Detecção de Mutantes Equivalentes com MutantBench: Uma Replicação Experimental e Avaliação de Baselines Supervisionados

Vinicius N. Lopes
Universidade Federal do Pampa (UNIPAMPA)
Alegrete, RS, Brasil
viniciusnunez.aluno@unipampa.edu.br

Elder Rodrigues
Universidade Federal do Pampa (UNIPAMPA)
Alegrete, RS, Brasil
elderrodrigues@unipampa.edu.br

Maicon Bernardino
Universidade Federal do Pampa (UNIPAMPA)
Alegrete, RS, Brasil
bernardino@acm.org

Resumo

A detecção de mutantes equivalentes permanece como um dos principais desafios do *Mutation Testing* devido ao elevado custo associado à identificação manual desses casos. Este trabalho apresenta uma replicação parcial baseada no benchmark *MutantBench*, avaliando classificadores supervisionados leves executados localmente. Foram avaliados *Logistic Regression*, *Linear SVM*, *Random Forest* e *KNN* com representação TF-IDF baseada em n-gramas de caracteres. O protocolo preserva a divisão oficial entre 1.652 instâncias de treinamento e 1.650 de teste e executa integralmente em CPU. O *KNN* apresentou o melhor desempenho entre os modelos avaliados, com 92,91% de acurácia, F1-score de 72,08% para a classe equivalente e F1 macro de 84,01%. O *Random Forest* obteve a maior revocação da classe equivalente, 71,89%, evidenciando perfis de erro distintos. A comparação com resultados da literatura é indireta e não demonstra equivalência entre abordagens. Os resultados estabelecem referências reproduzíveis e de baixo custo para futuras avaliações.

Palavras-chave

Teste de Mutação, Mutantes Equivalentes, Large Language Models, Engenharia de Software, Verificação e Validação

1 Introdução

O *Mutation Testing* é uma técnica de avaliação de testes baseada na inserção sistemática de pequenas alterações sintáticas em um programa. Cada alteração produz uma variante chamada mutante, que representa uma possível falha ou desvio comportamental. Uma suíte de testes é considerada mais efetiva quando consegue distinguir o programa original de seus mutantes, isto é, quando os testes falham na presença dessas modificações. Desde os trabalhos clássicos de DeMillo, Lipton e Sayward [3], a análise de mutação tem sido utilizada como critério rigoroso para investigar a qualidade de dados de teste, cobertura de comportamento e acoplamento entre falhas artificiais e falhas reais [6, 7, 11].

Apesar de seu valor analítico, o teste de mutação apresenta custos elevados. A execução de uma suíte para um grande número de mutantes pode ser cara, e a interpretação dos mutantes sobreviventes exige cuidado. Entre esses sobreviventes, os mutantes equivalentes constituem o caso mais problemático: eles modificam a forma sintática do programa, mas preservam seu comportamento observável para todas as entradas possíveis. Como não há caso de teste capaz de matar um mutante equivalente, sua presença reduz artificialmente o escore de mutação e pode levar a conclusões equivocadas sobre a qualidade da suíte [8, 9].

A detecção automática de mutantes equivalentes é, em geral, um problema indecidível, pois requer raciocinar sobre equivalência semântica entre programas. Na prática, pesquisadores e ferramentas recorrem a aproximações: técnicas baseadas em compilação, análise estática, restrições, execução dinâmica, classificação supervisionada ou modelos estruturais. Tais estratégias reduzem parte do custo, mas frequentemente apresentam baixa cobertura semântica ou dificuldade de generalização para programas, operadores e domínios não observados durante o treinamento.

Nos últimos anos, modelos pré-treinados para código e LLMs passaram a demonstrar resultados relevantes em tarefas de compreensão, busca, geração e transformação de programas [4, 5, 15, 16]. Em particular, Tian et al. [12] investigaram o uso de LLMs para detecção de mutantes equivalentes e reportaram ganhos expressivos em relação a abordagens tradicionais. Ainda assim, os resultados da literatura não eliminam questões práticas: modelos maiores podem exigir APIs externas, altos custos por requisição, maior latência, controle cuidadoso de *prompts* e mecanismos de reprodução que nem sempre estão disponíveis em ambientes acadêmicos ou industriais.

Este trabalho tem como objetivo realizar uma replicação parcial do protocolo experimental proposto por Tian et al. [12] utilizando o benchmark *MutantBench*, avaliando o desempenho de classificadores supervisionados leves executados localmente. Os resultados obtidos são comparados, de forma indireta, com resultados reportados na literatura para modelos pré-treinados e *Large Language Models* (LLMs), buscando analisar a competitividade dessas abordagens sob a perspectiva de custo computacional, reprodutibilidade e viabilidade prática.

Apesar dos avanços recentes obtidos por modelos de linguagem de grande porte, ainda existem poucos estudos independentes capazes de reproduzir esses resultados em ambientes computacionais convencionais. Além disso, permanece pouco claro até que ponto classificadores supervisionados tradicionais continuam competitivos quando comparados às abordagens baseadas em LLMs para a detecção de mutantes equivalentes.

Diante desse cenário, este trabalho realiza uma replicação experimental parcial baseada no benchmark MutantBench, buscando avaliar a reprodutibilidade dos resultados reportados na literatura e investigar a relação entre eficácia preditiva, custo computacional e disponibilidade de artefatos experimentais.

Com base nesses desafios, o estudo organiza a investigação em três questões de pesquisa. **RQ1**: qual é o desempenho de classificadores supervisionados leves na detecção de mutantes equivalentes utilizando o *MutantBench*? **RQ2**: como os resultados desses classificadores se posicionam em relação aos valores reportados na literatura, considerando as limitações de uma comparação indireta? **RQ3**: quais tipos de erro são mais frequentes entre os classificadores avaliados?

As contribuições deste artigo são: (i) uma replicação parcial reproduzível baseada no benchmark *MutantBench*; (ii) uma avaliação experimental de classificadores supervisionados leves executados em infraestrutura computacional convencional; (iii) uma comparação indireta, com métricas compatibilizadas, com resultados reportados na literatura; e (iv) uma análise dos diferentes perfis de erro dos classificadores.

2 Fundamentação e Trabalhos Relacionados

2.1 Teste de Mutação e Mutantes Equivalentes

O teste de mutação parte da hipótese de que pequenas alterações sintáticas podem simular falhas plausíveis cometidas por desenvolvedores. A qualidade da suíte é estimada pelo escore de mutação, geralmente calculado como a razão entre mutantes mortos e mutantes não equivalentes. Essa formulação torna os mutantes equivalentes um fator crítico: se eles forem contabilizados como mutantes vivos, o escore será subestimado e a análise poderá induzir esforço desnecessário de criação de testes.

