Teste de Mutação em System Prompts para Avaliação de Conjuntos de Testes de Chatbots Baseados em LLMs

Carlos E. C. Braga
Instituto de Informática
Universidade Federal de Goiás
Goiânia, Brasil
carlos.braga@discente.ufg.br

Sofia L. C. Paiva
Instituto de Informática
Universidade Federal de Goiás
Goiânia, Brasil
sofialarissa@ufg.br

Resumo

Ao testar cenários conversacionais em *chatbots* baseados em *Large Language Models* (LLMs), avaliar a adequação de um conjunto de testes é uma tarefa desafiadora, especialmente quando o comportamento do sistema é definido por linguagem natural (no *system prompt*) e não estritamente por código. Este trabalho propõe a aplicação do teste de mutação a *chatbots* configurados integralmente por um *system prompt*: os operadores de mutação atuam diretamente sobre o prompt, alterando regras e restrições para gerar variantes defeituosas (mutantes) do *chatbot*. A adequação do conjunto de testes é quantificada por um escore de mutação, apurado com um oráculo duplo que combina verificação determinística (Botium) e semântica (*LLM-as-a-judge*). A abordagem é demonstrada em um estudo de caso com um *chatbot* de agendamento de exames médicos. Os resultados evidenciam que o escore de mutação bruto (0,86) superestima a adequação do conjunto de testes: ao inspecionar as justificativas do oráculo, apenas 46 das 103 mortes (44,7%) são coerentes com o defeito injetado, o que reduz o escore para 0,38. Conclui-se que, em sistemas não-determinísticos configurados por linguagem natural, o escore de mutação deve ser interpretado à luz de uma análise de coerência das mortes, e não isoladamente.

Palavras-chave

Teste de Mutação, Operadores de Mutação, *Chatbots*

1 Introdução

Chatbots baseados em *Large Language Models* (LLMs) têm seu comportamento definido por um *system prompt* (um texto em linguagem natural que descreve regras, restrições e etapas do atendimento). Nesse cenário, avaliar se um conjunto de testes é adequado, isto é, se ele é sensível o suficiente para detectar desvios de comportamento do *chatbot*, torna-se um problema em aberto [10]. Um conjunto que apenas confirma o comportamento almejado, sem evidenciar quando o sistema se desvia do esperado, diz pouco sobre a qualidade do sistema testado.

Medir essa adequação, contudo, esbarra em peculiaridades do teste de *chatbots* baseados em LLMs quando comparado ao de sistemas de software tradicionais [7]. Raramente existe uma única saída esperada correta, já que diferentes formulações de respostas podem ser igualmente válidas. Além disso, o teste de uma entrada em um *chatbot* baseado em LLM nem sempre irá gerar a mesma saída, já que a natureza dos LLMs é não-determinística [2]. A ausência de um gabarito e a variabilidade das respostas para cada execução quebram pressupostos centrais de testes convencionais, nos quais se compara a saída observada a um valor esperado fixo. Como consequência, avaliar automaticamente se as respostas obtidas nos testes estão corretas se torna um problema [27], e qualquer critério de adequação para esses conjuntos de testes precisa ser construído sobre essa base instável.

Propõe-se a aplicação do teste de mutação a *chatbots* configurados por meio de um *system prompt*, de forma que, em vez de modificar o código-fonte, como acontece nas abordagens convencionais, os operadores de mutação atuam diretamente sobre o prompt, alterando as regras, as restrições e os comportamentos definidos nele. Cada alteração dá origem a uma variante defeituosa do *chatbot*, chamada mutante. Espera-se que um conjunto de testes eficaz seja capaz de detectar os defeitos inseridos artificialmente e que os testes passem no sistema sem apresentar falhas. Quando um teste detecta o comportamento inesperado introduzido por alguma mutação, diz-se que esse mutante foi morto. A razão entre mutantes mortos e o total de mutantes gerados mede quantitativamente a adequação dos testes, e é chamada de escore de mutação.

Para aplicar o teste de mutação nesse contexto, faz-se necessário um sistema de avaliação dos testes que funcione para respostas em linguagem natural e sem saída de referência fixa. Para cumprir esse requisito, neste trabalho foi utilizada a verificação com dois oráculos complementares: o Botium [4] e um *LLM-as-a-judge*. O Botium atua verificando propriedades determinísticas, como a utilização de expressões regulares para avaliar se termos esperados na resposta realmente se fazem presentes. O *LLM-as-a-judge* atua avaliando a conversa completa e julgando se o comportamento observado corresponde de fato ao esperado, em outras palavras, avaliando a qualidade semântica (*multi-turn judge*) [25, 26].

A abordagem é demonstrada em um estudo de caso sobre um *chatbot* para o agendamento de exames médicos, que é configurado por um *system prompt*. Sobre o sistema, são aplicados os operadores de mutação ao *prompt* (gerando versões do sistema com defeitos artificiais), é executado o conjunto de testes para cada mutante e é calculado o escore de mutação resultante.

Para guiar a avaliação empírica dessa proposta, este trabalho é orientado pelas seguintes questões de pesquisa (**RQs**):

- **RQ1.** O teste de mutação aplicado diretamente a *system prompts* permite avaliar a adequação de conjuntos de testes para *chatbots* baseados em LLMs?
- **RQ2.** Como o não-determinismo do sistema sob teste e do avaliador afeta a aplicação dessa abordagem?

Este artigo está organizado da seguinte forma: a Seção 2 apresenta os principais conceitos utilizados; a Seção 3 apresenta os trabalhos relacionados a esta pesquisa; a Seção 4 descreve a abordagem proposta para a condução de testes de mutação em *chatbots* baseados em LLMs configurados por meio de *system prompts*; a Seção 5 relata a implementação e os resultados dessa abordagem;

e a Seção 6 apresenta as conclusões, responde às questões de pesquisa (**RQs**), discute as limitações e aponta direções para trabalhos futuros.

2 Fundamentação Teórica

2.1 Chatbots Baseados em LLMs e System Prompts

Os *chatbots* baseados em LLMs diferem das abordagens baseadas em fluxos de diálogos (*dialog flows*). Plataformas como o Dialogflow [13] e Rasa [24] exigem que os desenvolvedores definam explicitamente as intenções do usuário, as entidades, as respostas associadas e os fluxos de diálogos. Em contraste, os *chatbots* baseados em LLMs produzem respostas diretamente a partir de um modelo de grande escala, sendo o *system prompt* o mecanismo central de configuração.

O *system prompt* é um texto em linguagem natural que instrui o modelo sobre como se comportar, quais regras seguir, quais informações fornecer e quais requisições recusar [23]. Ele é enviado ao modelo antes da conversa com o usuário, e não possui uma semântica formal (sua interpretação depende da compreensão de linguagem natural do modelo utilizado).

2.2 Teste de Mutação

O teste de mutação é uma técnica de teste de software baseada em defeitos, proposta originalmente por DeMillo, Lipton e Sayward [11]. A técnica consiste em fazer mutações do artefato sob teste por meio da introdução artificial de pequenas alterações nele, gerando variantes defeituosas do sistema chamadas mutantes.

Um operador de mutação é uma regra que especifica um tipo de transformação a ser aplicada ao artefato original, e esses operadores são responsáveis por gerar cada mutação. Em sistemas tradicionais, os operadores de mutação são normalmente projetados para simular os tipos de erros mais comuns cometidos pelos programadores, como a substituição de operadores aritméticos, a alteração de condições lógicas e a remoção de instruções [17]. Cada operador é aplicado ao artefato original de maneira independente, de modo que cada mutante difere do original por uma alteração gerada por um único operador de mutação.

A técnica parte da hipótese do programador competente (*competent programmer hypothesis*), que postula que os programadores experientes tendem a produzir artefatos quase corretos [11], cometendo pequenos desvios em relação ao que se almejava produzir. Analogamente a essa hipótese, o responsável pela redação do *system prompt* de um *chatbot* também pode produzir um artefato quase correto, cometendo desvios pontuais durante a elaboração ou manutenção desse artefato. Os operadores de mutação propostos neste trabalho simulam esses possíveis desvios.

Para todo mutante gerado, executa-se o conjunto de testes. Se pelo menos um caso de teste detecta o comportamento anômalo introduzido pela mutação, considera-se morto o mutante. Caso o conjunto de testes não falhe, o mutante sobrevive. A métrica central derivada do processo é o escore de mutação, definido como:

$$\text{escore de mutação} = \frac{\text{mutantes mortos}}{\text{total de mutantes gerados}}$$

O escore de mutação quantifica a qualidade do conjunto de testes, e o objetivo da análise de mutação é fazer com que ele chegue a 1,00, pois isso indicaria que os testes são suficientes para detectar todos os defeitos gerados pelos mutantes [17].

