Compact Abstract Graphs for Detecting Code Vulnerability with GNN Models

Luo Yu
Computer Science Electrical Engineering
University of Missouri-Kansas City
Kansas City, USA

Weifeng Xu
School of Criminal Justice
The University of Baltimore
Baltimore, USA

Dianxiang Xu
Computer Science Electrical Engineering
University of Missouri-Kansas City
Kansas City, USA

ACSAC 2022
December 5-9, 2022 · Austin, Texas, USA

UMKC

Agenda

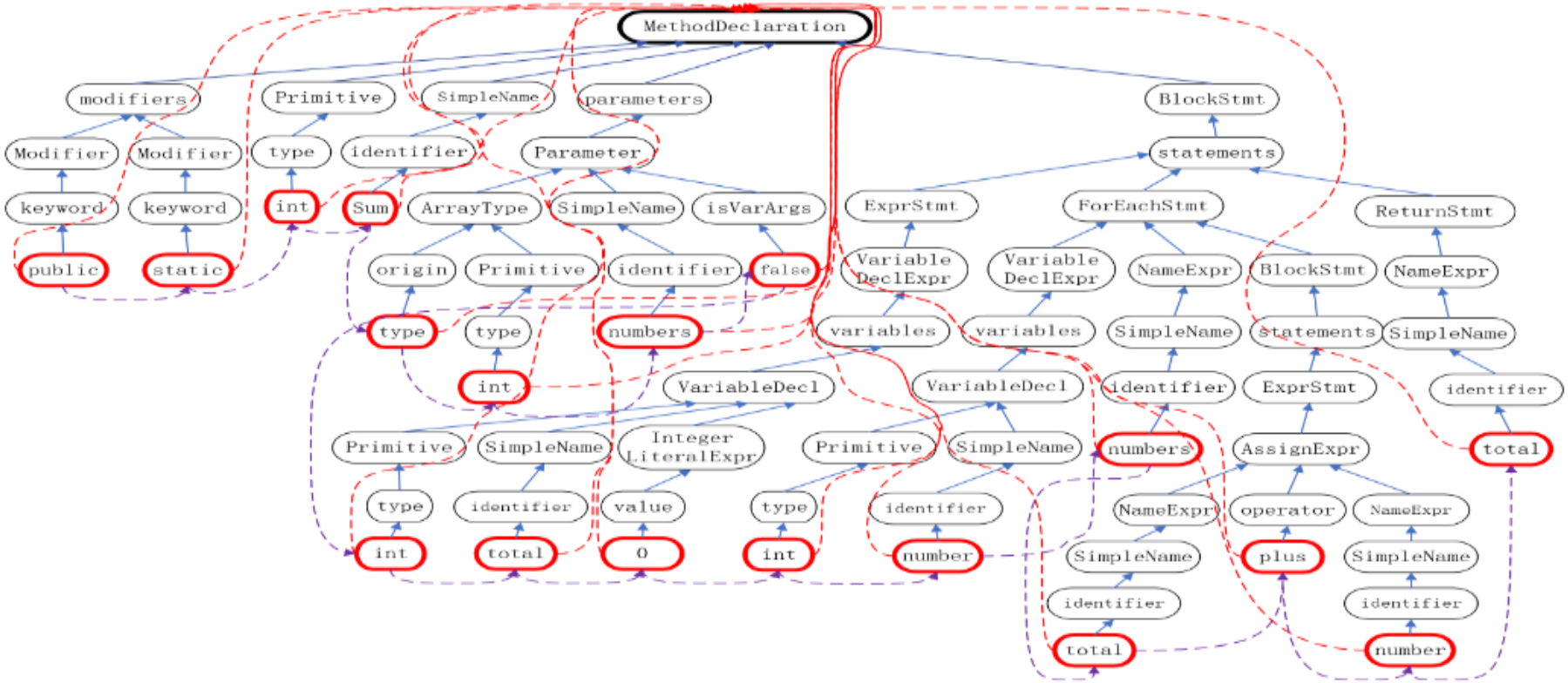
- Goals
- Abstract Graph (AG)
- Compact Abstract Graph (CAG)
- Framework
- Experiments
- Conclusion

Goals

To address the practical issue of source code representation for exploiting the state-of-the-art GNN models to detect a broad range of code vulnerabilities. We propose Compact Abstract Graphs (CAGs) of source code in different programming languages (e.g., Java and C) for efficient vulnerability prediction with various GNN models.

