# Detecção de Inconsistências Lógicas em Casos de Teste por Meio de Extração Semântica e Análise de Alcançabilidade

Christopher Ric Coelho Saar
Universidade Federal do Acre
Rio Branco, Brasil
christopher.saar@sou.ufac.br

Victor Alexandre Lima Ribeiro
Universidade Federal do Acre
Rio Branco, Brasil
victor.ribeiro@sou.ufac.br

Luan Reis de Lima
Universidade Federal do Acre
Rio Branco, Brasil
luan.reis@sou.ufac.br

Igsonfiure Rodrigues Félix
Universidade Federal do Acre
Rio Branco, Brasil
igsonfiure.felix@sou.ufac.br

Nicolas Riccieri Gardin Assumpção
Brock University
St. Catharines, Canadá
go25us@brocku.ca

Laura Costa Sarkis
Universidade Federal do Acre
Rio Branco, Brasil
laura.sarkis@ufac.br

Olacir Rodrigues Castro Júnior
Universidade Federal do Acre
Rio Branco, Brasil
olacir.junior@ufac.br

Marlon Amaro Coelho Teixeira
Instituto Federal do Acre
Rio Branco, Brasil
marlon.teixeira@ifac.edu.br

## Resumo

A identificação de inconsistências lógicas em artefatos de teste é um desafio crítico na engenharia de *software*, resultando frequentemente em retrabalho e elevados custos de manutenção. Revisões manuais tradicionais são morosas e suscetíveis a falhas, especialmente em sistemas de larga escala. Este trabalho propõe o *framework* de auditoria automática DILCAT, fundamentado em uma abordagem de decisão conservadora neuro-simbólica que integra a flexibilidade semântica de *Sentence-Transformers* com o poder preditivo de classificadores de *Machine Learning* supervisionados e Filtragem por Classes Gramaticais. Alinhada à norma ISO/IEC/IEEE 29119-3:2021, a metodologia propõe reduzir decisões heurísticas rígidas com uma classificação probabilística baseada em limiar de corte preventivo, capaz de categorizar as relações funcionais locais entre os casos de teste nas classes GAP, SEQUENTIAL_CHAIN e STANDALONE. Os resultados experimentais foram validados em um *dataset* orgânico e legado de 80 casos de teste da indústria de telecomunicações, os quais totalizam 406 instâncias de transição de estado, em conjunto com bases adicionais para testes fora de domínio. Os dados demonstram que a combinação de extração vetorial e aprendizado estatístico minimiza a incerteza linguística e mitiga falsos positivos funcionais. O *ground truth* foi estabelecido por três anotadores e validado pelo coeficiente estatístico Fleiss' Kappa, alcançando um índice de 0,74, o que indica consistência entre os anotadores dos dados usados para validação e corrobora com a robustez do *framework* no tratamento de ruídos terminológicos e na facilitação da prevenção de intermitências antes da etapa de automação. Por fim, a avaliação experimental indicou que o sistema atinge uma acurácia global de 98% no conjunto avaliado, mantendo uma estabilidade macro de 94% de acurácia global durante os ensaios de validação cruzada.

## Palavras-chave

Engenharia de *Software*, Engenharia de Testes, Engenharia de Requisitos, *Hard Skills*, Casos de Teste, Consistência de Casos de Teste, Processamento de Linguagem Natural.

## 1 Introdução

A garantia da qualidade em processos de teste de software constitui um dos pilares para a confiabilidade de sistemas modernos, uma vez que a identificação precoce de falhas reduz significativamente os custos de correção ao longo do ciclo de desenvolvimento [24]. Entretanto, a especificação de casos de teste ainda é frequentemente realizada em linguagem natural livre, introduzindo ambiguidades, informações implícitas e diferentes interpretações sobre ações, precondições e resultados esperados [31]. Em ambientes industriais de grande porte, nos quais milhares de casos de teste evoluem continuamente, essas ambiguidades dificultam a manutenção das suítes, favorecem redundâncias e comprometem a consistência lógica entre os testes [12].

As consequências desse problema tornam-se evidentes quando a execução de um caso de teste depende de estados produzidos por casos anteriores sem que essas dependências estejam explicitamente documentadas. Nesses cenários, precondições podem não ser satisfeitas por nenhum estado previamente estabelecido na suíte, caracterizando lacunas de alcançabilidade lógica entre os testes. Esse compartilhamento não mapeado de estados quebra o isolamento do ambiente de execução [12], tornando o resultado dos testes dependente da ordem de execução e favorecendo o surgimento de testes intermitentes (*flaky tests*), além de aumentar o retrabalho em pipelines de Integração Contínua (CI) [23].

Diversas abordagens têm sido propostas para mitigar esse problema. O uso de *Behavior-Driven Development* (BDD) busca reduzir ambiguidades por meio da padronização da escrita dos cenários em uma linguagem ubíqua compartilhada entre as partes interessadas [8]. Entretanto, sua adoção não é universal e frequentemente

encontra limitações relacionadas à curva de aprendizado, às particularidades organizacionais e à dificuldade de especificar cenários completos em projetos de grande escala. Mais recentemente, pesquisas têm explorado o emprego de Modelos de Linguagem de Grande Escala (LLMs) para apoiar a geração e a análise de casos de teste. Embora apresentem resultados promissores, essas abordagens tendem a amplificar inconsistências já presentes na documentação original, além de permanecerem suscetíveis a alucinações lógicas, inferindo dependências inexistentes ou deixando de identificar quebras reais de continuidade entre casos de teste [18]. Soma-se a isso a ausência de mecanismos determinísticos e auditáveis que permitam verificar, de forma reproduzível, a consistência lógica de suítes legadas escritas em linguagem natural [35].

Diante desse cenário, este trabalho tem o objetivo de propor um *framework* neuro-simbólico estruturado na norma ISO/IEC/IEEE 29119-3:2021 [15] capaz de detectar automaticamente inconsistências lógicas de alcançabilidade (dependências sistêmicas e lacunas funcionais) em suítes de testes escritas em linguagem natural livre de forma determinística, reproduzível e auditável, antes da etapa de automação, integrando técnicas de Processamento de Linguagem Natural, similaridade semântica e aprendizado de máquina supervisionado.

Para nortear a validação experimental e estabelecer o rigor metodológico da abordagem proposta, este estudo busca responder a três perguntas de pesquisa. A primeira investiga a possibilidade de extrair, de casos de teste escritos em linguagem natural livre, informações relativas a ações, entidades, pré-condições e pós-condições. A segunda avalia a eficácia da abordagem neuro-simbólica do *framework* (DILCAT) na identificação de lacunas lógicas reais em suítes de testes industriais orgânicas. Por fim, a terceira analisa de que forma a estratégia de classificação probabilística adotada pela abordagem gerencia situações de incerteza linguística, reduzindo a ocorrência de falsos positivos durante a auditoria de suítes de testes.

Este trabalho apresenta três contribuições principais. A primeira consiste em uma metodologia de extração semântica baseada na norma ISO/IEC/IEEE 29119-3:2021 para padronizar artefatos de teste provenientes de fontes heterogêneas. A segunda corresponde a um mecanismo de auditoria capaz de identificar automaticamente dependências sistêmicas e lacunas funcionais por meio da integração entre processamento de linguagem natural, similaridade semântica e aprendizado de máquina supervisionado. A terceira contribuição consiste na avaliação experimental da abordagem em um cenário industrial real da área de telecomunicações, complementado por uma matriz anonimizada e de domínio de comércio eletrônico para testes de estresse com alterações sintáticas e fora de domínio de telecomunicações. Os resultados evidenciaram que o sistema atingiu uma acurácia global de 98% frente ao gabarito homologado, mantendo uma estabilidade macro de 94% de acurácia global durante os ensaios de validação cruzada e alcançando um $F1$-Score de 0,88 na classificação de inconsistências lógicas de alcançabilidade (*GAPs*).

## 2 Fundamentação Teórica e Trabalhos Relacionados

Os testes desempenham um papel crucial no desenvolvimento de *software*, sendo conduzidos ao longo de todo o ciclo com o objetivo de identificar falhas o mais precocemente possível. A detecção precoce de falhas é significativamente menos custosa e gera menos impactos negativos do que sua identificação tardia, como ocorre quando o sistema já está em produção. Isso indica que o custo de correção pode aumentar de forma exponencial à medida que a falha se propaga ao longo das etapas do ciclo de desenvolvimento [24].