O problema tem sido discutido desde os primeiros estudos sobre teste de mutação e permaneceu como um dos principais entraves para adoção prática da técnica [6, 8, 9]. Offutt e Pan [9] investigaram a detecção automática de mutantes equivalentes e caminhos inviáveis, evidenciando a relação entre equivalência, raciocínio semântico e limitações de análise automática. Revisões posteriores classificaram estratégias em categorias como detecção, sugestão e prevenção de geração de mutantes equivalentes [8, 11].

2.2 Técnicas Tradicionais

Técnicas tradicionais para lidar com mutantes equivalentes buscam reduzir custo por meio de aproximações conservadoras. O *Trivial Compiler Equivalence* (TCE) compara representações geradas após compilação e otimização [10]. Se o código original e o mutante convergem para a mesma representação, o mutante pode ser classificado como equivalente. A vantagem do TCE está em sua eficiência e determinismo; sua aplicação, contudo, depende da infraestrutura, das otimizações e das representações produzidas pelo compilador. Além disso, a técnica captura apenas equivalências que se manifestam nesse nível, deixando fora equivalências dependentes de contexto, pré-condições ou propriedades semânticas mais profundas.

Outra família de abordagens utiliza aprendizado de máquina supervisionado. Mutantes são descritos por atributos léxicos, sintáticos, estruturais ou dinâmicos, e classificadores como SVM, Random Forest e redes neurais aprendem a distinguir equivalentes de não equivalentes. Essa linha reduz o esforço manual, mas depende da qualidade das características extraídas e de conjuntos de dados anotados, que são escassos e caros de produzir. O problema de generalização é relevante: padrões aprendidos em um conjunto de operadores ou projetos podem não se transferir para outros contextos.

Modelos estruturais de aprendizado profundo tentam reduzir a dependência de atributos manuais. O ASTNN, por exemplo, aprende representações de código a partir da árvore sintática abstrata, decompondo-a em unidades estruturais [17]. Representações baseadas em grafos também foram exploradas para capturar relações de controle, fluxo de dados e dependências semânticas [1]. Embora mais expressivas, essas técnicas aumentam o custo de treinamento e exigem infraestrutura mais robusta.

2.3 Modelos de Código e LLMs

Modelos pré-treinados para código, como CodeBERT, CodeT5, CodeT5+ e UniXCoder, foram propostos para aprender representações de programas a partir de grandes coleções de código-fonte e linguagem natural [4, 5, 15, 16]. Esses modelos diferem de classificadores tradicionais porque não dependem exclusivamente de atributos projetados manualmente; eles aprendem representações contextuais que podem capturar identificadores, estrutura, padrões de uso e relações entre fragmentos de código.

No contexto do teste de mutação, LLMs têm sido investigados tanto para gerar mutantes quanto para analisar sua utilidade e equivalência. Wang et al. [14] mostram que LLMs podem gerar mutações mais diversas e próximas de falhas reais, mas também observam problemas como baixa compilabilidade e maior ocorrência de mutantes inúteis ou equivalentes. Tian et al. [12], por sua vez, avaliam LLMs diretamente na detecção de mutantes equivalentes e relatam ganhos de desempenho em comparação com técnicas anteriores.

Como evolução recente, Danilov et al. [2] propuseram a *Cluster Purge Loss*, uma função de perda voltada à organização do espaço de embeddings do UniXCoder para a detecção de mutantes equivalentes. Esses estudos demonstram o potencial de modelos pré-treinados para código, mas também reforçam a necessidade de protocolos experimentais cuidadosos, pois os resultados podem depender da representação, da estratégia de ajuste, dos dados utilizados e dos recursos computacionais disponíveis.

3 Metodologia

3.1 Conjunto de Dados

O estudo utiliza o benchmark MutantBench, proposto por van Hijfte e Oprescu [13] e empregado por Tian et al. [12]. O conjunto contém pares de métodos Java e seus respectivos mutantes, anotados quanto à equivalência semântica. Sua adoção é adequada por três razões: permite comparação com resultados recentes, concentra-se em uma linguagem amplamente utilizada em ferramentas de teste e fornece uma base anotada para treinamento e avaliação de classificadores supervisionados.

Os experimentos seguiram integralmente a divisão oficial disponibilizada pelo MutantBench, utilizando o conjunto *Mutant_A* para treinamento (1652 instâncias) e o conjunto *Mutant_B* para teste (1650 instâncias). Não foi realizada amostragem adicional nem reparticionamento dos dados, preservando o protocolo experimental adotado por Tian et al. [12]. Essa decisão favorece a reprodutibilidade dos experimentos e permite comparação direta com resultados reportados na literatura.

Cada instância corresponde a um par formado por um método Java original e sua respectiva versão mutante. O pré-processamento preservou o código-fonte necessário para a representação dos pares e utilizou a divisão oficial do benchmark. Comentários, identificadores dos pares, rótulos de equivalência e metadados dos projetos não foram empregados como atributos de entrada dos classificadores.

3.2 Técnicas e Modelos

Esta pesquisa concentrou-se exclusivamente na avaliação experimental de classificadores supervisionados leves executados localmente, compreendendo *Logistic Regression*, *Linear SVM*, *Random Forest* e *KNN*. Esses modelos foram escolhidos por apresentarem baixo custo computacional, elevada reprodutibilidade e facilidade

de execução em infraestrutura convencional, permitindo investigar sua eficácia na detecção de mutantes equivalentes utilizando o benchmark *MutantBench*.

Os classificadores foram treinados utilizando representações TF-IDF baseadas em n-gramas de caracteres extraídos dos pares original-mutante. O desempenho foi avaliado por meio de métricas de classificação, incluindo acurácia, precisão, revocação (*recall*) e F1-score, além da análise das matrizes de confusão.

Os pares original-mutante foram convertidos em uma única entrada textual no formato `CODE_A`, código original, `CODE_B` e código mutante. A representação foi produzida com TF-IDF baseado em n-gramas de caracteres delimitados por palavras, utilizando `analyzer=char_wb`, intervalo de n-gramas entre 3 e 5, limite de 50.000 atributos, normalização L2 e preservação de letras maiúsculas e minúsculas. Os demais parâmetros não especificados mantiveram os valores padrão da biblioteca Scikit-Learn.

