

Multimodal source code vulnerability description model for vulnerability classification

Nunzio Saitta
Corso di Cognitive Computing A.A. 2024/2025

Indice

| | | |
|----------|------------------------------------|----------|
| 1 | Introduzione | 2 |
| 2 | Obiettivi | 2 |
| 3 | Metodologia | 2 |
| 3.1 | Preparazione dei dati | 2 |
| 3.2 | Architettura del modello | 2 |
| 3.3 | Funzione di loss | 2 |
| 3.4 | Addestramento | 3 |
| 4 | Risultati | 3 |
| 4.1 | Loss | 3 |
| 4.2 | Accuracy | 4 |
| 4.3 | Macro F1-score | 5 |
| 4.4 | Metriche per classe | 5 |
| 5 | Conclusioni | 5 |

1 Introduzione

La classificazione automatica delle vulnerabilità, presenti nel codice, in categorie *CWE* rappresenta un passo fondamentale per la comprensione e la mitigazione delle debolezze del software. In questo progetto si propone un approccio **multimodale**, che combina l'utilizzo di snippet di codice e di descrizioni delle vulnerabilità, con l'obiettivo di migliorare la classificazione. L'uso congiunto di codice e testo introduce la difficoltà di *alignment* tra due domini differenti. L'allineamento multimodale può migliorare la generalizzazione, ma bisogna rivolgere particolare attenzione alla semantica *CWE* e all'overlap tra categorie.

2 Obiettivi

- Allineare rappresentazioni di codice e descrizioni testuali in uno spazio condiviso.
- Sfruttare architetture transformer pre-addestrate (CodeBERT per gli snippet di codice, RoBERTa per le descrizioni).
- Addestrare il modello per predire la categoria *CWE* corrispondente.

3 Metodologia

3.1 Preparazione dei dati

I dati sono stati ottenuti da dataset pubblici come il National Vulnerability Database(NVD). Per ciascun record sono stati estratti:

- snippet di codice
- descrizione testuale della vulnerabilità
- label *CWE*

Il dataset è stato preprocessato per rimuovere rumore e normalizzare i dati:

- *Codice*: rimozione di commenti, normalizzazione degli spazi e delle tabulazioni.
- *Testo*: lowercasing, stopword removal, lemmatizzazione.

3.2 Architettura del modello

Il modello proposto è composto da:

- Encoder CodeBERT per il codice.
- Encoder RoBERTa per la descrizione testuale.
- Due proiezioni lineari per mappare gli embedding nello spazio comune.
- Una testa di classificazione lineare che riceve l'embedding del codice e produce la distribuzione sulle categorie *CWE*.

3.3 Funzione di loss

La loss complessiva è definita come:

$$\mathcal{L} = \alpha \mathcal{L}_{class} + \beta \mathcal{L}_{align}$$

dove \mathcal{L}_{class} è relativa alla classificazione *CWE* e \mathcal{L}_{align} è la loss relativa all' allineamento che forza le rappresentazioni di codice e testo corrispondenti ad avvicinarsi nello spazio latente.

3.4 Addestramento

Sono stati effettuati vari addestramenti con differenti configurazioni al fine di ottimizzare le performance. In generale, la modifica che ha portato maggiori benefici è stata l'introduzione di **WeighetedSampler**: per bilanciare le classi durante l'addestramento. La sua introduzione ha portato a un miglior bilanciamento del training con conseguenze importanti sulla convergenza della loss, mentre a livello di classificazione ci sono stati incrementi significativi per le classi meno rappresentate nel dataset.

Questa è la configurazione finale del modello, a cui fanno riferimento i risultati successivamente riportati:

- Ottimizzatore: AdamW con learning rate iniziale $2e-5$.
- Scheduler lineare con warmup del 10% degli step totali.
- Epoche: 12.
- Dataset: 70% training, 15% validation, 15% test.
- Batch size: 16.
- Coefficienti di loss: $\alpha = 0.7$, $\beta = 0.3$.
- Metriche: Accuracy, Precision, Recall, F1-score, Top-3 Accuracy.