Neste cenário, quando se faz necessário o desenvolvimento de testes em larga escala para contribuir com a qualidade do sistena, significa que problemas introduzidos ainda na especificação dos casos de teste, como precondições não satisfeitas ou resultados esperados que não alimentam o próximo teste da cadeia, tornam-se exponencialmente mais caros quanto mais tarde forem descobertos. Diante desse problema, a norma ISO/IEC/IEEE 29119-3:2021 [15] define a estrutura mínima que um caso de teste deve possuir, incluindo campos obrigatórios de precondições e resultados esperados. Essa estrutura fornece a semântica necessária para que estados sistêmicos possam ser comparados de forma determinística entre testes distintos, independentemente de quem os escreveu ou em que formato estão armazenados. O *framework* Detector de Inconsistências Lógicas em Casos de Teste (DILCAT) proposto neste trabalho adota essa norma como camada de interoperabilidade, normalizando artefatos de fontes heterogêneas antes de qualquer análise semântica, decisão metodológica que, conforme o mapeamento sistemático de Navarro e Ibarra [19], está presente em menos de 10% dos trabalhos da área.

Em uma direção complementar, Farooq *et al.* [8] realizaram uma revisão sistemática sobre o *Behavior-Driven Development* (BDD) e identificaram que, embora a metodologia seja eficaz para alinhar a comunicação entre *stakeholders* e mitigar ambiguidades individuais, ela apresenta lacunas críticas em cenários de larga escala. Os autores destacam que a identificação de falhas e a gestão de redundâncias tornam-se tarefas altamente onerosas à medida que o volume de requisitos aumenta, evidenciando a carência de mecanismos sistemáticos para validar a integridade de suítes de teste complexas. O DILCAT preenche justamente esse espaço operacional pós-especificação, atuando como um auditor de consistência lógica sobre a suíte consolidada antes da execução automatizada dos testes.

### 2.1 PLN e Mineração de Texto Aplicados a Casos de Teste

A aplicação de PLN a testes de software tem sido investigada como uma abordagem para automatizar diferentes atividades do processo de teste. Wagaw e Ayenew [3] realizaram uma revisão sistemática abrangendo 13 estudos primários sobre a geração automatizada de casos de teste por meio de Processamento de Linguagem Natural. Os autores identificaram sete técnicas e sete algoritmos empregados nesse contexto, destacando que o emprego de *parsing*, *tokenization* e *POS tagging* na extração de dados operacionais a partir de requisitos textuais e relatórios de falhas. O estudo aponta que essas técnicas podem contribuir para a melhoria do processo de teste, embora a seleção da metodologia mais adequada permaneça um desafio para

as equipes de engenharia. Entre as principais lacunas identificadas pelos autores, destaca-se a escassez de estudos empíricos abrangentes e de avaliações comparativas que permitam caracterizar as vantagens e limitações das ferramentas existentes.

Reimers e Gurevych [26] introduziram o *Sentence-BERT* (SBERT), uma modificação da arquitetura BERT que utiliza redes siamesas para derivar *embeddings* de sentenças semanticamente significativos, permitindo a comparação direta via similaridade de cosseno. Esta abordagem resolve o gargalo computacional do BERT original em tarefas de busca e agrupamento, reduzindo drasticamente o tempo de processamento em grandes coleções de dados. O *framework* DILCAT utiliza essa base tecnológica através dos modelos *all-MiniLM-L6-v2* e *paraphrase-multilingual-MiniLM-L12-v2,* implementações destiladas que equilibram a precisão semântica validada pelos autores com o baixo custo operacional necessário para auditorias em tempo real.

Este cenário proporciona a necessidade de criação de novas ferramentas. Nessa linha, Fischbach *et al.* [9] apresentaram a ferramenta CiRA (*Conditionals in Requirements Artifacts*), capaz de detectar e extrair relações causais em requisitos informais para a derivação automática do conjunto mínimo de casos de teste. Em avaliações conduzidas com três parceiros industriais, o *framework* demonstrou eficácia ao gerar automaticamente 71,8% de uma base de 578 casos de teste manuais, além de identificar 80 cenários relevantes omitidos pelos projetistas humanos. Contudo, embora o CiRA avance na síntese orientada por grafos de causa-efeito, sua operação é estritamente generativa a partir de requisitos bem formados. Diferente desta abordagem, o DILCAT atua na camada de auditoria de conformidade, sendo capaz de processar suítes de teste legadas e preexistentes para identificar falhas de alcançabilidade lógica entre estados sistêmicos.

## 2.2 Dependências de Estado e Testes Intermitentes

Em ambientes de desenvolvimento de *software*, a ocorrência de falhas em casos de teste, na ausência de modificações no código-fonte indica, na maioria dos casos, que a origem do problema não reside na implementação. O trabalho de Tahir *et al.* [30] realizou uma revisão multivocal cobrindo 560 artigos, dos quais 200 foram analisados em profundidade, e mapearam as causas de *flaky tests*. Entre as mais relevantes, a dependência de ordem de execução se destaca, onde o resultado de um teste é determinado não pelo código que ele testa, mas pelo estado residual que um teste anterior deixou no ambiente. Apesar de documentar extensivamente o problema e suas causas, o trabalho não propõe mecanismos de detecção preventiva na fase de especificação.

Já os trabalhos realizados por Peng *et al.* [23], confirmaram empiricamente, em *pipelines* de integração contínua reais, que o acesso não mapeado a recursos e variáveis compartilhadas constitui uma das principais causas de *flakiness* e de falsos negativos. No entanto, embora o estudo apresente uma avaliação empírica, sua análise opera predominantemente no nível do código-fonte, investigando dependências por meio de grafos de chamada e rastros de execução. Essa abordagem implica que a detecção de inconsistências ocorre apenas após a implementação dos testes, não contemplando a fase de especificação textual, na qual o custo de correção tende a ser significativamente inferior.

Por fim, no âmbito da formalização teórica do problema, Haidry e Miller [12] formalizaram a dependência funcional entre casos de teste como uma relação de precedência estrutural, na qual a satisfação de uma precondição é condição necessária para a validade do caso subsequente. Esse modelo é a base direta do Grafo de Dependência Direcionado do DILCAT, além de estar presente em trabalhos recentes sobre o tema [31, 32].

## 2.3 Mineração Semântica e Detecção de Redundância em Casos de Teste

A linha de pesquisa que mais diretamente antecede o DILCAT é a que aplica PLN para extrair dependências funcionais entre casos de teste escritos em linguagem natural. Tahvili *et al.* [31] exploraram essa direção ao utilizar *Doc2Vec* e algoritmos de agrupamento não supervisionado (*HDBSCAN*) para detectar dependências em suítes de teste de sistemas embarcados ferroviários. A conclusão central do trabalho corrobora a premissa deste artigo: o PLN é a saída técnica viável quando a rastreabilidade formal entre requisitos e testes está ausente ou incompleta. Contudo, a abordagem de Tahvili baseia-se na correlação entre similaridade semântica e dependência funcional, o que permite agrupar testes relacionados, mas não garante a verificação de alcançabilidade lógica.

Uma extensão sistemática conduzida por Tahvili *et al.* [32], comparou diversas técnicas de mineração de texto, incluindo métodos baseados em distância de *string*, compressão (NCD) e aprendizado de máquina (*Doc2Vec* e SBERT). O estudo evidenciou que, em cenários industriais de alta complexidade, o emprego de modelos de *Sentence-Transformers* combinados a estratégias rigorosas de pré-processamento (tokenização) supera as abordagens tradicionais na captura de dependências funcionais. Contudo, o trabalho mantém-se fundamentado no paradigma do agrupamento por similaridade semântica para fins de priorização e escalonamento.

Em contraste com essa abordagem, Pan *et al.* [20] introduziram o LTM (*Language model-based Test suite Minimization*), marcando a primeira aplicação de LLMs na minimização de suítes de teste via *embeddings* de código e busca evolutiva. Posteriormente, Pan *et al.* [21] expandiram essa abordagem com o *framework* RTM, voltado especificamente para a minimização de testes em linguagem natural guiada pela cobertura de requisitos. Avaliado com 736 casos industriais do setor automotivo, o RTM demonstra eficácia na redução de redundância, mas opera sob a premissa de que a rastreabilidade formal entre requisitos e testes está previamente estabelecida e consolidada.