Modelos estruturais, modelos pré-treinados para código e *Large Language Models* (LLMs) não foram executados nesta pesquisa. Os resultados apresentados para essas abordagens foram obtidos exclusivamente a partir da literatura, sendo utilizados como referência para uma comparação indireta com os classificadores supervisionados avaliados. Dessa forma, as conclusões referentes a essas categorias restringem-se à discussão comparativa dos resultados reportados, não constituindo uma avaliação experimental direta.

3.3 Métricas

Neste trabalho foram analisadas as métricas de acurácia, *precision*, *recall* e *F1-score*, considerando a classe "equivalente" como positiva. A métrica *precision* representa a proporção de mutantes classificados como equivalentes que realmente pertencem a essa classe, enquanto o *recall* indica a proporção de mutantes equivalentes corretamente identificados. O *F1-score* da classe equivalente foi utilizado como medida de equilíbrio entre ambos. Também foi calculado o F1 macro, correspondente à média não ponderada do F1 das duas classes, para permitir uma comparação metricamente compatível com os valores reportados por Tian et al. [12].

Além das métricas agregadas, foram analisadas as matrizes de confusão de cada classificador, permitindo identificar os padrões de erros, especialmente os falsos positivos e falsos negativos. Essa análise complementa as métricas tradicionais ao evidenciar os tipos de erro mais frequentes na detecção de mutantes equivalentes.

Aspectos relacionados ao custo operacional, como tempo de processamento, consumo de recursos computacionais e custos associados à utilização de modelos de grande porte, não foram avaliados experimentalmente nesta pesquisa. Esses aspectos são discutidos apenas com base nos resultados reportados na literatura, com o objetivo de contextualizar as diferenças entre classificadores supervisionados leves e abordagens baseadas em *Large Language Models* (LLMs).

3.4 Escopo da Replicação

Este trabalho realiza uma replicação parcial do estudo conduzido por Tian et al. [12]. O objetivo não é reproduzir integralmente todos os experimentos originalmente reportados, mas avaliar a reprodutibilidade dos resultados utilizando infraestrutura computacional convencional e classificadores supervisionados executados localmente.

Em particular, os resultados reportados para Large Language Models (LLMs) foram utilizados exclusivamente como referência comparativa extraída dos artefatos oficiais disponibilizados pelos autores. Nenhum modelo proprietário, incluindo GPT, Claude ou Gemini, foi executado durante esta pesquisa.

Os experimentos executados localmente concentraram-se em abordagens reproduzíveis em CPU, incluindo Logistic Regression, Linear SVM, Random Forest e KNN.

3.5 Ambiente Experimental

Os experimentos foram executados localmente utilizando Python e a biblioteca Scikit-Learn para treinamento e avaliação dos classificadores supervisionados Logistic Regression, Linear SVM, Random Forest e KNN. Todo o processamento, incluindo treinamento, inferência e cálculo das métricas de avaliação, foi realizado em CPU, sem utilização de GPU dedicada.

Essa configuração foi adotada para avaliar a viabilidade prática de abordagens supervisionadas leves em infraestrutura convencional. Diferentemente de abordagens baseadas em *Large Language Models* (LLMs), que frequentemente demandam hardware especializado ou serviços de inferência externos, os classificadores avaliados podem ser treinados e executados integralmente em CPU, preservando a reprodutibilidade dos experimentos e reduzindo os custos computacionais associados à execução.

Conforme descrito na Seção 3.1, os experimentos utilizaram integralmente a divisão oficial do MutantBench, preservando os conjuntos *Mutant_A* para treinamento e *Mutant_B* para teste, sem qualquer reparticionamento adicional dos dados.

Os scripts de execução, configurações experimentais e resultados produzidos foram organizados para permitir a reprodução dos experimentos. Os scripts, as configurações experimentais, os modelos treinados e os resultados estão disponíveis no repositório permanente informado na seção de Disponibilidade de Artefatos.

A configuração utilizada representa um ambiente de desenvolvimento convencional, demonstrando que os experimentos podem ser reproduzidos sem necessidade de hardware especializado ou infraestrutura de alto desempenho.

4 Resultados Experimentais

4.1 Desempenho dos Classificadores

Como etapa de validação do protocolo proposto, foi realizada uma replicação parcial do estudo de Tian et al. [12] utilizando o benchmark MutantBench.

Foram empregados os conjuntos hierárquicos A e B do benchmark, utilizados respectivamente para treinamento e avaliação dos modelos. O conjunto de treinamento contém 1.652 pares de programas, enquanto o conjunto de teste contém 1.650 pares.

A Tabela 3 apresenta a distribuição das classes nos conjuntos utilizados. Observa-se forte desbalanceamento, com aproximadamente 85% dos exemplos pertencentes à classe não equivalente e 15% à classe equivalente. Em função desse desbalanceamento, a análise não se restringe à acurácia, reportando também *precision*, *recall* e *F1-score* para a classe equivalente.

Foram avaliadas quatro abordagens supervisionadas leves executadas localmente: Regressão Logística, SVM Linear, Random Forest e KNN. Os classificadores supervisionados utilizaram representação TF-IDF baseada em n-gramas de caracteres extraídos dos pares original-mutante. A representação foi construída a partir do código do método original e do respectivo método mutante. Identificadores dos pares, identificadores de fragmentos, rótulos

Tabela 1: Configurações utilizadas nos classificadores supervisionados.

Modelo	Configuração
Logistic Regression	$C = 1{,}0$, penalização L2, solver `lbfgs`, `class_weight=balanced`, `max_iter=2000` e `random_state=42`.
Linear SVM	$C = 1{,}0$, perda `squared_hinge`, penalização L2, `class_weight=balanced`, `max_iter=1000` e `random_state=42`.
Random Forest	300 árvores, critério Gini, `max_features=sqrt`, `class_weight=balanced`, execução paralela e `random_state=42`.
KNN	$k = 5$, distância euclidiana, pesos uniformes e seleção automática do algoritmo de busca.

Tabela 2: Configuração do ambiente experimental.

Categoria	Configuração
Arquitetura	x86-64
Processador	Intel Core i7-12700H
Memória RAM	16 GB DDR4
Linguagem	Python 3.12.4
Biblioteca	Scikit-Learn 1.6.1
Benchmark	MutantBench
Treinamento	Mutant_A (1652 instâncias)
Teste	Mutant_B (1650 instâncias)
Execução	Local (CPU, sem aceleração por GPU)

Tabela 3: Distribuição das classes nos conjuntos de treinamento e teste.

