Assessment of Software Vulnerability Contributing Factors using XAI (eXplainable AI) Techniques

Contributor: Ding Li, Prof. Yan Liu

Presenter : Ding Li

Department of Electrical and Computer Engineering,

Concordia University, Montreal, Canada

Presentation Overview

- Introduction
  - Background of software vulnerability detection
  - XAI Feature importance explanation
- Related work
  - Factors in Code Representation Techniques
- Research Questions
- Methodology
  - Text-based factors assessment
  - Graph-based factors assessment (This talk focus in Graph-based)
- Experiment results (Graph-based factors assessing).
- Conclusion, Contribution, Reference, Discussion.

Background – software vulnerability detection

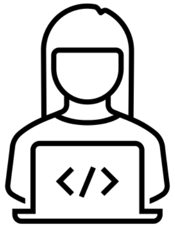
Software vulnerability:

- Flaws or weaknesses in a software program.

- can be exploited to perform unauthorized actions, such as breaching data or disrupting services[1].

How to detect software vulnerabilities?

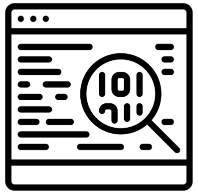
Manual Detection



Pros:
- In-depth understanding of the system's functionality

Cons:
- Time-consuming.
- Non scalable

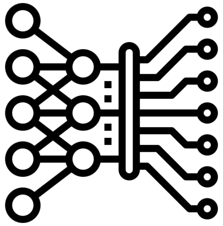
Static & Dynamic Analysis Tools



Pros:
- Automated and scalable

Cons:
- High false positives
- Limited by rule sets

Machine Learning Detection



Pros:
- Automated and scalable
- Continue learning from data
- Reduces false positives

Cons:
- Computationally expensive
- Transparent concerns

Information summarized from [3]

Background – software vulnerability detection

Give a vulnerable code snippet, what are the **contributing factors**, and how to **measure their features impact on** the prediction results of machine learning based detection approach?

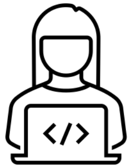
```
public void action(int data) throws Throwable
{

    /* POTENTIAL FLAW: Create a HashMap using data as the initial size.
    data may be very large, creating memory issues */
    HashMap intHashMap = new LinkedashMap(data);

}
```

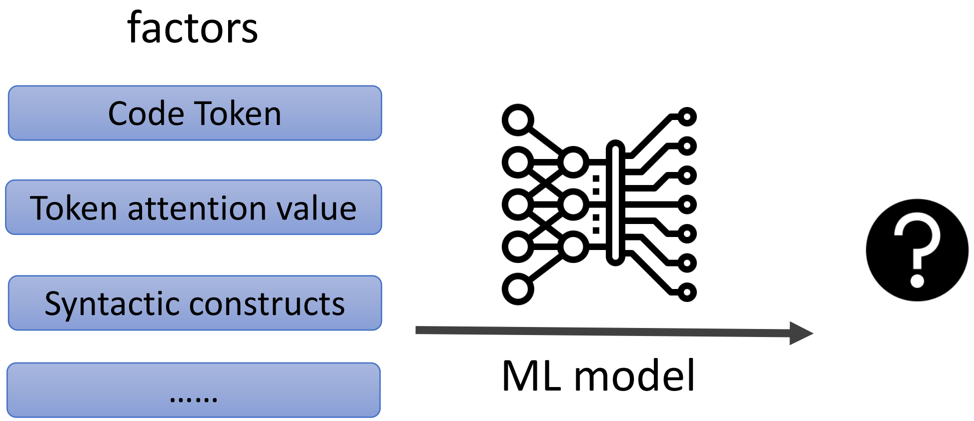
Figure 1. A code snippet example of vulnerability type (Memory Allocation with Excessive Size Value)

Factors that manual detection rely on:



- Semantic Tokens:
  - HashMap intHashMap, =
  - new, LinkedHashMap, data, ()
- Syntax Meanings:
  - LinkedHashMap *call* data
  - new *init* LinkedHashMap(data)
  - Hashmap *decl* LinkedHashMap(data)
  - LinkedHashMap(data) *expr* initHashMap

-> Memory Allocation with Excessive Size Value



What factors, and how the features affect the machine learning based detection decision?

Presentation Overview

- Introduction
  - Background of code vulnerability
  - XAI Feature importance explanation
- Related work
  - Factors in Code Representation Techniques
- Research Questions
- Methodology
  - Text-based factors assessment
  - Graph-based factors assessment
- Graph-based factors assessing results.
- Conclusion, Contribution, Reference, Discussion.

Background - XAI Feature Importance Explanation

XAI (eXplainable AI) feature importance explanation, as a branch of XAI method, helps user to understand the model's predictions and specific influence of individual features contributing to these predictions[2].

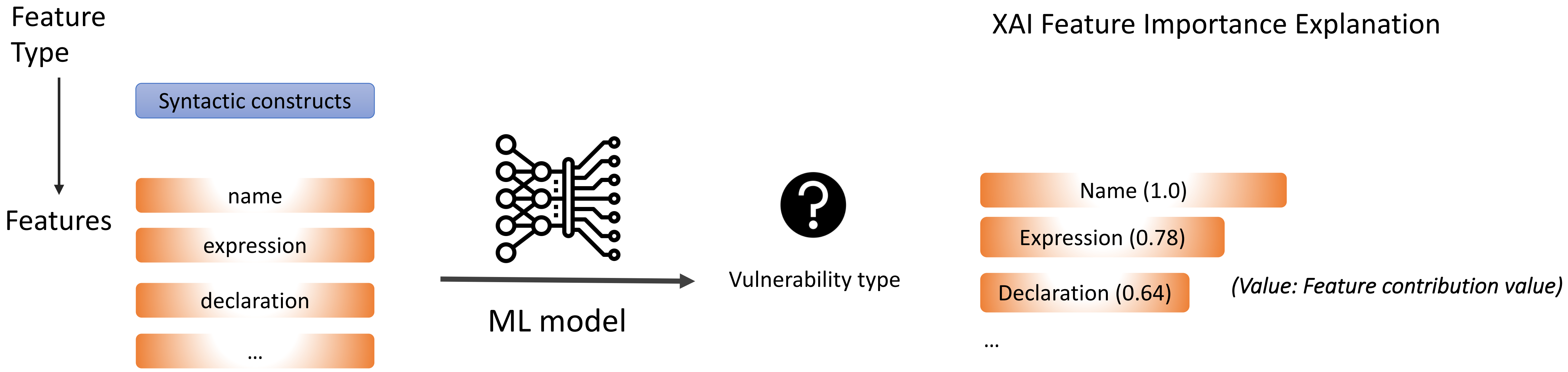


Figure2 : XAI (EXplainable AI) feature importance explanation gives the quantified results of feature's impact on model's predictions.

Presentation Overview

- Introduction
  - Background of code vulnerability
  - XAI Feature importance explanation
- Related work
  - Factors in Code Representation Techniques
- Research Questions
- Methodology
  - Text-based factors assessment
  - Graph-based factors assessment
- Experiment results (Graph-based factors assessing).
- Conclusion, Contribution, Reference, Discussion.