Abstract Graphs

```
public static int sum(int[] numbers) {
    int total = 0;
    for (int number : numbers) {
        total += number;
    }
    return total;
}
```



Compact Abstract Graphs (CAG)

Compress AG by two steps:

- Merging Single-Entry Node Sequences
- Merging Aggregation Structures

Merging Single-Entry Node Sequences

Formally, given abstract graph $\langle N, E, s\rangle$ and a node sequence $\langle n1, n2, ...nk\ \rangle$, merging the sequence results in a new abstract graph $\langle N', E', s\rangle$ such that N' = N \{$n1, n2, ...nk$ } ∪ {n} and E' = E \{($n1, n2$), ($n2, n3$), ..., ($nk-1, nk$), (α, $n1$), (nk, ω)} } ∪ {(α, n), (n, ω)}, where n is the new merged node, (α, $n1$) ∈ E, (nk , ω) ∈ E.

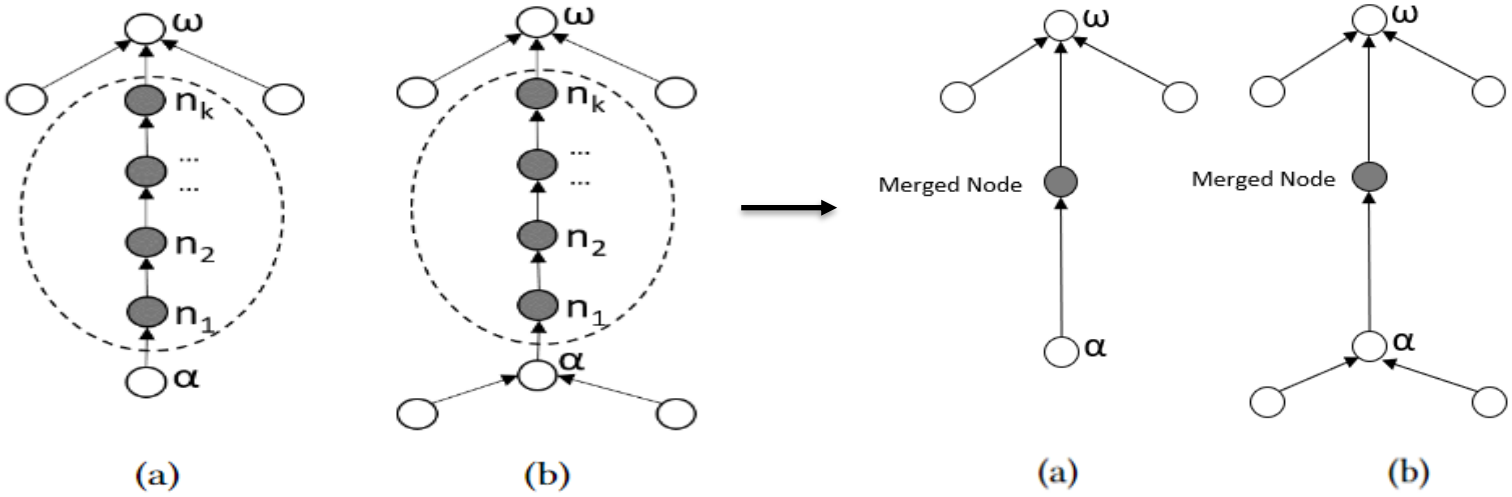


Figure 2: Patterns of Single-Entry Node Sequences

After Merging

Merging Single-Entry Node Sequences

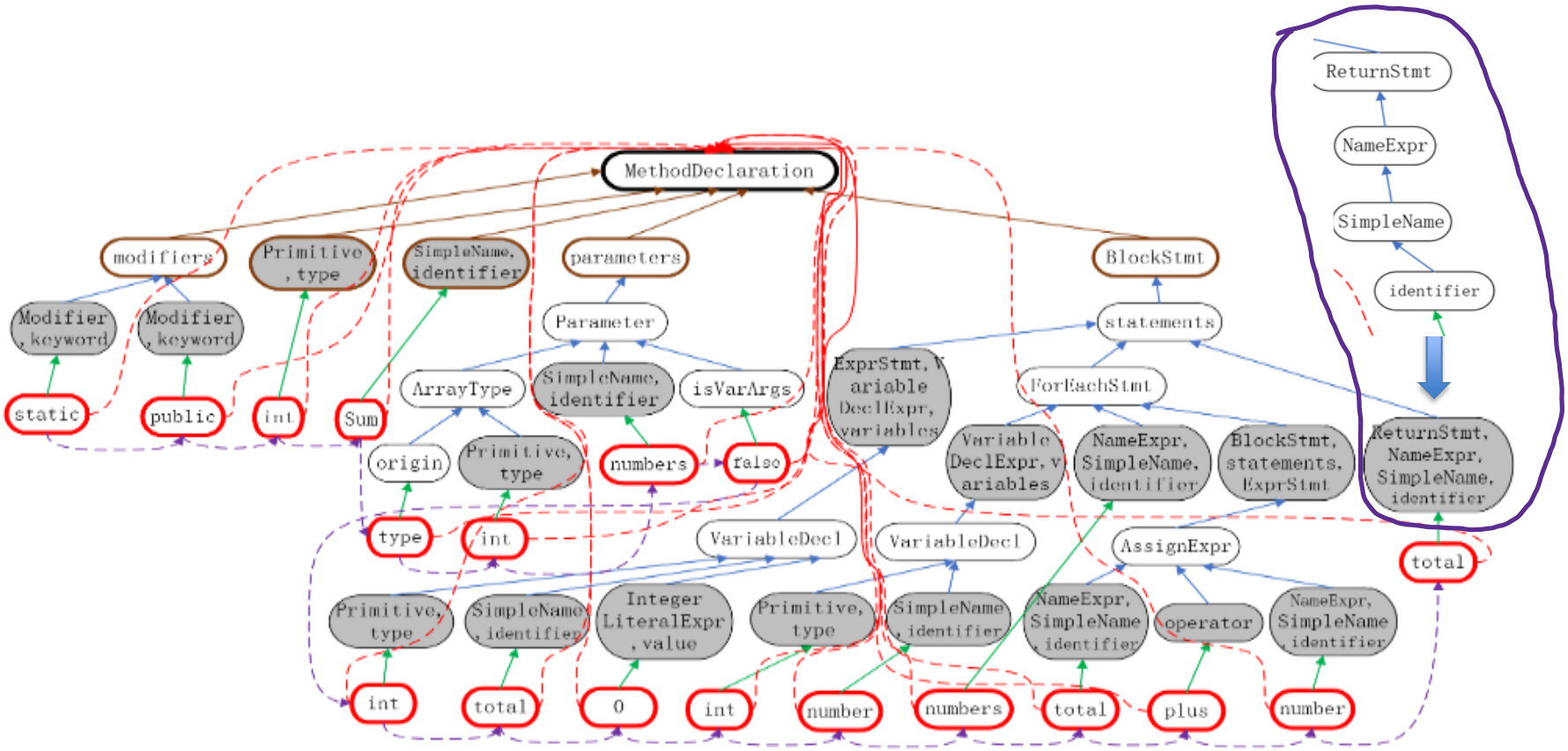


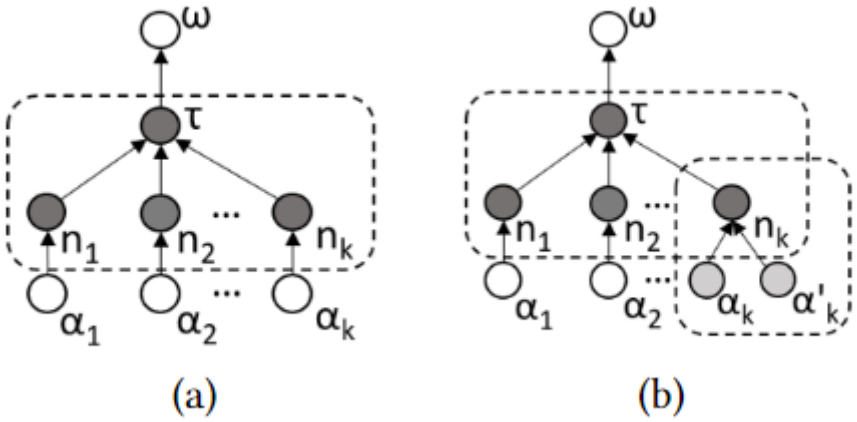
Figure 3: The AG after Merging Single-Entry Node Sequences

Merging Aggregation Structures

Whether an aggregation structure can be compressed depends on two conditions:

- Each child node ni in the structure has exactly one entry edge. If any child node has two or more entry edges, we cannot merge the aggregation structure; Otherwise, it would lose structural information.

