuído*, permitindo a visualização completa da cadeia de execução (*prompts*, chamadas de ferramentas e respostas).

4.1.1 OpenTelemetry (OTel). O OpenTelemetry (OTel) é um padrão aberto para instrumentação e observabilidade de sistemas distribuídos, permitindo a coleta unificada de *traces* (rastros de execução), métricas e logs [6]. Seu objetivo é registrar, de forma estruturada, como uma requisição percorre os diferentes componentes de um sistema ao longo do tempo.

No modelo do OTel, um *trace* representa a execução completa de uma operação do ponto de vista do sistema, enquanto um *span* corresponde a uma etapa individual dessa execução, associada a um intervalo de tempo específico (por exemplo, uma chamada de função ou uma requisição a um serviço externo). Dessa forma, um *trace* é composto por múltiplos *spans* organizados hierarquicamente.

No contexto de sistemas agênticos, cada etapa relevante do ciclo de decisão do agente — como a geração de uma resposta pelo modelo ou a invocação de uma ferramenta externa — pode ser registrada como um *span* dentro de um mesmo *trace*. Essa instrumentação permite observar, de maneira estruturada, a sequência de ações executadas pelo agente, bem como medir a latência e o custo associados a cada chamada de ferramenta ou interação com o modelo.

Do ponto de vista arquitetural, o OTel é independente do modelo de linguagem ou do *framework* utilizado, fornecendo um formato padronizado para exportação dos dados de observabilidade. Isso possibilita que os registros coletados sejam enviados para diferentes *backends* de análise (por exemplo, sistemas de monitoramento ou plataformas de avaliação), reduzindo o acoplamento tecnológico e contribuindo para a longevidade da infraestrutura de testes e validação.

4.1.2 Arize Phoenix. O Arize Phoenix é uma plataforma de análise e observabilidade voltada especificamente para modelos de linguagem e sistemas baseados em IA generativa [3]. A ferramenta foi projetada para operar sobre a infraestrutura do OpenTelemetry, funcionando como uma camada de análise para os dados instrumentados. Enquanto de ferramentas de instrumentação (como OTel), que apenas coletam dados de execução, o Phoenix fornece capacidades analíticas especializadas para interpretar e visualizar esses dados com foco em comportamento semântico e desempenho de modelos.

Uma das funcionalidades centrais do Phoenix é a capacidade de comparar e agrupar interações e saídas do modelo por meio de projeções de vetores de alta dimensão, como UMAP (Uniform Manifold Approximation and Projection). Nesse contexto, cada interação ou saída gerada pelo agente é representada como um vetor num espaço de alta dimensão (o chamado *embedding*), e a projeção reduz essa representação para um espaço visualizável (por exemplo, tridimensional). Isso permite identificar padrões emergentes, tais como:

- **Clusters semânticos de falhas:** grupos de respostas que apresentam características semelhantes, indicando potencialmente modos recorrentes de erro (por exemplo, alucinações temáticas).
- **Deriva de dados (*drift*):** mudanças progressivas na distribuição das respostas do modelo em relação a um conjunto de referência (o *Golden Dataset*), que podem sinalizar degradação de desempenho ao longo do tempo.

Além da visualização vetorial, o Phoenix oferece painéis dedicados à inspeção de aspectos específicos de sistemas agênticos, como os fragmentos recuperados em arquiteturas RAG (*Retrieval-Augmented Generation*), facilitando a identificação de falhas de recuperação de contexto (*Retrieval Failure*).

4.2 Ferramentas de Segurança (Safety & Security)

Módulos baseados em IA, em especial aqueles que operam por meio de interfaces conversacionais, estão sujeitos a classes específicas de ataques que não se manifestam em sistemas determinísticos tradicionais. A exposição direta a entradas em linguagem natural amplia significativamente a superfície de ataque, possibilitando técnicas como injeção de prompt, executar ações proibidas e extração indevida de informações sensíveis.

Nesse contexto, a validação de segurança e robustez não pode depender apenas de verificações funcionais convencionais, exigindo o uso de ferramentas especializadas de varredura e simulação de ataques, que transitam da sondagem estática para abordagens dinâmicas e automatizadas.