## 2.4 LLMs, Alucinações e Integração Neuro-Simbólica

Nos últimos anos, os LLMs emergiram como uma tecnologia promissora para a automação de testes, apresentando crescimento exponencial em publicações de área. Wang *et al.* [34] publicaram um *survey* abrangente analisando 102 estudos, destacando que o uso desses modelos concentra-se em testes unitários e reparo de programas. A conclusão mais relevante para este trabalho é a identificação de uma lacuna crítica: o raciocínio sobre sequências de estados e dependências complexas permanece um desafio, com a literatura

apresentando ausência de práticas voltadas a testes de integração. Li *et al.* [17] confirmaram empiricamente que, embora os LLMs demonstrem alto desempenho em cenários isolados, sua eficácia decai significativamente quando a tarefa exige o mapeamento de cadeias de estados entre múltiplos casos de teste.

No campo da síntese, Liubinskyi e Zvarych [18] propuseram o modelo NSRG, integrando restrições de Lógica Temporal Linear para mitigar a taxa de 57% de discrepância observada na geração puramente neural de testes. A resposta a esse conjunto de limitações passa pela integração neuro-simbólica, como apresentado pelos trabalhos de Hitzler *et al.* [13], onde argumentam que o acoplamento entre a flexibilidade do aprendizado profundo e o determinismo simbólico permite capturar o conhecimento de especialistas e favorecer o rigor necessário em domínios de engenharia de *software*. O DILCAT fundamenta-se nessa dualidade neuro-simbólica para preencher uma lacuna operativa distinta: enquanto as abordagens citadas focam na síntese assistida ou detecção em requisitos estruturados, o *framework* proposto especializa-se na auditoria de consistência lógica em suítes de teste legadas escritas em linguagem natural livre.

## 3 *Framework* DILCAT

A arquitetura da solução fundamenta-se na premissa de que a consistência de uma suíte de testes não é meramente uma propriedade sintática, mas reside na continuidade lógica de seus estados sistêmicos [18]. O *framework* proposto, denominado DILCAT, atua como uma camada de auditoria neuro-simbólica que mapeia dependências através de *embeddings* contextuais e valida a alcançabilidade lógica via teoria dos grafos [36].

### 3.1 Classificação Baseada em Aprendizado de Máquina

A abordagem proposta introduz uma alternativa que visa mitigar a dependência de árvores de decisão puramente heurísticas ou regras ontológicas rígidas, as quais frequentemente demandam um esforço manual elevado de manutenção e exibem baixa adaptabilidade a variações de escrita industriais. Em vez de operar sobre o texto em sua totalidade, o motor de execução realiza o processamento linguístico inicial isolando os núcleos substantivos e nomes próprios por meio de *POS-tagging*, alimentando os modelos *all-MiniLM-L6-v2* e *paraphrase-multilingual-MiniLM-L12-v2*, dos *Sentence-Transformers* com os componentes essenciais que delimitam os estados sistêmicos. Essa decisão de projeto permite que os classificadores estatísticos subsequentes processem a semântica das instruções concentrando-se nas entidades críticas do domínio, gerenciando a incerteza linguística sem a necessidade de uma exaustiva engenharia de regras manuais para cada novo cenário.

Esses vetores densos, gerados a partir do escopo textual completo associado aos atributos estruturais extraídos das seções normativas da ISO/IEC/IEEE 29119-3:2021 (precondições, passos de ação e resultados esperados), constituem a matriz de características estruturais (*feature vectors*) do modelo. O classificador supervisionado consome esses dados para determinar, matematicamente, a distribuição de probabilidade contínua de cada par de artefatos configurar uma transição funcional válida. Complementarmente, o *POS-tagging* via biblioteca *Stanza* é integrado ao ecossistema de forma paralela, atuando como um módulo de validação de controle e descontaminação semântica isolada, o que permite aferir o impacto do ruído de verbos operacionais genéricos sem desestruturar a entrada rica dos modelos estatísticos.

Para materializar a política de decisão conservadora e priorizar a retenção ativa de lacunas lógicas, o processo de tomada de decisão do classificador é calibrado por frentes assimétricas: a imposição de uma matriz de custo no treinamento, configurando uma razão de pesos de $50 : 1 : 2$ para as classes GAP, SEQUENTIAL_CHAIN e STANDALONE, respectivamente, e a introdução de um limiar restrito de corte probabilístico fixado em $\tau_{prob} = 0.20$ para a ativação da classe de inconsistência. Desse modo, se o modelo supervisionado manifestar uma probabilidade mínima de apenas 20% de quebra de fluxo, o par de artefatos é preventivamente rotulado como GAP. Esse mapeamento contínuo e protegido possibilita o rastreamento minucioso de divergências textuais, permitindo que o sistema gerencie múltiplos autores industriais com critério conservador de classificação e salvaguarde a integridade do pipeline de execução automatizada dos testes.

### 3.2 Formalização Lógica e Teoria dos Conjuntos

Para fundamentar o funcionamento do motor de auditoria, a lógica de consistência é expressa através da Teoria dos Conjuntos. Cada caso de teste $TC_i$ é abstraído pela tupla $\{ID, O, P, I, E, \tau\}$, as triplas semânticas extraídas no módulo anterior são mapeadas formalmente em $P$ que representa o conjunto de estados exigidos (precondições), e $E$, o conjunto de estados gerados como efeito (pós-condições).

Dada uma suíte de testes composta por $n$ instâncias, o conjunto de estados sistêmicos acumulados até o passo $k$ é definido pela união dos efeitos precedentes: $\mathcal{A}_k = \bigcup_{j=1}^{k} E_j$. Esta premissa de mutação e herança de estados tem ampla base na literatura, atestando-se que o sucesso de um teste depende rigorosamente das modificações de estado executadas previamente [7]. Sob esta ótica, e em alinhamento com definições formais de estruturas de dependência funcional [12], a condição de consistência lógica para o teste subsequente $TC_{k+1}$ é definida pela relação de subconjunto na Equação (1):

$$TC_{k+1} \text{ é consistente} \iff P_{k+1} \subseteq \mathcal{A}_k \tag{1}$$

Esta formalização permite classificar as inconsistências de forma axiomática. Como ilustrado na Figura 1, o ***gap* lógico** (ou estado inconsistente) ocorre precisamente quando a intersecção semântica entre o que é exigido ($P$) e o que já foi provido ($\mathcal{A}_k$) é incompleta, resultando em um conjunto diferença não vazio ($P_{k+1} \setminus \mathcal{A}_k \neq \emptyset$).

### 3.3 Modelagem de Contexto e Engenharia de Atributos de Transição

A arquitetura do auditor proposto estrutura a verificação de alcançabilidade lógica com foco na extração de características locais com fidelidade elevada, o que otimiza a matriz de entrada e evita a dispersão dimensional do vetor de atributos. Para compor a matriz de características que alimenta o classificador supervisionado, o mecanismo de captura de contexto avalia a consistência da transição funcional imediata em dois níveis complementares de granularidade. Primeiro, calcula-se a similaridade de cosseno em relação ao efeito

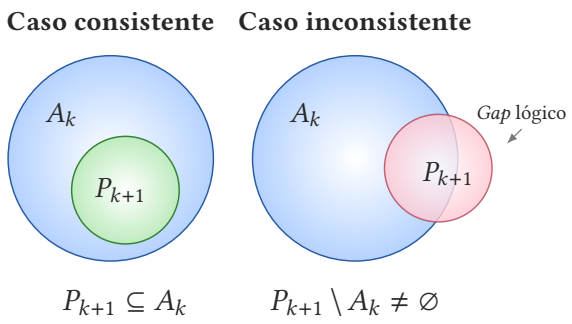


**Figura 1: Representação da agregação de estados em $A_k$, do caso consistente quando $P_{k+1} \subseteq A_k$ e do caso inconsistente quando $P_{k+1} \setminus A_k \neq \emptyset$.**

gerado no passo diretamente adjacente ($i-1$). Segundo, afere-se a aderência conceitual do passo atual ao bloco de configuração inicial do ambiente (*initial setup*). Essa decisão de *design* fundamenta-se na observação de que a integridade lógica em suítes de teste de telecomunicações depende criticamente da continuidade imediata entre passos vizinhos e da preservação das diretrizes macro estabelecidas no início da execução.