- The structure represents the part-whole relation of a programming construct.



Merging Aggregation Structures

Table 1: Categories of Aggregation Structures

Category	Label of the Parent Node
Expression	MethodCallExpr, FieldAccessExpr, BinaryExpr, ObjectCreationExpr, AssignExpr, ArrayCreation -Expr, CastExpr, ArrayAccessExpr, IntegerLiteral -Expr, UnaryExpr, InstanceOfExpr, SingleMember -AnnotationExpr, VariableDeclarationExpr, ConditionalExpr
Statement	statements, IfStmt, CatchClause, TryStmt, SwitchEntry, ForStmt, WhileStmt, catchClauses, entries, SwitchStmt, ExpressionStmt, ForEachStmt, LabeledStmt, DoStmt, AssertStmt, ThrowStmt, ReturnStmt, thrownExceptions, SynchronizedStmt
Declaration	VariableDeclarator, variables, values
Argument	arguments, typeArguments
Parameter	Parameter, parameters, typeParameters, TypeParameter
Type	ClassOrInterfaceType, ArrayType
Modifier	modifiers

Merging Aggregation Structures

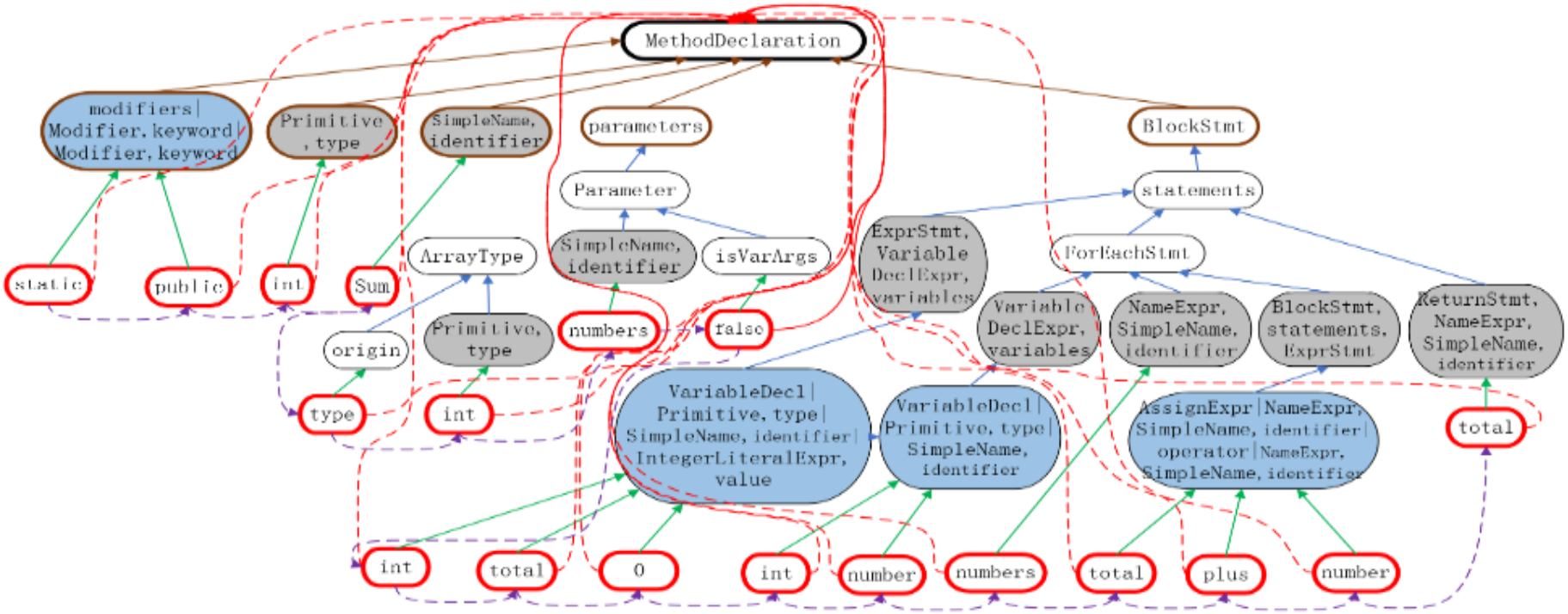


Figure 5: A Sample CAG

Framework

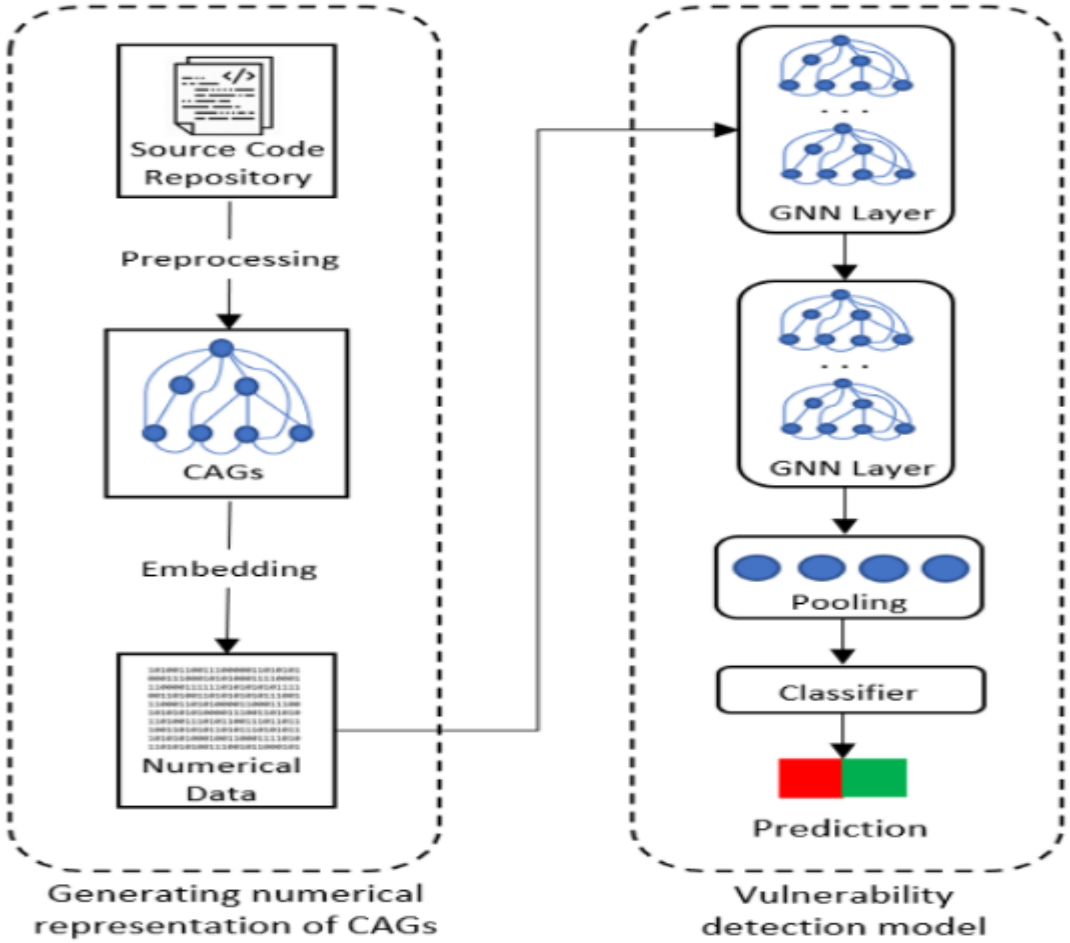


Figure 6: Framework for GNN-based Vulnerability Detection

GNN Models

- Graph Convolutional Networks (GCNs)
- Graph Attention Networks (GATs)
- Unified Message Passing Model (UniMP)
- GNNs with autoregressive moving average filter (ARMAConv)
- Residual Gated Graph ConvNets (ResGatedGCNs)
- Feature-Steered Graph Convolutions (FeaStNet)

Dataset

Table 2: The Datasets

Dataset	Language	#Vul. Types	#Positive Samples	#Negative Samples
Java Dataset	Java	114	37,350	68,480
C Dataset	C	106	58,459	126,170

Experiment

Table 3: Results of the Java Dataset (%)

GNN Model	Accuracy	Precision	Recall	F1
FeaStNet	94.70	94.14	93.97	94.05
ARMAConv	95.01	95.35	94.02	94.14
RGGCNs	95.05	95.88	93.99	94.84
GCNs	95.30	95.60	94.22	94.91
GATs	96.27	96.50	95.04	95.84
UniMP	**96.33**	**96.67**	**95.28**	**96.08**