4.2.1 Garak (LLM Vulnerability Scanner). Descrito por Derczynski et al. [8] como o "Nmap para LLMs", o **Garak** (*Generative AI Red-teaming and Assessment Kit*) é uma ferramenta de varredura de vulnerabilidades focada em sondagem (*probing*).

O Garak opera disparando milhares de *prompts* predefinidos (sondas) contra o modelo alvo, tentando eliciar comportamentos inseguros. Ele utiliza geradores de ataques (ex: injeção de prompt, toxicidade) e detectores automáticos para verificar se o ataque teve sucesso. Sua arquitetura extensível permite testar desde alucinação simples até vazamento de dados complexo, funcionando como uma primeira linha de defesa rápida na esteira de CI/CD.

4.2.2 Microsoft PyRIT (Python Risk Identification Toolkit). Enquanto o Garak foca em varredura ampla, o **PyRIT** foca em *Red Teaming* profundo e contextual. Apresentado por Lopez Munoz et al. [20], este framework introduz a automação de ataques de múltiplos turnos (*multi-turn*).

O diferencial do PyRIT é o uso de uma arquitetura de "Agente Atacante" (um LLM configurado para atacar) que conversa com o agente alvo. Isso permite simular ataques de engenharia social que exigem várias rodadas de diálogo para convencer o agente a liberar dados sensíveis ou executar ações proibidas (*jailbreak*). O framework integra um motor de pontuação (*Scoring Engine*) que classifica automaticamente o sucesso do ataque, permitindo a execução de campanhas de segurança em escala massiva sem revisão humana manual.

5 Métodos

A definição de uma taxonomia de testes e o estabelecimento de uma pirâmide de distribuição de esforço fornecem uma base organizacional para a avaliação da qualidade do sistema. No entanto, a execução tática da validação em sistemas agênticos exige métodos

específicos que superem as limitações das abordagens manuais e determinísticas tradicionais.

Biese [5] identifica que a prática predominante na indústria ainda é o teste manual *ad-hoc*, no qual desenvolvedores interagem com o sistema tentando encontrar falhas de forma orgânica. Essa abordagem é classificada como insustentável para sistemas agênticos devido a três limitações intrínsecas:

- **Cegueira de Cobertura (*Coverage Blindness*):** Testadores humanos tendem a validar apenas o "caminho feliz" ou cenários óbvios, negligenciando casos de borda e comportamentos adversariais.
- **Invisibilidade de Regressão:** Em sistemas probabilísticos, uma pequena alteração no *prompt* do sistema pode modificar comportamentos em cenários não relacionados. Testes manuais raramente detectam essas regressões laterais antes do *deploy*.
- **Impossibilidade de Escala:** A explosão combinatória de conversas de múltiplos turnos (*multi-turn*) torna inviável a verificação humana de todos os caminhos possíveis de diálogo dentro de um tempo hábil.

Diante dessas limitações, a validação de agentes baseados em LLM não pode depender exclusivamente de inspeções intuitivas (coloquialmente referidas como *"vibe checks"*), que são não escaláveis e sujeitas a viés de confirmação. Xia et al. [29] argumentam que a avaliação deve evoluir de um estágio terminal para uma função governante contínua, conceito denominado **EDDOps** (*Evaluation-Driven Development and Operations*).

Esta seção detalha os métodos contemporâneos para a execução de testes em IA Generativa, transitando de verificações *ad-hoc* para fluxos de trabalho sistemáticos que unificam a avaliação *offline* (desenvolvimento) e *online* (operação) em um ciclo de feedback fechado.