Ao restringir o escopo da engenharia de atributos à vizinhança direta e ao estado de origem, o DILCAT limita a ocorrência de arestas acidentais potencialmente induzidas por coincidências linguísticas de longa distância. Essa abordagem elimina a necessidade de buscas lineares globais semanticamente agnósticas ou varreduras retroativas computacionalmente onerosas, permitindo que o motor estatístico extraia padrões puramente baseados nos objetos de estado locais. Uma vez computados os vereditos probabilísticos locais para cada transição, o *framework* aciona sua camada simbólica para consolidar essas conexões em um Grafo de Dependência Direcionado gerenciado via motor de grafos (*NetworkX*), o que viabiliza o mapeamento matemático e visual do fluxo completo no diagnóstico final.

Se o modelo preditivo manifestar uma estimativa de risco igual ou superior ao limiar conservador fixado em $\tau_{prob} = 0,20$, indicando que uma precondição técnica atual não encontra amparo material estável no efeito precedente ou no *setup* de origem, a política proativa de segurança entra em ação. O motor invalida a transição sequencial imediata e retém o artefato sob a classe `GAP`. Esse comportamento conservador prioriza a identificação de possíveis inconsistências antes da etapa da execução automatizada, reduzindo o risco de propagação de suposições incorretas sobre o fluxo e possibilitando que inconsistências lógicas ocultas sejam expostas para auditoria humana antes de comprometerem a paralelização ou a execução automatizada dos testes.

## 3.4 Agregação de Vereditos e Unicidade Documental

Dada a natureza procedimental dos requisitos, um único caso de teste ($TC_i$) frequentemente contém múltiplos passos de execução ($I$). O *framework* implementa uma camada de agregação e pós-processamento para contribuir com a integridade da auditoria no nível documental.

A função de agregação consolida as $n$ triplas extraídas de um mesmo identificador único, adotando uma postura conservadora de auditoria: se qualquer passo procedimental for classificado como GAP devido a uma falha de alcançabilidade, o $TC_i$ integral é reportado como inconsistente. O *score* final é, portanto, definido pelo elo mais fraco da cadeia procedimental, conforme a Equação (2):

$$S_{1_{final}} = \min(S_{1_1}, S_{1_2}, \ldots, S_{1_n}) \tag{2}$$

Esta abordagem visa contribuir com o isolamento do ambiente de testes. Estudos anteriores indicam que a execução de testes contendo etapas com dependências não satisfeitas pode introduzir alterações indesejadas em estados compartilhados e variáveis do sistema, contribuindo para a ocorrência de intermitência (*flakiness*) e de falsos negativos em rotinas de integração contínua [30]. Ao invalidar o documento em sua totalidade diante da primeira quebra de fluxo lógico ($S_{1_{final}} < \tau$), a arquitetura busca reduzir execuções desnecessárias e a necessidade de análise manual de falhas lógicas em conjuntos de larga escala. Dessa forma, busca-se fornecer ao gestor de testes um diagnóstico unificado por artefato com informações mais objetivas para apoiar a tomada de decisão.

# 4 Materiais e Métodos

Esta seção detalha o arcabouço metodológico e o *pipeline* experimental concebido para enfrentar a ambiguidade em casos de teste, focando na interoperabilidade entre múltiplas fontes de dados.

## 4.1 Padronização via ISO/IEC/IEEE 29119-3:2021

Um dos maiores desafios na auditoria de suítes de testes provenientes de múltiplas fontes é a heterogeneidade das estruturas de dados. A ausência de um esquema comum impede a análise de consistência, pois os estados pregressos e as precondições futuras não partilham uma semântica estrutural mínima.

Para mitigar este problema, o *framework* adota a norma **ISO/IEC/IEEE 29119-3:2021** como modelo de referência. Esta norma define uma estrutura de campos essenciais que permite que requisitos de fontes distintas sejam normalizados em um esquema unificado. A conformidade com a norma contribui para que campos críticos como *Preconditions* ($P$) e *Expected Results* ($E$) sejam isolados de forma consistente, servindo como entradas determinísticas para o motor de similaridade. Assim, a ISO 29119-3:2021 atua como uma "camada de interoperabilidade", favorecendo a detecção de inconsistências mesmo em conjuntos híbridos compostos por múltiplos ficheiros e autores.

## 4.2 Dataset Industrial, Decomposição em Instâncias e Protocolo de Validação

Para submeter o DILCAT a um cenário real de estresse operacional, a modelagem experimental fundamentou-se em um *dataset* orgânico e legado proveniente da indústria de telecomunicações, composto por 80 casos de teste estruturados sob a norma ISO 29119-3:2021, identificados individualmente por `test_case_id`. Como

um mesmo caso de teste pode conter múltiplos passos de execução, os casos foram decompostos em unidades de análise no nível de passo/transição, resultando em 406 instâncias, cada uma mantendo associação com o respectivo `test_case_id` de origem. Cada uma dessas instâncias mantém associação com o respectivo `test_case_id` de origem. Dessa forma, as 406 instâncias não correspondem a casos de teste adicionais ou artificialmente gerados, mas à granularização dos 80 casos de teste originais para fins de classificação das relações funcionais entre seus passos. Complementarmente, foi incorporado um conjunto de dados adicionais atuando como testes de estresse fora de domínio com uma suíte focada em fluxos de um sistema de comércio eletrônico, contendo 38 casos de teste e 64 instâncias.

O estabelecimento do *ground truth* seguiu um protocolo cego e independente, conduzido por três engenheiros de *software* seniores especialistas no domínio do sistema, resultando na rotulagem das 406 instâncias de transição, nas respectivas classes `SEQUENTIAL_CHAIN` ($n = 306$; 75, 37%), `STANDALONE` ($n = 60$; 14, 78%) e `GAP` ($n = 40$; 9, 85%), com confiabilidade aferida pelo coeficiente Fleiss' Kappa [10] de 0, 74 (*concordância substancial*), dando suporte as avaliações utilizando deste dataset. Para evitar o vazamento de dados (*data leakage*) e facilitar a independência entre as etapas, a separação dos artefatos foi realizada estritamente no nível de `test_case_id` antes de qualquer procedimento de calibração. Foram alocados 80% dos casos de teste (64 casos; $\approx$ 325 instâncias) para o conjunto de desenvolvimento e 20% (16 casos; $\approx$ 81 instâncias) para o conjunto de teste independente (*hold-out*), o qual permaneceu completamente não visto durante o ajuste do modelo. No conjunto de desenvolvimento, a avaliação dos classificadores e a calibração de hiperparâmetros foram executadas via validação cruzada agrupada (`GroupKFold` com $K = 5$ e `random_state` = 42), assegurando que todas as instâncias pertencentes a um mesmo caso de teste fossem mantidas na mesma dobra (*fold*) em cada iteração, reportando-se os resultados por meio da média e desvio-padrão ($\mu \pm \sigma$).

O arquivo `telecom_matrix_training.csv` é representado em uma estrutura tabular de 406 instâncias, que não constituem um conjunto sintético independente nem correspondem a casos de teste artificialmente gerados, mas resultam da anonimização dos 80 casos de teste industriais em unidades que preservaram a essência estrutural adequada à análise das relações funcionais entre estados. A avaliação experimental também utilizou o arquivo `landing_page_ecommerce_eng_pt.csv`, englobando instâncias de um sistema de comércio eletrônico.

Para assegurar a reprodutibilidade integral do protocolo experimental, os classificadores supervisionados baseados em *Random Forest* e *XGBoost* foram configurados com parâmetros determinísticos e semente aleatória (*seed*) fixada em 42. O modelo Random Forest foi configurado com 100 estimadores (`n_estimators`), profundidade máxima de 5 (`max_depth`), critério `gini` e pesos de classe (`class_weight`) ajustados na proporção 50:1:2 para GAP, CHAIN e STANDALONE, respectivamente. O algoritmo XGBoost operou com 100 estimadores, profundidade máxima de 4, taxa de aprendizado de 0,05, função objetivo `multi:softprob` e ponderação assimétrica equivalente estabelecida via `sample_weight`.

