Detecção de vulnerabilidades em *bytecodes* de contratos inteligentes no Ethereum via *embeddings* do CodeBERT

Pedro Henrique F. S. Oliveira[1], Heder S. Bernardino[1], Saulo Moraes Villela[1], Edelberto Franco Silva[1], Jairo Francisco de Souza[1], Alex B. Vieira[1]

[1]Departamento de Ciência da Computação (DCC)
Universidade Federal de Juiz de Fora (UFJF) – Juiz de Fora, MG – Brasil

pedrohenrique.filgueiras@estudante.ufjf.br

{heder.bernardino,saulo.moraes,edelberto.franco}@ufjf.br

{jairo.souza,alex.borges}@ufjf.br

Abstract. *Ethereum is a cryptocurrency platform that allows the execution of smart contracts, autonomous programs that operate on a decentralized network. Vulnerabilities in these contracts represent significant financial and security risks in blockchain ecosystems, motivating the automation of the process of detecting them. This work studies the detection of vulnerabilities in Ethereum smart contracts using bytecode-derived embeddings. Embeddings are vector representations generated by language models that capture the structural characteristics of text. These representations were used as input for logistic regression, decision tree, and random forest algorithms in order to detect which contracts have vulnerabilities. The results show that the embeddings contain useful information to distinguish vulnerable from non-vulnerable contracts. The study also finds that altering the original data distribution during training significantly affects performance, highlighting the sensitivity of embedding-based approaches to sampling strategies.*

Resumo. *O Ethereum é uma plataforma de criptomoedas que permite a execução de contratos inteligentes, programas autônomos que operam em uma rede descentralizada. As vulnerabilidades nesses contratos representam grandes riscos financeiros e de segurança nos ecossistemas blockchain, motivando a automatização do processo de detectá-las. Este trabalho estuda a detecção de vulnerabilidades em contratos inteligentes Ethereum usando embeddings derivados de bytecode. Embeddings são representações vetoriais geradas por modelos de linguagem, que capturam as características estruturais de texto. Essas representações foram usadas como entrada para os algoritmos de regressão logística, árvore de decisão e floresta aleatória, com o fim de detectar quais contratos possuem vulnerabilidades. Os resultados mostram que os embeddings contêm informações úteis para distinguir contratos vulneráveis de não vulneráveis. O estudo também constata que a alteração da distribuição original dos dados durante o treinamento afeta significativamente o desempenho, destacando a sensibilidade das abordagens baseadas em embeddings às estratégias de amostragem.*

1. Introdução

Ethereum é uma plataforma descentralizada de ativos digitais baseada na tecnologia blockchain [Wood 2014]. Além de criptomoedas, essa plataforma suporta a implementação de aplicações programáveis e autoexecutáveis, conhecidas como contratos inteligentes. Diferentemente do software tradicional, os contratos inteligentes operam diretamente na blockchain, sem a necessidade de intermediários [Szabo 1996]. Isso abre a possibilidade para que desenvolvedores utilizem essa tecnologia além de ativos digitais, implementando aplicações descentralizadas (dApps) e realizando operações automatizadas em um ambiente baseado em consenso.

Embora os contratos inteligentes sejam ferramentas poderosas, eles também apresentam desafios de segurança significativos [Chu et al. 2023]. Uma vez implementado na rede Ethereum, o conteúdo do código não pode ser alterado. Portanto, quaisquer falhas lógicas ou de segurança ficarão permanentemente acessíveis. Diversos ataques exploram vulnerabilidades documentadas, causando perdas financeiras significativas, o que destaca a importância de uma análise de segurança rigorosa. Consequentemente, a detecção de vulnerabilidades antes da implementação é crucial em um ambiente descentralizado e imutável como as blockchains [Durieux et al. 2020, Feist et al. 2019].

As primeiras abordagens de análise de segurança para contratos Ethereum basearam-se principalmente em análise estática, execução simbólica e detecção baseada em regras. Ferramentas como Oyente, Mythril e Securify modelam o fluxo de execução do contrato para identificar vulnerabilidades conhecidas, como reentrância, dependência de timestamp e desordem de exceção [Tsankov et al. 2018, Di Angelo and Salzer 2019, Sharma et al. 2022]. Embora eficazes para contextos específicos e bem documentados, essas abordagens dependem de regras especializadas e apresentam limitações na generalização para padrões não previamente modelados.

Recentemente, técnicas de aprendizado de máquina têm demonstrado potencial para a detecção automática de vulnerabilidades em contratos inteligentes. Algoritmos de classificação podem analisar instâncias de contratos para detectar padrões de vulnerabilidade, reduzindo a necessidade de auditorias manuais com regras predefinidas. O uso de arquiteturas Transformer para modelagem de código-fonte também tem demonstrado resultados expressivos em tarefas como classificação de vulnerabilidades, detecção de bugs e análise semântica de programas [Feng 2020].

No entanto, uma limitação nesse tipo de aplicação é a escassez de dados pré-rotulados. Somado a isso, contratos Ethereum não são sempre verificados, ocasionando em códigos-fonte que podem não estar disponíveis publicamente. Modelos que utilizam informações em nível de código podem apresentar problemas com esse tipo de conjunto de dados de contratos.

Para superar esses desafios, este trabalho propõe o uso de um modelo baseados em BERT para construir representações de características diretamente a partir do *bytecode* de contratos inteligentes, que está disponível publicamente para todos os contratos implantados na rede Ethereum. Ao modelar o *bytecode* como uma representação sequencial e aproveitar os recursos de aprendizado contextual das arquiteturas Transformer, a abordagem proposta extrai *embeddings* semânticos que capturam padrões comportamentais relevantes sem depender de código-fonte verificado. Esses *embeddings* são então usados

como entrada para modelos de aprendizado de máquina subsequentes para detecção de vulnerabilidades, permitindo análises em escala e reduzindo a dependência de conjuntos de dados rotulados e informações em nível de código-fonte.