5.1 Eval-based Development e Análise de Erros

O desenvolvimento de sistemas agênticos consolidou um novo paradigma metodológico: o *Eval-driven Development* (EDD) [29]. Diferentemente do *Test-Driven Development* (TDD) [4] tradicional, focado na correção lógica de componentes isolados, o EDD foca na avaliação do comportamento emergente do sistema frente a objetivos subespecificados, isto é, metas descritas em nível abstrato, sem a definição explícita de todos os passos de execução. Por exemplo, ao receber um objetivo como "analisar um conjunto de documentos e produzir um relatório executivo", o agente decide autonomamente como decompor a tarefa, quais ferramentas utilizar e como integrar os resultados, fazendo com que o comportamento observado surja da interação entre modelo, *prompts* e arquitetura. Segundo o modelo de processo proposto por Xia et al. [29], o desenvolvimento de agentes é estruturado como um ciclo iterativo orientado por avaliações, no qual os resultados dos *evals* são utilizados como insumos para decisões de engenharia, como o refinamento de *prompt*, a reorganização do fluxo agêntico e ajustes na arquitetura do sistema.

Neste contexto, a unidade fundamental de trabalho deixa de ser apenas o caso de teste unitário e passa a ser o "Eval". Husain e Shankar [15] definem formalmente um *Eval* como a composição de três elementos:

(1) Um ***dataset*** de entradas;
(2) O **sistema ou módulo alvo** (*target*);
(3) Um **avaliador** (*scorer*) que quantifica a qualidade da saída.

5.1.1 Superando o "Vibe Check". Na fase inicial de desenvolvimento, é comum que engenheiros dependam do que Husain e Shankar [15] classificam como *"Vibe Check"*: a inspeção manual de algumas saídas para aferir qualitativamente se o sistema "parece" estar funcionando. Embora útil para prototipagem rápida, este método torna-se insustentável à medida que a complexidade do agente escala, devido à falta de rastreabilidade e à impossibilidade de detectar regressões sutis.

Para profissionalizar o ciclo, a literatura técnica e acadêmica converge para a formalização de *Golden Datasets*, conjuntos de dados curados contendo entradas representativas e, quando aplicável, a saída ideal (*ground truth*) ou critérios de sucesso explícitos. O processo de EDD exige que, antes de otimizar o código ou o *prompt* do agente, o engenheiro estabeleça uma *pipeline* de avaliação automatizada, frequentemente utilizando a técnica de *"LLM-as-a-Judge"* para atuar como o *scorer* em larga escala, substituindo a intuição humana por métricas observáveis.

5.1.2 A Metodologia de Análise de Erros. Métricas agregadas (como "85% de acurácia") se comportam como indicadores resultado (*lagging indicators*), pois descrevem o desempenho observado do sistema, mas oferecem pouca capacidade explicativa sobre as causas das falhas ou sobre como melhorá-lo. Conforme argumentado por Husain e Shankar [15], bons escores globais não implicam necessariamente que o sistema funcione adequadamente em cenários reais, o que torna a **Análise de Erros Qualitativa** uma prática central no desenvolvimento de sistemas agênticos.

Este método consiste na inspeção granular dos *traces* (registros detalhados de execução que incluem passos de raciocínio, chamadas de ferramentas e retornos de API). Ao identificar uma falha nos testes, o engenheiro não deve apenas corrigir o código, mas categorizar a falha para atacar a causa raiz. Com base na taxonomia de falhas discutida por Husain [15], os erros em agentes geralmente derivam de:

- **Déficit de Contexto:** O agente não possuía a informação necessária para responder (geralmente uma falha de recuperação/RAG).
- **Incapacidade do Modelo:** O modelo base falhou em seguir instruções complexas ou raciocinar sobre o contexto fornecido (indicando necessidade de melhoria no *prompt* ou troca de modelo).
- **Falha de Ferramenta:** O agente alucinou parâmetros inválidos ou utilizou a ferramenta incorreta para a tarefa.

O ciclo de *EDDOps* se fecha quando essa análise de erros resulta em novos casos de teste adicionados ao *Golden Dataset*, prevenindo a recorrência da falha em iterações futuras [29].

5.2 Avaliação de RAG (Retrieval-Augmented Generation)

A validação de sistemas agênticos frequentemente depende da capacidade do modelo de recuperar e utilizar conhecimento externo de forma confiável. Diferentemente de modelos puramente generativos, onde a avaliação foca na fluência ou criatividade, sistemas RAG exigem métricas que auditem a integridade factual e a precisão

da recuperação. A literatura atual divide a avaliação de RAG em duas abordagens principais: Avaliação Independente e Avaliação Ponta-a-Ponta (End-to-End) [12].