O pipeline de execução do DILCAT integra de forma híbrida diferentes tecnologias e componentes de infraestrutura que atuam em sinergia. A análise sintática e a estruturação linguística são realizadas pela biblioteca *Stanza*, responsável pelo *Dependency Parsing* e pela filtragem por classes gramaticais (*POS-Tagging*), atuando como o motor de descontaminação semântica ao isolar exclusivamente os núcleos substantivos e nomes próprios (NOUN/PROPN) das instâncias para reduzir o ruído vetorial induzido por verbos repetitivos. Esses núcleos conceituais são convertidos em vetores densos de 384 dimensões pela biblioteca *Sentence-Transformers* através dos modelos *all-MiniLM-L6-v2* e *paraphrase-multilingual-MiniLM-L12-v2*, baseado na técnica de destilação de auto-atenção, projetando as representações de estado em um espaço latente capaz de mitigar o problema de sinonímia técnica. Como alternativa à formulação manual de regras ou heurísticas estáticas de domínio, o ecossistema incorpora modelos de classificação supervisionados de *Machine Learning*, que ingerem a matriz de características estruturais e computam distribuições de probabilidade contínuas para determinar a pertinência de cada transição. Por fim, o motor de grafos *NetworkX* é utilizado para visualizar esses vereditos preditivos em um Grafo de Dependência Direcionado (DiGraph), no qual os casos de teste são modelados como nós e as arestas representam a consolidação lógica da dependência estrutural.

A integração dessas ferramentas estabelece um fluxo neurosimbólico de transformação da especificação livre em um modelo formal de consistência lógica. Inicialmente, as sentenças são submetidas à análise linguística estruturada baseada em árvores de dependência [6, 29]. A extração vetorial no espaço latente permite a unificação de termos funcionalmente equivalentes sob uma mesma representação canônica, viabilizando a comparação conceitual dos estados sistêmicos. Por fim, o modelo preditivo pondera as *features* textuais contra os dados de transição espelhados na matriz de treino e alimenta o motor de grafos. Essa modelagem permite verificar se uma necessidade técnica atual (precondição) possui amparo real no efeito gerado no passo diretamente anterior ou no bloco de configuração inicial do ambiente, identificando os *GAPs* lógicos locais antes do início da implementação do código automatizado.

## 4.3 Procedimento Experimental

O fluxo metodológico da pesquisa, detalhado na infraestrutura de *scripts* executáveis e sistematizado na Figura 2, transita do requisito bruto ao relatório final de auditoria através de quatro fases integradas que operacionalizam a abordagem probabilística:

(1) **Fase 1 - Entrada e Estruturação:** Realiza-se a ingestão multiformato e o *parsing* dos artefatos textuais heterogêneos, padronizando e isolando o conjunto de casos de teste para os campos normatizados e estruturais da ISO/IEC/IEEE 29119-3:2021.
(2) **Fase 2 - Processamento Semântico e Vetorial:** Aplica-se a análise linguística (PLN) via *POS-tagging* para a descontaminação do texto e extração de candidatos a estados, seguida pelo mapeamento vetorial densificado em espaço latente via *Sentence-Transformers*, facilitando a coesão terminológica do domínio.
(3) **Fase 3 - Modelagem e Classificação por Machine Learning:** Nesta etapa, os vetores de características (*feature vectors*) alimentam o classificador supervisionado. O motor de *Machine Learning* infere a relação Ação $\rightarrow$ Estado e computa

os *scores* de probabilidade contínuos para as classes funcionais, gerando os dados de conectividade que constroem o Grafo de Dependência Direcionado.

(4) **Fase 4 - Organização e Saída:** O sistema executa os algoritmos de alcançabilidade na estrutura de grafos, validando de forma matemática as cadeias de dependência. O processo resulta na análise quantitativa de divergências textuais, coleta de métricas de validação cruzada e na geração automática do Relatório de Consistência visual para apoio à decisão.

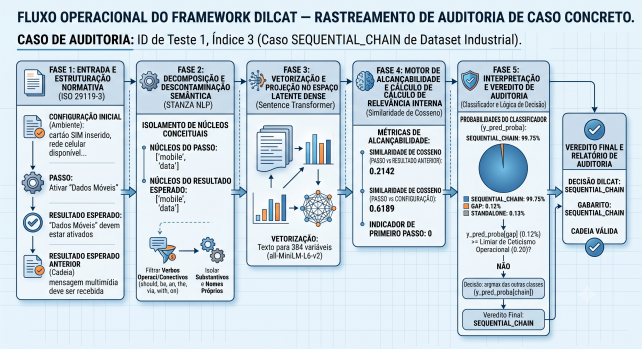


**Figura 2: Fluxo operacional do *framework* DILCAT exemplificando uma cadeia sequencial consistente.**

## 5 Resultados e Discussão

A análise experimental conduzida demonstra que o DILCAT, ao incorporar modelos preditivos de *Machine Learning*, opera sob uma postura conservadora de classificação, dando primazia à detecção de inconsistências em detrimento da tolerância a falsos aceites, de forma a operar sob ótica de minimização de danos, priorizando a segurança e o isolamento do ambiente de testes sobre a similaridade estatística genérica ou aproximações textuais cosméticas [2, 27, 33, 37]. No ecossistema de garantia de qualidade, validar uma dependência funcional inexistente (falso positivo) induz à automação de suítes instáveis, gerando intermitências crônicas e falsos negativos em rotinas de CI [11, 22]. Portanto, o *framework* utiliza as distribuições probabilísticas geradas pelos classificadores como uma métrica de segurança contra o ruído linguístico.

### 5.1 Análise Quantitativa e Matrizes de Confusão

Para evidenciar a dinâmica operacional e mapear o impacto das escolhas arquiteturais de processamento, a Tabela 1 consolida os resultados experimentais obtidos, estruturada de forma hierárquica por conjunto de dados e ordenada de maneira decrescente pelo $F1$-Score específico da classe GAP. Sob a perspectiva de segurança estabelecida pelo *framework*, a capacidade de identificar com precisão as quebras de fluxo funcional (GAPs) sobrepõe-se à busca por uma acurácia global inflada. Esse compromisso de engenharia fica nitidamente demonstrado na análise de estresse fora de domínio (1`anding_page_ecommerce_eng_pt.csv`): o algoritmo Random Forest exibe um desempenho geral aparentemente ideal, com acurácia de 0,94, mas seu *Recall* de 0,00 na classe GAP denuncia uma falha crítica ao permitir que todas as inconsistências passem de forma silenciosa pela auditoria. Em contrapartida, o modelo XGBoost associado ao extrator *all-MiniLM-L6-v2* manifesta o critério conservador preconizado pelo ecossistema, atingindo um *Recall* unitário (1,00) na captura de GAPs. Embora essa postura conservadora reduza a precisão local (0,14) devido ao impacto de um domínio inédito, essa configuração prioriza o recall da classe GAP, ainda que com redução substancial da precisão no cenário fora de domínio. No ensaio de validação cruzada agrupada (`GroupKFold`, $K = 5$), o DILCAT obteve uma acurácia global média de $0,94 \pm 0,02$, com $F1$-Score médio de $0,72 \pm 0,04$ na classe minoria (`GAP`) no arquivo `telecom_matrix_training.csv`. Por sua vez, a avaliação conduzida sobre o conjunto de teste independente (*hold-out* contendo 20% dos casos mantidos completamente não vistos durante a calibração) consolidou a performance da abordagem com acurácia global de 0,98 e $F1$-Score de 0,88 para a classe `GAP`.

### 5.2 Rastreamento Semântico Contínuo: Um Caso Concreto

Para mitigar a abstração do pipeline neuro-simbólico e demonstrar a viabilidade operacional do DILCAT, detalha-se o fluxo de processamento de uma instância real extraída do conjunto de dados industriais de telecomunicações, correspondente ao Índice 3 do identificador de testes de número 1, conforme ilustrado na Figura 2.

Na primeira fase, o *framework* executa a ingestão e a estruturação do artefato baseando-se nos campos preconizados pela norma ISO/IEC/IEEE 29119-3:2021. O ambiente armazena o bloco de *setup* inicial (composto pelos estados de conectividade celular e ativação de dados), a ação textual do passo atual ("*Enable Mobile data*") e o resultado esperado correspondente ("*Mobile data should be enabled*"). Como o caso avaliado não constitui a instrução de abertura da suíte, o motor lógico recupera e retém o efeito gerado pelo passo imediatamente anterior na cadeia de execução: "*multimedia message should be received*".