A execução de contratos inteligentes no Ethereum ocorre por meio da Ethereum Virtual Machine (EVM), uma máquina de pilha com conjunto de instruções próprio, sem tipagem explícita e com fluxo de controle dinâmico [Wood 2014]. Sendo uma codificação de baixo-nível, o *bytecode* da EVM perde abstrações semânticas presentes no código-fonte, como nomes de variáveis e tipos estruturados. Além disso, padrões como chamadas indiretas e saltos condicionais dificultam a reconstrução do fluxo de controle. Essas características tornam a análise em nível de *bytecode* desafiadora para a detecção automática de vulnerabilidades.

Além de possibilitar a detecção de vulnerabilidades a partir do *bytecode*, os *embeddings* contextuais produzidos pela abordagem proposta, baseada em BERT, fornecem uma base flexível para múltiplas estratégias de aprendizado. Essas representações aprendidas podem ser usadas para engenharia de recursos, análise de similaridade e aumento de dados, contribuindo para conjuntos de dados de treinamento mais ricos e informativos. Neste trabalho, o aumento de dados é explorado como uma dessas aplicações, visando mitigar as limitações impostas pela escassez de dados rotulados, gerando instâncias de treinamento sintéticas a partir de representações de contratos existentes.

Essa estratégia baseada em representação de código via BERT ilustra como as representações baseadas em *embeddings* podem aprimorar a robustez e a generalização de modelos de detecção de vulnerabilidades. Até onde sabemos, este trabalho está entre os primeiros a investigar o uso de técnicas de aumento de dados orientadas por *embeddings* no contexto da segurança de contratos inteligentes no Ethereum.

A abordagem de detecção de contratos inteligentes com vulnerabilidades foi avaliada por meio de treinamento de modelos de classificação de aprendizado de máquina, com um arcabouço de variadas métricas de avaliação. A abordagem se mostrou capaz de gerar resultados robustos, com um AUC-ROC de 0,8844 para o modelo de Floresta Aleatória, apontando uma linha de estudo promissora para a detecção de padrões de *bytecode* com representações de *embeddings*.

O restante deste documento está organizado da seguinte forma: na Seção 2 analisamos trabalhos relacionados na área de detecção de contratos inteligentes vulneráveis. Na Seção 3, detalhamos os conjuntos de dados utilizados e as etapas de pré-processamentos necessárias. Na Seção 4, discutimos e entramos em detalhes na metodologia utilizada, seguida pelas Seções 5 e 6, que discutem resultados, comparações e conclusões com planos futuros, respectivamente.

2. Trabalhos Relacionados

Nesta seção, são analisados estudos que formam uma base referencial relevante para este trabalho. Destacamos diferenças em metodologias, situando este trabalho no contexto do campo de segurança de contratos inteligentes.

Em [Dong et al. 2025], os autores apresentam uma avaliação empírica de técnicas de aumento de dados aplicadas à análise de código-fonte. Embora este estudo não se concentre em criptomoedas ou contratos inteligentes, ele fornece uma comparação abran-

gente de 25 métodos de aumento de dados em quatro arquiteturas de redes neurais profundas, oferecendo insights valiosos sobre a eficácia das estratégias de aumento na análise de código baseada em aprendizado. Os resultados indicam que os métodos de aumento baseados em *embeddings*, como Mixup e SenMixup, consistentemente alcançaram as maiores melhorias de precisão. Em cenários com poucos dados, a aplicação do Mixup levou a ganhos de precisão superiores a 12%, demonstrando seu potencial para mitigar a escassez de dados rotulados.

O estudo analisa ainda a robustez do modelo em cenários adversários, mostrando que técnicas tradicionais de aumento de dados orientadas a texto, como a Deleção Aleatória (RD), alcançaram melhor desempenho em termos de robustez, embora com ganhos gerais limitados. Notavelmente, as amostras geradas não precisavam representar código válido ou compilável. Apesar dessa limitação, os dados aumentados ainda foram eficazes na melhoria do desempenho do modelo, sugerindo que a consistência semântica no nível de representação pode ser suficiente para certas tarefas de aprendizado.

Em contraste, o trabalho apresentado em [Hwang et al. 2024] adota uma perspectiva fundamentalmente diferente, focando no aumento de dados especificamente no domínio de contratos inteligentes. Os autores propõem as Redes de Geração Guiadas por Compilador (CGGNet), uma arquitetura projetada para gerar contratos inteligentes sintática e semanticamente válidos. O principal objetivo dessa abordagem é garantir que as amostras aumentadas sejam compiláveis, preservando assim a correção da execução. O desempenho da CGGNet é avaliado em comparação com arquiteturas recorrentes comumente usadas para modelagem de sequências, como RNNs e LSTMs, com o modelo proposto alcançando uma taxa de sucesso de geração de 59,71%.

Embora esses estudos demonstrem a eficácia do aumento de dados na análise de código-fonte e na geração de contratos inteligentes, eles operam principalmente no nível do código-fonte e se concentram na compreensão geral do código ou na síntese de contratos. Em contraste, o presente trabalho visa o problema da detecção de vulnerabilidades em contratos inteligentes do Ethereum e opera diretamente em *bytecode*, que está disponíveis para todos os contratos implantados na rede. Ao aproveitar *embeddings* contextuais derivados de modelos baseados em BERT, este trabalho explora o aumento de dados como uma estratégia complementar dentro de uma estrutura mais ampla orientada por *embeddings*, abordando tanto a escassez de dados rotulados quanto a disponibilidade limitada de código-fonte verificado.