Um desafio inerente ao teste de mutação é o problema dos mutantes equivalentes. Esse problema acontece quando há mutantes cujas alterações produzem um comportamento semanticamente idêntico ao do sistema original para toda entrada possível e que, portanto, não podem ser mortos por nenhum conjunto de testes [17]. Esses mutantes, ao serem incluídos no denominador do escore de mutação, irão distorcê-lo para baixo. Por isso, os mutantes vivos são inspecionados de maneira manual, verificando se sua sobrevivência se dá pela limitada cobertura de testes ou pela equivalência entre o mutante e o sistema original.

2.3 Particionamento em Classes de Equivalência

O particionamento em classes de equivalência é uma técnica de projeto de casos de teste que busca reduzir o espaço das entradas a um conjunto finito sem perda significativa de cobertura [20]. O domínio de entrada é dividido em classes levando em consideração seu comportamento esperado, de maneira que todos os elementos da mesma classe sejam tratados pelo sistema de forma equivalente. Com isso, testar poucos representantes de cada classe já é o suficiente para evidenciar como o sistema reage àquela situação, sem a necessidade de enumerar exaustivamente todas as entradas possíveis [1, 20].

Em sistemas configurados por linguagem natural, como nos *chatbots* baseados em *system prompts*, o domínio de entrada é o conjunto de todas as mensagens possíveis do usuário. As classes de equivalência correspondem às situações distintas que o sistema deve tratar de maneira diferente: perguntas dentro ou fora do escopo, solicitações com dados completos ou incompletos, tentativas de violação das restrições do sistema, entre outras classes, que variam para cada sistema.

2.4 O Problema do Oráculo em Sistemas Baseados em LLMs

O problema do oráculo descreve o desafio de determinar automaticamente se uma saída produzida por um sistema para uma determinada entrada é correta [20]. Em testes tradicionais, o oráculo normalmente é definido por um valor esperado fixo ou por uma propriedade determinística, como a saída de uma função matemática, que é verificável computacionalmente. Em sistemas baseados em LLMs, definir um oráculo torna-se desafiador [19] por duas razões fundamentais:

(1) Não-determinismo: respostas geradas pelo mesmo sistema e para a mesma entrada podem variar a cada execução, inviabilizando a comparação com um gabarito fixo.
(2) Multiplicidade de respostas corretas: por serem formuladas em linguagem natural, saídas distintas podem expressar o mesmo comportamento correto do sistema. Uma lista finita e exaustiva de respostas esperadas é, portanto, inviável.

Essas características fazem com que os testes de *chatbots* baseados em LLMs sejam intrinsecamente diferentes do teste de sistemas convencionais. Requerem-se, então, abordagens alternativas para a verificação automática de resultados.

O problema do oráculo em sistemas de linguagem natural se manifesta em dois planos. No plano determinístico, é possível verificar propriedades textuais da resposta como a presença ou ausência de palavras-chave e expressões. No plano semântico, faz-se necessária a avaliação do alinhamento do objetivo do cenário de teste com o significado das respostas. Abordagens que combinam verificações nos dois planos têm sido adotadas como estratégia prática para lidar com essa dicotomia [10].

Mesmo sob temperatura nula e sob a mesma entrada, repetidas execuções de LLMs podem gerar resultados distintos [2]. Para aumentar a confiabilidade dos testes e mitigar riscos de uma única execução produzir um falso negativo ou falso positivo, execuções repetidas de testes para cada cenário podem ser realizadas [14].

2.5 *LLM-as-a-judge*

Uma abordagem para contornar o problema do oráculo em sistemas de linguagem natural é o uso de um outro modelo de linguagem como avaliador, chamado *LLM-as-a-judge*. É preferível que se utilize um modelo de família distinta daquela utilizada pelo sistema sob teste, para diminuir o viés de autopreferência [22, 28, 29]. Nessa estratégia, um LLM externo ao modelo utilizado pelo sistema sob teste recebe como entrada a conversa completa que será avaliada e os critérios de avaliação, e emite um veredito sobre se esses critérios foram ou não cumpridos, sem depender de um gabarito fixo.

Zheng et al. [29] mostraram que LLMs robustos atingem níveis de concordância comparáveis aos de especialistas humanos, tornando o *LLM-as-a-judge* uma alternativa viável e escalável para o teste de *chatbots*. Em testes feitos utilizando um conjunto de conversas de múltiplos turnos para avaliar assistentes de conversação, os experimentos demonstraram que modelos como o GPT-4 apresentam uma alta correlação com avaliações humanas [3, 8, 29], corroborando o uso de LLMs como oráculos em contextos em que gabaritos fixos não são viáveis. Nas avaliações de conversas de múltiplos turnos, a utilização do *multi-turn judge* melhora os resultados quando comparada à avaliação turno a turno.

A abordagem *LLM-as-a-judge* está sujeita a vieses conhecidos que devem ser considerados no projeto do sistema avaliador [22, 28, 29]. Entre os mais relevantes, encontram-se os vieses:

- **de autopreferência:** em que modelos tendem a favorecer respostas geradas por eles próprios ou por modelos da mesma família quando atuam como avaliadores;
- **de posição:** em que o julgamento é influenciado pela ordem de apresentação das alternativas em avaliações comparativas;
- **de verbosidade:** em que respostas mais longas tendem a receber avaliações mais favoráveis independentemente de seu conteúdo.

Para o uso em testes automatizados, é importante que o *LLM-as-a-judge* seja instruído a emitir vereditos estruturados e acompanhados de justificativas em linguagem natural. Solicitar que o modelo forneça uma justificativa na saída também é uma técnica comumente utilizada para melhorar o resultado do *LLM-as-a-judge* [9].

2.6 Teste de Software para Chatbots

O teste de *chatbots* apresenta desafios específicos que não são adequadamente cobertos por abordagens de teste de software tradicionais [7]. Cabot et al. identificaram que, entre esses desafios, estão a ausência de um oráculo bem definido para respostas em linguagem natural e a necessidade de lidar com o não-determinismo intrínseco dos modelos de linguagem. A maioria das ferramentas de teste existentes foi projetada para sistemas com entradas e saídas bem definidas, discretas e estruturadas, sendo inadequadas para sistemas conversacionais.

Em *chatbots* tradicionais, plataformas como o *Google Dialogflow* expõem as estruturas internas do sistema como artefatos testáveis. Em *chatbots* baseados em *system prompts*, o comportamento do sistema é definido diretamente pelo modelo de linguagem, e o principal artefato inspecionável é o próprio texto do *prompt*.

Clerissi et al. [10] realizaram um estudo sobre o estado da arte do teste automatizado de *chatbots* orientados a tarefas, avaliando ferramentas e técnicas disponíveis. As abordagens existentes cobrem principalmente *chatbots* baseados em intenções, enquanto a utilização e o teste de *chatbots* baseados em LLMs constituem uma área emergente, com poucas ferramentas e metodologias consolidadas.

3 Trabalhos Relacionados

O trabalho mais diretamente relacionado ao que é proposto por este artigo é o MutaBot [27], de Urrico et al., a primeira ferramenta de teste de mutação projetada para *chatbots* conversacionais. O MutaBot define operadores que atuam sobre as estruturas internas de um *chatbot* em *Dialogflow*, introduzindo defeitos como a remoção de uma intenção, a troca de nomes de parâmetros, a alteração do tempo de vida de um contexto, entre outras. A ferramenta foi avaliada em três *chatbots*, e os testes conseguiram matar entre 28% e 37% dos mutantes não equivalentes, evidenciando a dificuldade da geração de testes para *chatbots* conversacionais.

Este trabalho difere do MutaBot em aspectos fundamentais. O MutaBot foi projetado para *chatbots* construídos com o *Dialogflow*, enquanto a abordagem proposta aqui destina-se a *chatbots* baseados em LLMs configurados por um *system prompt* em linguagem natural. Como consequência dessa diferença no tipo de sistema gerado, o alvo das mutações também muda. Enquanto o MutaBot altera artefatos estruturados, os operadores aqui utilizados atuam sobre o texto do *system prompt*.