Related work - Factors in Code Representation Techniques

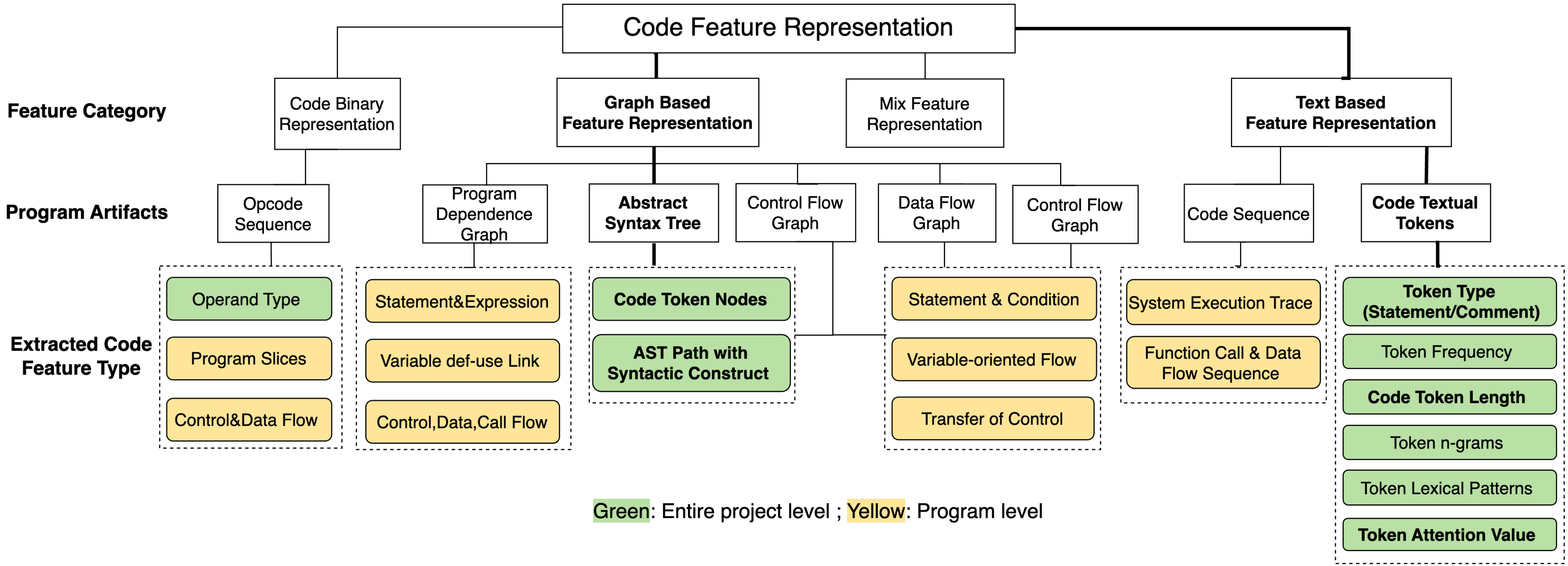


Figure 3: The taxonomy of factors under various code feature representation techniques.
our contribution: extension to the feature factor graininess from work [4].

Related work

Text-based code vulnerability detection：

- primarily focus on refining processes and improving model for higher detection accuracy, transferring knowledge from nature language process (Transformer models[5,6,7], CodeBERT[8], etc).
- Factors Explanation:
  - **Token type:** Both code body and comments matters[9]; Transformer-based model also value separator symbols (commas, etc.)[10].
  - **Token Frequency:** Preserving token frequency improves model performance [11].
  - **Token Length:** Limiting token length leads to information loss [12] (max 512 tokens in [12]).
  - **Token Attention value:**
    - In NLP task, attention values potentially indicate token importance[13], however caution is needed for this conclusion[14].
    - In code vulnerability task, attention value explanations stay at individual code snippets level[15,16] by mapping attention value, lack of cross-validation with XAI methods for representing importance.

Related work

Graph-based code vulnerability detection：

- Code Representation:
  - Abstract Syntax Tree is majority of the exiting study [4], but combinations multiple graphs based on AST become recently trend [3].
  - State-of-the-art models Code2Vec(AST) [18], GraphCodeBert(DFG)[19], Devign (Combine)[20], GraphVecCode(AST)[21].
- Factors explanation:
  - Code2Vec[18] and MIL[22]techniques provide explainability at the AST path level, suggesting the importance of paths on individual code snippets.
  - Refer to syntactic constructs, *names, identifier, and parameter* play a significant role in vulnerability tasks, as highlighted by various studies[23,24,25].
  - CWE(Common Weakness Enumeration) developed the weakness type and gather similar types into a tree structure.

Despite insights on certain crucial identifiers, a gap exists in the complete evaluation of all syntactic constructs across different vulnerability types, suggesting the need for further exploration in this area.

Research Questions

RQ1. How do measure the code textual factors influence on the performance of transformer-based models in code vulnerability detection tasks?

Text-based

Three factors: Code Token Length, Code Token Type, Code Token Attention Value

(This talk focus RQ2&3)

RQ2. How do syntactic constructs in Abstract Syntax Trees (AST) contribute to model's prediction for different software vulnerability types?

Graph-based

Aim to identify and quantify the impact of syntactic constructs linked to code vulnerabilities

RQ3. How do the CWE similarity summarized by syntactic constructs' importance explanations align with expert-defined results?

Graph-based

To evaluate the effectiveness of similarity results from XAI approach with expert-defined baseline.

Presentation Overview

- Introduction
  - Background of code vulnerability
  - XAI Feature importance explanation
- Related work
  - Factors in Code Representation Techniques
- Research Questions
- Methodology
  - Text-based factors assessment
  - Graph-based factors assessment
- Experiment results (Graph-based factors assessing).
- Conclusion, Contribution, Reference, Discussion.

Methodology Graph-based factors assessment – Syntactic constructs feature explanation (RQ2)

To answer RQ2: **How do syntactic constructs in Abstract Syntax Trees (AST) contribute to model's prediction for different software vulnerability types?**

