Capitolo 1

Metodologia

Questa sezione descrive in dettaglio l'approccio seguito per affrontare il problema della classificazione degli smart contracts. Questo capitolo è fondamentale per comprendere come sono stati raccolti, pre-processati e utilizzati i dati, come è sono stati configurati e addestrati i modelli e quali strumenti e tecniche sono stati impiegati per ottenere i risultati presentati.

La metodologia adottata in questa tesi è suddivisa nelle seguenti fasi principali:

1. Raccolta e preparazione dei dati: esplorazione del dataset utilizzato, delle tecniche di pre-processing applicate e delle modalità di suddivisione dei dati per l'addestramento e la valutazione.

2. Modellazione: descrizione dell'architettura dei modelli utilizzati, delle scelte di configurazione e delle strategie di addestramento. .

Ogni fase sarà trattata in modo dettagliato, evidenziando le scelte metodologiche compiute e le motivazioni alla base di tali scelte. Questo approccio sistematico garantisce trasparenza e replicabilità del lavoro svolto, consentendo ad altri di comprendere e, eventualmente, replicare i risultati ottenuti.

1.1 Esplorazione dei dati

Il dataset [**?**] utilizzato in questo progetto è un dataset disponibile pubblicamente sulla piattaforma HuggingFace una delle più importanti piattaforme per il Natural Language Processing. HF è un'infrastruttura open-source che fornisce accesso a una vasta gamma di modelli di deep learning pre-addestrati, tra cui alcuni dei più avanzati nel campo del NLP. Questo dataset contiene informazioni su 106.474 SmartContracts pubblicati sulla rete Ethereum. Ogni elemento nel dataset è composto da quattro elementi:

- **Address**: l'indirizzo del contratto

- **SourceCode**: il codice sorgente del contratto, scritto in linguaggio Solidity
- **ByteCode**: il codice bytecode del contratto, ottenuto a partire dalla compilazione del codice sorgente utilizzando il compilatore di Solidity. Questo bytecode è quello che viene eseguito sulla macchina virtuale di Ethereum (EVM).
- **Slither**: il risultato dell'analisi statica del contratto con Slither, un tool open-source per l'analisi statica di contratti scritti in Solidity. Questo risultato è un array di valori che vanno da 1 a 5, dove ogni numero rappresenta la presenza di una vulnerabilità e 4 rappresenta un contratto safe, cioè privo di vulnerabilità.

Le vulnerabilità che sono state prese in questo lavoro sono le seguenti:

- Access-Control
- Arithmetic
- Other
- Reentrancy
- Safe
- Unchecked-Calls

Prima della costruzione dei modelli è stata affrontata una fase di analisi esplorativa dei dati. Questa fase è stata svolta per comprendere meglio la struttura del dataset e dei contratti da classificare, per individuare eventuali problemi. A livello pratico, questa fase di analisi esplorativa dei dati è stata eseguita utilizzando il linguaggio Python, con l'ausilio di librerie come Pandas, NumPy, Matplotlib e Seaborn per l'analisi e la visualizzazione dei dati. Il dataset è diviso in tre sottoinsiemi: training, validation e test set. Il dataset di training è composto da 79.641 contratti, il dataset di validazione da 10.861 contratti e il dataset di test da 15.972 contratti. Tutte le informazioni sono presenti per tutti i contratti tranne l'informazione relativa al bytecode, che risulta essere assente per pochissimi contratti come visibile nella Tabella 1.1. Per ottenere una visione d'insieme

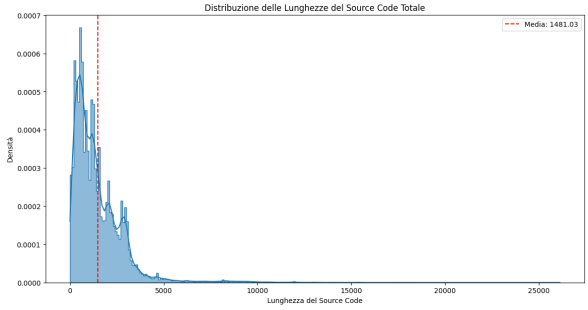
Dataset	**Count**	**%**
Train	227	0.285%
Test	51	0.319%
Validation	30	0.276%