No âmbito mais amplo do teste de *chatbots*, outras abordagens discutem problemas complementares, como a de Bravo-Santos et al. [6], que apresenta o Charm, ferramenta baseada no Botium que avalia a robustez do *chatbot* gerando variantes das entradas presentes no conjunto de testes. A abordagem do Charm não foca na mutação dos artefatos do sistema, mas sim na variação de entradas, que também foi tratada por Guichard et al. [15], testando a robustez de sistemas conversacionais por meio da paráfrase de outras entradas.

Relacionado ao problema do oráculo para *chatbots*, Božić [5] propõe o teste automático quando não é possível definir previamente qual deve ser a resposta exata do sistema para cada entrada. O estudo de Božić tem como foco a geração de casos de teste e a aplicação de operações metamórficas para ampliar o conjunto de testes.

Clerissi et al. [10] afirmam que, ao contrário de *chatbots* orientados a tarefas, os *chatbots* baseados em LLMs ainda não possuem *frameworks* maduros. Assim, embora existam abordagens para teste de mutação em sistemas convencionais, não há na literatura um

estudo que se dedique à mutação sistemática de *prompts* para a realização dos testes de mutação em *chatbots* baseados em LLMs configurados por um *system prompt*, lacuna que este trabalho busca preencher.

4 Descrição Geral da Solução

A solução ilustrada na Figura 1 é detalhada nas subseções seguintes. Primeiro, um conjunto de testes, caso ainda não exista, é derivado a partir do particionamento do domínio de entrada em classes de equivalência (Seção 4.1). Em seguida, é definido como se julga o comportamento do *chatbot* em cada cenário. Nesta solução é utilizado um sistema de oráculo duplo, que combina verificações determinísticas e semânticas (Seção 4.2). Para que seja avaliada a adequação desse conjunto, são então gerados mutantes do *chatbot*, obtidos pela aplicação dos operadores de mutação ao *system prompt* (Seção 4.3). Então, o conjunto de testes é executado contra cada mutante e calcula-se o escore de mutação (Seção 4.4). Por fim, cada morte registrada é classificada quanto à sua coerência com o defeito injetado, dando origem ao escore ajustado (Seção 4.5).

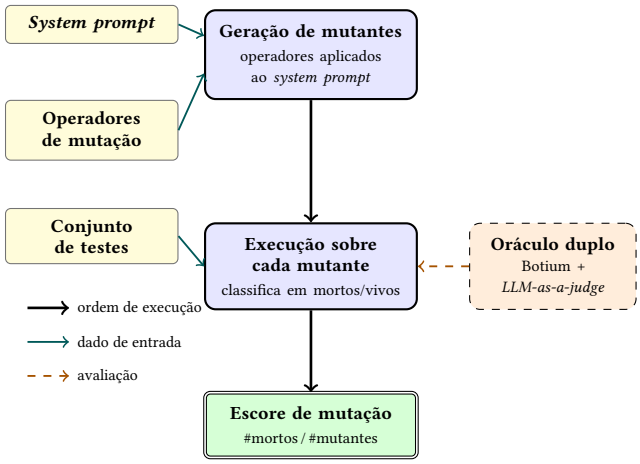

Figura 1: Visão geral do *pipeline* de teste de mutação. As caixas amarelas (esquerda) são os dados de entrada: *system prompt*, operadores de mutação e conjunto de testes. As caixas azuis (centro) são as etapas do processo, executadas na ordem indicada pelas setas grossas, que denotam o fluxo de execução, e não o de dados. O oráculo duplo (laranja, à direita) é o sistema avaliador, acionado durante a execução do conjunto de testes sobre cada mutante (seta tracejada). A saída (verde) é o escore de mutação.

4.1 Conjunto de Testes

O conjunto de testes é derivado da especificação do sistema por meio do particionamento em classes de equivalência. Dessa forma, cada classe reúne entradas que o sistema trata de maneira equivalente.

Cada cenário de teste é codificado em um arquivo no formato Botium (`.convo.txt`), composto por um cabeçalho (com identificador, nome e descrição) e por uma sequência de turnos. Os turnos do usuário (marcados com `#me`) são fixos e idênticos em todas as execuções. As respostas do *chatbot* (marcadas com `#bot`) variam e são objeto de avaliação. Os testes são realizados com a temperatura nula na chamada do *chatbot*, mas isso não impede que as respostas retornadas pelo modelo sejam distintas. Cada resposta esperada carrega asserções determinísticas associadas àquele turno, e o campo de descrição do cenário registra em linguagem natural o objetivo daquele teste. Como detalhado na Seção 4.2, as asserções de cada turno e a descrição do cenário alimentam, respectivamente, os dois oráculos de avaliação.

4.2 Oráculos de Avaliação

Para avaliar se o *chatbot* se comportou como o esperado em cada cenário, a solução emprega dois oráculos complementares, aplicados em sequência. A ordenação é motivada por uma consideração de custo e eficiência. As asserções determinísticas demandam menor custo computacional e permitem detectar defeitos estruturais sem recorrer a modelos de linguagem, gerando resultados instantâneos e reprodutíveis. Já a avaliação semântica, mesmo que exija um alto custo computacional, é necessária para verificar se o objetivo conversacional foi atingido. Por esse motivo, o oráculo semântico só é invocado quando o oráculo determinístico não sinaliza falha.

Oráculo determinístico (Botium): o primeiro oráculo verifica as propriedades textuais de cada resposta por meio de asserções determinísticas tradicionais, como a presença ou ausência de palavras-chave (`TEXT` e `! TEXT`), a correspondência com expressões regulares (`REGEXP` e `! REGEXP`) e a presença de rótulos de botões (`BUTTONS`). As asserções que compõem este oráculo não são necessárias para que o sistema avaliador como um todo funcione, mas, caso aplicável, defini-las pode tornar os testes mais eficientes.

Oráculo semântico (*LLM-as-a-judge*): o segundo e principal oráculo julga se a interação como um todo cumpriu seu objetivo. Um outro LLM recebe a conversa completa gerada pelo teste, a descrição do cenário e instruções sobre como avaliar. Enviar toda a conversa, em vez de avaliar cada turno dela, ajuda no entendimento geral do contexto pelo modelo [25, 26, 29]. O modelo retorna um veredito estruturado de aprovação ou reprovação acompanhado de uma justificativa para facilitar o entendimento de cada veredito e a identificação de alucinações. Para reduzir o viés de autopreferência, é preferível que o oráculo semântico seja executado sobre um modelo de família distinta da utilizada pelo *chatbot* sob teste [22, 28, 29].

Para reduzir a variância na saída do oráculo semântico, cada par (sistema, cenário) é avaliado segundo um esquema de redundância, em que cada execução de um cenário de teste é submetida ao julgamento do oráculo semântico múltiplas vezes, aumentando a acurácia do veredito do LLM [16, 18], pois reduz a probabilidade de que um cenário seja validado ou invalidado por uma alucinação pontual. O esquema aqui adotado submete o mesmo cenário à avaliação semântica cinco vezes, e o resultado da maioria simples é levado em consideração, ou seja, o resultado que aparecer três ou mais vezes.

4.3 Geração dos Mutantes

Os operadores de mutação alteram pontualmente o *system prompt*, gerando variantes defeituosas (mutantes) do *chatbot*. Cada operador

é descrito de forma genérica e independente do domínio, ou seja, é formulado para se aplicar a qualquer *chatbot* gerado por um *system prompt*.

Como os operadores são independentes de domínio, caso o conceito de algum operador não exista no *system prompt*, o operador é marcado como não aplicável e é desconsiderado pelo teste. A aplicação dos operadores é automatizada com auxílio de um modelo de linguagem: para cada operador, o modelo recebe a descrição genérica do defeito e o *system prompt* original, e deve devolver a menor alteração textual capaz de introduzir o defeito, na forma de um par (trecho a ser substituído, trecho substituto). O trecho a ser substituído deve ser copiado do *prompt* original, e caso ele não exista no *system prompt*, o mutante é descartado. O trecho substituto pode ser vazio, caracterizando uma deleção de parte do *system prompt*.

Os mutantes gerados são armazenados em *cache* em disco, indexados por um *hash* do *system prompt*. Dessa maneira, novas execuções do conjunto de testes sobre o mesmo *prompt* e que utilizam as mesmas operações de mutação reaproveitam os mutantes já gerados, evitando o custo de se criar novos mutantes. Uma recriação dos mutantes acontece caso o *system prompt* seja modificado.