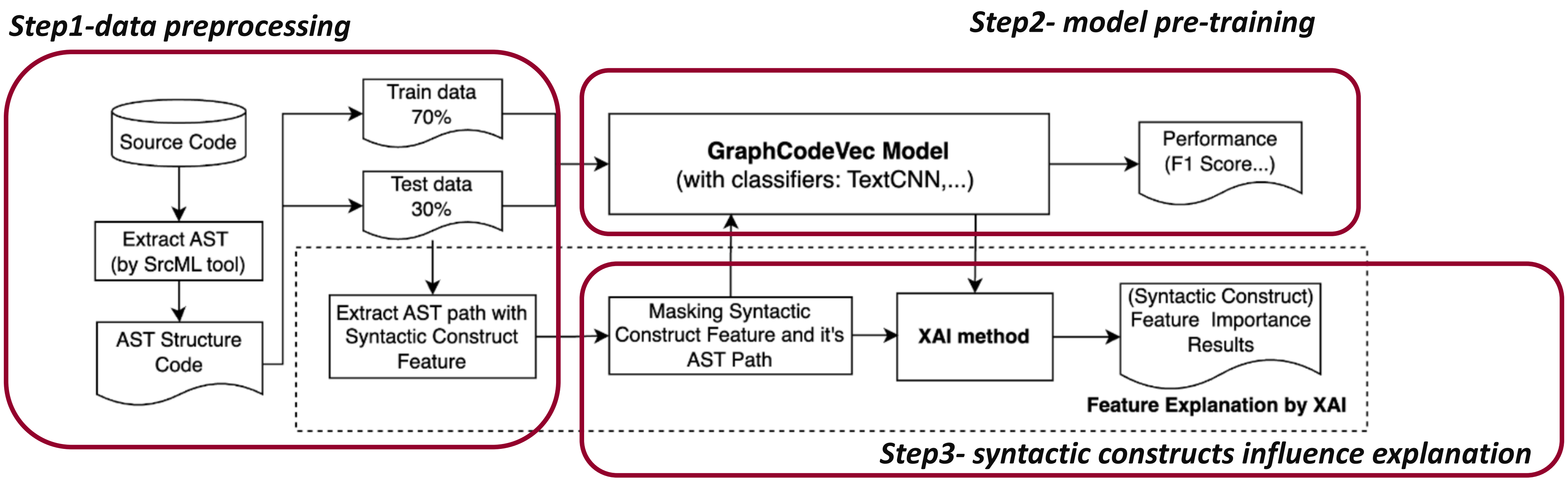


Figure 4. The overall framework of explainable syntactic constructs factors evaluation

Dataset: Juliet, OWASP, Draper benchmark projects.

GraphCodeVec[21]: novel sota model for creating a generalizable graph-based, task-agnostic code learning that leverages Graph Convolutional Networks (GCN)

XAI methods: SHAP[26], Mean-Centroid Preddiff[13].

Methodology Graph-based factors assessment -CWE Similarity (RQ3)

To answer RQ3: **How do the CWE similarity summarized by syntactic constructs' importance explanations align with expert-defined results?**

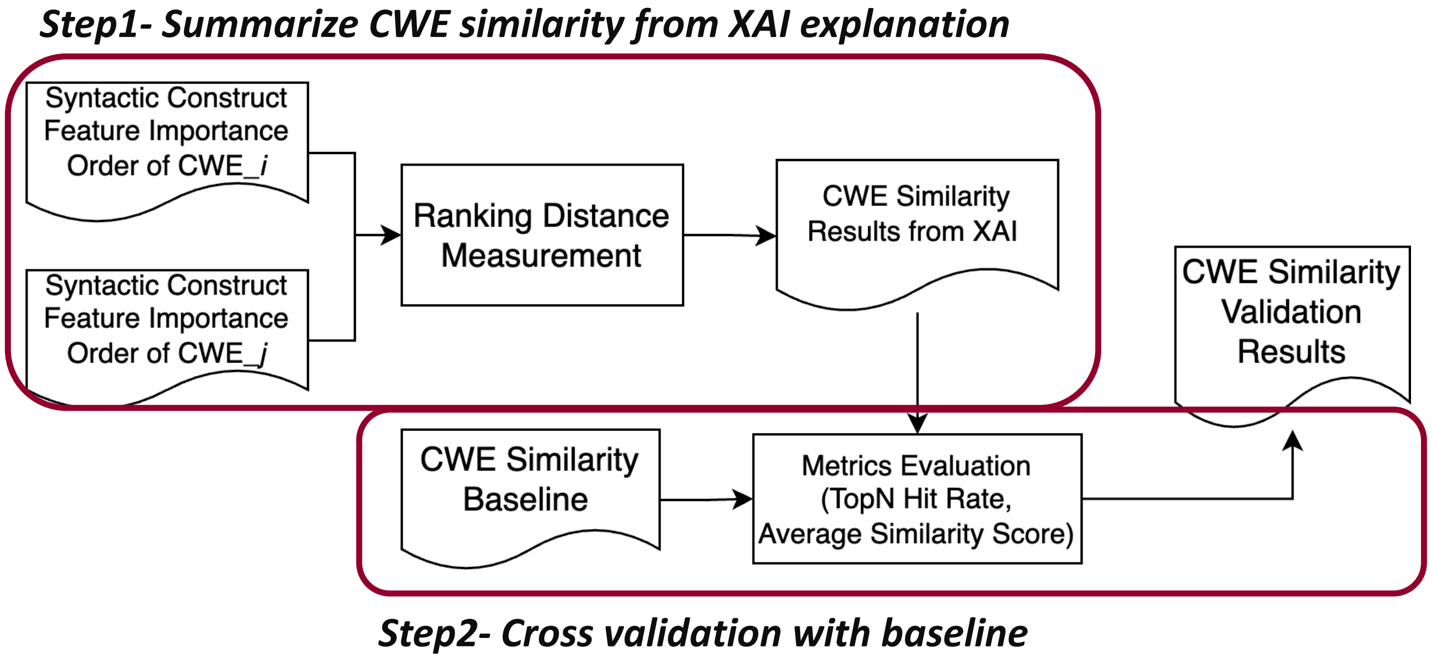


Figure 5. The overall framework of XAI summarized CWE similarity validation with baseline

Step1- Summarize CWE similarity from XAI explanation

- Given two CWEs' feature importance orders, CWE similarity value is ρ :

$$\rho(i,j) = \frac{K_\tau(Order_{CWE_i}, Order_{CWE_j})}{\max(K_\tau)}$$

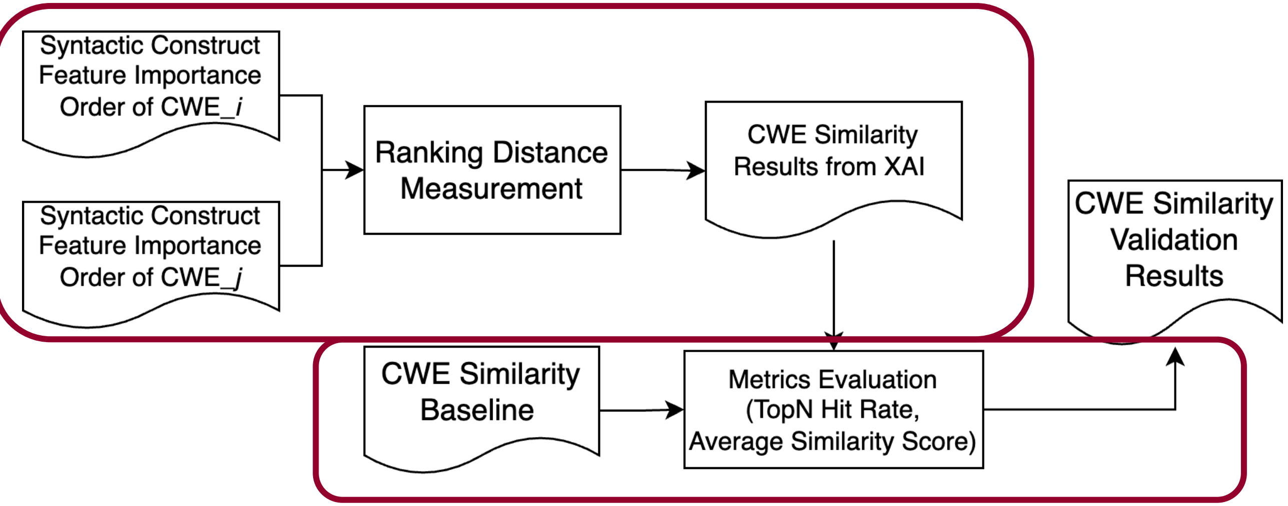
Where $K\tau$ Kendal tau ranking distance.