Tabella 1.1: Conteggio e Percentuale di Contratti Senza Bytecode per Dataset

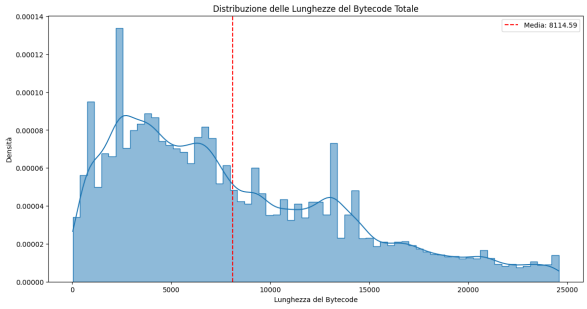
delle lunghezze dei contratti, abbiamo calcolato la lunghezza media del source code e del

bytecode. Prima del preprocessing le lunghezze medie di SourceCode e ByteCode sono rispettivamente di 3155 token e 8114 token. Abbiamo visualizzato la distribuzione delle lunghezze del source code utilizzando un istogramma. Per migliorare la leggibilità del grafico, abbiamo raggruppato i dati per quanto riguarda il source code in intervalli di 500 token. L'istogramma è accompagnato da una linea che indica la lunghezza media dei token rappresentata con una linea tratteggiata rossa.

L'inclusione delle lunghezze medie fornisce un punto di riferimento utile per interpretare le distribuzioni e confrontare i singoli esempi di codice rispetto alla media del dataset. Queste analisi sono fondamentali per le successive fasi di preprocessing e modellazione, garantendo che i modelli possano gestire efficacemente la variabilità presente nei dati. Sul bytecode non è stato applicato nessun tipo di preprocessing per ridurre la dimensione dei dati. Per quanto riguarda il codice sorgente sono stati eliminati tutti i commenti e le funzioni getter monoistruzione, cioè tutte quelle funzioni `getX()` le quali abbiano come unica istruzione una istruzione di return, poichè sono state assunte come funzioni corrette, l'eliminazione di queste stringhe è avvenuta tramite una ricerca delle stringhe effettuata con una regex. Abbiamo unito i set di dati di addestramento, test e validazione in un unico DataFrame per analizzare le lunghezze del source code e del bytecode. In particolare, sono state calcolate rispettivamente le lunghezze del codice sorgente e del bytecode. Effettuando le rimozioni dei commenti la media del numero di token del sourcecode scende a 1511 token, mostrando come la rimozione dei commenti abbia un grande impatto sulla lunghezza media del codice. Rimuovendo anche le funzioni getter monoistruzione la lunghezza media del source code scende a 1481 token. Poichè



(a) Distribuzione delle Lunghezze del Source Code dopo il preprocessing



(b) Distribuzione delle Lunghezze del Bytecode dopo il preprocessing

Figura 1.1: Distribuzioni delle lunghezze del source code e del bytecode.

successivamente andremo a classificare i contratti con dei modelli nella famiglia BERT che prendono in input sequenze di token lunghe al massimo 512 token abbiamo calcolato la percentuale di contratti che non superano questa soglia e in alcuni suoi multipli, per capire quanti contratti riusciamo a classificare per intero e quanti verranno troncati. I risultati sono mostrati nella Tabella 1.2.

Metrica	Sotto 512	Sotto 1024	Sotto 1536	Media
Source Code (%)	21.90	46.04	64.77	62.21
Bytecode (%)	1.56	6.31	8.75	58.69

Tabella 1.2: Percentuale di contratti sotto varie lunghezze in token.

Diventa però importante notare, che per molti casi di contratti che superano i 5000 token questi sono così lunghi poichè riportano in calce al contratto anche il codice sorgente di librerie esterne, che non è di interesse per la classificazione delle vulnerabilità.

1.1.1 Distribuzione delle Classi e Matrici di Co-occorrenza

Successivamente, la fase di esplorazione dei dati ha previsto l'analisi delle classi di vulnerabilità dei dati. In questa sezione, presentiamo la distribuzione delle classi e le matrici di co-occorrenza per i dataset di addestramento, test e validazione. Si precisa che i risultati di seguito proposti si riferiscono già al dataset da cui sono stati sottratti i contratti privi di bytecode.