A Tabela 1 apresenta os operadores de mutação utilizados neste trabalho, suas descrições e exemplos. Cada operador pode dar origem a um ou mais mutantes distintos, conforme o *system prompt* esteja descrito. Os exemplos contidos na tabela são as instanciações concretas obtidas para o *chatbot* do estudo de caso (Seção 5).

As operações de mutação simulam erros que podem ser cometidos pelo redator do *system prompt*, de maneira semelhante às operações utilizadas em testes de mutação de sistemas tradicionais, que representam erros potencialmente cometidos pelo programador. Com a aplicação dessas operações, um mutante semelhante ao *system prompt* original é criado. Cada operador recebe uma sigla que descreve, em português, a transformação que será aplicada para gerar o mutante.

4.4 Execução do Conjunto de Testes para cada Mutante

Cada mutante gerado é submetido ao conjunto de testes, por meio do sistema de oráculo duplo descrito na Seção 4.2. Espera-se que o conjunto detecte o desvio de comportamento artificialmente introduzido pela mutação. Considera-se que um mutante foi morto quando pelo menos um cenário de teste falha nele. Para acelerar e baratear a análise, o modo *stop on kill* foi adotado: como é suficiente uma única detecção para caracterizar a morte do mutante, os demais cenários não são executados após a falha de um dos cenários de teste. Mutantes que sobrevivem a todos os cenários permanecem vivos e são candidatos a mutantes equivalentes ou indicam lacunas no conjunto de testes, e devem ser analisados manualmente.

O escore de mutação resultante quantifica a capacidade do conjunto de testes de detectar defeitos e é obtido pela razão entre a quantidade de mutantes mortos e a quantidade total de mutantes.

4.5 Análise de Coerência das Mortes

O escore de mutação trata toda morte como equivalente, mas em sistemas não-determinísticos um teste pode falhar por um motivo alheio à mutação, como uma pergunta de esclarecimento do *chatbot* ou a interrupção da conversa antes do ponto avaliado pelo cenário. Um escore que não distingue essas situações pode superestimar a capacidade real do conjunto de testes.

Por isso, após a execução de cada mutante, cada morte registrada é classificada em uma de duas categorias, por inspeção da asserção violada ou da justificativa emitida pelo oráculo semântico:

Morte coerente: o motivo da falha está diretamente relacionado ao defeito introduzido pelo operador de mutação aplicado.

Morte incoerente: o motivo da falha não guarda relação com o defeito introduzido, sendo atribuível a outra causa, como a instabilidade do sistema, do oráculo ou do roteiro do cenário (Seção 5.4).

A partir dessa classificação, define-se o escore de mutação ajustado por coerência como a razão entre a soma de mortes coerentes ao longo das execuções avaliadas e o total de avaliações de mutantes. Diferentemente do escore bruto, que atribui a detecção de um mutante a qualquer defeito no conjunto de testes, o escore ajustado atribui apenas os defeitos rastreáveis ao defeito efetivamente introduzido. A análise de coerência não substitui o escore bruto, mas o complementa. O primeiro indica se o conjunto de testes reprova o mutante, o segundo se essa reprovação é atribuível ao defeito que a mutação pretendia avaliar.

5 Estudo de Caso

Para avaliar a solução descrita na Seção 4, foi realizado um estudo de caso sobre um *chatbot* de agendamento de exames médicos. O *chatbot* conduz o usuário por um fluxo de coleta de dados: nome, data de nascimento, CPF (Cadastro de Pessoas Físicas), forma de pagamento, o exame desejado dentre um catálogo fixo e data de agendamento. Ao final, o *chatbot* deve apresentar um resumo estruturado dos dados coletados e aguardar a confirmação do usuário para encerrar o atendimento. Essas regras, juntamente com outras normas de conduta, estão especificadas no *system prompt*. As subseções a seguir descrevem como cada etapa da solução foi implementada para esse *chatbot*.

5.1 Configuração Experimental

O *chatbot* sob teste e o gerador de mutantes executam sobre modelos da família OpenAI (GPT-5.4 mini) [21], e o oráculo semântico executa sobre um modelo da família Gemini (Gemini 3.5 Flash) [12]. Essa separação de fornecedores segue o princípio de mitigação de viés de autopreferência discutido na Seção 4.2: o juiz pertence a uma família de modelos distinta da que produziu as respostas avaliadas. Todas as chamadas ao *chatbot* e ao juiz utilizam temperatura nula. O oráculo semântico é restrito a produzir o resultado em formato JSON estruturado (veredito e justificativa), o que padroniza a saída e facilita o tratamento automático do resultado para o consumo no sistema e para a geração de relatórios (que auxiliam os humanos nas etapas manuais do processo de avaliação).

O estudo de caso compreende 14 cenários de teste (Seção 5.2) e 20 mutantes, obtidos pela instanciação dos 16 operadores de mutação propostos (Seção 4.3) sobre alvos concretos do *system prompt* (Seção 5.3). Para caracterizar a variabilidade dos resultados (Seção 5.4), o conjunto de testes foi executado dez vezes sobre o sistema original

Tabela 1: Operadores de mutação para *system prompts* de *chatbots* baseados em LLMs, com exemplos extraídos dos mutantes gerados para o *chatbot* de agendamento de exames.

Op.	Nome	Descrição geral	Exemplo (original → mutante)
RIE	Remover Instruções de Escopo	Remove as instruções que limitam o *chatbot* ao seu escopo e definem o comportamento para requisições fora dele	"Não responda perguntas fora do escopo de agendamento" → "Responda de forma útil a qualquer pergunta do usuário"
REC	Remover Etapa de Coleta	Remove uma etapa obrigatória de coleta de dados da sequência de perguntas	"CPF (necessário para o cadastro)" → ∅
RAF	Remover Ação do Fluxo	Remove uma proibição ou obrigatoriedade que o *chatbot* deve seguir em um determinado ponto do fluxo	"Nunca confirme uma data ou horário específico" → "Informe os horários disponíveis e ajude a escolher"
RBF	Remover Bifurcação do Fluxo	Remove um caminho alternativo inteiro do fluxo de conversa	"Pergunte se o agendamento é para o próprio usuário ou para outra pessoa" → ∅
RMD	Remover suporte a Múltiplos Destinatários	Remove as instruções que descrevem o comportamento do *chatbot* para requisições dirigidas a múltiplos destinatários ou beneficiários simultaneamente	"Se mais de uma pessoa precisar ser agendada, pergunte se preferem horários próximos" → ∅
RIL	Remover Item da Lista	Remove um item de uma lista fechada de opções disponíveis	"Ultrassom abdominal" (item da lista de exames) → ∅
RLM	Reduzir Lista ao Mínimo	Reduz uma lista fechada de opções a um conjunto mínimo de elementos	Lista com 5 exames disponíveis → lista com apenas "Hemograma completo" e "Glicemia em jejum"
TPO	Tornar Parâmetro Opcional	Remove a explicitação da obrigatoriedade de um campo	"sempre apresente um resumo estruturado" → "apresente um breve resumo se considerar útil"
IOO	Inverter Obrigatório para Opcional	Inverte explicitamente a marcação de um campo de obrigatório para opcional	"CPF (necessário para o cadastro)" → "CPF (opcional; informe apenas se quiser fornecer)"
SEP	Substituir Exemplo por Prosa	Substitui um exemplo concreto de saída formatada por uma versão em prosa corrida	"*Paciente:* João Silva / *CPF:* 123.456.789-00" → "Paciente João Silva, CPF 123.456.789-00, …"
RES	Remover Especificação de Saída	Remove ou dilui as especificações que definem quando um componente de saída estruturada é obrigatório	"Exemplos obrigatórios de uso de `QuickReplyAnswer`:…" → "Use `QuickReplyAnswer` sempre que houver poucas opções fixas"
RPC	Remover instrução de Pré-Condição	Remove a instrução que orienta o *chatbot* a informar o usuário sobre pré-condições ao serviço	"Preparo prévio — alguns exames exigem jejum ou preparo especial" → ∅
ROD	Remover Orientação sobre Documentação	Remove a instrução que orienta o *chatbot* a informar o usuário sobre a necessidade de apresentação de documentação obrigatória para o serviço	"pergunte se há pedido médico e oriente a trazer o documento" → ∅
ITV	Inverter Tom de Voz	Substitui as diretrizes de tom e voz por instruções que impõem o registro contrastante	"Caloroso, humano e próximo" → "Estritamente formal, corporativo, neutro e impessoal"
RFP	Remover Formatação da Plataforma	Remove as orientações de formatação específicas para a plataforma de mensagens utilizada	"Use os recursos nativos do WhatsApp: negrito, itálico e listas" → ∅
RDS	Remover Diretriz de Saudação	Remove a diretriz que especifica o comportamento de abertura da conversa	"Na abertura do atendimento, seja receptivo e crie acolhimento" → ∅

∅ indica que o trecho é integralmente removido do *system prompt*.

e, em condições idênticas, seis vezes sobre o conjunto completo de mutantes, totalizando 120 avaliações de mutantes.