CWE Similarity Baseline: https://cwe.mitre.org/data/definitions/1000.html

Methodology Graph-based factors assessment -CWE Similarity (RQ3)

To answer RQ3: **How do the CWE similarity summarized by syntactic constructs' importance explanations align with expert-defined results?**

Step1- Summarize CWE similarity from XAI explanation



Step2- Cross validation with baseline

Figure 5. The overall framework of XAI summarized CWE similarity validation with baseline

Step2- Cross validation with baseline

We define three **metrics** to compare CWE similarity from our XAI approach and baseline.

- **TopN Hit Rate**: if CWE similarity pair in baseline is within the TopN similar of XAI results:

$$\text{Top-N Hit Rate}^{CWE_i} = \begin{cases} 1, & \text{if hit condition} \\ 0, & \text{otherwise} \end{cases}$$

- **Avg Similarity score** : calculates the average normalized similarity score for all CWEs within a category in the baseline table

$$\overline{S}_i = \frac{1}{|CWE_i^{similar}|} \sum \frac{\rho(i, CWE_i^{similar})}{\max \rho(i)}$$

- **Mean Reciprocal Rank** : calculates the reciprocal of the rank of the first correct answer, within the XAI ranking list.

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i}$$

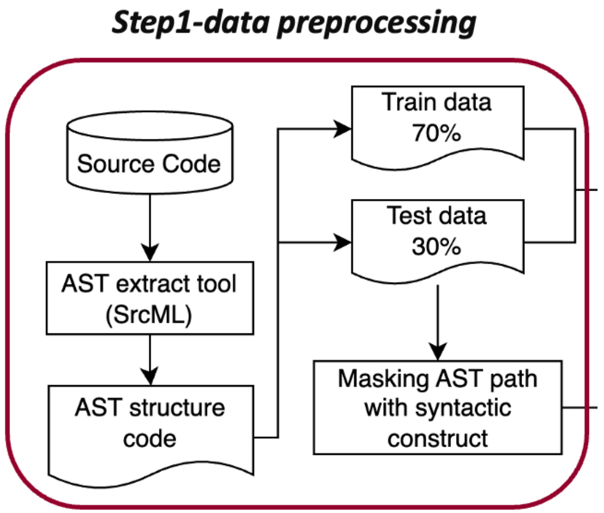
CWE Similarity Baseline: https://cwe.mitre.org/data/definitions/1000.html

Presentation Overview

- Introduction
  - Background of code vulnerability
  - XAI Feature importance explanation
- Related work
  - Factors in Code Representation Techniques
- Research Questions
- Methodology
  - Text-based factors assessment
  - Graph-based factors assessment
- Experiment results (Graph-based factors assessing).
- Conclusion, Contribution, Reference, Discussion.

Experiment Results - Syntactic constructs feature explanation (RQ2)

Step 1-data preprocessing



From Figure 4

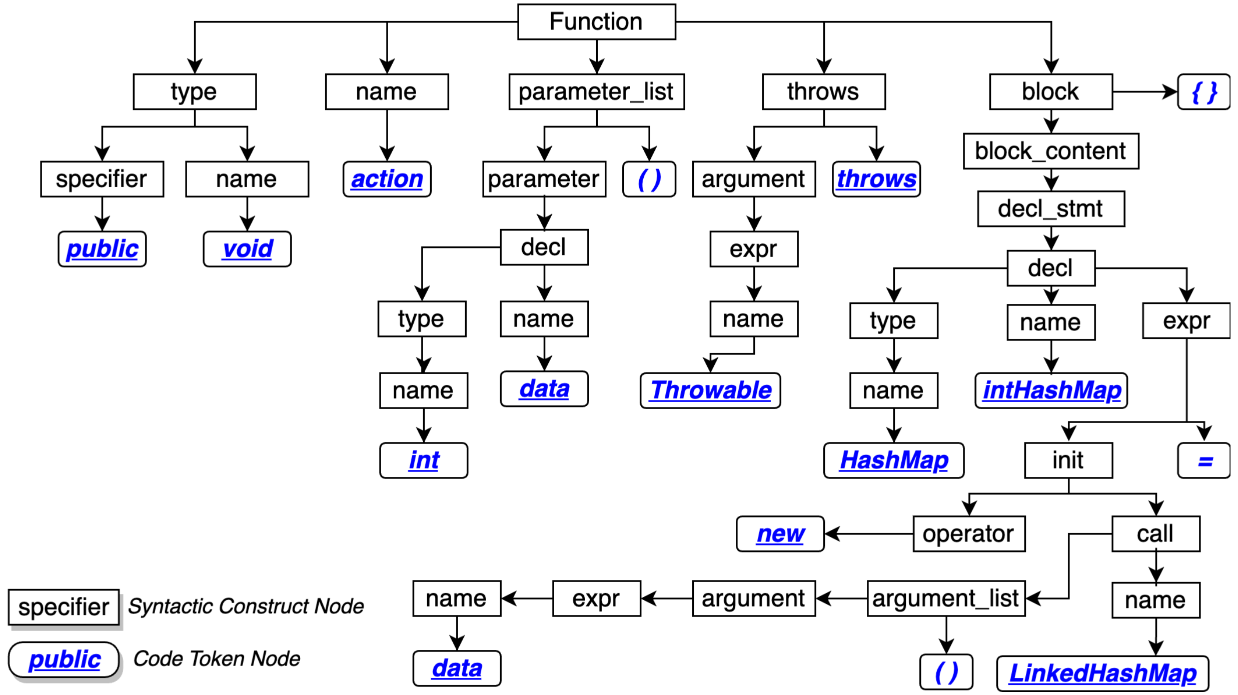


Figure 6-2. AST structure code: extract AST information of the code snippet, includes code token node, and the AST path.

```
public void action(int data) throws Throwable {
    /* POTENTIAL FLAW: Create a HashMap using data
    as the initial size. Data may be very large,
    creating memory issues */
    HashMap intHashMap = new LinkedHashMap(data);
}
```

Figure 6-1. Source Code: a code snippet of CWE789

```
#Code Tokens:
{public void action int data throws Throable HashMap
    intHashMap}
#AST Paths:
{public|void|specifier↑-type-name↓, ...
HashMap|intHashMap|name↑-type↑-decl-name↓,
LinkedHashMap|data|name↑-call-argument_list↓-
    argument↓-expr↓-name↓,
...}
```

```
# Masking AST Paths with "decl" as branch node:
{public|void|specifier↑-type-name↓, ...
***,
LinkedHashMap|data|name↑-call-argument_list↓-
    argument↓-expr↓-name↓,
...}
```

Figure 6-3. Masking AST path with syntactic construct (left unmarked, right marked)

Experiment Results - Syntactic constructs feature explanation (RQ2)