3. Conjunto de Dados

O conjunto de dados utilizado neste estudo foi construído a partir de códigos-fonte de contratos inteligentes disponíveis publicamente. Os contratos rotulados como vulneráveis foram inicialmente obtidos do SmartBugs [Di Angelo et al. 2023], conjunto de dados que agrega contratos Ethereum com problemas de segurança documentados. Para expandir a cobertura e obter as representações de todos os contratos inteligentes em *bytecode*, dados da plataforma Etherscan[1] foram integrados.

A Tabela 1 dispõe do conjunto de atributos disponíveis com o conjunto de dados em questão. Para este trabalho, apenas as informações do *bytecode* e os rótulos de vulne-

[1] `https://etherscan.io/`

rabilidade foram utilizados para o treinamento dos modelos. O uso de atributos adicionais em relação ao *bytecode* é uma possível expansão visando trabalhos futuros.

Tabela 1. Formato do conjunto de dados.

Atributo	Descrição
contract_address	Hash identificadora
is_verified	Se o contrato foi verificado no Etherscan
code_length	Tamanho do código gerado
tx_count	Número de transações feitas
balance	Saldo da conta em Wei
bytecode	Código-fonte do contrato inteligente em bytecode
unsafe	Rótulo de presença ou ausência de vulnerabilidade

O conjunto de dados final consiste em 47.451 contratos inteligentes Ethereum, dos quais 34.066 contêm vulnerabilidades conhecidas e 13.385 são rotulados como não vulneráveis, resultando em uma razão de desbalanceamento de aproximadamente 2,55. Este conjunto de dados constitui a base para todos os experimentos de extração de *embeddings* e classificação subsequentes.

4. Metodologia

Esta seção descreve o fluxo metodológico adotado neste trabalho. Ela detalha como o *bytecode* de contratos inteligentes é processado e transformado, como os *embeddings* são obtidos usando um modelo pré-treinado e como essas representações são utilizadas em classificadores de aprendizado de máquina para detecção de vulnerabilidades.

4.1. Visão Geral

Neste trabalho, nós geramos características por meio de vetores de *embeddings* com o uso de CodeBERT, um modelo baseado em *transformers*, pré-treinado para a interpretação de código-fonte. Sequências de instruções derivadas do *bytecode* de contratos inteligentes são tokenizadas e fornecidas como entrada para o modelo pré-treinado. Os *embeddings* contextualizados resultantes são agregados em vetores. Neste trabalho, esses *embeddings* servem como representações semânticas de alto nível do comportamento do contrato e são posteriormente utilizados como características de entrada para classificadores clássicos de aprendizado de máquina no processo de detecção de vulnerabilidades. A Figura 1 ilustra como o CodeBERT é acoplado à arquitetura deste trabalho.

Após a geração dos vetores de *embeddings*, inicia-se o processo de treinamento dos modelos de classificação. A Figura 2 ilustra o processo. Inicialmente, particionamos o conjunto de dados gerado em k subconjuntos (*folds*), conforme a estratégia de validação cruzada. Em cada iteração, um dos *folds* é reservado como um subconjunto próprio para teste, permitindo que o restante dos *folds* seja usado para o treinamento.

Para cada *fold*, antes de realizar o treinamento em si, é aplicada uma abordagem de balanceamento que faz com que ambas as classes tenham quantidades equiparáveis de amostras, evitando o favorecimento de uma classe majoritária no treinamento. O balanceamento é aplicado apenas ao conjunto de treinamento, de forma que os modelos serão avaliados em conjuntos de dados desbalanceados, em um cenário fiel à realidade.

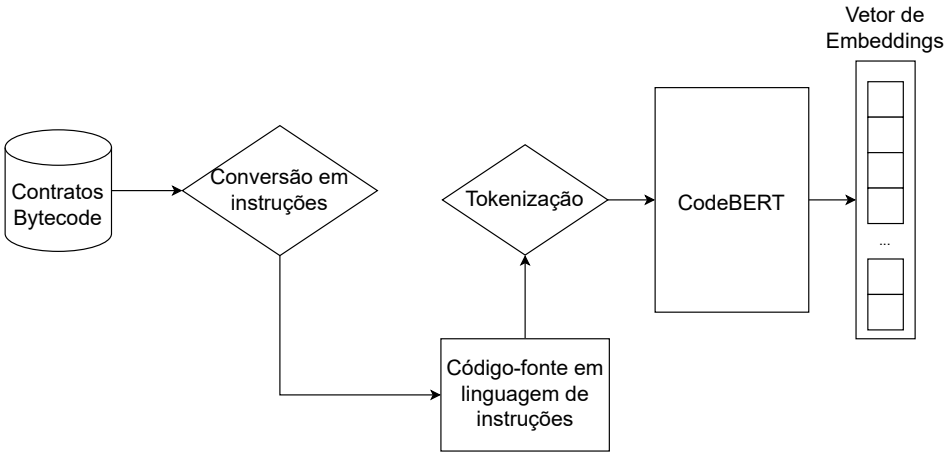

Figura 1. Extração de vetor de *embeddings* de contratos inteligentes usando CodeBERT.