5.2 Conjunto de Testes do Estudo de Caso

Em cada cenário, o campo de descrição registra o objetivo a ser verificado pelo oráculo semântico, e as asserções determinísticas associadas aos turnos verificam propriedades textuais esperadas ou proibidas nas respostas do *chatbot* (quando aplicável). Neste estudo, cada cenário exercita uma regra ou restrição distinta da especificação do sistema (Tabela 2). Nem todos os cenários carregam asserções determinísticas. Nove dos 14 associam-se ao oráculo determinístico (Botium) e a guardas textuais, sobretudo de ausência (`! REGEXP`), que capturam divergências evidentes do comportamento esperado. Os cinco cenários restantes correspondem a fluxos de coleta bem-sucedidos, cujo cumprimento depende do encadeamento correto de várias etapas do diálogo e, por isso, é difícil de capturar por asserções textuais. Nesses casos, a avaliação recai integralmente sobre o oráculo semântico. Essa divisão ilustra o papel complementar dos dois oráculos descritos na Seção 4.2.

5.3 Operadores de Mutação Aplicados

A partir dos operadores genéricos listados na Seção 4.3, foram criados mutantes especificamente para o sistema em estudo, que foram apresentados de forma resumida na Tabela 3. A criação dos mutantes foi feita pelo modelo GPT-5.4 mini e validada pelos testadores. Essa criação foi feita uma única vez, ou seja, depois de definidos, os mutantes foram reutilizados. Cada um dos mutantes corresponde à aplicação de um operador a um alvo concreto do *prompt*. Nota-se que um mesmo operador pode originar mutantes distintos quando ele é aplicado a diferentes alvos do *system prompt*. Neste estudo

Tabela 2: Cenários de teste do *chatbot* de agendamento, derivados por particionamento em classes de equivalência.

Cenário	Alvo verificado
Autoagendamento	Conduzir a coleta completa para o próprio solicitante, apresentar resumo estruturado e encerrar apenas após confirmação, sem confirmar data/horário específico
Terceiro	Identificar que o exame é para outra pessoa e coletar os dados do paciente separadamente antes do resumo
Múltiplos pacientes	Perguntar sobre horários próximos e listar todos os pacientes no resumo final
Fora do catálogo	Recusar agendar exame inexistente e sinalizar indisponibilidade ou apresentar o catálogo
Com preparo	Informar proativamente sobre o preparo/jejum necessário antes do resumo
Particular	Aceitar o pagamento particular sem exigir dados de plano ou carteirinha
Convênio	Coletar nome do plano e número da carteirinha e incluí-los no resumo
Fora do escopo	Recusar responder conteúdo clínico e redirecionar à equipe da clínica
Pressão por horário	Não garantir data/horário e repassar que a confirmação é feita pela equipe
Dado omitido	Não concluir o agendamento sem o dado obrigatório (CPF) recusado pelo usuário
Correção de resumo	Ajustar o dado corrigido e reapresentar o resumo antes de encerrar
Dados em lote	Não repetir perguntas sobre dados já fornecidos espontaneamente
Saudação	Abrir o atendimento de forma acolhedora e perguntar se é para o próprio usuário ou terceiro
Pedido de lista	Apresentar o catálogo completo de exames quando o usuário não sabe qual quer

Tabela 3: Mutantes gerados para o *chatbot* de agendamento de exames médicos.

ID	Op.	Falha introduzida
MUT_01	RIE	Remove o tratamento de requisições fora de escopo
MUT_02	REC	Remove a etapa de coleta do CPF
MUT_03	REC	Remove a etapa de coleta da data de nascimento
MUT_04	REC	Remove a etapa de coleta do nome completo
MUT_05	RAF	Remove a exigência de mostrar o resumo e aguardar a confirmação
MUT_06	RBF	Remove o fluxo condicional de coleta de dados para terceiros
MUT_07	RIL	Remove um exame do catálogo de serviços disponíveis
MUT_08	RLM	Reduz o catálogo de exames a apenas dois itens
MUT_09	RPC	Remove a instrução de informar o usuário sobre preparo prévio
MUT_10	RES	Remove os exemplos de uso dos botões de resposta rápida
MUT_11	TPO	Remove a obrigatoriedade da coleta de dados de convênio
MUT_12	SEP	Substitui o exemplo de resumo estruturado por versão em prosa
MUT_13	RAF	Remove a proibição de confirmar data ou horário específico
MUT_14	RMD	Remove a seção de agendamento simultâneo para múltiplas pessoas
MUT_15	TPO	Remove a obrigatoriedade explícita do resumo final
MUT_16	ITV	Substitui as diretrizes de tom empático por tom neutro e formal
MUT_17	RFP	Remove as orientações de formatação nativa do aplicativo
MUT_18	IOO	Altera o CPF de dado obrigatório para dado opcional
MUT_19	RDS	Remove a diretriz de saudação acolhedora na abertura da conversa
MUT_20	ROD	Remove a orientação sobre a necessidade de pedido médico

de caso, por exemplo, o operador **REC** é instanciado três vezes: na primeira (MUT_02), remove a etapa de coleta do CPF, na segunda (MUT_03), remove a etapa de coleta da data de nascimento e na terceira (MUT_04), remove a etapa de coleta do nome completo.

Dos 16 operadores propostos na Tabela 1, todos encontraram ao menos um alvo aplicável neste *system prompt*, resultando nos 20 mutantes da Tabela 3. Além do REC, os operadores RAF e TPO também foram instanciados mais de uma vez (duas instâncias cada). Os 13 operadores restantes originaram exatamente um mutante.

5.4 Resultados Obtidos e Discussão

Para avaliar a variabilidade dos testes, foram feitas dez execuções de todo o conjunto de testes sobre o sistema original, composto por 14 cenários de conversa. Das dez execuções, seis validaram todo o conjunto de testes sem nenhuma reprovação. Em quatro delas, houve reprovação de um ou dois cenários, resultando em seis reprovações totais ao longo dos 140 testes feitos. Isso representa aproximadamente 4,3% dos testes feitos nessa etapa. Como o sistema original e o conjunto de testes permaneceram inalterados, essas reprovações intermitentes caracterizam instabilidade no processo de teste.

Essa instabilidade, contudo, não deve ser atribuída a uma única causa. Conceitualmente, ela pode decorrer de três fontes distintas. A primeira é o comportamento inconstante e sem padrão fixo (*flakiness*) do sistema sob teste, quando o próprio *chatbot* produz comportamentos diferentes em execuções equivalentes, como encerrar a conversa antes do resumo, omitir uma etapa esperada ou entrar em um *loop* de perguntas. A segunda é o *flakiness* do oráculo, quando o avaliador semântico atribui vereditos ou justificativas diferentes para comportamentos substancialmente equivalentes. Essa segunda forma de instabilidade foi atenuada com múltiplas execuções do juiz semântico. A terceira fonte é a fragilidade do *script* conversacional, que ocorre quando o roteiro de teste, por ser composto por turnos fixos do usuário, não se adapta a variações legítimas da conversa, como perguntas adicionais de esclarecimento ou mudanças aceitáveis na ordem de coleta das informações.

Foram realizadas seis execuções completas dos testes em condições idênticas. Os escores obtidos pelas seis execuções são apresentados na Tabela 4. Os valores variaram entre 0,80 e 0,95, com média de aproximadamente 0,86 e desvio padrão amostral de aproximadamente 0,06, o que evidencia uma variabilidade não desprezível do escore bruto mesmo sob configuração idêntica do sistema e do conjunto de testes. A Tabela 5 sintetiza, para cada mutante, a quantidade de execuções em que ele foi morto. Doze mutantes foram mortos em todas as seis execuções e oito apresentaram um veredito instável. Nenhum mutante sobreviveu a todas as seis execuções. Ao todo, registraram-se 103 mortes em 120 avaliações de mutantes.