5.2.1 Paradigmas de Avaliação. A **Avaliação Independente** analisa os módulos de recuperação e geração separadamente. Para o módulo de recuperação, avalia-se a qualidade dos documentos recuperados utilizando métricas clássicas de sistemas de busca, como *Hit Rate*, *Mean Reciprocal Rank* (MRR) e *Precision*, com o objetivo de medir a capacidade do sistema em filtrar ruído e priorizar documentos relevantes [12]. Já para o módulo de geração, avalia-se a capacidade do LLM de sintetizar uma resposta coerente dado um contexto injetado, isolando-o de falhas de recuperação.

Por outro lado, a **Avaliação Ponta-a-Ponta** examina a resposta final gerada pelo sistema para uma entrada do usuário. Tradicionalmente, métricas baseadas em sobreposição de n-gramas eram utilizadas, mas estas têm se mostrado insuficientes para RAG, pois não capturam a precisão semântica e factual necessária para aplicações agênticas [9].

5.2.2 Métricas "Reference-Free" e o Framework Ragas. Diante da escassez de *Ground Truth* (respostas de referência anotadas por humanos) em ambientes de produção, consolidou-se o uso de métricas "Reference-Free", calculadas através do paradigma *LLM-as-a-Judge* (onde um LLM atua como avaliador). O framework **Ragas** (*Retrieval Augmented Generation Assessment*) estabelece três métricas fundamentais para auditar a tríade *Query-Context-Answer* [9], na qual *Query* corresponde à pergunta do usuário, *Context* ao conjunto de documentos recuperados pelo mecanismo de busca, e *Answer* à resposta final gerada pelo modelo.:

- **Fidelidade (Faithfulness):** Mede se a resposta gerada é factualmente consistente com o contexto recuperado, visando detectar alucinações. O cálculo envolve decompor a resposta em afirmações atômicas e verificar se cada uma pode ser inferida a partir do contexto [9]. Uma pontuação baixa indica que o agente está gerando informações não fundamentadas nos dados fornecidos [12].
- **Relevância da Resposta (Answer Relevance):** Avalia se a resposta aborda diretamente a pergunta do usuário, penalizando respostas incompletas ou redundantes. O Ragas calcula isso gerando perguntas potenciais a partir da resposta e medindo a similaridade de cosseno entre estas e a pergunta original [9].
- **Relevância do Contexto (Context Relevance):** Quantifica a relação sinal-ruído nos documentos recuperados. O objetivo é garantir que o contexto contenha apenas as informações necessárias para responder à pergunta, minimizando custos computacionais e o risco de o modelo degradar performance com textos excessivamente longos [9].

Adicionalmente, destaca-se a importância do **Context Recall** como uma métrica suplementar, que mede a capacidade do modelo de recuperar todas as informações relevantes necessárias para responder a uma questão, alinhando-se a relatórios técnicos recentes da OpenAI [12].

5.3 Scenario Testing: Simulação de Agentes em Escala

Enquanto a avaliação baseada em métricas quantifica a qualidade de respostas atômicas (como na avaliação de RAG), a validação de fluxos conversacionais longos e de comportamentos emergentes exige métodos que considerem o comportamento integrado do sistema ao longo de múltiplos turnos. Nesse contexto, Biese [5] propõe o paradigma de ***Scenario Testing***, no qual agentes de IA são utilizados para testar outros agentes por meio da simulação sistemática de cenários realistas.

Diferentemente de testes unitários ou avaliações isoladas de componentes, o *Scenario Testing* tem como objetivo principal validar o comportamento global do agente em jornadas completas de interação com o usuário, avaliando se critérios funcionais e não funcionais são satisfeitos ao longo do diálogo. Trata-se, portanto, de um método especialmente adequado para sistemas conversacionais baseados em múltiplos turnos e uso de ferramentas.

No contexto da taxonomia proposta neste trabalho, o *Scenario Testing* aplica-se predominantemente aos níveis de **System Testing** e **System Integration Testing**, nos quais se busca verificar a correta integração entre raciocínio do modelo, uso de ferramentas e manutenção de estado conversacional.