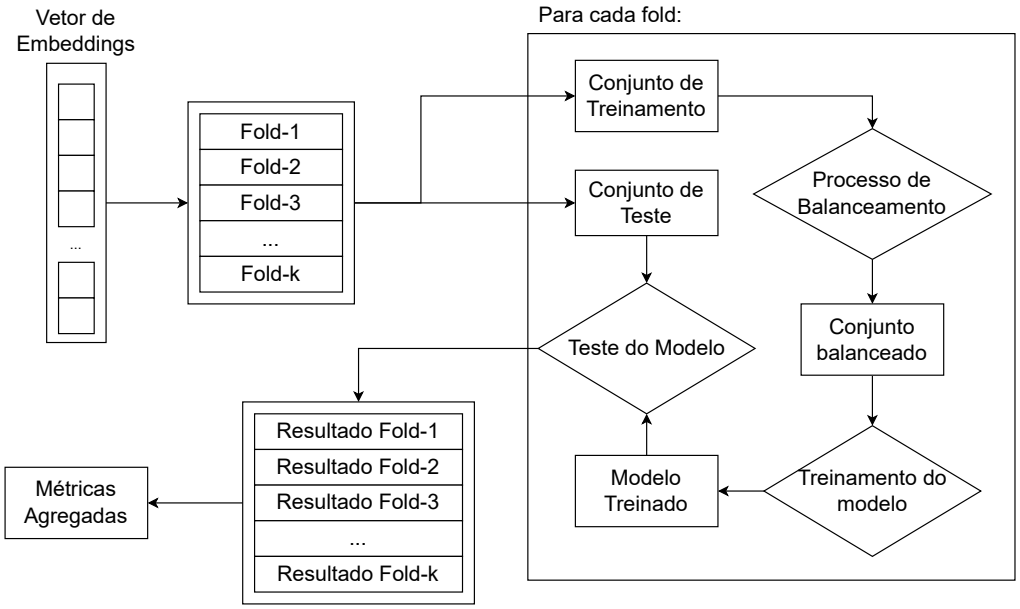

Figura 2. Metodologia para treinamento e avaliação de modelos de classificação.

A seguir, o modelo de classificação é treinado com o conjunto de treinamento balanceado. Após isso, é feita uma previsão utilizando o conjunto de teste, e os resultados são armazenados para avaliação posterior. Esse processo se repete para cada *fold*.

Com todos os resultados de previsão agregados, métricas clássicas de avaliação de problemas de classificação são aplicadas. Dessa forma, o procedimento integra o resultado de todos os subconjuntos da validação cruzada com o balanceamento das classes, mitigando vieses e desbalanceamentos.

Os experimentos foram efetuados utilizando a biblioteca scikit-learn no Python, na versão 3.8.8. Todos os modelos utilizaram uma configuração padrão da biblioteca para hiperparâmetros dos modelos.

4.2. *Bytecode*

No Ethereum, contratos inteligentes são executados na forma de *bytecode*, codificação que consiste em instruções de baixo nível interpretadas pela Máquina Virtual Ethereum (EVM) [Wood 2014]. Um *bytecode* define o comportamento operacional de um contrato e inclui operações baseadas em pilha, instruções de fluxo de controle e interações com o estado do blockchain. Embora compacto e eficiente para execução, o *bytecode* bruto não é diretamente adequado para análises de alto nível, visto que não é uma codificação que contenha elementos de linguagem natural.

Neste trabalho, a biblioteca pyevmasm[2] foi utilizada para converter o *bytecode* da EVM em sua sequência de instruções correspondente no nível de operações. Essas instruções fornecem uma representação mais estruturada e interpretável do comportamento do contrato, servindo como uma forma intermediária mais próxima das operações legíveis por humanos, ao mesmo tempo que preserva a semântica original do programa.

4.3. CodeBERT

O BERT (Bidirectional Encoder Representations from Transformers) é um modelo de representação de linguagem construído sobre a arquitetura Transformer, que utiliza mecanismos de autoatenção para modelar relações entre tokens [Vaswani et al. 2017, Devlin et al. 2019]. Ao contrário dos modelos sequenciais tradicionais, o BERT processa tokens nas duas direções, permitindo que a representação de cada token dependa tanto do contexto à esquerda quanto ao à direita. Ele é pré-treinado usando técnicas de autoaprendizagem supervisionada que permitem ao modelo aprender a estrutura sintática, as relações semânticas e as dependências de longo alcance no texto.

O CodeBERT aumenta o escopo do BERT para o domínio de códigos-fonte, sendo pré-treinado em grandes corpora de código-fonte, permitindo que ele aprenda representações que capturam a sintaxe específica da programação e padrões estruturais [Feng 2020]. O CodeBERT é capaz de codificar tokens de código de uma forma que preserva tanto as informações lexicais quanto as funcionais, tornando seus *embeddings* adequados para tarefas subsequentes de engenharia de software, como detecção de padrões e análise de vulnerabilidades.

4.4. Modelos de Classificação

Para os experimentos realizados neste trabalho, foram utilizados os seguintes modelos de aprendizado de máquina: Árvore de Decisão [Safavian and Landgrebe 1991], Floresta Aleatória [Breiman 2001] e Regressão Logística [Wright 1995]. Todos os modelos foram executados usando a biblioteca scikit-learn com a configuração padrão de hiperparâmetros.

A regressão logística é um modelo de classificação linear que estima a probabilidade de um resultado binário usando uma função logística (sigmoide) aplicada a uma combinação ponderada de características de entrada. Apesar de sua simplicidade, é amplamente utilizada devido à sua interpretabilidade e eficácia quando os limites de classe são aproximadamente lineares no espaço de características.