Durante a fase de decomposição sintática e descontaminação linguística, o pipeline neural remove conectivos e estruturas auxiliares genéricas para isolar os núcleos conceituais de significado. Esse processo reduz a expressão da ação atual aos *tokens* [*enable*, *mobile*, *data*] e a especificação do resultado esperado aos termos [*mobile*, *data*, *enabled*], impedindo que verbos procedimentais repetitivos inflem artificialmente as métricas de proximidade.

Na sequência, o ecossistema projeta esses núcleos textuais em um espaço latente denso por meio do modelo de *embeddings* destilados all-MiniLM-L6-v2, gerando representações matemáticas de 384 variáveis. O motor de alcançabilidade calcula então as distâncias lineares e aponta uma similaridade de cosseno tênue de 0,2142 entre o passo atual e o efeito precedente. Contudo, o algoritmo detecta uma forte ancoragem contextual de 0,6189 em relação ao bloco de configuração inicial do ambiente, consolidando o vetor de características estruturais que alimenta a inteligência preditiva.

Na fase final de tomada de decisão, o classificador supervisionado consome a matriz densa e computa a distribuição contínua de probabilidades. O modelo manifesta uma certeza expressiva ao atribuir 99,75% de probabilidade de pertencimento para a classe de encadeamento consistente (SEQUENTIAL_CHAIN), reservando apenas 0,12% para a classe GAP e 0,13% para a classe STANDALONE.

**Tabela 1: Consolidado de Métricas de Classificação do DILCAT Ordenado pelo Desempenho da Classe GAP**

| Sentence Transformer | Algorithm | Accuracy | F1 (CHAIN) | F1 (STANDALONE) | Precision (GAP) | Recall (GAP) | F1 (GAP) | F1 (Weighted) |
|---|---|---|---|---|---|---|---|---|
| **Dataset: telecom_kappa_original.csv (Avaliação Hold-out** | | | | | | | | |
| all-MiniLM-L6-v2 | XGBOOST | **0.98** | **0.99** | **0.99** | **0.94** | 0.82 | **0.88** | **0.98** |
| paraphrase-multilingual-MiniLM-L12-v2 | XGBOOST | **0.98** | **0.99** | 0.96 | 0.92 | **0.85** | **0.88** | **0.98** |
| all-MiniLM-L6-v2 | RANDOM FOREST | 0.97 | **0.99** | 0.97 | 0.89 | 0.80 | 0.84 | 0.97 |
| paraphrase-multilingual-MiniLM-L12-v2 | RANDOM FOREST | 0.96 | 0.98 | 0.95 | 0.83 | 0.72 | 0.77 | 0.96 |
| **Dataset: telecom_matrix_training.csv (Validação Cruzada GroupKFold - K=5)** | | | | | | | | |
| all-MiniLM-L6-v2 | RANDOM FOREST | **0.94** | **0.97** | **0.94** | 0.70 | 0.75 | **0.72** | **0.94** |
| all-MiniLM-L6-v2 | XGBOOST | **0.94** | **0.97** | **0.94** | **0.71** | 0.72 | **0.72** | **0.94** |
| paraphrase-multilingual-MiniLM-L12-v2 | RANDOM FOREST | 0.93 | **0.97** | 0.92 | 0.63 | **0.78** | 0.70 | **0.94** |
| paraphrase-multilingual-MiniLM-L12-v2 | XGBOOST | 0.93 | **0.97** | 0.91 | 0.62 | 0.72 | 0.67 | 0.93 |
| **Dataset: landing_page_ecommerce_eng_pt.csv (Análise de Estresse Fora do Domínio)** | | | | | | | | |
| all-MiniLM-L6-v2 | XGBOOST | 0.70 | **0.98** | 0.63 | **0.14** | **1.00** | **0.25** | 0.76 |
| paraphrase-multilingual-MiniLM-L12-v2 | XGBOOST | 0.77 | **0.98** | 0.75 | 0.12 | 0.67 | 0.21 | 0.82 |
| all-MiniLM-L6-v2 | RANDOM FOREST | **0.94** | **0.98** | **0.94** | 0.00 | 0.00 | 0.00 | **0.92** |
| paraphrase-multilingual-MiniLM-L12-v2 | RANDOM FOREST | **0.94** | **0.98** | **0.94** | 0.00 | 0.00 | 0.00 | **0.92** |

Como a estimativa de inconsistência funcional permaneceu substancialmente abaixo do limiar conservador adotado pela política de segurança ($\tau = 0,20$), o DILCAT emite o veredito de dependência estrutural legítima, em concordância com o (*ground truth*) chancelado pelo comitê de engenheiros seniores.

## 5.3 O Impacto da Descontaminação Semântica e Filtros POS-Tagging

A estabilidade e precisão do modelo de *Machine Learning* em dados orgânicos ruidosos são apoiadas pela etapa de Descontaminação Semântica, operada via *POS-tagging* do Stanza [25]. Em manuais de teste industriais, verbos operacionais como *"click"*, *"verify"*, *"show"* ou *"tap"* apresentam frequências elevadas e baixa carga discriminativa de domínio. Sem o tratamento adequado, essas estruturas gramaticais repetitivas atuam como um ruído vetorial latente, inflando artificialmente a similaridade de cosseno de base dos *Sentence-Transformers* para a faixa de 0,60 a 0,75, o que pode comprometer ou até inviabilizar qualquer classificação ao favorecer a ocorrência de falsos positivos [1, 28].

A exclusão dinâmica das *stopwords* técnicas e o isolamento dos núcleos substantivos e nomes próprios (`NOUN/PROPN`) reduziram a similaridade base de ruído, que variava de 0,60 a 0,75 no texto bruto, para o patamar de 0,15 a 0,30. Essa descontaminação força os modelos de *Machine Learning* a extraírem padrões baseados estritamente nos objetos de estado e entidades reais do domínio. Como resultado, a precisão do sistema evoluiu de um intervalo instável entre 0,45 e 0,70 para a marca de 1,00, mantendo sua eficácia mesmo nos cenários de maior estresse léxico e abstração.

## 5.4 Análise Qualitativa: Persistência de Contexto vs. Incerteza

O mapeamento probabilístico de divergências residuais revelou o fenômeno da Persistência Indevida de Contexto [14, 16]. Em cenários complexos onde múltiplos estados são empilhados, a representação vetorial denotativa pode induzir o classificador a um comportamento de otimismo semântico, ignorando ações destrutivas de estados anteriores.

Um exemplo empírico mapeado ocorreu no processamento de fluxos contendo rotinas de *Logout* ou interrupção de sessão. Ao identificar uma correspondência vetorial forte com o estado inicial de autenticação no início da suíte, o classificador tendeu a validar dependências em passos tardios, desconsiderando o efeito destrutivo da ação intermediária de desconexão. Este achado delimita o escopo operacional do *framework*: o DILCAT atua como um validador de alcançabilidade semântica e dependência funcional estática [4]. Todavia, a mitigação completa desse fenômeno e a verificação de lógica temporal estrita exigem, como evolução de pesquisa, a integração de camadas superiores de lógica proposicional para monitorar o ciclo de vida volátil dos estados sistêmicos [5].

# 6 Ameaças à Validade

A análise da robustez e da confiabilidade do DILCAT exige a consideração de ameaças multifacetadas que podem influenciar a generalização e a estabilidade dos resultados preditivos reportados.

## 6.1 Validade Interna

A principal limitação interna da abordagem reside no fenômeno da persistência indevida de contexto, fenômeno no qual a similaridade

vetorial permanece alta mesmo após ações destrutivas no sistema, discutida na análise qualitativa. O *framework* valida se um estado sistêmico é semanticamente alcançável através de representações vetoriais densas, mas o modelo de *Machine Learning* indicou que a principal limitação da análise estática baseada em PLN ocorre em sequências que contêm ações de descontinuidade explícita, tal como uma sequência de *login* e *logout*, que são ações destrutivas que podem ocorrer no meio dos testes. Essa escolha de *design* delega a verificação de fluxo dinâmico e a lógica temporal formal para iterações futuras do *framework*. Outra ameaça interna reside na parametrização dos hiperparâmetros e na calibração fina das distribuições probabilísticas do classificador. Para mitigar vieses de sobreajuste (*overfitting*) na matriz de treino sintética (`telecom_matrix_training.csv`), aplicou-se um protocolo de validação cruzada exaustiva, fazendo com que as fronteiras de decisão do modelo preditivo fossem penalizadas na presença de alta entropia ou termos truncados nas 406 instâncias avaliadas.