Conjunto	Não equivalente	Equivalente
Treinamento	1402	250
Teste	1401	249

de equivalência e metadados dos projetos não foram utilizados como atributos.

A Tabela 4 apresenta os resultados obtidos no conjunto de teste. Como referência trivial, o classificador majoritário, que sempre prevê a classe não equivalente, alcança 84,91% de acurácia, mas F1 igual a zero para a classe equivalente. O KNN apresentou o melhor desempenho entre os modelos avaliados, alcançando acurácia de 92,91%, F1 de 72,08% para a classe equivalente e F1 macro de 84,01%. Embora o *Random Forest* tenha obtido menor acurácia global, apresentou o maior *recall* para a classe equivalente (71,89%), indicando maior capacidade de recuperação de mutantes equivalentes.

Tabela 4: Desempenho dos classificadores no conjunto de teste.

Modelo	Acc.	Prec.	Rec.	F1-E	F1-M
Majoritário	0,8491	0,0000	0,0000	0,0000	0,4592
Reg. Logística	0,8436	0,4866	0,6586	0,5597	0,7323
SVM Linear	0,8612	0,5336	0,6386	0,5814	0,7491
Random Forest	0,8964	0,6393	0,7189	0,6767	0,8075
KNN	0,9291	0,8882	0,6064	0,7208	0,8401

Na tabela, F1-E representa o F1 da classe equivalente e F1-M representa o F1 macro. A comparação com o baseline majoritário mostra por que a acurácia, isoladamente, é insuficiente: a Regressão Logística obtém acurácia inferior à regra constante, apesar de recuperar parte dos mutantes equivalentes.

Observa-se que o KNN apresentou o melhor desempenho geral, atingindo F1-score de 72,08%. Esse resultado sugere que a similaridade lexical entre métodos originais e mutantes permanece uma característica relevante para a identificação de equivalência semântica.

Por outro lado, o Random Forest apresentou maior recall, indicando maior sensibilidade na recuperação de mutantes equivalentes, embora à custa de uma redução na precisão.

A Tabela 5 apresenta as matrizes de confusão consolidadas dos classificadores avaliados. O KNN reduziu falsos positivos, classificando incorretamente apenas 19 mutantes não equivalentes como equivalentes, enquanto o Random Forest recuperou 179 dos 249 mutantes equivalentes do conjunto de teste.

Tabela 5: Matrizes de confusão dos classificadores avaliados.

Logistic Regression

Real \ Pred.	NE	E
NE	**1228**	173
E	85	**164**

Linear SVM

Real \ Pred.	NE	E
NE	**1262**	139
E	90	**159**

Random Forest

Real \ Pred.	NE	E
NE	**1300**	101
E	70	**179**

KNN

Real \ Pred.	NE	E
NE	**1382**	19
E	98	**151**

NE = Não Equivalente E = Equivalente

Os resultados indicam que abordagens supervisionadas leves ainda representam alternativas competitivas para detecção de mutantes equivalentes, especialmente em cenários com restrições de custo computacional e infraestrutura.

4.2 Análise dos Tipos de Erro

A análise das matrizes de confusão evidencia diferenças relevantes no comportamento dos classificadores em relação aos tipos de erro produzidos. A Logistic Regression apresentou a maior incidência de falsos positivos, enquanto a Linear SVM também apresentou um número relativamente elevado desse tipo de erro quando comparada aos demais classificadores.

O KNN apresentou o comportamento mais conservador, produzindo apenas 19 falsos positivos, o menor valor entre todos os classificadores avaliados. Em contrapartida, deixou de identificar 98 mutantes equivalentes, aumentando a quantidade de falsos negativos. Esse comportamento indica uma tendência do modelo em evitar classificar mutantes como equivalentes quando não há evidências suficientes de similaridade, reduzindo o risco de descartar mutantes não equivalentes.

Por outro lado, o Random Forest apresentou o maior *recall* para a classe equivalente, recuperando 179 dos 249 mutantes equivalentes do conjunto de teste. Entretanto, esse ganho ocorreu à custa de 101 falsos positivos, indicando um comportamento mais permissivo na identificação de equivalência.

A distribuição dos falsos positivos (FP) e falsos negativos (FN) para cada classificador é resumida na Tabela 6. Observa-se que os modelos apresentam perfis distintos de classificação, refletindo diferentes compromissos entre precisão e capacidade de recuperação de mutantes equivalentes. Enquanto o KNN minimiza a

Tabela 6: Distribuição dos tipos de erro por classificador.

Classificador	FP	FN	Perfil
Logistic Regression	173	85	Maior FP
Linear SVM	139	90	Intermediário
Random Forest	101	70	Maior recall
KNN	19	98	Mais conservador

ocorrência de falsos positivos, tornando suas decisões mais conservadoras, o Random Forest prioriza a recuperação de mutantes equivalentes, aceitando uma incidência maior desse tipo de erro. Esses resultados sugerem que os modelos podem desempenhar papéis complementares em um processo de detecção de mutantes equivalentes, dependendo do objetivo da aplicação, como minimizar descartes incorretos ou maximizar a recuperação de mutantes equivalentes para análise posterior.

Os resultados obtidos indicam que as diferenças entre os classificadores decorrem principalmente do equilíbrio entre falsos positivos e falsos negativos. Dessa forma, a escolha do modelo depende do perfil de erro considerado mais aceitável em cada cenário de Mutation Testing. Em contextos nos quais o descarte incorreto de mutantes não equivalentes representa maior risco, abordagens mais conservadoras, como o KNN, mostram-se mais adequadas. Por outro lado, quando o objetivo é maximizar a recuperação de mutantes equivalentes para reduzir o esforço de inspeção manual, modelos como o Random Forest podem representar uma alternativa mais apropriada.

Embora o MutantBench disponibilize informações sobre os operadores de mutação utilizados na geração dos mutantes, esta replicação concentrou-se na avaliação agregada do desempenho dos classificadores, em conformidade com o objetivo de comparar baselines supervisionados leves. Assim, não foi realizada uma estratificação dos erros por operador de mutação ou por padrões específicos de transformação. Embora a presente replicação não tenha realizado uma análise estratificada por operador de mutação, os resultados evidenciam que os classificadores apresentam comportamentos complementares, sugerindo que diferentes perfis de erro podem ser explorados em estratégias híbridas de classificação. Uma investigação nesse nível de detalhe poderá identificar quais categorias de mutantes apresentam maior dificuldade para cada classificador e orientar o emprego de modelos estruturais ou LLMs apenas nos casos em que sua utilização seja efetivamente necessária.