[2] `https://github.com/crytic/pyevmasm`

Uma Árvore de Decisão é um modelo não linear que divide o espaço de características em regiões por meio de uma série de regras de decisão hierárquicas. Em cada nó, o algoritmo seleciona uma característica e um limiar que melhor divide os dados de acordo com uma medida de impureza. Esse processo continua recursivamente, formando uma estrutura de árvore onde as folhas correspondem às previsões de classe. As árvores de decisão podem capturar interações complexas entre características e limites de decisão não lineares, mas são propensas a sobreajuste se não forem devidamente restringidas.

A Floresta Aleatória é um comitê que combina múltiplas árvores de decisão para melhorar o desempenho preditivo e a robustez. Cada árvore é treinada em uma amostra dos dados e, a cada divisão, um subconjunto aleatório de características é considerado, introduzindo diversidade entre as árvores. As previsões finais são obtidas pela agregação das saídas das árvores individuais, geralmente por meio de votação majoritária. Essa abordagem de conjunto reduz a variância, mitiga o sobreajuste e permite que o modelo capture padrões complexos em espaços de *embeddings* de alta dimensionalidade.

O conjunto de dados usado neste estudo apresenta classes desbalanceadas. Como distribuições desbalanceadas podem enviesar os classificadores, diferentes estratégias de balanceamento de dados foram exploradas.

A subamostragem aleatória reduz o desequilíbrio de classes, removendo amostras da classe majoritária até que ambas as classes tenham representação igual. Essa abordagem preserva os dados originais da classe minoritária de forma integral, sacrificando amostras da classe majoritária. No entanto, essa abordagem pode descartar exemplos informativos e reduzir a diversidade de padrões disponíveis para aprendizado.

Já a sobreamostragem é uma estratégia que busca aumentar a representação da classe minoritária no conjunto de dados, em vez de remover exemplos da classe majoritária. Dessa forma, essa abordagem objetiva preservar todos os dados originais e, ao mesmo tempo, reduzir o viés em direção à classe majoritária.

Para a sobreamostragem, utilizamos o SMOTE (Synthetic Minority Oversampling Technique) [Chawla et al. 2002]. O SMOTE lida com o desequilíbrio gerando amostras sintéticas da classe minoritária, interpolando entre uma instância da classe minoritária e seus vizinhos mais próximos no espaço de características. Esse processo aumenta a representação da classe minoritária, mantendo a variabilidade, ajudando os classificadores a aprender regiões de decisão mais amplas para a classe minoritária.

Outra técnica que optamos por utilizar para a sobreamostragem é o Mixup, que é uma técnica de aumento de dados que cria novas amostras de treinamento combinando linearmente pares de instâncias [Zhang et al. 2017]. Dadas duas amostras, tanto as características de entrada quanto os rótulos são interpolados com um coeficiente λ em uma distribuição Beta, incentivando os modelos a se comportarem de forma mais linear entre exemplos conhecidos. Quando aplicado a vetores de *embeddings*, o Mixup promove limites de decisão mais suaves e pode melhorar a robustez, embora possa obscurecer as distinções de classe se as classes subjacentes não estiverem bem separadas.

Para obter estimativas robustas do desempenho do modelo, empregamos validação cruzada k-*fold* em toda a nossa avaliação. Na validação cruzada k-*fold*, o conjunto de dados é dividido em k partes iguais; cada parte, por sua vez, serve como um conjunto de validação separado, enquanto as k - 1 partes restantes são usadas para treinamento.

Esse processo gera k avaliações independentes, que podem ser agregadas para fornecer uma avaliação abrangente do desempenho de generalização em todo o conjunto de dados. Comparada a uma única divisão treino/teste, a validação cruzada k-*fold* reduz a variância nas estimativas de desempenho e aproveita todos os dados disponíveis tanto para treinamento quanto para validação. Neste trabalho, usamos k = 5.

4.5. Métricas de Avaliação

Para avaliar o desempenho do modelo de forma abrangente, foram utilizadas múltiplas métricas de avaliação, considerando que confiar em uma única métrica, como a acurácia, pode fornecer uma visão limitada. As definições matemáticas detalhadas para algumas métricas estão dispostas na Tabela 2.

Tabela 2. Fórmulas matemáticas para algumas métricas de avaliação.

Acurácia (Acc.)	Precisão (P)	Revocação (R)	F_β-*score*	MCC
$\frac{TP+TN}{TP+TN+FP+FN}$	$\frac{TP}{TP+FP}$	$\frac{TP}{TP+FN}$	$\frac{(1+\beta^2)\,P\cdot R}{\beta^2 P+R}$	$\frac{TP\cdot TN-FP\cdot FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$

A acurácia mede a proporção de instâncias classificadas corretamente entre todas as amostras. Embora intuitiva, pode ser excessivamente otimista em cenários desbalanceados, visto que um modelo enviesado para a classe majoritária pode alcançar alta acurácia independente da qualidade do modelo.

A precisão quantifica a proporção de casos positivos previstos que são de fato positivos. Ela reflete a confiabilidade das previsões positivas e é particularmente importante quando os falsos positivos são custosos. A revocação mede a proporção de contratos vulneráveis reais corretamente identificados pelo modelo. Essa métrica pesa mais a falha em detectar uma vulnerabilidade (falso negativo).

O F_1-*score* é a média harmônica da precisão e da revocação, equilibrando ambas as métricas em um único valor. É útil quando há necessidade de equilibrar a detecção de vulnerabilidades e a limitação de falsos positivos. Já o F_2-*score* é uma variação do F_β-*score* que atribui maior peso à revocação do que à precisão.

A AUC-ROC (Área sob a curva ROC) avalia a capacidade do modelo de classificar instâncias em todos os limiares de decisão possíveis. Ela compara a taxa de verdadeiros positivos contra a taxa de falsos positivos e fornece uma medida de separabilidade entre classes independente de limiar. Valores mais próximos de 1 para essa área possuem melhor capacidade de classificação, enquanto 0,5 representa o comportamento aleatório.