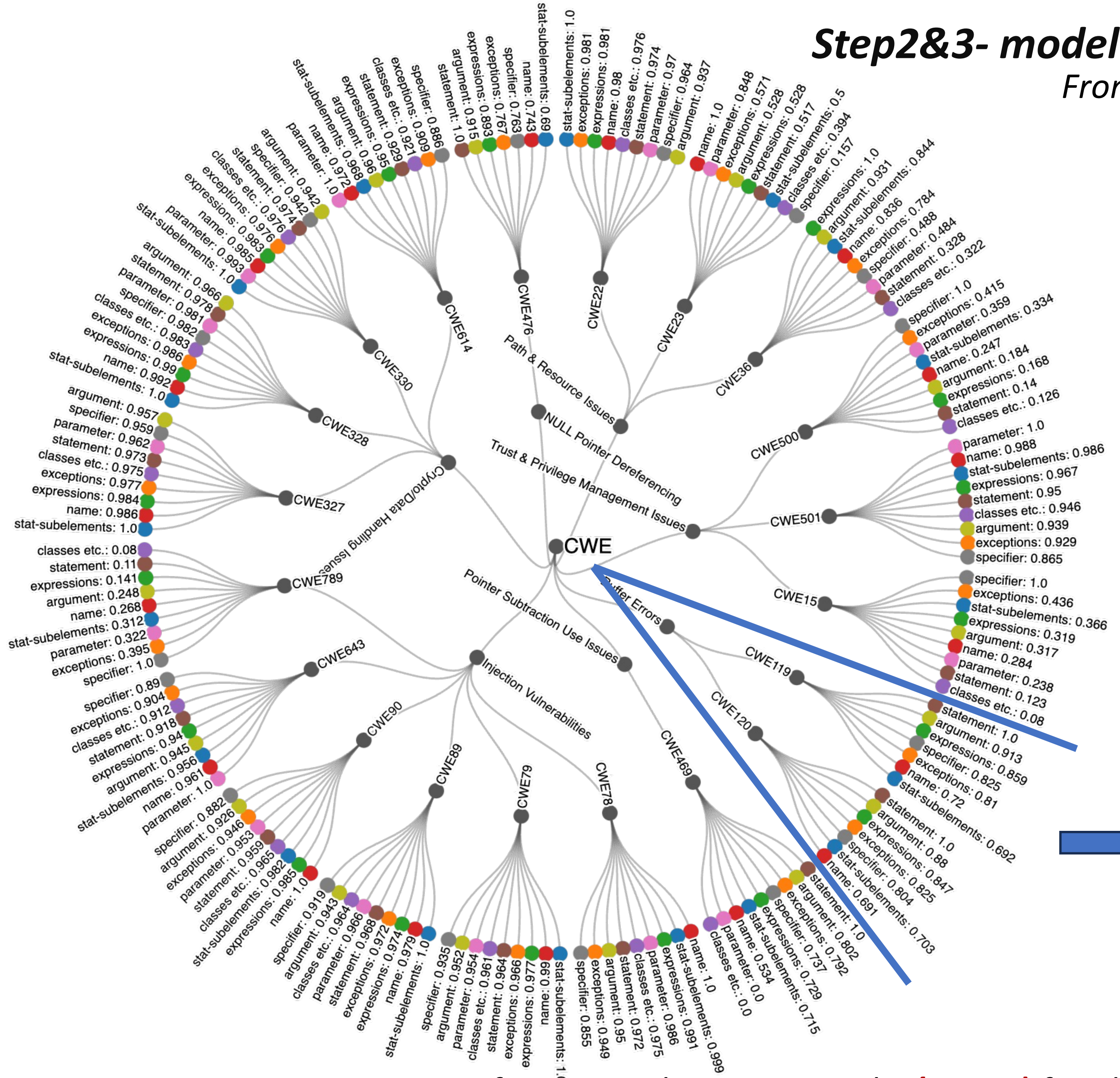
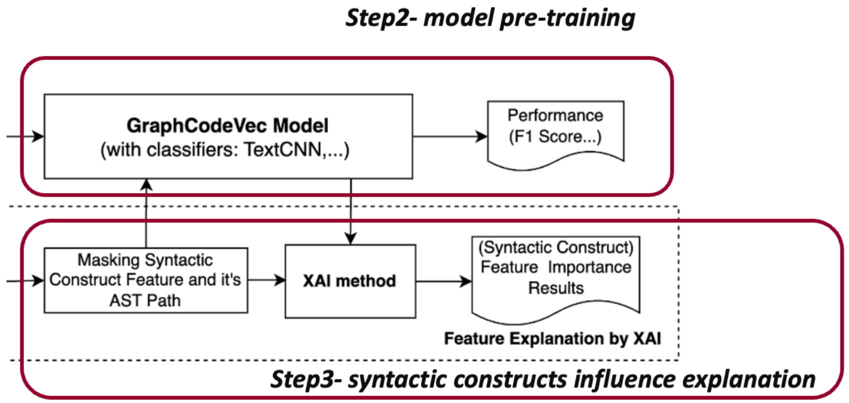


Figure 7: syntactic constructs feature explanations results **(Step 3)** for all CWEs

Step2&3- model pre-training, syntactic constructs influence explanation

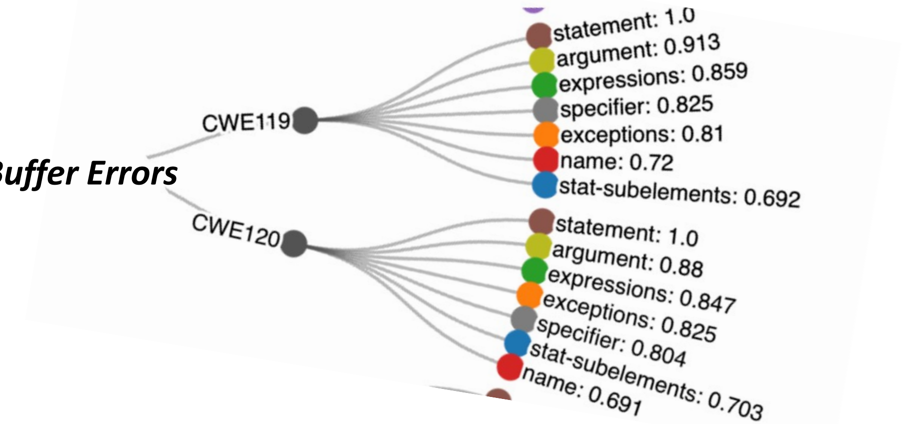
From step2, we observe GraphCodeVec + TextCNN perform consistent well.



From Figure 4

Syntactic construct features and their contribution value

Buffer Errors



Answering Research Questions

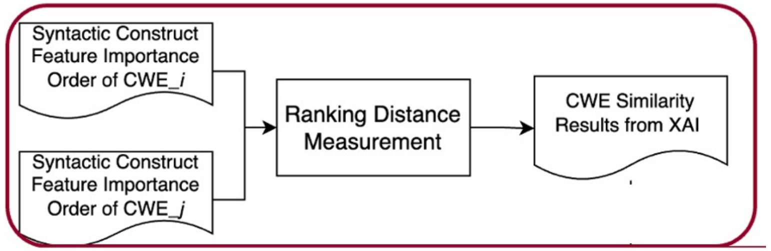
RQ2. How do syntactic constructs in Abstract Syntax Trees (AST) contribute to model's prediction for different software vulnerability types?