4 Risultati

4.1 Loss

L'andamento della loss totale mostra una riduzione regolare per tutte le epoche, indice di un apprendimento stabile. La loss di classificazione decresce più rapidamente mentre la loss relativa all'allineamento diminuisce più lentamente. Questo indica che il modello riesce goà a classificare dalla prime epoche e che poiché l'allineamento tra embedding di codice e testo richiede più tempo per stabilizzarsi. La convergenza complessiva indica che il bilanciamento dei pesi tra le due loss non introduce instabilità.

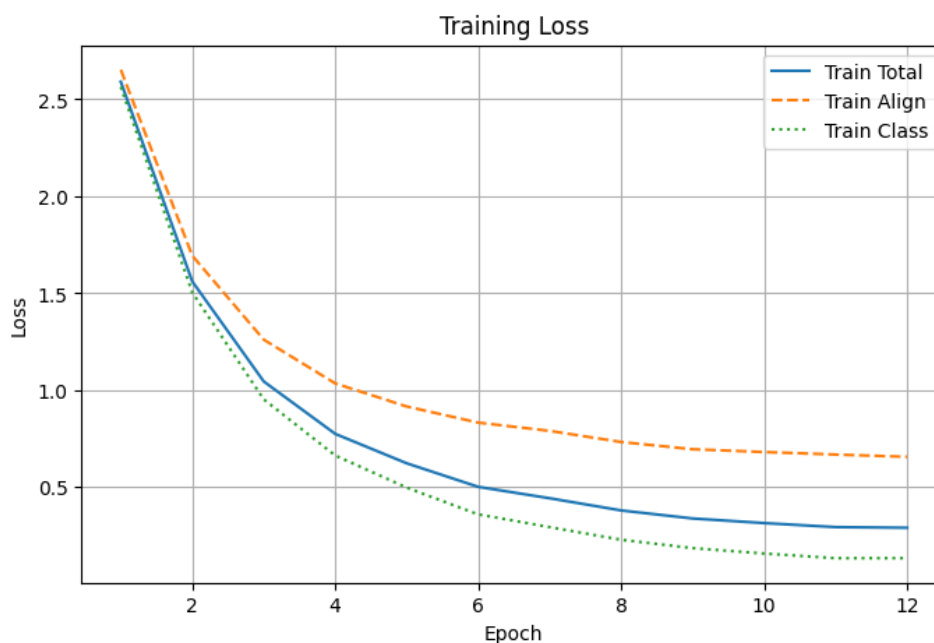


Figura 1: Andamento Train loss

4.2 Accuracy

L'accuratezza cresce rapidamente e si stabilizza tra le epoche 8 e 10. La stabilità delle curve di validazione e test conferma la solidità del modello e l'assenza di fluttuazioni anomale. I valori del test set si stabilizzano intorno al 65%.