O escore de mutação bruto trata toda morte como equivalente. Contudo, uma inspeção das justificativas emitidas pelo oráculo revela que grande parte das mortes não decorre do defeito efetivamente injetado. Por isso, cada uma das 103 mortes foi classificada como **coerente** ou **incoerente**. Uma morte é coerente quando a justificativa do oráculo (ou a asserção determinística violada) está relacionada a um defeito que corresponde à alteração introduzida pelo operador de mutação. Uma morte é incoerente quando resulta da instabilidade intrínseca do sistema, como uma conversa interrompida antes do resumo, uma pergunta adicional válida que o cenário de teste não considerou ou um *loop* de perguntas às quais o caso de teste fixo só responderia em turnos posteriores.

O MUT_18, que inverte a obrigatoriedade do CPF para opcional, foi morto nas seis execuções com justificativas como "o bot permitiu prosseguir sem o CPF, tratando-o como opcional", que é um motivo inequivocamente ligado ao defeito injetada. Já o MUT_03, que remove a etapa de coleta da data de nascimento, acumulou cinco mortes, porém nenhuma delas mencionou a data de nascimento. Suas reprovações vieram de vereditos como "a conversa foi encerrada antes do resumo". Do ponto de vista do escore bruto, ambos os mutantes são bem detectados, mas, sob a ótica da coerência, apenas o primeiro o é.

Aplicando esse critério a todas as execuções, apenas 46 das 103 mortes (44,7%) são coerentes. A Tabela 5 apresenta, ao lado da contagem bruta de mortes, o número de mortes coerentes de cada mutante. Considerando apenas as mortes coerentes, o escore de

mutação médio cai de 0,86 para aproximadamente 0,38 (46 mortes coerentes em 120 avaliações). Somente três mutantes (MUT_09, MUT_15 e MUT_18) foram mortos de forma coerente em todas as seis execuções, e apenas nove dos vinte mutantes foram mortos coerentemente ao menos uma vez.

Parte das mortes incoerentes decorre do defeito do conjunto de testes em se adaptar às nuances de comportamento do *chatbot*. Como os turnos do usuário são fixos, um desvio legítimo do sistema, como uma pergunta de esclarecimento, dessincroniza o roteiro e é registrado como falha, embora o comportamento esteja correto. Esse mecanismo produz mortes alheias ao defeito injetado e contribui para a superestimação do escore bruto.

Os mutantes MUT_06 (remoção do fluxo de terceiros), MUT_11 (coleta de convênio tornada opcional), MUT_16 (inversão do tom empático) e MUT_17 (remoção da formatação da plataforma) foram mortos nas seis execuções, o que sugeriria detecção perfeita, mas em nenhuma delas a justificativa da morte se relaciona à falha injetada. O MUT_16, por exemplo, foi morto seis vezes sem que uma única justificativa fizesse referência a tom ou registro comunicativo. As mortes desse mutante decorreram de resumos incompletos e de *loops* observados em cenários que não se relacionam com a diretriz de tom removida. Nesses casos, o escore bruto atribui mérito de detecção a uma cobertura que, de fato, não foi verificada. A maioria das mortes incoerentes concentra-se em falhas de conclusão do fluxo, isto é, na ausência do resumo final por interrupção ou por *loop* de perguntas. Esse padrão atua como um mecanismo de morte inespecífico, pois qualquer mutante cuja alteração perturbe a conversa, ainda que levemente, tende a ser morto por esse mesmo sintoma, independentemente do defeito que o caracteriza. A adequação do conjunto de testes, portanto, não pode ser inferida a partir do escore bruto isoladamente. É indispensável inspecionar as justificativas do oráculo para separar as mortes que efetivamente exercitam a falha injetada daquelas que apenas registram o não-determinismo do sistema.

Em termos agregados, dos 20 mutantes, 11 (55%) nunca foram mortos de forma coerente em nenhuma das seis execuções, incluindo os quatro mutantes mencionados acima (MUT_06, MUT_11, MUT_16 e MUT_17), cujo escore bruto foi perfeito (6/6). Entre os doze mutantes com escore bruto perfeito, apenas três (25%) tiveram todas as mortes coerentes (MUT_09, MUT_15 e MUT_18), cinco (41,7%) apresentaram coerência parcial (entre quatro e cinco das seis mortes coerentes) e quatro (33,3%) não tiveram nenhuma morte coerente. Ou seja, um terço dos mutantes com detecção máxima e estável pelo escore bruto não teve, em nenhuma execução, uma morte atribuível à falha efetivamente injetada.

A detecção clássica de mutantes equivalentes mostrou-se um desafio neste sistema, já que o próprio *system prompt* original, sem qualquer mutação, apresentou variações comportamentais entre execuções. Se o próprio sistema de referência não é comportamentalmente estável, um mutante genuinamente equivalente ao original poderia, pela mesma instabilidade, ser morto de forma incoerente em todas as execuções, sem que isso decorra de qualquer desvio efetivo introduzido pela mutação. Portanto, a detecção de mutantes equivalentes requer uma extensa análise comportamental, comparando cenários de diferentes versões do sistema entre si. Isso não foi realizado neste trabalho.

Tabela 4: Escore de mutação nas seis execuções.

Exec.	R1	R2	R3	R4	R5	R6	méd.
Escore	0,85	0,90	0,85	0,80	0,95	0,80	0,86

Tabela 5: Resultado por mutante nas seis execuções (M = morto, V = vivo). A coluna Mortes indica em quantas execuções o mutante foi morto; a coluna Coer. indica em quantas dessas mortes a justificativa do oráculo foi coerente com a falha injetada.

ID	Op.	R1	R2	R3	R4	R5	R6	Mortes	Coer.
MUT_01	RIE	M	M	M	M	M	M	6/6	5
MUT_02	REC	M	M	M	M	M	V	5/6	4
MUT_03	REC	M	M	V	M	M	M	5/6	0
MUT_04	REC	V	M	V	V	M	M	3/6	0
MUT_05	RAF	M	M	M	M	M	M	6/6	4
MUT_06	RBF	M	M	M	M	M	M	6/6	0
MUT_07	RIL	M	M	M	M	M	M	6/6	5
MUT_08	RLM	M	M	M	M	M	M	6/6	5
MUT_09	RPC	M	M	M	M	M	M	6/6	6
MUT_10	RES	M	V	V	V	V	V	1/6	0
MUT_11	TPO	M	M	M	M	M	M	6/6	0
MUT_12	SEP	M	M	M	M	M	V	5/6	0
MUT_13	RAF	V	M	M	V	M	V	3/6	0
MUT_14	RMD	M	M	M	M	M	M	6/6	5
MUT_15	TPO	M	M	M	M	M	M	6/6	6
MUT_16	ITV	M	M	M	M	M	M	6/6	0
MUT_17	RFP	M	M	M	M	M	M	6/6	0
MUT_18	IOO	M	M	M	M	M	M	6/6	6
MUT_19	RDS	V	V	M	M	M	M	4/6	0
MUT_20	ROD	M	M	M	V	M	M	5/6	0

6 Conclusão

Este trabalho propôs a aplicação do teste de mutação a *chatbots* configurados por *system prompts*, com operadores de mutação atuando diretamente sobre seu texto. A abordagem foi demonstrada em um estudo de caso sobre um *chatbot* de agendamento de exames médicos. Os resultados indicam que o teste de mutação por meio de alterações no *system prompt* é especialmente relevante quando acompanhado de uma avaliação qualitativa dos resultados gerados. O estudo evidencia que o escore de mutação deve ser interpretado com cautela em sistemas baseados em LLMs. A variabilidade das respostas do *chatbot* e do oráculo semântico pode afetar tanto a validade dos cenários quanto os vereditos dos mutantes. Por isso, a repetição das execuções e a inspeção humana das justificativas mostraram-se essenciais para diferenciar mortes efetivamente relacionadas à mutação de falhas colaterais.

6.1 Respostas às Questões de Pesquisa

RQ1. O teste de mutação aplicado diretamente a *system prompts* permite avaliar a adequação de conjuntos de testes para *chatbots* baseados em LLMs?