4.3 Análise Estatística

Tabela 7: Intervalos de confiança de 95% obtidos por bootstrap agrupado pelo método original, com 10.000 reamostragens.

Modelo	F1 equivalente	F1 macro
Reg. Logística	0,5597 [0,2944; 0,7757]	0,7323 [0,5886; 0,8529]
SVM Linear	0,5814 [0,3007; 0,7987]	0,7491 [0,5990; 0,8712]
Random Forest	0,6767 [0,4519; 0,8349]	0,8075 [0,6946; 0,8887]
KNN	0,7208 [0,4178; 0,8723]	0,8401 [0,6874; 0,9171]

Os intervalos agrupados são relativamente amplos e apresentam sobreposição entre os classificadores, especialmente para o F1 da classe equivalente. Isso evidencia a sensibilidade dos resultados à composição dos métodos presentes no conjunto de teste e recomenda cautela na interpretação da ordenação pontual dos modelos.

A divisão oficial entre treinamento e teste foi preservada nos resultados principais para manter comparabilidade direta com Tian et al. [12]. Por esse motivo, a validação cruzada não foi utilizada para selecionar hiperparâmetros nem para gerar os valores da avaliação principal.

Como análise complementar de generalização, foi realizada uma validação cruzada estratificada em cinco partes, agrupada pelo método original. Essa avaliação impede o compartilhamento do mesmo método entre treinamento e teste e permite analisar a estabilidade dos classificadores em métodos não observados.

4.4 Distribuição dos Operadores de Mutação

Tabela 8: Operadores mais frequentes no conjunto de teste, segundo os artefatos de Tian et al. [12].

Operador	Associações	Percentual
AOIS	368	18,52%
ROR	344	17,31%
AORB	302	15,20%
VDL	177	8,91%
ROD	169	8,51%
AOIU	104	5,23%
LOI	103	5,18%
CDL	100	5,03%
CR	63	3,17%
SEOD	50	2,52%
Outros	207	10,42%

4.5 Auditoria de Sobreposição e Sensibilidade

Foi realizada uma auditoria para investigar se o desempenho dos classificadores poderia ser influenciado por sobreposição lexical entre treinamento e teste. Não foram encontrados identificadores de pares repetidos entre os conjuntos. Entretanto, 145 das 1.650 instâncias de teste, correspondentes a 8,79%, possuem o par completo formado pelo método original e pelo método mutante idêntico ao de pelo menos uma instância de treinamento. A comparação foi realizada sobre a representação textual utilizada na auditoria, após a mesma normalização aplicada aos fragmentos analisados.

Também foi observado que 1.649 das 1.650 instâncias de teste compartilham o mesmo método original com alguma instância de treinamento. Além disso, os 19 programas representados no conjunto de teste também aparecem no treinamento. Na representação TF-IDF utilizada pelo KNN, 1.615 instâncias de teste, ou 97,88%, apresentaram similaridade de cosseno igual ou superior a 0,99 com o vizinho de treinamento mais próximo.

Os identificadores dos pares, os identificadores dos fragmentos e os rótulos não foram utilizados como atributos dos modelos. Também não foram encontradas expressões explícitas relacionadas aos rótulos de equivalência dentro dos fragmentos. Assim, não foi identificado vazamento direto do rótulo. A elevada sobreposição entre métodos, contudo, permite que os classificadores explorem regularidades lexicais específicas dos métodos e programas.

Como análise de sensibilidade, os 145 pares com conteúdo repetido foram removidos do teste. Nessa avaliação, o KNN obteve acurácia de 0,9256, F1 de 0,7241 para a classe equivalente e F1

macro de 0,8406. A proximidade com os resultados do conjunto completo indica que as duplicações exatas não explicam isoladamente o desempenho observado. Entretanto, a generalização para métodos não observados permanece limitada pela construção da divisão oficial.

Para complementar a divisão oficial, foi realizada uma validação cruzada estratificada em cinco partes, agrupada pelo identificador do método original. Esse protocolo impede que o mesmo método apareça simultaneamente no treinamento e no teste. Os valores correspondem à média e ao desvio padrão entre as cinco partes.

Tabela 9: Resultados da validação cruzada agrupada por método.

Modelo	F1 equivalente	F1 macro
Reg. Logística	$0,5407 \pm 0,1747$	$0,7057 \pm 0,1434$
SVM Linear	$0,3706 \pm 0,2426$	$0,6112 \pm 0,1678$
Random Forest	$0,1282 \pm 0,1261$	$0,5150 \pm 0,0747$
KNN	$0,1706 \pm 0,2448$	$0,5476 \pm 0,1355$

Os resultados apresentam maior variação e desempenho inferior ao observado na divisão oficial. Em particular, o F1 macro médio do KNN diminuiu de 0,8401 para 0,5476. Esse resultado indica que parte do desempenho obtido na divisão oficial está associada ao compartilhamento de métodos entre treinamento e teste. Portanto, os valores principais devem ser interpretados como desempenho sobre a divisão oficial do MutantBench, e não como evidência de generalização para métodos ou projetos inéditos.

5 Discussão

Além do desempenho preditivo observado nos experimentos, os resultados permitem discutir a relação entre eficácia, custo computacional e reprodutibilidade. Resultados reportados na literatura para abordagens baseadas em *Large Language Models* (LLMs) indicam desempenho superior em diversos cenários, embora essas abordagens não tenham sido executadas neste estudo. Em contrapartida, os classificadores supervisionados leves avaliados localmente alcançaram desempenho competitivo utilizando infraestrutura computacional convencional. Esse resultado sugere que abordagens tradicionais continuam apresentando valor prático em cenários com restrições de recursos computacionais ou requisitos elevados de reprodutibilidade experimental.

Ao contrário de modelos proprietários sujeitos a atualizações frequentes, mudanças de versão e limitações de acesso, os classificadores reproduzidos localmente podem ser executados, auditados e comparados de forma independente pela comunidade científica, favorecendo a reprodutibilidade dos experimentos.

Os resultados obtidos reforçam que classificadores supervisionados leves continuam desempenhando um papel relevante na detecção de mutantes equivalentes, não apenas como baselines experimentais, mas também como alternativas viáveis em cenários com restrições de infraestrutura computacional. Além de apresentarem boa capacidade preditiva, esses modelos oferecem baixo custo de treinamento, facilidade de reprodução e menor dependência de recursos externos quando comparados a abordagens baseadas em Large Language Models (LLMs).