## 6.2 Validade de Construto

A eficácia da abordagem neuro-simbólica proposta é intrinsecamente dependente da qualidade do processo de Descontaminação Semântica. Caso o pipeline do Stanza falhe no isolamento sintático de núcleos substantivos (`NOUN`/`PROPN`) devido a desvios gramaticais extremos, a presença de verbos operacionais genéricos residuais pode atuar como ruído latente. Esse ruído infla a similaridade base e induz o modelo de *Machine Learning* a estimar falsos positivos de dependência. Para mitigar essa ameaça, o ecossistema incorpora a padronização estrutural da norma ISO/IEC/IEEE 29119-3:2021, facilitando para que o classificador processe apenas campos deterministicamente isolados (precondições e efeitos). Além disso, a construção do *Ground Truth* aplicado sobre as 406 instâncias de transição foi submetida ao cálculo do coeficiente Fleiss' Kappa com três anotadores especialistas independentes. O índice obtido de 0,74 minimiza os riscos de subjetividade humana na definição do gabarito utilizado para treinar e validar o modelo.

## 6.3 Validade Externa

Embora o *framework* tenha demonstrado uma boa estabilidade métrica ao operar sobre os dados estruturados, o escopo da validação industrial concentrou-se originalmente no domínio de 80 casos de teste legados da indústria de telecomunicações. Reconhece-se que a generalização direta desses achados para outros ecossistemas corporativos (como o setor financeiro ou de saúde) requer cautela. O tamanho contido do conjunto de referência inicial é justificado pelo custo computacional e humano elevado exigido pelo protocolo de anotação cega por múltiplos especialistas seniores. Contudo, as distâncias vetoriais e a dispersão terminológica de estados críticos em outros domínios podem apresentar sutilezas que exijam uma recalibragem dos pesos das *features* textuais e dos limiares probabilísticos do classificador. Como estratégia de mitigação e para contornar as restrições de acordos de confidencialidade (*Non-Disclosure Agreements – NDA*), o desdobramento e espelhamento das regras desses casos reais na matriz anonimizada de 406 instâncias de transição atuou como um balizador de estabilidade, permitindo avaliar os limites de degradação do modelo sob cenários reais de estresse léxico.

# 7 Conclusão

Este trabalho propôs e validou um *framework* neuro-simbólico para a auditoria automática de consistência em suítes de teste legadas escritas em linguagem natural livre. A abordagem desenvolvida estabelece uma alternativa prática para mitigar a dependência de modelos heurísticos rígidos de decisão por meio de um pipeline preditivo baseado em *Machine Learning* supervisionado. A solução demonstrou que o acoplamento entre a flexibilidade semântica de *embeddings* gerados via *Sentence-Transformers* e o rigor estrutural da norma ISO/IEC/IEEE 29119-3:2021 permite mitigar os efeitos da entropia e da ambiguidade na especificação de requisitos.

A avaliação experimental indicou uma acurácia global de 94% no conjunto avaliado, composto por casos de teste da indústria de telecomunicações, em que os resultados indicaram ainda que a filtragem via *POS-tagging* esteve associada à redução da similaridade atribuída ao ruído gramatical e à melhoria das métricas observadas. Além disso, o *framework* se mostrou capaz de identificar situações de maior ambiguidade e *GAPs* lógicos nas instâncias avaliadas, antes do início do processo de execução dos casos de teste.

Como contribuição para o direcionamento de pesquisas futuras, mapeia-se a necessidade de mitigar o fenômeno da *Persistência Indevida de Contexto*. A abordagem proposta envolve a concepção de um *State Tracker* volátil que incorpore anotações de polaridade de verbos via *Stanza*, classificando as ações procedimentais em geradoras ou destrutivas de estado. Essa evolução arquitetural permitirá que o motor transicione da análise estática de alcançabilidade semântica para a Verificação Formal de Modelos Temporais, assegurando de forma determinística que estados desativados ou invalidados não sejam propagados ao longo de cadeias sequenciais complexas de execução.

## Disponibilidade de Artefatos

Devido a acordos de confidencialidade com os parceiros industriais, o conjunto de dados primário, contendo os casos de teste reais do domínio de telecomunicações, não pode ser disponibilizado publicamente.

Contudo, para favorecer a transparência e a reprodutibilidade técnica do *framework*, foram depositados no repositório Zenodo (https://zenodo.org/records/21120571) os seguintes materiais:

- **Datasets de Benchmarking:** conjuntos de dados utilizados nos experimentos, incluindo o `telecom_matrix_training`, que representa a estrutura tabular resultante da decomposição dos 80 casos de teste industriais em 406 instâncias de passo/transição, preservando a associação com seus respectivos `test_case_id`, e o `landing_page_ecommerce_eng_pt`, utilizado na análise fora de domínio;
- **Código-Fonte:** implementação completa do auditor, incluindo os módulos de processamento linguístico via Stanza, geração de representações semânticas via Sentence-Transformers, classificação supervisionada e análise das relações de dependência.

## Agradecimentos

Esta pesquisa foi realizada com apoio de recursos da Lei de Informática (Lei Federal nº 8.387/1991), por meio de convênio firmado com a Universidade Federal do Acre (UFAC). As pessoas autoras

agradecem à ferramenta de inteligência artificial Manus[1] no apoio à criação e edição da Figura 2 utilizada neste trabalho. Ao Claude[2], utilizado para apoio na criação do *dataset* contendo instâncias de um sistema de comércio eletrônico.

## Referências

[1] Malak Al-Hassan, Bilal Abu-Salih, Esra'a Alshdaifat, Ahmad Aloqaily, and Ali Rodan. 2024. An improved fusion-based semantic similarity measure for effective collaborative filtering recommendations. *International Journal of Computational Intelligence Systems* 17, 1 (2024), 45.

[2] Paul Ammann and Jeff Offutt. 2017. *Introduction to Software Testing* (2nd ed.). Cambridge University Press, Cambridge. doi:10.1017/9781316771273

[3] Halima Ayenew and Mekonnen Wagaw. 2024. Software Test Case Generation Using Natural Language Processing (NLP): A Systematic Literature Review. *Artificial Intelligence Evolution* 5, 1 (Jan. 2024), 1–10. doi:10.37256/aie.5120243220

[4] Christel Baier and Joost-Pieter Katoen. 2008. *Principles of Model Checking.* MIT Press, Cambridge, MA, USA.

[5] Andreas Bauer, Martin Leucker, and Christian Schallhart. 2011. Runtime Verification for LTL and TLTL. *ACM Transactions on Software Engineering and Methodology* 20, 4, Article 14 (Sept. 2011), 64 pages. doi:10.1145/2000799.2000800

[6] Adam Berger, Stephen A Della Pietra, and Vincent J Della Pietra. 1996. A maximum entropy approach to natural language processing. *Computational linguistics* 22, 1 (1996), 39–71.

[7] Matteo Biagiola, Andrea Stocco, Filippo Ricca, and Paolo Tonella. 2020. Dependency-Aware Web Test Generation. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)* (2020-10). IEEE, IEEE, Los Alamitos, CA, USA, 175–185. doi:10.1109/ICST46399.2020.00027

[8] Muhammad Shoaib Farooq, Uzma Omer, Amna Ramzan, Mansoor Ahmad Rasheed, and Zabihullah Atal. 2023. Behavior driven development: A systematic literature review. *IEEE access* 11 (2023), 88008–88024.

[9] Jannik Fischbach, Julian Frattini, Andreas Vogelsang, Daniel Mendez, Michael Unterkalmsteiner, Andreas Wehrle, Pablo Restrepo Henao, Parisa Yousefi, Tedi Juricic, Jeannette Radduenz, and Carsten Wiecher. 2023. Automatic creation of acceptance tests by extracting conditionals from requirements: NLP approach and case study. *Journal of Systems and Software* 197 (2023), 111549. doi:10.1016/j.jss.2022.111549

[10] Joseph L Fleiss. 1971. Measuring nominal scale agreement among many raters. *Psychological bulletin* 76, 5 (1971), 378.