A AUC-PR concentra-se na relação entre precisão e revocação em diferentes limiares. Ao contrário da AUC-ROC, ela enfatiza o desempenho na classe minoritária (vulnerável) em vez de ser influenciada por verdadeiros negativos.

Por fim, o MCC é uma métrica baseada em correlação que considera todos os quatro resultados da matriz de confusão (verdadeiros positivos, verdadeiros negativos, falsos positivos e falsos negativos) [Boughorbel et al. 2017]. Ele fornece uma medida equilibrada mesmo quando os tamanhos das classes diferem e varia de -1 (discordância total) a +1 (previsão perfeita), com 0 indicando desempenho aleatório.

5. Resultados

5.1. Análise dos Classificadores

Nesta seção, discorremos sobre o desempenho dos modelos de classificação avaliados utilizando o conjunto de métricas descritas na Seção 4.5. A Tabela 3 mostra uma visão comparativa de diferentes modelos e estratégias de balanceamento, destacando os melhores valores de desempenho para cada métrica, com seus valores em negrito. Por meio desta visão comparativa, desenvolvemos a análise que visa entender como cada estratégia de predição está desempenhando de forma relativa e absoluta.

Tabela 3. Resultados dos classificadores para códigos-fonte vulneráveis.

Bal.	Alg.	Acc.	Pre.	Rec.	F_1-*score*	F_2-*score*	TP	TN	MCC	AUC-ROC	AUC-PR
Nenhum	DT	0,8307	0,8609	0,9114	0,8854	0,9008	31047	8370	0,5655	0,7896	0,8571
	RF	**0,8613**	0,8581	**0,9667**	**0,9092**	**0,9498**	**32931**	7939	**0,6406**	**0,8844**	**0,9396**
	LR	0,8262	0,8246	0,9628	0,8883	0,9316	32798	6408	0,5396	0,8229	0,9039
Sub.	DT	0,4845	0,7236	0,4562	0,5596	0,4926	15541	7449	0,0115	0,5004	0,7065
	RF	0,5015	0,7256	0,4916	0,5861	0,5255	16746	6333	0,0166	0,4998	0,7139
	LR	0,5255	0,7320	0,5349	0,6181	0,5654	18223	6712	0,0328	0,5198	0,7165
SMOTE	DT	0,8108	0,8737	0,8598	0,8667	0,8625	29290	9150	0,5370	0,7921	0,8608
	RF	0,8410	**0,8801**	0,9012	0,8905	0,8969	30701	**9203**	0,6005	0,8829	0,9391
	LR	0,7921	0,8593	0,8496	0,8544	0,8515	28941	8647	0,4914	0,8282	0,9084
Mixup	DT	0,8219	0,8661	0,8896	0,8777	0,8848	30304	8698	0,5516	0,7928	0,8618
	RF	0,8481	0,8747	0,9203	0,8969	0,9108	31350	8893	0,6121	0,8792	0,9375
	LR	0,7934	0,8573	0,8545	0,8559	0,8550	29108	8539	0,4912	0,8273	0,9079

Diante de todas as diferentes configurações avaliadas na Tabela 3, a Floresta Aleatória sem balanceamento obteve o melhor desempenho geral, com uma AUC-ROC de 0,8844, e também liderou na maioria das outras métricas de avaliação. Isso indica que o modelo foi capaz de capturar a estrutura dos dados sem o uso de reamostragem, apesar do desequilíbrio de classes. Os modelos de Floresta Aleatória, em geral, mostraram um comportamento forte e estável também nos experimentos com sobreamostragem.

Em contraste, a subamostragem aleatória levou a uma severa degradação no desempenho de todos os modelos, com valores de AUC-ROC próximos a 0,5, ou seja, efetivamente aleatórios. Esse comportamento foi consistente em todos os modelos testados, indicando uma consequência específica da transformação dos dados, e não de um modelo específico. Uma explicação provável é que a subamostragem removeu uma grande parte das amostras da classe majoritária que oferecem maior contribuição informativa para separação de classes, ajudando a definir um limite de decisão mais claro. Como consequência, as amostras remanescentes teriam maior nível de ambiguidade, confundindo o classificador.

Embora o conjunto de dados resultante tenha se tornado balanceado em termos de classes, ele também se tornou mais homogêneo e menos informativo. Os modelos teriam sido então treinados principalmente com exemplos "difíceis", reduzindo sua capacidade de aprender uma classificação assertiva entre as classes.

Essas descobertas destacam um ponto importante: o balanceamento de classes por si só não gerou melhores resultados. Nesse caso, preservar a distribuição completa

dos dados reteve mais informações sobre como o código vulnerável e o não vulnerável diferem no espaço de *embeddings*, o que permitiu que a Floresta Aleatória explorasse padrões sutis, porém consistentes.

Observando as quantidades de verdadeiros negativos (TN), nota-se que, com exceção do cenário de subamostragem aleatória, a aplicação de estratégias de sobreamostragem levou a um aumento no número de verdadeiros negativos identificados pelos modelos, o que sugere que o enriquecimento dos dados de treinamento com representações adicionais da classe minoritária melhorou a capacidade dos modelos de caracterizar o limite da classe majoritária, levando a um menor número de contratos não vulneráveis classificados erroneamente como vulneráveis. No entanto, essas melhorias na contagem de verdadeiros negativos não se traduziram consistentemente em ganhos substanciais nas métricas gerais, indicando que o desempenho global permaneceu limitado pela separabilidade dos *embeddings*.