5.3.1 Arquitetura de Simulação Multi-Agente. A implementação técnica do *Scenario Testing* baseia-se em uma arquitetura de simulação composta por três entidades distintas interagindo em um ambiente controlado, conforme descrito por Biese [5]:

Agent Under Test (AUT): O sistema agêntico alvo da validação. Ele opera sem saber que está em uma simulação, respondendo conforme sua programação original.

User Simulator Agent: Um agente configurado com uma "Persona" específica e um objetivo claro (ex: "Você é um cliente impaciente tentando obter um reembolso fora do prazo"). Este agente é instruído a mimetizar comportamentos humanos, incluindo erros de digitação, ambiguidades e resistência a sugestões, gerando uma pressão realista sobre o AUT.

Judge Agent: Um terceiro agente (geralmente um modelo mais robusto, como GPT-4o ou Claude 3.5 Sonnet) que observa a transcrição da interação. Diferente das métricas simples de similaridade, o Juiz avalia critérios subjetivos definidos no cenário, como "O agente manteve a empatia?" ou "O agente negou o reembolso seguindo a política da empresa?".

5.4 Red Teaming e Segurança: Validação de Escopo Negativo

Enquanto os testes funcionais e de integração verificam se o agente executa corretamente o que foi programado para fazer (*Escopo Positivo*), a validação de segurança exige uma inversão fundamental de perspectiva para o **Escopo Negativo**: verificar se o agente é robusto o suficiente para **não** executar ações perigosas ou não autorizadas.

Diante desse cenário, são necessários testes especializados, capazes de simular comportamentos adversariais direcionados a essas vulnerabilidades próprias de agentes. A literatura contemporânea converge para a prática de *Red Teaming* (Time Vermelho), uma

metodologia de simulação adversarial destinada a sondar e explorar falhas de segurança antes que atores maliciosos o façam.

5.4.1 A Necessidade de Automação no Red Teaming. Historicamente, o *Red Teaming* dependia de especialistas humanos tentando "quebrar" o modelo manualmente. Ganguli et al. [11] demonstraram que, embora o treinamento com reforço humano (RLHF) torne os modelos mais resistentes a ataques casuais, a abordagem baseada exclusivamente na criatividade humana é difícil de escalar e apresenta retornos decrescentes do ponto de vista defensivo, pois cada novo caso identificado exige esforço crescente para cobrir regiões cada vez menores do espaço de ataque.

Mais recentemente, a introdução de frameworks como o **PyRIT** (*Python Risk Identification Toolkit*) pela Microsoft [20] sinaliza a transição para o *Red Teaming Automatizado*. O PyRIT propõe uma arquitetura onde um "Agente Atacante" (*Red Teaming Bot*) gera milhares de permutações de *prompts* maliciosos contra o "Agente Alvo", enquanto um "Motor de Pontuação" (*Scoring Engine*) avalia automaticamente se o ataque teve sucesso. Essa automação é crucial para cobrir vetores de ataque complexos, como vazamento de dados em conversas de múltiplos turnos (*multi-turn*), que seriam inviáveis de testar manualmente em larga escala.

5.4.2 Taxonomia de Riscos e Técnicas Adversariais. A validação de segurança em agentes deve cobrir, no mínimo, três categorias críticas de vulnerabilidade, conforme evidenciado pelos estudos analisados:

- **Jailbreaking e Ataques Adversariais Universais:** O objetivo do *Jailbreaking* é contornar os mecanismos de alinhamento ético do modelo. Zou et al. [30] revelaram uma fragilidade alarmante nos LLMs modernos: a existência de "sufixos adversariais universais". Através de otimização baseada em gradiente (*Greedy Coordinate Gradient*), é possível descobrir sequências de caracteres aparentemente sem sentido que, quando anexadas a um *prompt* proibido (ex: "como construir uma bomba"), induzem o modelo a obedecer à instrução, ignorando seus treinos de segurança. A validação, portanto, não pode confiar apenas em listas estáticas de palavras proibidas.
- **Prompt Injection (Injeção de Prompt):** Em sistemas agênticos, a injeção visa sobrescrever as instruções do sistema (*System Prompt*) para sequestrar o controle do fluxo. O framework PyRIT destaca a importância de testar tanto injeções diretas (do usuário) quanto indiretas (onde o agente ingere conteúdo malicioso de um documento externo) [20].
- **PII Leakage (Vazamento de Dados):** A capacidade de um agente reter contexto abre riscos de engenharia social. Testes de *PII Leakage* verificam se o agente pode ser persuadido a revelar informações sensíveis (chaves de API, e-mails, CPFs) contidas em sua base de conhecimento (RAG). O critério de sucesso deste teste é a detecção automática de padrões de dados sensíveis na resposta do agente pelo *Scoring Engine*.

5.4.3 O Critério de Sucesso: Recusa (Refusal). No contexto de testes de segurança, o comportamento esperado para uma entrada maliciosa é a **Recusa** (*Refusal*). O agente deve identificar a violação de suas diretrizes e responder com uma negativa neutra.

A métrica de qualidade, portanto, não é apenas a taxa de recusa, mas o equilíbrio entre Segurança e Utilidade. Ganguli et al. [11] alertam para o risco de modelos excessivamente defensivos que recusam solicitações inócuas (*False Refusals*). Assim, uma suíte de testes de segurança robusta deve incluir tanto *prompts* adversariais quanto *prompts* limítrofes legítimos, garantindo que os *guardrails* não inutilizem a funcionalidade do agente.

5.4.4 Scenario Testing e Red Teaming Automatizado. A principal vantagem da metodologia proposta é a capacidade de paralelizar a execução de cenários adversariais por meio de *Scenario Testing*. Diferentemente de casos de teste manuais e pontuais, a arquitetura baseada em simulação permite a injeção sistemática de cenários complexos e raros, que dificilmente seriam concebidos por testadores humanos, mas que representam alto risco em ambientes de produção.

Pode-se configurar o *User Simulator* para atuar explicitamente como um atacante (*Red Teaming*), gerando *prompts* maliciosos com o objetivo de realizar injeção de *prompt*, induzir *jailbreaks* ou extrair dados sensíveis do sistema sob teste. Caso o AUT (Agent Under Test) falhe em acionar seus mecanismos de proteção (*guardrails*), o *Judge Agent* identifica automaticamente a violação e sinaliza o cenário como falho para posterior análise humana.

Esse arranjo transforma a validação de segurança de um processo essencialmente manual e linear em uma operação de engenharia contínua, automatizada e escalável [5]. Em termos taxonômicos, esse tipo de teste posiciona-se no nível de *System Testing*, pois avalia o comportamento emergente do agente em cenários completos de ponta a ponta, incluindo diálogos multi-turno e uso de ferramentas externas.

Sob a ótica de escopo, o *Scenario Testing* adversarial materializa a validação do **Escopo Negativo**, pois o objetivo não é confirmar que o agente executa corretamente suas funções previstas, mas verificar se ele é capaz de recusar ações perigosas ou não autorizadas. Assim, os cenários deixam de representar apenas trajetórias funcionais esperadas e passam a incorporar trajetórias proibidas, cobrindo explicitamente classes de risco como *Prompt Injection*, *Jailbreaking* e *PII Leakage*.

Dessa forma, o *Scenario Testing* não se limita a validar requisitos funcionais, mas torna-se um instrumento sistemático de engenharia de segurança para sistemas agênticos, integrando práticas de *Red Teaming* ao ciclo regular de testes.

6 Conclusão

Este trabalho analisou a validação de sistemas agênticos baseados em LLMs. Dada a natureza híbrida (determinística e probabilística) desses sistemas, conclui-se que as práticas tradicionais de teste, isoladamente, são insuficientes.

6.1 Contribuições

O estudo oferece um arcabouço para aplicações mais confiáveis, destacando-se: (i) organização conceitual sobre o impacto de arquiteturas agênticas na validação; (ii) taxonomia adaptada ao CT-AI; (iii) uma pirâmide de testes bipartida; (iv) revisão de métodos; e (v) análise de ferramentas.