Sob essa perspectiva, os perfis de erro observados motivam, como hipótese para trabalhos futuros, o uso de classificadores leves como filtros iniciais. Os experimentos deste artigo não avaliam cobertura por limiar, taxa de encaminhamento ou desempenho de um pipeline híbrido e, portanto, não fornecem evidência para afirmar sua eficácia.

A comparação indireta com resultados reportados na literatura sugere que parte do desempenho alcançado por modelos mais complexos pode ser aproximada por classificadores supervisionados leves em determinados cenários. Essa observação não implica equivalência entre as abordagens, mas evidencia que baselines simples, de baixo custo e facilmente reproduzíveis continuam representando referências importantes para a avaliação de novas técnicas.

Sob a perspectiva de custo computacional, os resultados experimentais deste trabalho mostram que modelos leves podem ser executados integralmente em hardware convencional, enquanto aspectos relacionados ao custo operacional de modelos de grande porte são discutidos apenas com base na literatura. Essa diferença reforça a importância de considerar não apenas métricas de desempenho, mas também fatores como reprodutibilidade, facilidade de implantação e requisitos computacionais.

Além dos aspectos relacionados ao desempenho preditivo, os resultados obtidos possuem implicações práticas relevantes para pesquisadores e equipes de teste de software. Em cenários reais, a análise manual de mutantes equivalentes continua representando um custo significativo para a adoção de *Mutation Testing*.

Os resultados deste estudo sugerem que classificadores supervisionados leves podem atuar como filtros iniciais para reduzir o espaço de busca de mutantes potencialmente equivalentes. A partir dessa perspectiva, modelos mais sofisticados, incluindo LLMs, poderiam ser empregados apenas nos casos ambíguos. Essa possibilidade, entretanto, constitui uma proposta conceitual e não foi avaliada experimentalmente nesta pesquisa.

Outra contribuição importante refere-se à reprodutibilidade experimental. Diferentemente de modelos proprietários sujeitos a alterações frequentes de versão e disponibilidade, os classificadores avaliados neste estudo podem ser executados integralmente em ambiente local, favorecendo a auditoria dos resultados e a replicação independente por outros pesquisadores.

5.1 Comparação com o Estudo Original

Os resultados dos modelos baseados em LLMs foram obtidos a partir do artigo e dos artefatos disponibilizados por Tian et al. [12]. Esses modelos não foram reexecutados no ambiente experimental deste trabalho. Portanto, a comparação apresentada nesta seção é indireta e não representa uma reprodução completa dos experimentos do estudo original.

Para responder à RQ2, a Tabela 10 apresenta o F1 macro do melhor modelo executado neste estudo e valores selecionados reportados por Tian et al. [12]. A utilização da mesma métrica evita comparar o F1 da classe equivalente, reportado nos demais resultados deste artigo, com o F1 macro utilizado no estudo original.

O KNN com TF-IDF apresentou F1 macro numericamente superior aos valores selecionados do estudo original. Esse resultado mostra que representações lexicais leves podem constituir baselines relevantes no MutantBench, especialmente em cenários com restrições de infraestrutura.

Entretanto, o resultado não demonstra que o KNN seja superior aos modelos baseados em LLMs. As abordagens utilizam diferentes representações, hiperparâmetros, processos de treinamento e ambientes de execução. Além disso, os modelos reportados por

Tabela 10: Comparação indireta com valores reportados por Tian et al. [12]. Apenas o KNN com TF-IDF foi executado neste estudo.

Fonte	Técnica	F1 macro
Este estudo	KNN com TF-IDF	0,8401
Tian et al.	KNN	0,7215
Tian et al.	ASTNN	0,7000
Tian et al.	CodeBERT	0,6920
Tian et al.	UniXCoder	0,8188
Tian et al.	CodeT5+	0,8188

Tian et al. não foram reexecutados neste trabalho. Assim, a tabela permite posicionar os resultados no mesmo conjunto de dados e na mesma escala métrica, mas não sustenta superioridade causal nem equivalência entre as abordagens.

5.2 Recomendações para Reprodutibilidade

Embora este trabalho tenha avaliado exclusivamente classificadores supervisionados leves, futuras pesquisas que comparem diferentes categorias de abordagens para detecção de mutantes equivalentes devem adotar um protocolo experimental padronizado para garantir transparência e reprodutibilidade.

Cada instância avaliada deve registrar o método original, o mutante correspondente, o operador de mutação utilizado, o rótulo de referência e metadados relevantes do projeto. Além disso, recomenda-se normalizar as entradas preservando identificadores, tipos e contexto mínimo necessário para a decisão. Quando informações forem omitidas durante o pré-processamento, essa decisão deve ser explicitamente documentada.

Em estudos que incluam técnicas determinísticas, classificadores supervisionados e Large Language Models (LLMs), recomenda-se executar inicialmente abordagens de menor custo computacional, como o Trivial Compiler Equivalence (TCE) e classificadores supervisionados leves, permitindo identificar casos triviais e reduzir o número de instâncias encaminhadas para modelos mais complexos.

Caso LLMs sejam utilizados, recomenda-se registrar versão do modelo, estratégia de prompting, parâmetros de inferência, temperatura, limite de tokens, tempo de resposta, custo estimado por consulta e, para modelos locais, informações de hardware e quantização. Esses registros são fundamentais para permitir comparações reproduzíveis entre diferentes estudos.

Tabela 11: Artefatos recomendados para garantir a reprodutibilidade de futuras avaliações experimentais.

Artefato	Conteúdo registrado
Dataset derivado	Identificadores dos mutantes, rótulos, operadores e divisão entre treino, validação e teste.
Prompts	Template completo, exemplos usados em *few-shot* e instruções de saída.
Configuração	Modelo, versão, temperatura, limite de tokens, quantização e hardware.
Resultados brutos	Predição, justificativa, tempo, custo e eventuais respostas inválidas.
Scripts	Rotinas de execução, cálculo de métricas e geração de tabelas.

Além das métricas quantitativas, recomenda-se realizar análise qualitativa dos erros, priorizando falsos positivos, pois esses casos podem descartar incorretamente mutantes não equivalentes e mascarar limitações da suíte de testes. Quando estudos envolverem LLMs, recomenda-se ainda distinguir separadamente avaliações zero-shot e few-shot, evitando vazamento de exemplos entre os conjuntos de treinamento e teste.