Considerando que os modelos se basearam apenas em *embeddings* CodeBERT, sem etapas de *feature engineering* ou ajuste de hiperparâmetros, os resultados apresentados são promissores. Alcançar um alto desempenho de discriminação diretamente a partir de *embeddings* genéricos de código pré-treinadaos sugere que as representações contemplam padrões significativos relevantes para a vulnerabilidades em contratos inteligentes.

5.2. Análise dos Embeddings

Para melhor compreender a estrutura dos *embeddings* obtidos, uma Análise de Componentes Principais (PCA) foi aplicada aos vetores dos contratos e projetada em duas dimensões, conforme a Figura 3. O primeiro componente principal (PC1) explica 82,7% da variância total, enquanto o segundo componente principal (PC2) explica 10,7%, resultando em 93,4% da variância combinada capturada na projeção 2D.

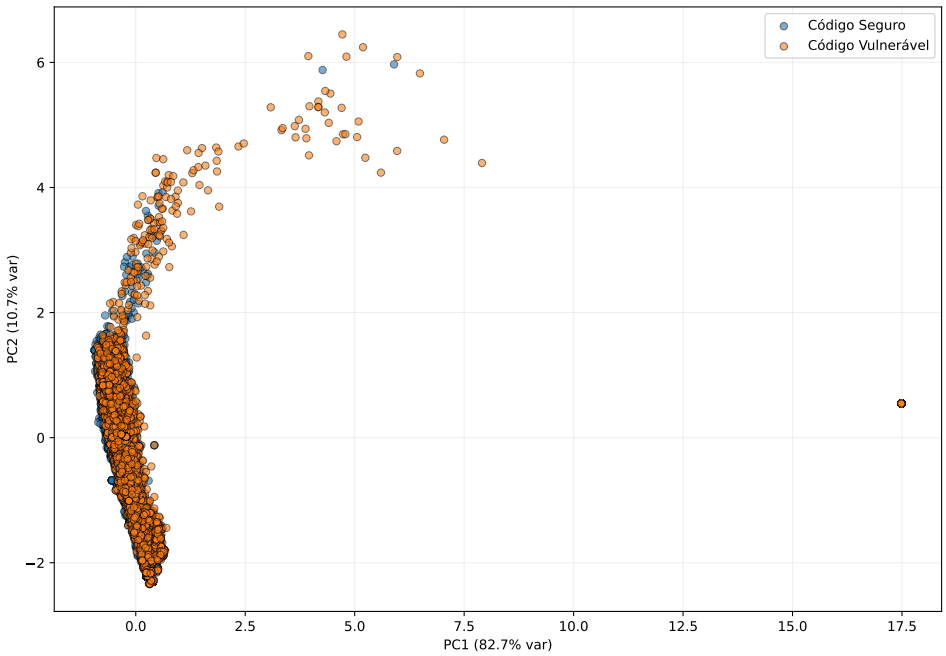

Figura 3. *Embeddings* em duas dimensões pós redução de dimensionalidade.

A projeção revela dois padrões distintos. Primeiro, observa-se um grande e denso agrupamento de amostras que varia principalmente ao longo do PC2. Dentro dessa região, amostras de código vulneráveis e não vulneráveis estão fortemente misturadas, sugerindo que uma porção da variância corresponde a diferenças que não estão diretamente relacionadas à vulnerabilidade. Essa região provavelmente reflete padrões de programação comuns, estruturas sintáticas compartilhadas ou convenções específicas do domínio de código-fonte Solidity.

Em segundo lugar, um grupo menor e mais distinto de amostras emerge em valores mais altos ao longo do PC1. Essa região contém um número desproporcionalmente grande de instâncias de código vulneráveis, formando um subespaço parcialmente separável. Isso sugere que o PC1 captura propriedades estruturais ou semânticas correlacionadas com a vulnerabilidade, como fluxos de código atípicos, uso incomum de APIs ou outros padrões negativos que se desviam da distribuição dominante do código. Nesse sentido, a vulnerabilidade parece estar associada a uma sub-região específica do espaço, em vez de estar distribuída uniformemente.

Essas observações indicam que embora as informações relacionadas à vulnerabilidade estejam presentes nos *embeddings* gerados pelo modelo pré-treinado sem o uso de retreinamentos, o sinal está localizado e inserido em uma sub-região do espaço amostral. Com isso, um modelo não-linear baseado em comitê como é o modelo de Floresta Aleatória foi capaz de isolar essa sub-região, enquanto separações lineares mais simples podem ter tido dificuldades devido à área densamente misturada.

Esta visualização ajuda também a explicar a forte degradação de desempenho observada sob subamostragem aleatória. A projeção mostrou que as amostras vulneráveis não estão distribuídas uniformemente pelo espaço de *embeddings*, mas sim concentradas em uma sub-região, alinhada principalmente com o primeiro componente principal, enquanto a maioria dos pontos de dados de ambas as classes ocupa um grupo mais volumoso e densamente misturado. A subamostragem aleatória remove uma porção substancial das amostras da classe majoritária sem levar em consideração sua localização nessa estrutura, o que pode ter ocasionado em perda de relações importantes no espaço de características. Como resultado, o modelo é treinado em um subconjunto mais homogêneo, que carece do contraste contextual mais amplo presente nos dados originais. Quando avaliado na distribuição completa, isso leva a uma separação pior e a um desempenho de classificação degradado, refletido no colapso da AUC-ROC.

A visualização corrobora os resultados quantitativos, mostrando que os *embeddings* de código contêm informações estruturadas relevantes para a detecção de vulnerabilidades, mesmo na ausência de técnicas mais refinadas.