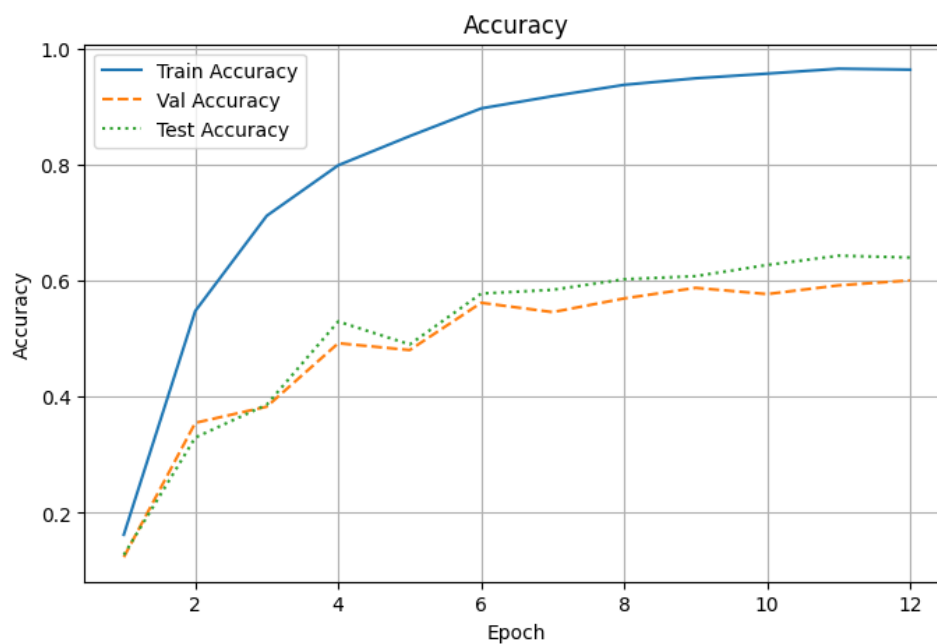


Figura 2: Andamento Accuracy

La Top-3 Accuracy raggiunge circa l'85%, indicando che nella maggior parte dei casi la classe corretta rientra tra le prime tre predizioni del modello.

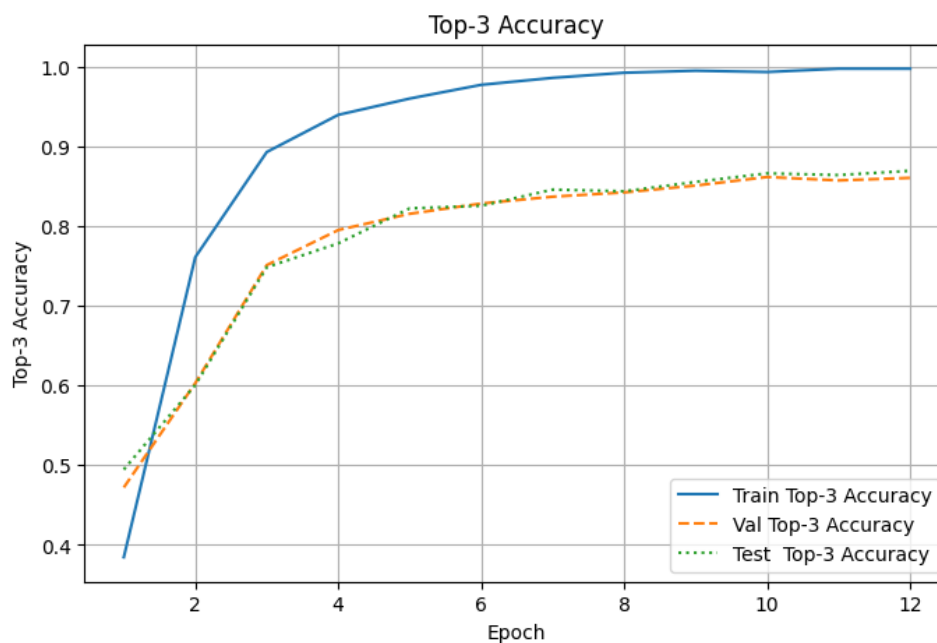


Figura 3: Andamento Top3 Accuracy

4.3 Macro F1-score

Il valore di Macro-F1 cresce rapidamente nelle prime epoche e tende a stabilizzarsi dopo la sesta, indicando una convergenza del modello. L'andamento simile di validation e test suggerisce una buona capacità di classificazione generale.