Aim to identify and quantify the impact of syntactic constructs linked to code vulnerabilities

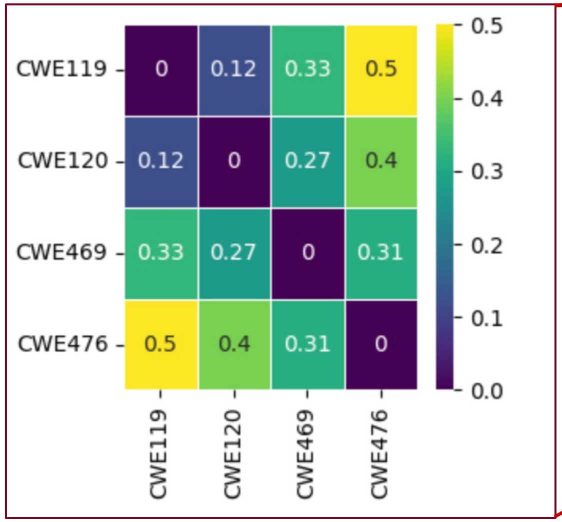
- The importance of syntactic constructs varies from CWE type, and the dataset.
- However, constructs such as statement, name, and parameters have a general high impact on code vulnerability types.
  - ✓ *Similar findings that names, identifier(statement), and parameter* play a significant role in vulnerability tasks, in studies[23,24,25]
- Several CWE type sharing high similarity based on feature importance order. (CWE 78,79, 89)
  - ✓ As a motivation of RQ3

Experiment Results - CWE Similarity (RQ3)

Step1- Summarize CWE similarity from XAI explanation



From Figure 5



- ➢ *CWE120 and CWE119 are more similar.*
- ➢ *CWE469 & CWE 476 are less similar with CWE 119&120.*

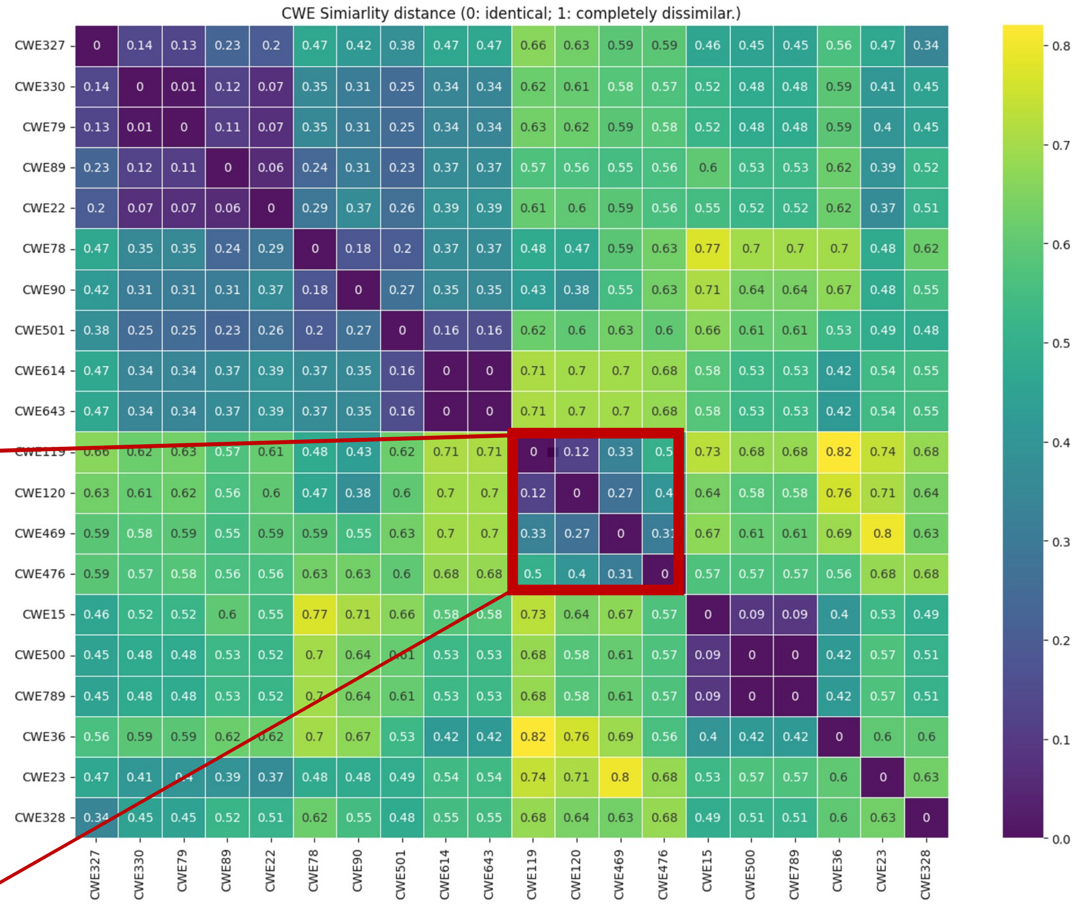
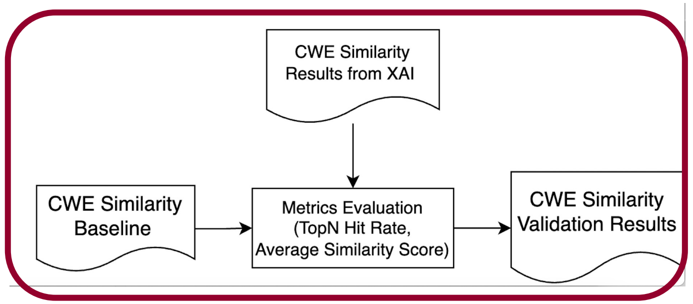


Figure 8: CWE similarity distance value from syntactic construct feature importance based on XAI approach

Experiment Results - CWE Similarity (RQ3)

Step2: CWE similarity results cross validation with baseline



From Figure 5

Table1: CWE categorized by baseline similarities

Category	Similar CWEs
Path traversal and resource management issues	CWE22, CWE23, CWE36
Trust boundaries and privilege management	CWE500, CWE501, CWE15
Buffer errors	CWE119, CWE120
Injection vulnerabilities	CWE78, CWE79, CWE89, CWE90, CWE643, CWE789
Cryptographic and sensitive data handling issues	CWE327, CWE328, CWE330, CWE614
Use of pointer subtraction to determine size	CWE469
NULL pointer dereference	CWE476

Table2: CWE Similarity Evaluation Results

TABLE VII: CWE Similarity Evaluation Results

CWE	Top-1	Top-3	Top-5	MRR	$\overline{S}$
CWE23	1	1	1	0.572	0.802
CWE327	1	1	1	0.736	0.628
CWE330	1	1	1	0.728	0.247
CWE79	1	1	1	0.728	0.250
CWE89	0	1	1	0.630	0.328
CWE22	0	0	0	0.115	0.118
CWE78	1	1	1	0.687	0.622
CWE90	1	1	1	0.743	0.610
CWE501	1	1	1	0.767	0.774
CWE614	1	1	1	0.738	0.761
CWE643	1	1	1	0.738	0.761
CWE328	0	0	1	0.233	0.620
CWE36	0	0	0	0.122	0.661
CWE15	1	1	1	1	1
CWE500	1	1	1	1	1
CWE789	1	1	1	1	1
CWE469	-	-	-	-	-
CWE476	-	-	-	-	-
CWE119	1	1	1	1	1
CWE120	1	1	1	1	1
Average	**0.778**	**0.833**	**0.889**	**0.696**	**0.677**

Note: Top-1/3/5 represents the Top-N Hit rate, MRR represents Mean Reciprocal Rank, and $\overline{S}$ represents the Average Normalized Similarity Score. CWE469 and CWE476 do not have a similar CWE in the datasets scope.