5.3 Pipeline Híbrido Proposto

Com base nos resultados experimentais apresentados na Seção 5, propõe-se uma arquitetura conceitual para integração entre técnicas determinísticas, classificadores supervisionados leves e Large Language Models (LLMs). Essa arquitetura não foi implementada nem validada experimentalmente neste trabalho, sendo apresentada como uma direção para pesquisas futuras.

O pipeline é composto por três estágios. No primeiro, técnicas determinísticas de baixo custo, como o Trivial Compiler Equivalence (TCE), removem mutantes cuja equivalência pode ser identificada de forma conservadora. No segundo estágio, classificadores supervisionados ou modelos estruturais priorizam os casos com maior probabilidade de equivalência. Por fim, apenas os casos considerados ambíguos são encaminhados para análise por LLMs.

Essa organização busca reduzir o custo computacional e financeiro, reservando modelos de maior custo apenas para situações em que as abordagens anteriores não apresentam confiança suficiente. Além disso, permite manter decisões simples associadas a técnicas reproduzíveis, enquanto casos semanticamente mais complexos são analisados por modelos mais expressivos.

A arquitetura proposta é ilustrada na Figura 1, que sintetiza os três estágios do processo de decisão.

Como ilustrado na Figura 1, decisões conservadoras obtidas pelo TCE podem ser aceitas diretamente, enquanto casos classificados com baixa confiança seguem para avaliação por LLMs. Situações em que ainda não há consenso permanecem disponíveis para auditoria manual. Em ambientes de integração contínua, essa arquitetura também permite operar em um modo rápido, utilizando apenas filtros leves, ou em um modo completo, empregando LLMs somente quando necessário.

6 Ameaças à Validade

Validade interna. Os resultados deste estudo podem ser influenciados pelas configurações adotadas para os classificadores supervisionados, incluindo a representação TF-IDF, os hiperparâmetros dos modelos e a divisão entre treinamento e teste. Embora esses fatores tenham sido mantidos consistentes durante os experimentos, diferentes configurações podem produzir resultados distintos. Caso futuros experimentos incluam *Large Language Models* (LLMs), parâmetros como temperatura, estratégias de *prompting*, limite de tokens e versão do modelo também poderão influenciar os resultados obtidos.

Validade externa. A avaliação foi realizada exclusivamente sobre o benchmark *MutantBench*, composto por programas Java. Embora esse benchmark permita comparações consistentes com estudos anteriores, os resultados podem não se generalizar para outras linguagens de programação, operadores de mutação, domínios de aplicação ou cenários industriais.

Validade de construção. A eficácia foi avaliada por meio das métricas de acurácia, *precision*, *recall*, *F1-score* e matrizes de confusão. Embora essas métricas sejam amplamente utilizadas na literatura, elas não capturam outros aspectos relevantes,

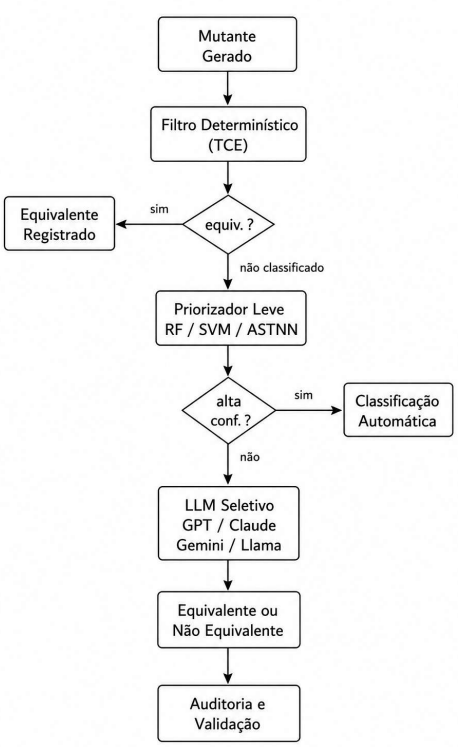

Figura 1: Arquitetura conceitual do pipeline híbrido proposto.

como interpretabilidade, facilidade de integração em ambientes de desenvolvimento ou custo operacional de abordagens mais complexas.

Validade de conclusão. Os resultados dependem da distribuição das amostras disponíveis no *MutantBench* e da representação adotada para os dados. Além disso, a comparação com modelos estruturais e LLMs foi realizada de forma indireta, utilizando resultados reportados na literatura, não constituindo uma comparação experimental direta. Dessa forma, as conclusões referentes a essas abordagens devem ser interpretadas com essa limitação em mente.

7 Limitações do Estudo

Apesar dos cuidados metodológicos adotados, este trabalho apresenta limitações que delimitam o escopo de interpretação dos resultados. A avaliação está restrita ao MutantBench e à linguagem Java; portanto, os resultados podem não generalizar para outras linguagens, operadores de mutação ou contextos industriais.

Outra limitação refere-se à unidade de análise. O estudo considera pares original-mutante em nível de método, sem contemplar necessariamente o contexto completo dos projetos, dependências externas ou interações entre componentes. Em cenários reais, essas informações podem influenciar a interpretação semântica dos modelos.

Além disso, o estudo não contempla estratégias extensivas de *fine-tuning* para todos os modelos avaliados. Essa escolha prioriza um protocolo reproduzível e compatível com ambientes acadêmicos de recursos limitados. A seleção dos modelos também é influenciada por restrições práticas de infraestrutura, memória, disponibilidade computacional e custo de APIs comerciais.

Outra limitação importante refere-se à infraestrutura computacional disponível durante os experimentos. A execução de modelos de maior porte e estratégias extensivas de *fine-tuning* não foi realizada devido ao elevado custo computacional e às limitações de hardware disponíveis para a pesquisa. Consequentemente, os experimentos priorizaram abordagens reproduzíveis em equipamentos convencionais.

Essas limitações não invalidam a proposta, mas indicam que as conclusões devem ser interpretadas como evidências iniciais e como base para avaliações futuras com benchmarks, linguagens e configurações mais amplas.

Além disso, não foram realizados testes estatísticos pareados, como o teste de McNemar, para comparar formalmente as previsões dos classificadores no conjunto de teste. Portanto, as diferenças observadas entre as métricas pontuais devem ser interpretadas como descritivas e não como evidência de significância estatística entre os modelos.

8 Conclusão

Este trabalho apresentou uma replicação parcial do protocolo experimental proposto por Tian *et al.*, utilizando o benchmark *MutantBench* para avaliar classificadores supervisionados leves executados localmente. Foram investigados os modelos *Logistic Regression*, *Linear SVM*, *Random Forest* e *KNN*, empregando representação TF-IDF baseada em n-gramas de caracteres. Os resultados obtidos foram comparados, de forma indireta, com valores reportados na literatura para abordagens mais complexas, incluindo modelos baseados em *Large Language Models* (LLMs).