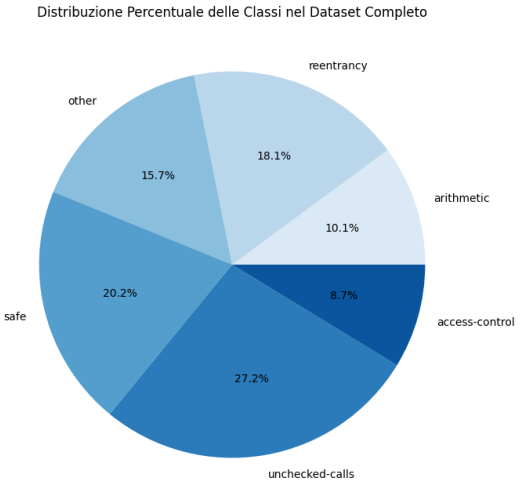
Distribuzione delle Classi

La Tabella 1.3 mostra la distribuzione delle classi per i tre dataset. È evidente che la classe 'unchecked-calls' è la più frequente in tutti e tre i dataset, mentre la classe 'access-control' è la meno rappresentata.

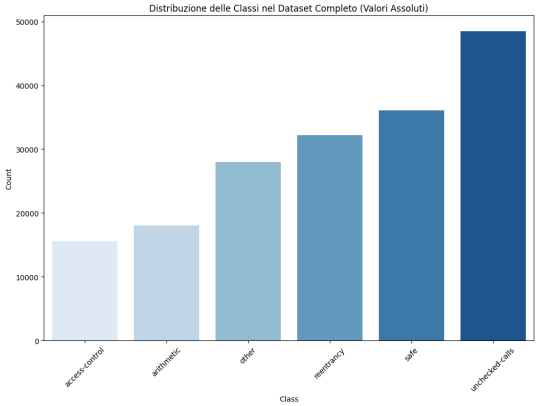
Class	Train		Test		Validation		Full	
	Count	**%**	**Count**	**%**	**Count**	**%**	**Count**	**%**
access-control	11619	8.71%	2331	8.71%	1588	8.73%	15538	8.72%
arithmetic	13472	10.10%	2708	10.12%	1835	10.09%	18015	10.10%
other	20893	15.67%	4193	15.67%	2854	15.69%	27940	15.67%
reentrancy	24099	18.07%	4838	18.09%	3289	18.08%	32226	18.08%
safe	26979	20.23%	5405	20.20%	3676	20.21%	36060	20.23%
unchecked-calls	36278	27.21%	7276	27.20%	4951	27.21%	48505	27.21%

Tabella 1.3: Distribuzione delle Classi nei Dataset di Addestramento, Test, Validazione e Completo

Le Figure 1.2a e 1.2b mostrano rispettivamente la distribuzione percentuale e assoluta delle classi nell'intero dataset. Queste visualizzazioni forniscono una panoramica chiara della frequenza delle diverse classi all'interno del dataset, evidenziando le differenze di distribuzione tra le classi. Dalla distribuzione delle classi nei diversi dataset, possiamo osservare che:



(a) Distribuzione Percentuale delle Classi



(b) Distribuzione Assoluta delle Classi

Figura 1.2: Distribuzioni delle Classi nell'intero dataset, in termini relativi e assoluti.

- Le classi sono distribuite in modo abbastanza uniforme nei dataset di addestramento, test e validazione, con percentuali simili tra i tre split per classe
- La classe 'unchecked-calls' è la più frequente in tutti e tre i dataset, con una presenza significativa soprattutto nel dataset di addestramento (36278 occorrenze).
- La classe 'access-control' è la meno frequente, con il numero più basso di occorrenze nel dataset di validazione (1588 occorrenze).
- Le classi 'safe' e 'reentrancy' sono anche abbastanza rappresentate

Matrici di Co-occorrenza

Le Tabelle 1.3, 1.4 e 1.5 mostrano le matrici di co-occorrenza per i dataset di addestramento, test e validazione rispettivamente. Le matrici di co-occorrenza indicano la frequenza con cui ogni coppia di classi appare insieme nello stesso elemento.

In questa sezione, vengono presentate le matrici di co-occorrenza per ogni split del dataset, mostrando sia in termini assoluti che relativi il numero di cooccorrenze tra le varie classi.