Table 4: Experiment Results of the C Dataset (%)

GNN Model	Accuracy	Precision	Recall	F1
FeaStNet	91.55	90.77	92.14	91.29
ARMAConv	92.01	90.71	92.75	91.78
RGGCNs	92.49	90.79	92.98	91.91
GCNs	92.50	90.92	92.61	91.93
GATs	93.14	**92.51**	93.43	92.89
UniMP	**93.21**	92.48	**93.47**	**92.90**

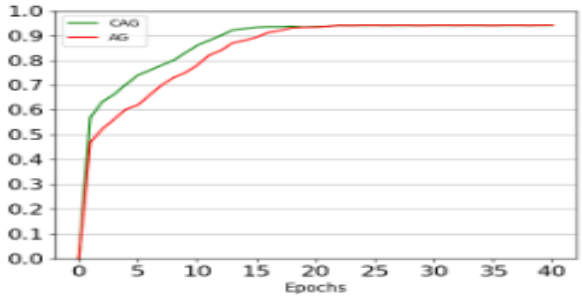
Table 5: Results of the Combined Java/C Dataset (%)

GNN Model	Accuracy	Precision	Recall	F1
FeaStNet	92.59	91.39	93.00	91.68
ARMAConv	92.75	91.72	92.05	91.88
RGGCNs	93.54	91.83	94.59	92.38
GCNs	93.49	91.39	93.74	92.43
GATs	94.07	92.28	94.72	93.39
UniMP	**94.09**	**92.29**	**94.77**	**93.40**

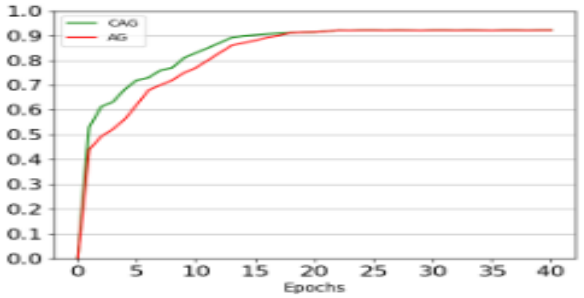
Effectiveness of Graph Reduction

Table 6: Compression Ratios

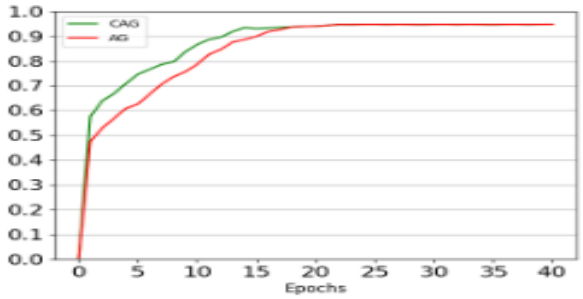
Graph Representation	Node #	Node Reduction	Edge #	Edge Reduction	Time (min/10Epoch)	Training Time Reduction	Storage Size (GB)
AGs	23,186,375	-	31,325,795	-	31.0	-	422
CAGs	10,746,584	53.65%	18,885,202	39.71%	22.5	27.42%	190.3