Entre os modelos avaliados, o KNN apresentou o melhor desempenho geral, alcançando 92,91% de acurácia e F1-score de 72,08% para a classe equivalente. A análise das matrizes de confusão mostrou que o KNN apresentou o menor número de falsos positivos entre os classificadores avaliados, enquanto o Random Forest recupera uma quantidade maior de mutantes equivalentes, evidenciando diferentes perfis de utilização conforme o objetivo da aplicação.

Os resultados obtidos estabelecem baselines reproduzíveis para a detecção de mutantes equivalentes em cenários com restrições de infraestrutura. O possível uso desses classificadores como filtros iniciais permanece uma hipótese a ser avaliada por meio de limiares de confiança, cobertura e carga encaminhada a modelos posteriores.

Como limitações, este estudo restringiu-se à execução experimental de classificadores supervisionados leves. Modelos estruturais, modelos pré-treinados para código e LLMs não foram executados, sendo considerados apenas por meio de resultados reportados na literatura. Além disso, os experimentos foram realizados exclusivamente sobre o benchmark *MutantBench*, o que limita a generalização dos resultados para outros conjuntos de dados e linguagens de programação.

Os resultados apresentados reforçam que classificadores supervisionados leves permanecem relevantes para a pesquisa em Mutation Testing, constituindo uma base reproduzível e de baixo custo para comparação com futuras abordagens baseadas em modelos estruturais e Large Language Models.

Disponibilidade de Artefatos

Os artefatos experimentais, incluindo scripts, configurações, modelos treinados e resultados, estão disponíveis no Zenodo em https://doi.org/10.5281/zenodo.21091024.

Agradecimentos

Maicon Bernardino é financiado pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), Bolsa nº 307969/2026-6.

Referências

[1] Miltiadis Allamanis, Marc Brockschmidt, and Mahmoud Khademi. 2018. Learning to Represent Programs with Graphs. In *International Conference on Learning Representations*. OpenReview.net, Vancouver, Canada. https://openreview.net/forum?id=BJOFETxR-

[2] Adelaide Danilov, Aria Nourbakhsh, and Christoph Schommer. 2025. Cluster Purge Loss: Structuring Transformer Embeddings for Equivalent Mutants Detection. arXiv:2507.20078 [cs.LG] doi:10.48550/arXiv.2507.20078

[3] Richard A. DeMillo, Richard J. Lipton, and Frederick 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

[4] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*. Association for Computational Linguistics, Online, 1536–1547. doi:10.18653/v1/2020.findings-emnlp.139

[5] Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. 2022. UniXcoder: Unified Cross-Modal Pre-Training for Code Representation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Dublin, Ireland, 7212–7225. doi:10.18653/v1/2022.acl-long.499

[6] 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

[7] René Just, Darioush Jalali, Laura Inozemtseva, Michael D. Ernst, Reid Holmes, and Gordon Fraser. 2014. Are Mutants a Valid Substitute for Real Faults in Software Testing?. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2014)*. ACM, 654–665. doi:10.1145/2635868.2635929

[8] Lech Madeyski, Wojciech Orzeszyna, Richard Torkar, and Mariusz Jozala. 2014. Overcoming the Equivalent Mutant Problem: A Systematic Literature Review and a Comparative Experiment of Second Order Mutation. *IEEE Transactions on Software Engineering* 40, 1 (2014), 23–42. doi:10.1109/TSE.2013.44

[9] A. Jefferson Offutt and Jie Pan. 1997. Automatically Detecting Equivalent Mutants and Infeasible Paths. *Software Testing, Verification and Reliability* 7, 3 (1997), 165–192. doi:10.1002/(SICI)1099-1689(199709)7:3<165::AID-STVR143>3.0.CO;2-U

[10] Mike Papadakis, Yue Jia, Mark Harman, and Yves Le Traon. 2015. Trivial Compiler Equivalence: A Large Scale Empirical Study of a Simple, Fast and Effective Equivalent Mutant Detection Technique. In *Proceedings of the 37th IEEE/ACM International Conference on Software Engineering*. IEEE, 936–946. doi:10.1109/ICSE.2015.103

[11] Mike Papadakis, Marinos Kintis, Jie Zhang, Yue Jia, Yves Le Traon, and Mark Harman. 2019. Mutation Testing Advances: An Analysis and Survey. In *Advances in Computers*. Vol. 112. Elsevier, 275–378. doi:10.1016/bs.adcom.2018.03.015

[12] Zhao Tian, Honglin Shu, Dong Wang, Xuejie Cao, Yasutaka Kamei, and Junjie Chen. 2024. Large Language Models for Equivalent Mutant Detection: How Far Are We? arXiv:2408.01760 [cs.SE] https://arxiv.org/abs/2408.01760

[13] Lars van Hijfte and Ana Oprescu. 2021. MutantBench: An Equivalent Mutant Problem Comparison Framework. In *2021 IEEE 14th International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 7–12. doi:10.1109/ICSTW52544.2021.00015

[14] Bo Wang, Mingda Chen, Youfang Lin, Mike Papadakis, and Jie M. Zhang. 2024. An Exploratory Study on Using Large Language Models for Mutation Testing. arXiv:2406.09843 [cs.SE] https://arxiv.org/abs/2406.09843

[15] Yue Wang, Hung Le, Akhilesh Deepak Gotmare, Nghi D. Q. Bui, Junnan Li, and Steven C. H. Hoi. 2023. CodeT5+: Open Code Large Language Models for Code Understanding and Generation. arXiv:2305.07922 [cs.CL] https://arxiv.org/abs/2305.07922

[16] Yue Wang, Weishi Wang, Shafiq Joty, and Steven C. H. Hoi. 2021. CodeT5: Identifier-Aware Unified Pre-Trained Encoder-Decoder Models for Code Understanding and Generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Punta Cana, Dominican Republic, 8696–8708. doi:10.18653/v1/2021.emnlp-main.685

[17] Jian Zhang, Xu Wang, Hongyu Zhang, Hailong Sun, Kaixuan Wang, and Xudong Liu. 2019. A Novel Neural Source Code Representation Based on Abstract Syntax Tree. In *Proceedings of the 41st International Conference on Software Engineering (ICSE 2019)*. IEEE Press, Montreal, QC, Canada, 783–794. doi:10.1109/ICSE.2019.00086