- Our CWE similarity summary from XAI effectively align with baseline with 77.8% Top1 Hit rate.

Answering Research Questions

RQ3. How do the CWE similarity summarized by syntactic constructs' importance explanations align with expert-defined results?

To evaluate the effectiveness of similarity results from XAI approach with expert-defined baseline.

- Our CWE similarity evaluation method efficiently identifies related CWEs, achieving a hit rate of 77.8% for the most similar CWE (Top-1) and 88.9% for the top five similar CWEs (Top-5).
- In our evaluation, only two instances - CWE22 and CWE36 (2 out of 20) did not meet the baseline similarities.

Presentation Overview

- Introduction
  - Background of code vulnerability
  - XAI Feature importance explanation
- Related work
  - Factors in Code Representation Techniques
- Research Questions
- Methodology
  - Text-based factors assessment
  - Graph-based factors assessment
- Experiment results (Graph-based factors assessing).
- Conclusion Contribution, Reference, Discussion.

Conclusion Contribution

- ✓ We extend the taxonomy of code representation techniques by examining them at the feature factor level.
- ✓ Our study provides a comprehensive evaluation of the importance of all syntactic constructs, complementing previous studies that focused only on top-valued constructs.
- ✓ By leveraging rankings of syntactic constructs, we effectively analyze and validate CWE similarity, comparing our results to expert-defined baselines to confirm the effectiveness of our XAI explanation approach.

Reference & Discussion

1. N. I. of Standards and Technology. "Vulnerability Definition". Computer Security Resource Center. Online at: csrc.nist.gov/glossary/term/vulnerability.

2. Huang, J., Wang, Z., Li, D., Liu, Y. (2022). "The Analysis and Development of an XAI Process on Feature Contribution Explanation". In: IEEE International Conference on Big Data, Osaka, Japan. Pages 5039-5048. DOI: 10.1109/BigData55660.2022.10020313.

3. Lin, G., Wen, S., Han, Q.-L., Zhang, J., Xiang, Y. (2020). "Software Vulnerability Detection Using Deep Neural Networks: A Survey". In: Proceedings of the IEEE, Vol. 108, No. 10, Pages 1825–1848.

4. Hanif, Hazim, et al. "The rise of software vulnerability: Taxonomy of software vulnerabilities detection and machine learning approaches." Journal of Network and Computer Applications 179 (2021): 103009.

5. Bojanowski, P., Grave, E., Joulin, A., Mikolov, T. (2017). "Enriching Word Vectors with Subword Information". In: Transactions of the Association for Computational Linguistics, Vol. 5, Pages 135–146.

6. Svyatkovskiy, A., Zaytsev, V., Sundaresan, N. (2019). "Semantic Source Code Models Using Identifier Embeddings". In: IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER). Pages 554–565.

7. Loyola, P., Matzger, B., Schiele, G. (2019). "Import2vec Learning Embeddings for Software Libraries". In: 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE). Pages 1106–1108.

8. Feng, Zhangyin, et al. "Codebert: A pre-trained model for programming and natural languages." arXiv preprint arXiv:2002.08155 (2020).

9. Feng, Z., et al. (2020). "CodeBERT: A Pre-trained Model for Programming and Natural Languages". Online at: arXiv preprint arXiv:2002.08155.

10. Li, D., Liu, Y., Huang, J., Wang, Z. (2023). “A Trustworthy View on Explainable Artificial Intelligence Method Evaluation”. In: Computer, Vol. 56, No. 4. Pages 50–60.

11. Sharma, R., Chen, F., Fard, F., Lo, D. (2022). “An Exploratory Study on Code Attention in BERT”. In: Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension. Pages 437–448.

12. Zheng, W., Gao, J., Wu, X., Xun, Y., Liu, G., Chen, X. (2020). “An Empirical Study of High-Impact Factors for Machine Learning-based Vulnerability Detection”. In: 2020 IEEE 2nd International Workshop on Intelligent Bug Fixing (IBF). Pages 26–34.

13. Yuan, X., Lin, G., Tai, Y., Zhang, J. (2022). “Deep Neural Embedding for Software Vulnerability Discovery: Comparison and Optimization”. In: Security and Communication Networks, Vol. 2022. Pages 1–12.

14. Vashishth, S., Upadhyay, S., Tomar, G. S., Faruqui, M. (2019). “Attention Interpretability across NLP Tasks”. Online at: arXiv preprint arXiv:1909.11218.

15. Jain, S., Wallace, B. C. (2019). “Attention is Not Explanation”. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers). Pages 3543–3556.

16. Mao, Y., Li, Y., Sun, J., Chen, Y. (2020). “Explainable Software Vulnerability Detection Based on Attention-based Bidirectional Recurrent Neural Networks”. In: 2020 IEEE International Conference on Big Data (Big Data). Pages 4651–4656.

17. Duan, X., Wu, J., Ji, S., Rui, Z., Luo, T., Yang, M., Wu, Y. (2019). “VulSniper: Focus Your Attention to Shoot Fine-grained Vulnerabilities”. In: IJCAI. Pages 4665–4671.

18. Alon, U., et al. (2019). "Code2Vec: Learning Distributed Representations of Code". In: Proceedings of the ACM on Programming Languages 3.POPL. Pages 1-29.

19. Guo, D., et al. (2020). "GraphCodeBERT: Pre-training Code Representations with Data Flow". Online at: arXiv preprint arXiv:2009.08366.

20. Zhou, Y., et al. (2019). "Devign: Effective Vulnerability Identification by Learning Comprehensive Program Semantics via Graph Neural Networks". In: Advances in Neural Information Processing Systems 32.

21. Ding, Z., et al. (2023). "Towards Learning Generalizable Code Embeddings using Task-agnostic Graph Convolutional Networks". In: ACM Transactions on Software Engineering and Methodology 32.2. Pages 1-43.

22. Hariharan, M., et al. (2022). "Proximal Instance Aggregator Networks for Explainable Security Vulnerability Detection". In: Future Generation Computer Systems, Vol. 134. Pages 303-318.

23. Applis, L., Panichella, A., van Deursen, A. (2021). “Assessing Robustness of ML-based Program Analysis Tools Using Metamorphic Program Transformations”. In: 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE). Pages 1377–1381.

24. Rabin, M. R. I., Bui, N. D., Wang, K., Yu, Y., Jiang, L., Alipour, M. A. (2021). “On the Generalizability of Neural Program Models with Respect to Semantic-Preserving Program Transformations”. In: Information and Software Technology, Vol. 135. Article No. 106552.