6. Conclusões e Trabalhos Futuros

Este trabalho investigou a viabilidade de detectar contratos inteligentes que possuem vulnerabilidades usando apenas *embeddings* resultantes de um CodeBERT pré-treinado. Os resultados demonstram que os *embeddings* já codificam informações estruturais e semânticas relevantes para a detecção de vulnerabilidades. Em particular, o modelo de Floresta Aleatória obteve um desempenho robusto (AUC-ROC = 0,8844), indicando que modelos não lineares podem explorar efetivamente padrões presentes no espaço dos *embeddings*.

Os experimentos também possibilitaram comparações de estratégias de balanceamento. A subamostragem aleatória, apesar de balancear as classes, degradou consistentemente o desempenho em todos os modelos, com valores de AUC-ROC aproximando-se do aleatório. Isso sugere que a remoção de amostras da classe majoritária reduziu a diversidade e os padrões necessários para uma separação eficaz das classes. Em contraste, a preservação da distribuição natural dos dados manteve a variação de informação e permitiu que os modelos distinguissem melhor entre código vulnerável e não vulnerável.

Além disso, a visualização baseada em PCA mostrou que as informações relacionadas à vulnerabilidade parecem estar concentradas em uma sub-região específica do espaço de *embeddings*, reforçando a ideia de que as representações pré-treinadas já contêm sinais relevantes para a tarefa, embora localizados.

Diversas direções podem expandir e aprofundar este estudo. Os experimentos atuais se basearam principalmente em configurações de modelo padrão e no vetor de *embeddings* completo. O ajuste de hiperparâmetros pode melhorar ainda mais o desempenho. Além disso, técnicas de *feature engineering* podem ajudar a identificar os componentes mais relevantes para a tarefa, potencialmente melhorando a generalização e a interpretabilidade. Embora os modelos clássicos de aprendizado de máquina tenham apresentado bom desempenho, arquiteturas mais expressivas ou modelos mais complexos poderiam capturar melhor padrões sutis no espaço amostral.

Análises adicionais de estratégias de aumento de dados e reamostragem também são relevantes. O forte impacto negativo da subamostragem aleatória sugere que nem todas as técnicas de balanceamento são benéficas. Trabalhos futuros devem comparar sistematicamente abordagens alternativas, como ponderação de classes, estratégias de amostragem sintética mais avançadas ou aumento de dados em nível de representação. Compreender como diferentes técnicas afetam o comportamento dos modelos será importante para o desenvolvimento da arquitetura de treinamento em cenários de desequilíbrio de classes.

Ao explorar essas direções, pesquisas futuras podem se basear na linha de base estabelecida aqui e avançar em direção a sistemas mais precisos, robustos e interpretáveis para a detecção automatizada de vulnerabilidades de contratos inteligentes.

Referências

Boughorbel, S., Jarray, F., and El-Anbari, M. (2017). Optimal classifier for imbalanced data using matthews correlation coefficient metric. *PloS one*, 12(6):e0177678.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357.

Chu, H., Zhang, P., Dong, H., Xiao, Y., Ji, S., and Li, W. (2023). A survey on smart contract vulnerabilities: Data sources, detection and repair. *Information and Software Technology*, 159:107221.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 con-*

ference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers), pages 4171–4186.

Di Angelo, M., Durieux, T., Ferreira, J. F., and Salzer, G. (2023). Smartbugs 2.0: An execution framework for weakness detection in ethereum smart contracts. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 2102–2105. IEEE.

Di Angelo, M. and Salzer, G. (2019). A survey of tools for analyzing ethereum smart contracts. In *Proceedings of the 2019 IEEE international conference on decentralized applications and infrastructures (DAPPCON)*, pages 69–78.

Dong, Z., Hu, Q., Guo, Y., Zhang, Z., Cordy, M., Papadakis, M., Le Traon, Y., and Zhao, J. (2025). Boosting source code learning with text-oriented data augmentation: an empirical study. *Empirical Software Engineering*, 30(3):68.

Durieux, T., Ferreira, J. F., Abreu, R., and Cruz, P. (2020). Empirical review of automated analysis tools on 47,587 ethereum smart contracts. In *Proceedings of the ACM/IEEE 42nd International conference on software engineering*, pages 530–541.

Feist, J., Grieco, G., and Groce, A. (2019). Slither: a static analysis framework for smart contracts. In *Proceedings of the 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*, pages 8–15.

Feng, Z. (2020). Codebert: A pre-trained model for program-ming and natural languages. *arXiv preprint arXiv:2002.08155*.

Hwang, S.-J., Ju, S. H., and Choi, Y.-H. (2024). Cggnet: compiler-guided generation network for smart contract data augmentation. *IEEE Access*, 12:97515–97532.

Safavian, S. R. and Landgrebe, D. (1991). A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, 21(3):660–674.

Sharma, N., Sharma, S., et al. (2022). A survey of mythril, a smart contract security analysis tool for evm bytecode. *Indian Journal of Natural Sciences*, 13(75):51003–51010.

Szabo, N. (1996). Smart contracts: building blocks for digital markets. *EXTROPY: The journal of transhumanist thought*, 18(2):28.

Tsankov, P., Dan, A., Drachsler-Cohen, D., Gervais, A., Buenzli, F., and Vechev, M. (2018). Securify: Practical security analysis of smart contracts. In *Proc. of the 2018 ACM SIGSAC conference on computer and communications security*, pages 67–82.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Wood, G. (2014). Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151:1–32.

Wright, R. E. (1995). Logistic regression.

Zhang, H., Cisse, M., Dauphin, Y. N., and Lopez-Paz, D. (2017). mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*.