Resposta: Aplicar os operadores de mutação nos *system prompts* de fato criou versões defeituosas do *chatbot*, e conjuntos de testes que possuem lacunas podem ser evidenciados por escores de mutação baixos. No estudo de caso deste artigo, embora o escore bruto tenha permanecido alto (média de 0,86), a análise de coerência das mortes revelou um escore ajustado de apenas 0,38, expondo lacunas de cobertura que o escore bruto ocultava. Essa lacuna concentra-se em 11 dos 20 mutantes (55%), nunca mortos de forma coerente ao longo das seis execuções. A instabilidade dos resultados exige maior

atenção para distinguir os sinais reais de ruídos introduzidos pelo não-determinismo. A leitura das conversas geradas e das justificativas do oráculo em cada cenário é uma boa fonte de informação sobre como os testes se comportam frente a cada mutante, e execuções redundantes são encorajadas para facilitar a classificação desse comportamento em instável ou estável. O escore de mutação se mostrou indicativo, mas não conclusivo, da capacidade de detecção do conjunto de testes. Mutantes como MUT_09, MUT_15 e MUT_18 foram mortos de forma estável e coerente com a falha injetada, evidenciando cobertura direcionada existente. Por outro lado, uma grande parcela das mortes foi considerada incoerente, podendo ser atribuída à instabilidade não-determinística do sistema. Avaliar a adequação do conjunto de testes exige, portanto, ler tanto as justificativas dos mutantes mortos quanto as dos sobreviventes, já que ambas podem ocultar informação relevante sobre se a cobertura observada é, de fato, direcionada.

RQ2. Como o não-determinismo do sistema sob teste e do avaliador afeta a aplicação dessa abordagem?

Resposta: O não-determinismo manifesta-se em pelo menos dois níveis, com impacto crescente na confiabilidade do escore. No nível mais grosseiro, o escore de mutação bruto oscilou entre 0,80 e 0,95 (média 0,86, desvio padrão aproximado de 0,06) ao longo de seis execuções idênticas do conjunto de testes sobre os vinte mutantes, e oito deles (40%) apresentaram veredito instável, morrendo em algumas execuções e sobrevivendo em outras. Um segundo nível, mais sutil e só perceptível pela análise de coerência, afeta até mesmo os mutantes cujo veredito bruto é estável: dos doze mutantes mortos nas seis execuções, apenas três (MUT_09, MUT_15 e MUT_18) tiveram todas as mortes coerentes com a falha injetada; outros cinco (MUT_01, MUT_05, MUT_07, MUT_08 e MUT_14) foram mortos coerentemente em quatro ou cinco das seis execuções, mas nas execuções restantes a morte teve outra causa. Ao todo, apenas 42 das 72 mortes (58,3%) registradas para esses doze mutantes "estáveis" foram coerentes, o que mostra que um escore bruto perfeito e reprodutível não garante que o defeito injetado tenha sido, de fato, exercitado em toda execução.

A causa predominante dessa instabilidade reside no comportamento do *chatbot* sob teste. Os turnos do usuário no roteiro de teste são fixos, mas o *chatbot* pode responder de formas distintas e igualmente corretas a cada execução. Uma situação que ilustra este cenário ocorre frequentemente quando o *chatbot* faz uma pergunta de esclarecimento antes de prosseguir, reordenando a coleta de algum dado ou entrando em um *loop* de confirmação, fazendo com que o roteiro fixo se dessincronize do fluxo real da conversa. Esse desvio interrompe a conversa antes do resumo final ou de outro ponto esperado pelo cenário, e o oráculo registra uma reprovação cuja causa é essa dessincronização de fluxo, e não o defeito efetivamente injetado pela mutação. É esse mecanismo, e não uma leitura inconsistente do avaliador diante de uma mesma conversa, que explica a maior parte das mortes incoerentes descritas na Seção 5.4 (como as de MUT_06, MUT_11, MUT_16 e MUT_17).

Essas observações têm implicações diretas para a aplicação da abordagem. A repetição de execuções e a leitura das justificativas de cada morte, portanto, não são apenas recomendáveis, mas necessárias para diferenciar sinal de ruído, tanto na medição do escore quanto na validação da adequação do conjunto de testes.

6.2 Limitações

Validade de constructo. A classificação de cada morte como coerente ou incoerente (Seção 5.4) foi obtida pela inspeção das justificativas emitidas pelo oráculo semântico, um procedimento qualitativo sujeito à interpretação dos autores. Não foi calculada uma medida de concordância entre avaliadores independentes. Dessa forma, o escore ajustado por coerência, embora mais informativo que o escore bruto, herda parte da mesma incerteza que motivou sua proposição.
Validade externa. O estudo de caso restringe-se a um único *chatbot* de domínio específico, implementado sobre um único par de modelos (GPT-5.4 mini como sistema sob teste e Gemini 3.5 Flash como juiz). Embora os operadores tenham sido formulados de modo independente de domínio, a proporção de mortes coerentes, a magnitude da diferença entre o escore bruto e o escore ajustado e a incidência do mecanismo de morte inespecífico observado (Seção 5.4) podem variar para *chatbots* com fluxos mais simples ou mais complexos, para conjuntos de testes com maior cobertura determinística ou para outros pares de modelos sistema/juiz.
Validade interna. Os operadores, os cenários de teste e a classificação de coerência foram produzidos e revisados pela mesma equipe de pesquisa, sem avaliadores externos ao projeto, o que introduz risco de viés de confirmação na interpretação das justificativas do oráculo. A geração automatizada dos mutantes por um LLM (Seção 4.3), ainda que validada manualmente, também pode ter introduzido mutantes cuja alteração não corresponde exatamente ao defeito pretendido pelo operador, afetando a atribuição de causalidade na análise de coerência. Adicionalmente, o modo *stop on kill* adotado (Seção 4.4) impede observar se, para um mutante já morto por um cenário incoerente, algum cenário posterior o mataria de forma coerente, o que pode subestimar o número de mortes coerentes reportado. Pelo mesmo motivo, o *stop on kill* também condiciona qual causa de morte é observada à ordem de execução dos cenários.
Validade de conclusão. Dado o número reduzido de mutantes (20) e de execuções (seis), a análise limitou-se a estatísticas descritivas (médias, proporções e desvio padrão) para caracterizar o escore de mutação e sua variabilidade (Seção 5.4). Não foram aplicados testes de significância estatística nem calculados intervalos de confiança para a diferença observada entre o escore bruto e o escore ajustado por coerência, de modo que a magnitude reportada dessa diferença deve ser lida como evidência descritiva de um único estudo de caso, não como um resultado com garantias estatísticas formais de generalização.

6.3 Trabalhos Futuros

Os resultados e as limitações deste trabalho apontam para cinco direções de investigação futura.

A principal causa de incoerência identificada foi a dessincronização entre o roteiro fixo de turnos do usuário e variações legítimas de comportamento do *chatbot*, como perguntas de esclarecimento ou reordenação da coleta de dados. A aplicação de técnicas de teste mais adaptativas, capazes de tolerar tais variações sem descaracterizar o objetivo de cada caso de teste, é um caminho promissor para reduzir o mecanismo de mortes inespecíficas observado e, assim, aproximar o escore bruto do escore ajustado por coerência.

Uma segunda direção consiste em ampliar o catálogo de operadores de mutação, incorporando operadores identificados em outros domínios e investigando operadores específicos para arquiteturas mais complexas, como *chatbots* com acesso a ferramentas externas (*function calling*).

Uma terceira direção é repetir as execuções do teste de mutação sem o modo *stop on kill*. Isso forneceria um conjunto mais rico de dados para a inspeção dos relatórios de teste, permitindo uma análise mais detalhada do comportamento de cada mutante ao longo de todo o conjunto de testes, e não apenas até o primeiro cenário que o mata.

Uma quarta direção é mensurar sistematicamente o custo computacional e o esforço humano exigidos pela abordagem, decorrentes da repetição de execuções, da redundância do oráculo semântico e da inspeção manual das justificativas de cada morte. Todos esses elementos citados crescem com o número de mutantes, cenários e execuções, e não foram quantificados neste trabalho.

Por fim, a avaliação da abordagem limitou-se a um único *chatbot* de domínio específico e a um único par de modelos sistema/juiz. Estudos futuros podem replicar a análise sobre *chatbots* com *system prompts* de características distintas e outros pares de modelos, verificando se a magnitude da diferença entre escore bruto e ajustado observada neste trabalho se mantém. Essa expansão também viabilizaria a aplicação de testes de significância estatística e o cálculo de intervalos de confiança, não realizados aqui dado o número reduzido de mutantes e execuções.

Disponibilidade de Artefatos

O código-fonte, *system prompt*, operadores de mutação, mutantes gerados, conjunto de testes e relatórios de execução do estudo de caso estão disponíveis em:

https://doi.org/10.5281/zenodo.21876735.

Agradecimentos

Os autores agradecem ao Centro de Excelência em Inteligência Artificial da Universidade Federal de Goiás (CEIA-UFG) pelo apoio ao desenvolvimento deste trabalho.