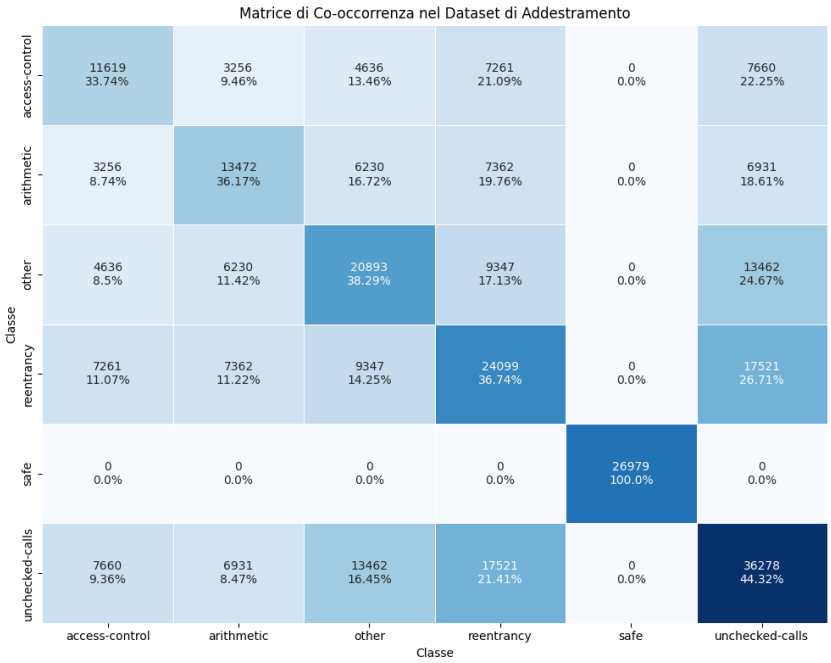


Figura 1.3: Matrice di Co-occorrenza nel Dataset di Addestramento

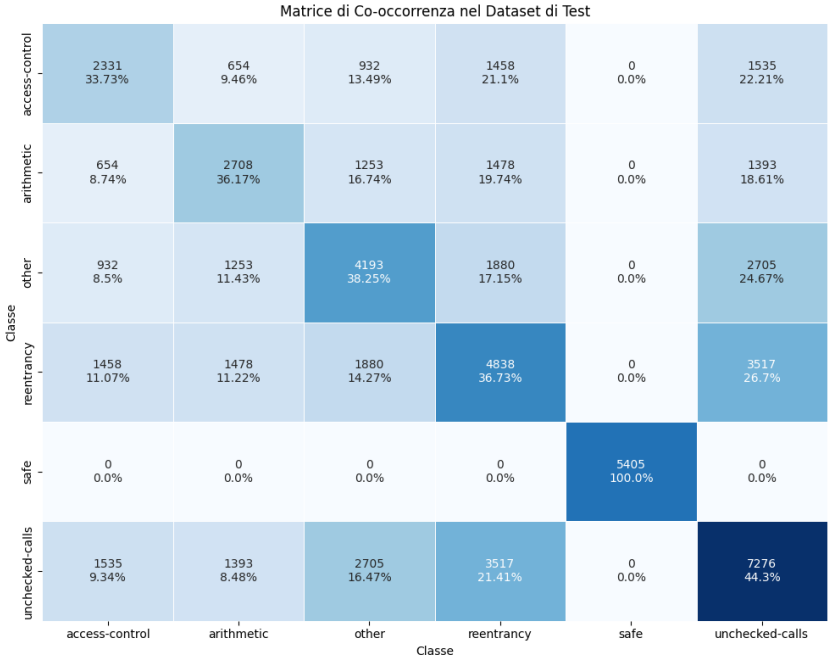


Figura 1.4: Matrice di Co-occorrenza nel Dataset di Test

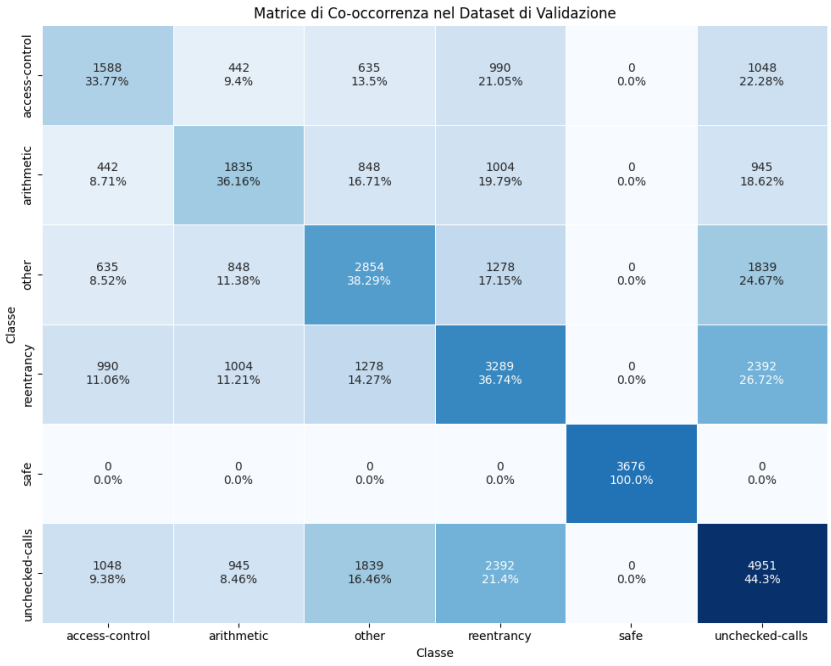


Figura 1.5: Matrice di Co-occorrenza nel Dataset di Validazione

Analizzando le matrici di co-occorrenza, notiamo che:

- La classe safe, che rappresenta i contratti privi di vulnerabilità, correttamente non apparte contemporaneamente a nessuna delle altre classi.
- Le classi 'unchecked-calls' co-occorrono frequentemente con 'reentrancy', 'other', e 'access-control'. Questo suggerisce che i contratti con chiamate non verificate spesso presentano anche altri tipi di vulnerabilità.
- Le classi 'arithmetic' e 'reentrancy' mostrano una co-occorrenza significativa, suggerendo che le vulnerabilità aritmetiche possono spesso essere associate a problemi di rientro.

Questi risultati evidenziano l'importanza di considerare la co-occorrenza delle classi quando si analizzano le vulnerabilità nei contratti intelligenti, poiché molte vulnerabilità non si verificano in isolamento ma tendono a manifestarsi insieme ad altre.

1.2 BERT

Il modello BERT (Bidirectional Encoder Representations from Transformers) rappresenta un pilastro fondamentale nel campo del Natural Language Processing (NLP),

grazie alla sua capacità di comprensione del contesto delle parole all'interno di una frase o di un testo più ampio. Questo modello, sviluppato da Google, si basa sull'architettura dei transformer, una classe di modelli neurali che ha dimostrato notevole successo nell'analisi del linguaggio naturale.

BERT si distingue per la sua capacità bidirezionale di elaborare il contesto linguistico. A differenza dei modelli NLP precedenti, che processavano il testo in modo sequenziale, interpretando le parole una dopo l'altra, BERT considera sia il contesto precedente sia quello successivo di ciascuna parola all'interno di una frase. Questo approccio bidirezionale consente a BERT di catturare relazioni semantiche più complesse e di fornire una rappresentazione più accurata del significato del testo.