25. Yang, Z., Shi, J., He, J., Lo, D. (2022). “Natural Attack for Pre-trained Models of Code”. In: Proceedings of the 44th International Conference on Software Engineering. Pages 1482–1493.

26. Lundberg, S. M., Lee, S.-I. (2017). “A Unified Approach to Interpreting Model Predictions”. In: Advances in Neural Information Processing Systems, Vol. 30.

Appendix 1 - Syntactic constructs feature explanation (RQ2)

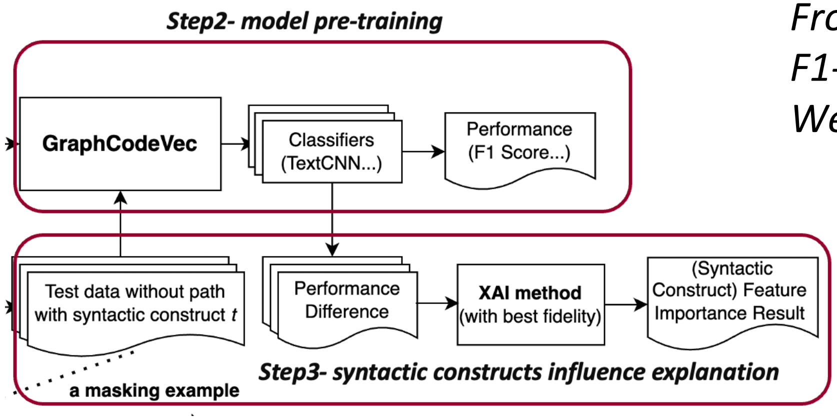
Syntactic Constructs and Categories in the software

TABLE I: Syntactic Constructs in Abstract Syntax Tree

Meta Syntactic Constructs [84]	Syntactic Constructs
Name, Base Elements	<name>, <block_comment>, <literal>,...
Statements	<assert>,<block>,<break>,<case>, <if_stmt>,<continue>,<default>,<do>, <empty_stmt>,<expr_stmt>,<for>, <if_stmt>,<label>,<return>,<switch>, <while>,<default>,...
Statement subelements	<expr>,<condition>,<control>,<else>, <iftype="elseif">,<expr>,<if>,<incr>, <then>,<type> ,<block_content>,...
Specifiers	<specifier>,...
Declarations, Definitions, Initializations	<lambda>,<function>,<decl_stmt>, <decl>,<init> ,<new>,...
Classes, Interfaces, Annotations, and Enums	<annotation>,<class>,<static>, <annotation_defn>, ...
Expressions	<call>, ...
Exceptions	<finally>,<throw>,<throws>,<try> ,<catch>,...

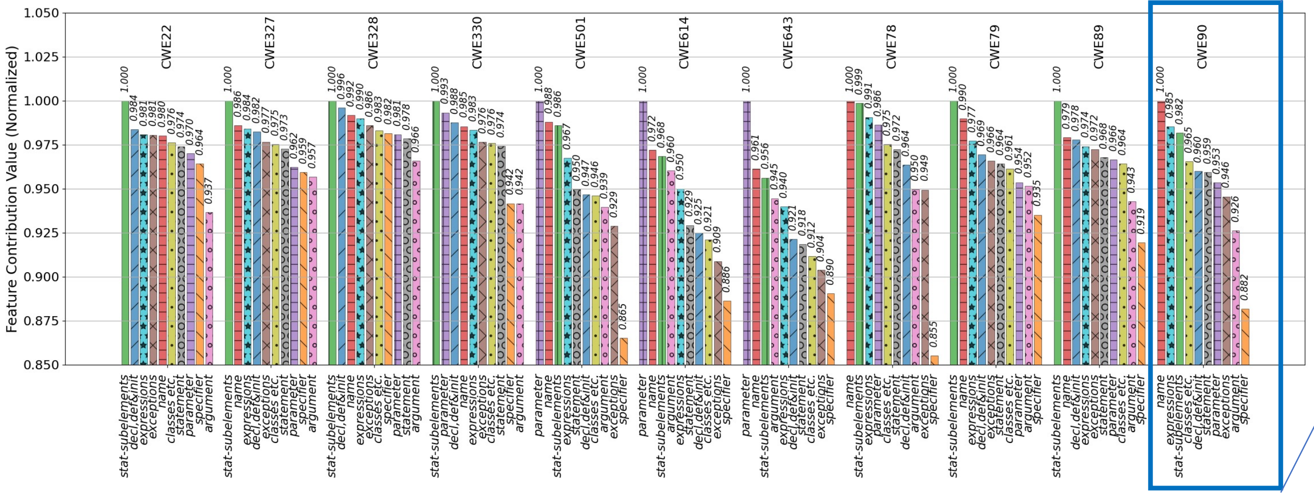
Appendix 2 - Syntactic constructs feature explanation (RQ2)

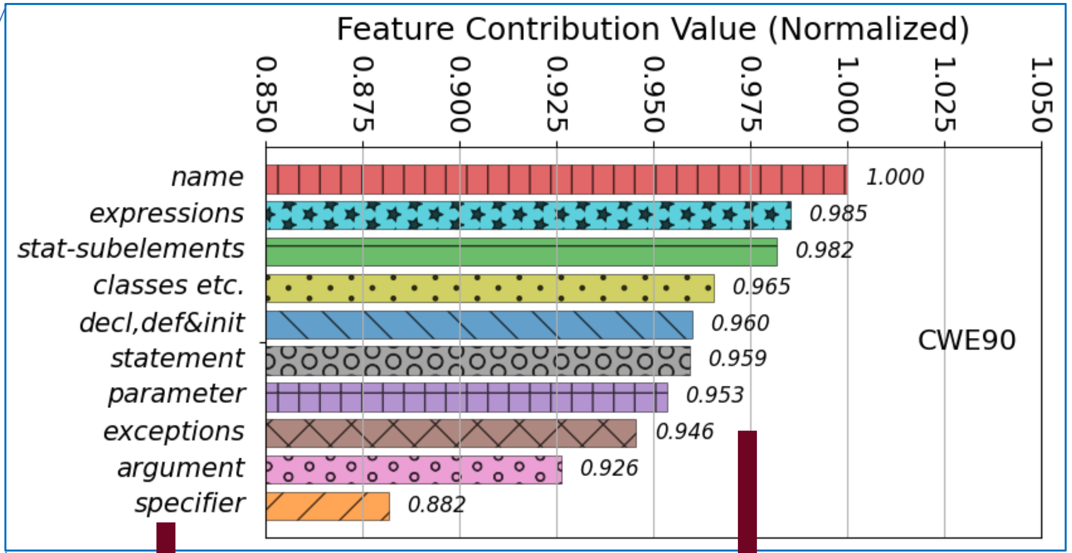
Step2&3- model pre-training, syntactic constructs influence explanation



From Figure 4

From step2, we observe GraphCodeVec + TextCNN perform consistent well (88.4%, 89.9% on F1-Score for Juliet, Draper dataset) than GraphCodeVec + Random Forest or Transformer. We preform XAI based on GraphCodeVec + TextCNN.





Meta syntactic construct features and their feature importance order

Related Feature Contribution Value

*Figure 7: Feature explanations results **(Step 3)** for CWE from OWASP dataset.*