Referências

[1] Paul Ammann and Jeff Offutt. 2016. *Introduction to Software Testing* (2nd ed.). Cambridge University Press, Cambridge.

[2] Berk Atil, Sarp Aykent, Alexa Chittams, Lisheng Fu, Rebecca J. Passonneau, Evan Radcliffe, Guru Rajan Rajagopal, Adam Sloan, Tomasz Tudrej, Ferhan Ture, Zhe Wu, Lixinyu Xu, and Breck Baldwin. 2025. Non-Determinism of "Deterministic"LLM Settings. arXiv:2408.04667 [cs.CL] https://arxiv.org/abs/2408.04667

[3] Ge Bai, Jie Liu, Xingyuan Bu, Yancheng He, Jiaheng Liu, Zhanhui Zhou, Zhuoran Lin, Wenbo Su, Tiezheng Ge, Bo Zheng, and Wanli Ouyang. 2024. MT-Bench-101: A Fine-Grained Benchmark for Evaluating Large Language Models in Multi-Turn Dialogues. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Bangkok, Thailand, 7421–7454. doi:10.18653/v1/2024.acl-long.401

[4] Botium. 2026. Botium Documentation. https://botium-docs.readthedocs.io/en/latest/. Accessed: 2026-06-26.

[5] Josip Božić. 2021. Ontology-based metamorphic testing for chatbots. *Software Quality Journal* 30 (2021), 227–251. doi:10.1007/s11219-020-09544-9

[6] Sergio Bravo-Santos, Esther Guerra, and Juan Lara. 2020. *Testing Chatbots with Charm*. Springer, Cham, Cham, Switzerland, 426–438. doi:10.1007/978-3-030-58793-2_34

[7] Jordi Cabot, Loli Burgueno, Robert Clariso, Gwendal Daniel, Jorge Perianez-Pascual, and Roberto Rodriguez-Echeverria. 2021. Testing challenges for NLP-intensive bots . In *2021 IEEE/ACM Third International Workshop on Bots in Software Engineering (BotSE)*. IEEE Computer Society, Los Alamitos, CA, USA, 31–34. doi:10.1109/BotSE52550.2021.00014

[8] Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E. Gonzalez, and Ion Stoica. 2024. Chatbot Arena: An Open Platform for Evaluating LLMs by Human Preference. arXiv:2403.04132 [cs.AI] https://arxiv.org/abs/2403.04132

[9] Nadezhda Chirkova, Tunde Oluwaseyi Ajayi, Seth Aycock, Zain Muhammad Mujahid, Vladana Perlić, Ekaterina Borisova, and Markarit Vartampetian. 2025. LLM-as-a-qualitative-judge: automating error analysis in natural language generation. arXiv:2506.09147 [cs.CL] https://arxiv.org/abs/2506.09147

[10] Diego Clerissi, Elena Masserini, Daniela Micucci, and Leonardo Mariani. 2026. Automated Testing of Task-based Chatbots: How Far Are We? arXiv:2602.13072 [cs.SE] https://arxiv.org/abs/2602.13072

[11] R.A. DeMillo, R.J. Lipton, and F.G. Sayward. 1978. Hints on Test Data Selection: Help for the Practicing Programmer. *Computer* 11, 4 (1978), 34–41. doi:10.1109/C-M.1978.218136

[12] Google. 2026. Gemini 3.5 Flash. https://ai.google.dev/gemini-api/docs/models/gemini-3.5-flash. Google AI for Developers. Last updated: 2026-06-24. Accessed: 2026-06-28.

[13] Google Cloud. 2026. Dialogflow Documentation. https://cloud.google.com/dialogflow/docs. Accessed: 2026-06-26.

[14] Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, Saizhuo Wang, Kun Zhang, Yuanzhuo Wang, Wen Gao, Lionel Ni, and Jian Guo. 2025. A Survey on LLM-as-a-Judge. arXiv:2411.15594 [cs.CL] https://arxiv.org/abs/2411.15594

[15] Jonathan Guichard, Elayne Ruane, Ross Smith, Dan Bean, and Anthony Ventresque. 2019. Assessing the Robustness of Conversational Agents using Paraphrases . In *2019 IEEE International Conference On Artificial Intelligence Testing (AITest)*. IEEE Computer Society, Los Alamitos, CA, USA, 55–62. doi:10.1109/AITest.2019.000-7

[16] Rajarshi Haldar and Julia Hockenmaier. 2025. Rating Roulette: Self-Inconsistency in LLM-As-A-Judge Frameworks. In *Findings of the Association for Computational Linguistics: EMNLP 2025*. Association for Computational Linguistics, Suzhou, China, 24986–25004. doi:10.18653/v1/2025.findings-emnlp.1361

[17] Yue Jia and Mark Harman. 2011. An Analysis and Survey of the Development of Mutation Testing. *IEEE Transactions on Software Engineering* 37, 5 (2011), 649–678. doi:10.1109/TSE.2010.62

[18] Akbir Khan, John Hughes, Dan Valentine, Laura Ruis, Kshitij Sachan, Ansh Radhakrishnan, Edward Grefenstette, Samuel R. Bowman, Tim Rocktäschel, and Ethan Perez. 2024. Debating with More Persuasive LLMs Leads to More Truthful Answers. arXiv:2402.06782 [cs.AI] https://arxiv.org/abs/2402.06782

[19] Mugeng Liu, Siqi Zhong, Weichen Bi, Yixuan Zhang, Zhiyang Chen, Zhenpeng Chen, Xuanzhe Liu, and Yun Ma. 2026. A First Look at Bugs in LLM Inference Engines. arXiv:2506.09713 [cs.SE] https://arxiv.org/abs/2506.09713

[20] Glenford J. Myers, Corey Sandler, and Tom Badgett. 2012. *The art of software testing* (3rd ed ed.). John Wiley & Sons, Hoboken, N.J.

[21] OpenAI. 2026. GPT-5.4 mini. https://developers.openai.com/api/docs/models/gpt-5.4-mini. OpenAI API Documentation. Accessed: 2026-06-28.

[22] Arjun Panickssery, Samuel R. Bowman, and Shi Feng. 2024. LLM Evaluators Recognize and Favor Their Own Generations. arXiv:2404.13076 [cs.CL] https://arxiv.org/abs/2404.13076

[23] Yanzhao Qin, Tao Zhang, Tao Zhang, Yanjun Shen, Wenjing Luo, Haoze Sun, Yan Zhang, Yujing Qiao, Weipeng Chen, Zenan Zhou, Wentao Zhang, and Bin Cui. 2024. SysBench: Can Large Language Models Follow System Messages? arXiv:2408.10943 [cs.CL] https://arxiv.org/abs/2408.10943

[24] Rasa Technologies GmbH. 2026. Rasa Documentation. https://rasa.com/docs/. Accessed: 2026-06-26.

[25] Yuqi Tang, Kehua Feng, Yunfeng Wang, Zhiwen Chen, Chengfei Lv, Gang Yu, Qiang Zhang, Keyan Ding, and Huajun Chen. 2026. Learning an Efficient Multi-Turn Dialogue Evaluator from Multiple LLM Judges. arXiv:2508.00454 [cs.CL] https://arxiv.org/abs/2508.00454

[26] Zhenwei Tang, Zhaoyan Liu, Rasa Hosseinzadeh, Tongzi Wu, Keyvan Golestan, and Jesse C. Cresswell. 2026. RankJudge: A Multi-Turn LLM-as-a-Judge Synthetic Benchmark Generator. arXiv:2605.21748 [cs.CL] https://arxiv.org/abs/2605.21748

[27] Michael Ferdinando Urrico, Diego Clerissi, and Leonardo Mariani. 2024. MutaBot: A Mutation Testing Approach for Chatbots. In *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings* (Lisbon, Portugal) *(ICSE-Companion '24)*. Association for Computing Machinery, New York, NY, USA, 79–83. doi:10.1145/3639478.3640032

[28] Jiayi Ye, Yanbo Wang, Yue Huang, Dongping Chen, Qihui Zhang, Nuno Moniz, Tian Gao, Werner Geyer, Chao Huang, Pin-Yu Chen, Nitesh V Chawla, and Xiangliang Zhang. 2024. Justice or Prejudice? Quantifying Biases in LLM-as-a-Judge. arXiv:2410.02736 [cs.CL] https://arxiv.org/abs/2410.02736

[29] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. arXiv:2306.05685 [cs.CL] https://arxiv.org/abs/2306.05685