6.2 Limitações e Trabalhos Futuros

As limitações incluem o caráter conceitual sem validação empírica e a rápida obsolescência das ferramentas de IA. Trabalhos futuros devem focar em estudos de caso industriais, métricas para fluxos *multi-step*, e na integração entre observabilidade e segurança. Em suma, a confiabilidade agêntica exige evoluir dos testes tradicionais para a gestão contínua do comportamento probabilístico.

References

[1] Bhashithe Abeysinghe and Ruhan Circi. 2024. The Challenges of Evaluating LLM Applications: An Analysis of Automated, Human, and LLM-Based Approaches. arXiv:2406.03339 [cs.CL] https://arxiv.org/abs/2406.03339

[2] Anthropic. 2024. *Building Effective Agents.* Retrieved January 23, 2026 from https://www.anthropic.com/research/building-effective-agents

[3] Arize AI. [n. d.]. *Phoenix: AI Observability Evaluation.* Retrieved January 23, 2026 from https://arize.com/docs/phoenix

[4] Kent Beck. 2002. *Test-Driven Development: By Example.* Addison-Wesley, Boston, MA.

[5] Pascal Biese. 2025. *Scenario Testing: A New Paradigm for Making AI Agents More Reliable.* Retrieved January 23, 2026 from https://www.llmwatch.com/p/scenario-testing-a-new-paradigm-for

[6] CNCF. 2025. *OpenTelemetry Documentation.* Retrieved January 23, 2026 from https://opentelemetry.io/docs/

[7] Mike Cohn. 2009. *Succeeding with Agile: Software Development Using Scrum.* Addison-Wesley.

[8] Leon Derczynski, Erick Galinkin, Jeffrey Martin, Subho Majumdar, and Nanna Inie. 2024. garak: A Framework for Security Probing Large Language Models. arXiv:2406.11036 [cs.CL] https://arxiv.org/abs/2406.11036

[9] Shahul Es, Jithin James, Luis Espinosa-Anke, and Steven Schockaert. 2025. Ragas: Automated Evaluation of Retrieval Augmented Generation. arXiv:2309.15217 [cs.CL] https://arxiv.org/abs/2309.15217

[10] Mohamed Amine Ferrag, Fatima Alwahedi, Ammar Battah, Bilel Cherif, Abdechakour Mechri, Norbert Tihanyi, Tamas Bisztray, and Merouane Debbah. 2025. Generative AI in cybersecurity: A comprehensive review of LLM applications and vulnerabilities. *Internet of Things and Cyber-Physical Systems* 5 (2025), 1–46. doi:10.1016/j.iotcps.2025.01.001

[11] Deep Ganguli, Liane Lovitt, Jackson Kernion, Amanda Askell, Yuntao Bai, Saurav Kadavath, Ben Mann, Ethan Perez, Nicholas Schiefer, Kamal Ndousse, Andy Jones, Sam Bowman, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Nelson Elhage, Sheer El-Showk, Stanislav Fort, Zac Hatfield-Dodds, Tom Henighan, Danny Hernandez, Tristan Hume, Josh Jacobson, Scott Johnston, Shauna Kravec, Catherine Olsson, Sam Ringer, Eli Tran-Johnson, Dario Amodei, Tom Brown, Nicholas Joseph, Sam McCandlish, Chris Olah, Jared Kaplan, and Jack Clark. 2022. Red Teaming Language Models to Reduce Harms: Methods, Scaling Behaviors, and Lessons Learned. arXiv:2209.07858 [cs.CL] https://arxiv.org/abs/2209.07858

[12] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2024. Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv:2312.10997 [cs.CL] https://arxiv.org/abs/2312.10997

[13] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish

Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. 2024. The Llama 3 Herd of Models. arXiv:2407.21783 [cs.AI] https://arxiv.org/abs/2407.21783

[14] Daya Guo, Dejian Yang, et al. 2025. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature* 645, 8081 (Sept. 2025), 633–638. doi:10.1038/s41586-025-09422-z

[15] Hamel Husain and Shreya Shankar. 2026. *LLM Evals: Everything You Need to Know.* Retrieved January 23, 2026 from https://hamel.dev/blog/posts/evals-faq