Il funzionamento di BERT può essere compreso attraverso due fasi principali: l'addestramento e l'utilizzo.

Durante la fase di addestramento, BERT viene esposto a enormi quantità di testo, proveniente da varie fonti e domini. Utilizzando un processo noto come "pre-addestramento", il modello apprende i modelli linguistici e il contesto delle parole. Questo pre-addestramento coinvolge due compiti principali: la predizione di parole mascherate e la predizione della successione di frasi. Nel primo compito, BERT impara a prevedere le parole mancanti in una frase fornita, mentre nel secondo compito, il modello apprende a determinare se due frasi sono consecutive in un testo o sono state estratte casualmente da testi diversi.

Una volta completata la fase di addestramento, BERT può essere utilizzato per una vasta gamma di compiti NLP senza la necessità di ulteriori addestramenti specifici. Durante l'utilizzo, il modello riceve in input una sequenza di token, che possono essere parole, frammenti di testo o segmenti di frasi. Ogni token viene rappresentato come un vettore di caratteristiche, derivato dal contesto bidirezionale fornito da BERT durante l'addestramento. Queste rappresentazioni vettoriali possono essere utilizzate per svariati compiti, come classificazione di testo, analisi del sentiment, risposta alle domande, traduzione automatica e molto altro ancora.

In sintesi, il modello BERT ha rivoluzionato il campo del NLP introducendo una comprensione più approfondita e contestualizzata del linguaggio naturale. La sua capacità di catturare il contesto bidirezionale delle parole ha portato a miglioramenti significativi nelle prestazioni dei sistemi NLP su una vasta gamma di compiti e applicazioni. BERT rimane un pilastro fondamentale nell'ambito della comprensione automatica del linguaggio umano, consentendo a macchine e sistemi di interagire e comprendere il linguaggio umano in modo più naturale e preciso.