[11] Zhaoqiang Guo, Tingting Tan, Shiran Liu, Xutong Liu, Wei Lai, Yibiao Yang, Yanhui Li, Lin Chen, Wei Dong, and Yuming Zhou. 2023. Mitigating false positive static analysis warnings: Progress, challenges, and opportunities. *IEEE Transactions on Software Engineering* 49, 12 (2023), 5154–5188.

[12] Shifa-e-Zehra Haidry and Tim Miller. 2013. Using Dependency Structures for Prioritization of Functional Test Suites. *IEEE Transactions on Software Engineering* 39, 2 (2013), 258–275.

[13] Pascal Hitzler, Aaron Eberhart, Monireh Ebrahimi, Md Kamruzzaman Sarker, and Lu Zhou. 2022. Neuro-symbolic approaches in artificial intelligence. *National Science Review* 9, 6 (06 2022), nwac035. doi:10.1093/nsr/nwac035

[14] Xinmeng Huang, Shuo Li, Mengxin Yu, Matteo Sesia, Hamed Hassani, Insup Lee, Osbert Bastani, and Edgar Dobriban. 2024. Uncertainty in Language Models: Assessment through Rank-Calibration. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 284–312. doi:10.18653/v1/2024.emnlp-main.18

[15] ISO. 2021. IEEE/ISO/IEC International Standard for Software and systems engineering–Software testing–Part 3:Test documentation. *ISO/IEC/IEEE 29119-3:2021(E)* 2021, 29119-3 (2021), 1–98. doi:10.1109/IEEESTD.2021.9591577

[16] Lorenz Kuhn, Yarin Gal, and Sebastian Farquhar. 2023. Semantic Uncertainty: Linguistic Invariances for Uncertainty Estimation in Natural Language Generation. *International Conference on Learning Representations* (2023), 19 pages. doi:10.48550/arXiv.2302.09664

[17] Yihao Li, Pan Liu, Haiyang Wang, Jie Chu, and W Eric Wong. 2025. Evaluating large language models for software testing. *Computer Standards & Interfaces* 93 (2025), 103942.

[18] B. B. Liubinskyi and M. M. Zvarych. 2026. A Graph-Based Model for Automatic Test Case Generation from Textual Requirements with Hierarchical Coverage. *Mathematical Modeling and Computing* 13, 1 (2026), 311–320. doi:10.23939/mmc2026.01.311

[19] Jordy Navarro and Ronald Ibarra. 2026. Automatic test case generation using natural language processing: A systematic mapping study. *Information and Software Technology* 189 (2026), 107929. doi:10.1016/j.infsof.2025.107929

[20] Rongqi Pan, Taher A. Ghaleb, and Lionel C. Briand. 2024. LTM: Scalable and Black-Box Similarity-Based Test Suite Minimization Based on Language Models. *IEEE Transactions on Software Engineering* 50, 11 (2024), 2890–2907. doi:10.1109/TSE.2024.3465143

[21] Rongqi Pan, Feifei Niu, Lionel C. Briand, and Hanyang Hu. 2026. Requirements Coverage-Guided Minimization for Natural Language Test Cases. *ACM Transactions on Software Engineering and Methodology* (Feb. 2026), 24 pages. doi:10.1145/3798164 Just Accepted.

[22] Yun Peng, Shuqing Li, Wenwei Gu, Yichen Li, Wenxuan Wang, Cuiyun Gao, and Michael R. Lyu. 2022. Revisiting, Benchmarking and Exploring API Recommendation: How Far Are We? *IEEE Transactions on Software Engineering* 49, 4 (2022), 1876–1897. doi:10.1109/TSE.2022.3197600

[23] Zi Peng, Tse-Hsun Chen, and Jinqiu Yang. 2022. Revisiting Test Impact Analysis in Continuous Testing From the Perspective of Code Dependencies. *IEEE Transactions on Software Engineering* 48, 6 (2022), 1979–1993. doi:10.1109/TSE.2020.3045914

[24] Roger S Pressman. 2015. Software engineering: a practitioner's approach.

[25] Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. 2020. Stanza: A Python Natural Language Processing Toolkit for Many Human Languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, Asli Celikyilmaz and Tsung-Hsien Wen (Eds.). Association for Computational Linguistics, Online, 101–108. doi:10.18653/v1/2020.acl-demos.14

[26] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (Eds.). Association for Computational Linguistics, Hong Kong, China, 3982–3992. doi:10.18653/v1/D19-1410

[27] Cynthia Rudin. 2019. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature machine intelligence* 1, 5 (2019), 206–215.

[28] Serhad Sarica and Jianxi Luo. 2021. Stopwords in technical language processing. *PLOS ONE* 16, 8 (2021), e0254937. doi:10.1371/journal.pone.0254937

[29] Zhengyan Shi, Giuseppe Castellucci, Simone Filice, Saar Kuzi, Elad Kravi, Eugene Agichtein, Oleg Rokhlenko, and Shervin Malmasi. 2025. Ambiguity Detection and Uncertainty Calibration for Question Answering with Large Language Models. In *Proceedings of the 5th Workshop on Trustworthy NLP (TrustNLP 2025)*, Trista Cao, Anubrata Das, Tharindu Kumarage, Yixin Wan, Satyapriya Krishna, Ninareh Mehrabi, Jwala Dhamala, Anil Ramakrishna, Aram Galystan, Anoop Kumar, Rahul Gupta, and Kai-Wei Chang (Eds.). Association for Computational Linguistics, Albuquerque, New Mexico, 41–55. doi:10.18653/v1/2025.trustnlp-main.4

[30] Amjed Tahir, Shawn Rasheed, Jens Dietrich, Negar Hashemi, and Lu Zhang. 2023. Test flakiness' causes, detection, impact and responses: A multivocal review. *Journal of Systems and Software* 206 (2023), 111837. doi:10.1016/j.jss.2023.111837

[31] Sahar Tahvili, Leo Hatvani, Michael Felderer, Wasif Afzal, and Markus Bohlin. 2019. Automated Functional Dependency Detection Between Test Cases Using Doc2Vec and Clustering. In *2019 IEEE International Conference On Artificial Intelligence Testing (AITest)* (2019-04). IEEE, Newark, CA, USA, 19–26. doi:10.1109/AITest.2019.00-13

[32] Sahar Tahvili, Leo Hatvani, Michael Felderer, Francisco Gomes de Oliveira Neto, Wasif Afzal, and Robert Feldt. 2025. Comparative analysis of text mining and clustering techniques for assessing functional dependency between manual test cases. *Software Quality Journal* 33, 24 (2025), 1–36. doi:10.1007/s11219-025-09722-7

[33] Guido Tebes, Luis Olsina, Denis Peppino, and Pablo Becker. 2020. TestTDO: A Top-Domain Software Testing Ontology. In *Proceedings of the XXIII Iberoamerican Conference on Software Engineering, CIbSE 2020*, Claudia P. Ayala, Leonardo Murta, Daniela Soares Cruzes, Eduardo Figueiredo, Carla Silva, Jose Luis de la Vara, Breno de França, Martín Solari, Guilherme Horta Travassos, and Ivan Machado (Eds.). Curran Associates, Curitiba, Paraná, Brazil, 364–377.

[34] Junjie Wang, Yuchao Huang, Chunyang Chen, Zhe Liu, Song Wang, and Qing Wang. 2024. Software Testing With Large Language Models: Survey, Landscape, and Vision. *IEEE Transactions on Software Engineering* 50, 4 (2024), 911–936. doi:10.1109/TSE.2024.3368208

[35] Zhenzhen Yang, Rubing Huang, Chenhui Cui, Nan Niu, and Dave Towey. 2025. Requirements-Based Test Generation: A Comprehensive Survey. *ACM Trans. Softw. Eng. Methodol.* (Oct. 2025). doi:10.1145/3771727 Just Accepted.

[36] Dongran Yu, Bo Yang, Dayou Liu, Hui Wang, and Shirui Pan. 2023. A survey on neural-symbolic learning systems. *Neural Netw.* 166, C (Sept. 2023), 105–126. doi:10.1016/j.neunet.2023.06.028

[37] Zeyu Yu, Ning Fang, Song Wang, and Yiling Ling. 2024. Software Testing with Large Language Models: Survey, Landscape, and Vision. *IEEE Transactions on Software Engineering* 50, 4 (2024), Detailed pagination or article ID. arXiv:2307.07221 [cs.SE] doi:10.1109/TSE.2024.3368208

[1] https://manus.im
[2] https://claude.ai/