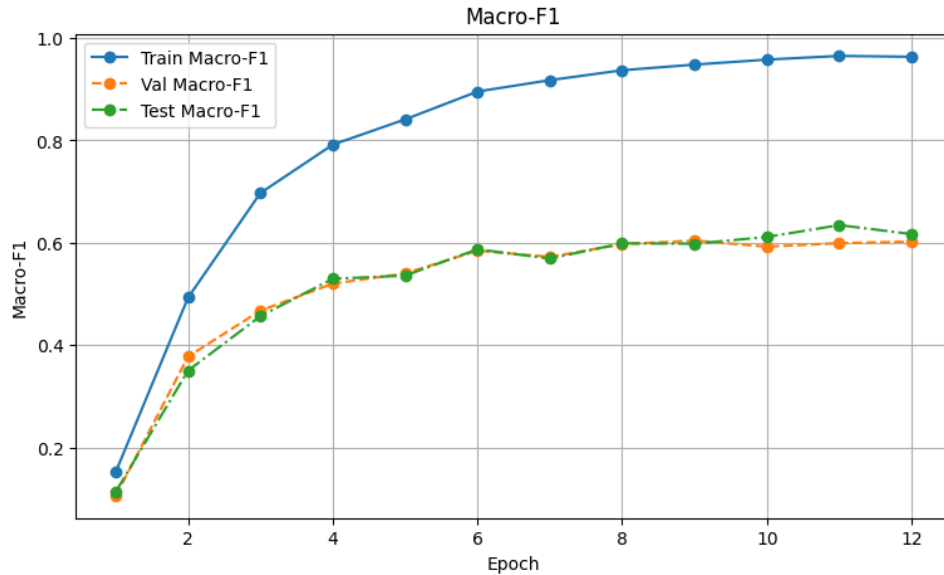


Figura 4: Andamento Macro F1-score

4.4 Metriche per classe

Le metriche per classe riportate nella seguente tabella mostrano le prestazioni classificazione, con valori di F1-score compresi tra 0.47 e 0.95 a seconda della tipologia di vulnerabilità (CWE). Contribuiscono alla qualità delle predizioni sia chiarezza dei pattern nel codice sia la possibile vicinanza semantica tra classi CWE.

| Label | Precision | Recall | F1-score | Support |
|---------|-----------|--------|----------|---------|
| CWE-732 | 1.000 | 0.909 | 0.952 | 11 |
| CWE-119 | 0.693 | 0.712 | 0.702 | 323 |
| CWE-284 | 0.724 | 0.656 | 0.689 | 32 |
| CWE-404 | 0.571 | 0.800 | 0.667 | 5 |
| CWE-125 | 0.667 | 0.667 | 0.667 | 84 |
| CWE-190 | 0.638 | 0.667 | 0.652 | 45 |
| CWE-20 | 0.607 | 0.654 | 0.630 | 182 |
| CWE-476 | 0.700 | 0.568 | 0.627 | 37 |
| CWE-79 | 0.571 | 0.667 | 0.615 | 6 |
| CWE-787 | 0.636 | 0.538 | 0.583 | 26 |
| CWE-362 | 0.625 | 0.541 | 0.580 | 37 |
| CWE-415 | 0.429 | 0.667 | 0.522 | 9 |
| CWE-200 | 0.536 | 0.462 | 0.497 | 80 |
| CWE-416 | 0.469 | 0.479 | 0.474 | 48 |
| CWE-59 | 0.500 | 0.333 | 0.400 | 9 |

Tabella 1: Metriche di valutazione per classe sul test set

5 Conclusioni

Il lavoro ha evidenziato l'efficacia di un approccio multimodale per la classificazione di vulnerabilità software, basato sull'allineamento tra codice sorgente e descrizioni testuali. L'architettura proposta, ha permesso di ottenere un buon equilibrio tra accuratezza e allineamento delle rappresentazioni. L'analisi

dei risultati evidenzia una Macro-F1 di circa 0.60. Le curve di loss confermano la corretta convergenza del modello e l'efficacia del bilanciamento tra le due componenti. È importante notare che l'assenza di una correlazione diretta tra il numero di esempi per classe e le metriche dipende, come detto in precedenza, anche dall'utilizzo del campionamento bilanciato durante la fase di train.

La vicinanza semantica tra alcune classi CWE può aver influenzato le performance, con pattern di codice simili che possono portare confusione tra categorie affini. Questo spiegherebbe in parte anche il valore della Top-3 Accuracy, che risulta significativamente più alto rispetto alla singola accuracy, infatti raggiunge circa l'85%, indicando che nella maggioranza dei casi la classe corretta rientra tra le prime tre predizioni del modello. Questo risultato conferma che il modello è in grado di riconoscere con buona affidabilità la tipologia di vulnerabilità, anche se non individua esattamente la classe. In conclusione, il modello sviluppato fa emergere come l'allineamento tra codice e linguaggio naturale può migliorare e supportare la classificazione CWE.