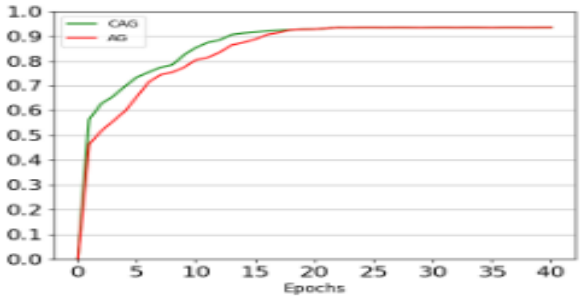
(a) Accuracy



(b) Precision

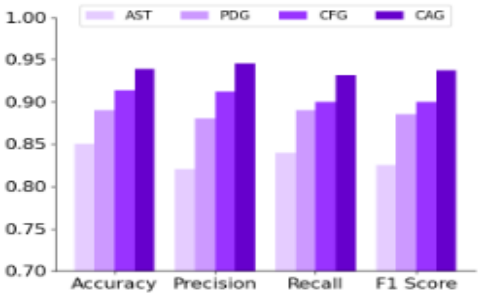


(c) Recall

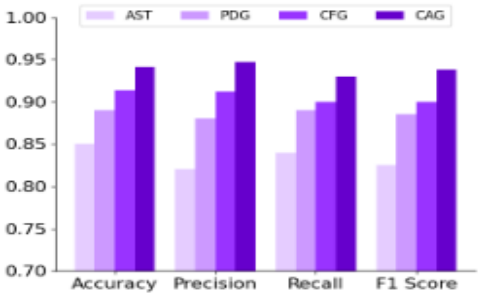


(d) F1 Score

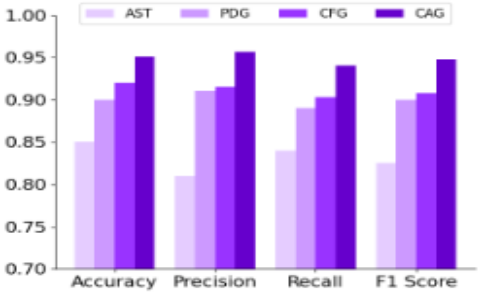
Comparison with ASTs, CFGs and PDGs



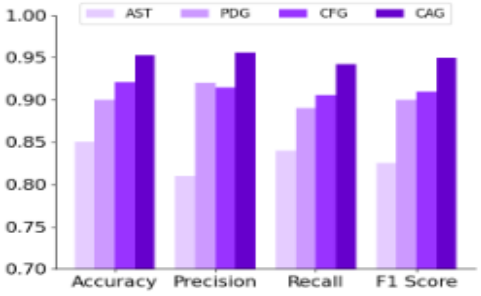
(a) FeaStNet



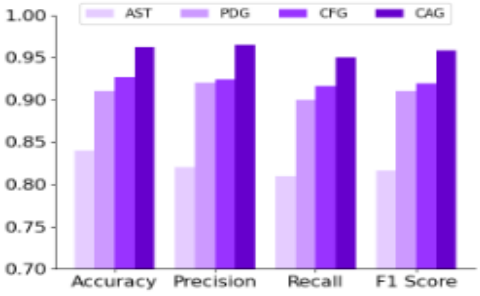
(b) ARMAConv



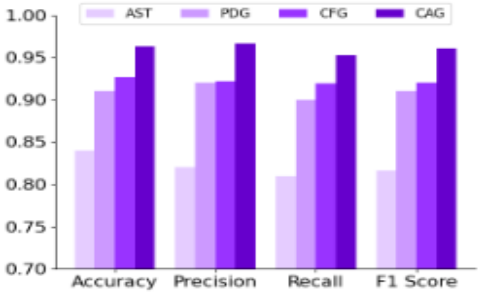
(c) RGGCNs



(d) GCNs



(e) GATs



(f) UniMP

Comparison with the Related Works

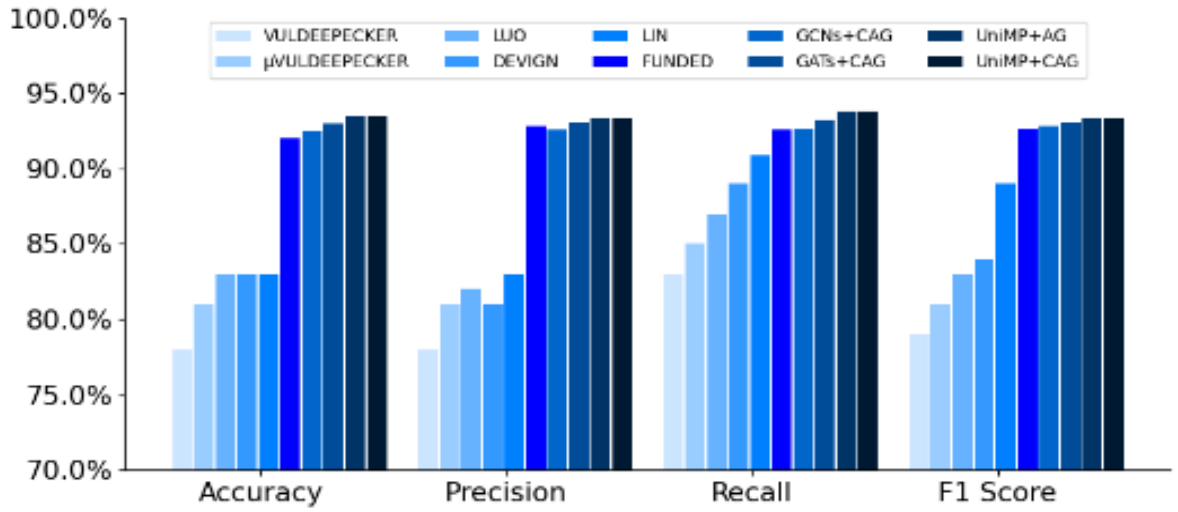


Figure 10: Comparison Result on TIFS Dataset

Comparison with the Related Works

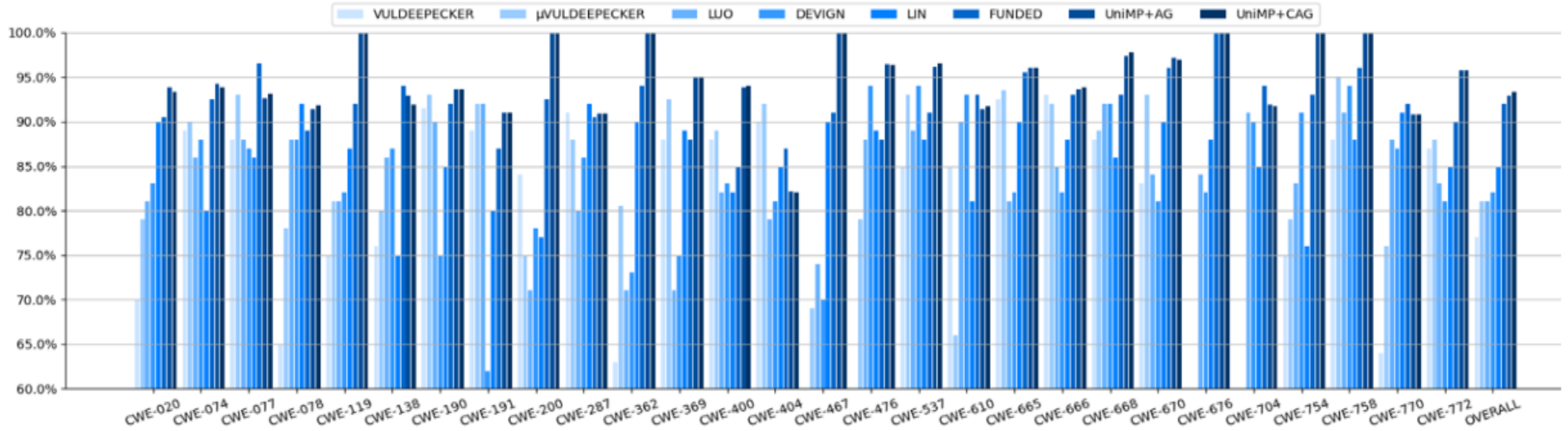


Figure 9: Comparison of Individual Vulnerabilities in the TIFS Dataset

Applications

Table 7: The Applications

Project	Lan.	KLOC	Classes	Methods	#Vul.
Apache JMeter	Java	122	1,921	9,306	160
Elasticsearch	Java	366	4,836	26,646	405
FFmpeg	C	615	3,478	36,505	627
OpenSSL	C	361	2,203	19,922	630
GitHub	J&C	-	-	1,506	1,506
Total	-	1,464	12,438	93,132	2,575

GitHub refers to historical routines extracted from GitHub projects

Table 8: Prediction Results for Real-World Applications (%)

GNN Model	A	P	R	F1	FPR	FNR
FeaStNet	93.3	87.4	74.6	80.5	2.5	25.4
ARMAConv	93.3	91.1	70.7	79.6	1.6	29.3
RGGCNs	93.3	90.3	70.8	79.4	1.7	29.2
GCNs	95.1	96.9	76.0	85.2	0.6	24.0
GATs	95.3	94.6	79.2	86.2	1.0	20.8
UniMP	96.9	94.3	88.5	91.3	1.2	11.5

Conclusion

- Presented CAGs as a novel source code representation for predicting software vulnerabilities.
- Using Java and C datasets with 220 types of vulnerabilities have demonstrated that CAGs are much more efficient than ASTs, CFGs, and PDGs.
- Applied to more than 2,500 vulnerabilities collected from real-world software projects.

Thank You