[16] International Software Testing Qualifications Board (ISTQB). 2021. *Certified Tester AI Testing (CT-AI) Syllabus v1.0.* Retrieved January 23, 2026 from https://istqb.org

[17] International Software Testing Qualifications Board (ISTQB). 2024. *Certified Tester Foundation Level Syllabus v4.0.1.* Retrieved January 29, 2026 from https://istqb.org

[18] JUnit Team. 2017. *JUnit 5 User Guide.* JUnit Project. Retrieved February 06, 2026 from https://junit.org/junit5/docs/current/user-guide/

[19] Samantha Lock. 2022. What is AI chatbot phenomenon ChatGPT and could it replace humans? Retrieved February 16, 2026 from https://www.theguardian.com/technology/2022/dec/05/what-is-ai-chatbot-phenomenon-chatgpt-and-could-it-replace-humans

[20] Gary D. Lopez Munoz, Amanda J. Minnich, Roman Lutz, Richard Lundeen, Raja Sekhar Rao Dheekonda, Nina Chikanov, Bolor-Erdene Jagdagdorj, Martin Pouliot, Shiven Chawla, Whitney Maxwell, Blake Bullwinkel, Katherine Pratt, Joris de Gruyter, Charlotte Siska, Pete Bryan, Tori Westerhoff, Chang Kawaguchi, Christian Seifert, Ram Shankar Siva Kumar, and Yonatan Zunger. 2024. PyRIT: A Framework for Security Risk Identification and Red Teaming in Generative AI System. arXiv:2410.02828 [cs.CR] https://arxiv.org/abs/2410.02828

[21] Andrew Ng. 2024. What's next for AI agentic workflows. Video by Sequoia Capital. Retrieved January 23, 2026 from https://www.youtube.com/watch?v=sal78ACtGTc

[22] OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. 2024. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL] https://arxiv.org/abs/2303.08774

[23] Ipek Ozkaya. 2023. Application of Large Language Models to Software Engineering Tasks: Opportunities, Risks, and Implications. *IEEE Software* 40, 3 (2023), 4–8. doi:10.1109/MS.2023.3248401

[24] Gemini Team, Rohan Anil, et al. 2025. Gemini: A Family of Highly Capable Multimodal Models. arXiv:2312.11805 [cs.CL] https://arxiv.org/abs/2312.11805

[25] Jose Calvo-Manzano Villalón, Gonzalo Cuevas Agustin, Tomás San Feliu Gilabert, and José de Jesús Jiménez Puello. 2015. A taxonomy for software testing projects. In *2015 10th Iberian Conference on Information Systems and Technologies (CISTI).* 1–6. doi:10.1109/CISTI.2015.7170545

[26] Junjie Wang, Yuchao Huang, Chunyang Chen, Zhe Liu, Song Wang, and Qing Wang. 2024. Software Testing with Large Language Models: Survey, Landscape, and Vision. arXiv:2307.07221 [cs.SE] https://arxiv.org/abs/2307.07221

[27] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science* 18, 6 (March 2024). doi:10.1007/s11704-024-40231-1

[28] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, Zhangyue Yin, Shihan Dou, Rongxiang Weng, Wensen Cheng, Qi Zhang, Wenjuan Qin, Yongyan Zheng, Xipeng Qiu, Xuanjing Huang, and Tao Gui. 2023. The Rise and Potential of Large Language Model Based Agents: A Survey. arXiv:2309.07864 [cs.AI] https://arxiv.org/abs/2309.07864

[29] Boming Xia, Qinghua Lu, Liming Zhu, Zhenchang Xing, Dehai Zhao, and Hao Zhang. 2025. Evaluation-Driven Development and Operations of LLM Agents: A Process Model and Reference Architecture. arXiv:2411.13768 [cs.SE] https://arxiv.org/abs/2411.13768

[30] Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J. Zico Kolter, and Matt Fredrikson. 2023. Universal and Transferable Adversarial Attacks on Aligned Language Models. arXiv:2307.15043 [cs.CL] https://arxiv.org/abs/2307.15043