UNSW SYDNEY

The work was published at ISSTA 2022, Daejeon, South Korea

Path-Sensitive Code Embedding via Contrastive Learning for Software Vulnerability Detection

Yulei Sui
University of New South Wales, Australia

joint work with Xiao Cheng, Guanqin Zhang, Haoyu Wang,

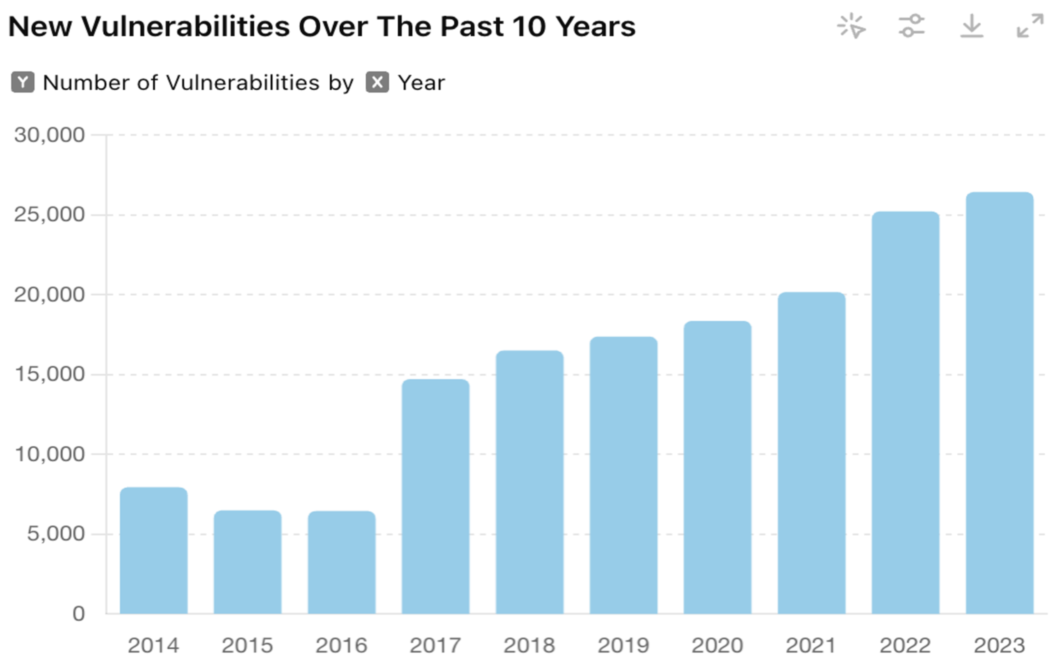
Contribution

A new **path-sensitive code embedding** utilizing

- precise **path-sensitive value-flow analysis**
- a **pretrained value-flow path encoder** via self-supervised **contrastive learning**

to significantly **boost the performance** and **reduce the training costs** of later **path-based prediction** models to precisely pinpoint vulnerabilities.

Software Vulnerability



Static Vulnerability Detector

SVF

Flawfinder

coverity

Pinpoint

......

CHECKMARX

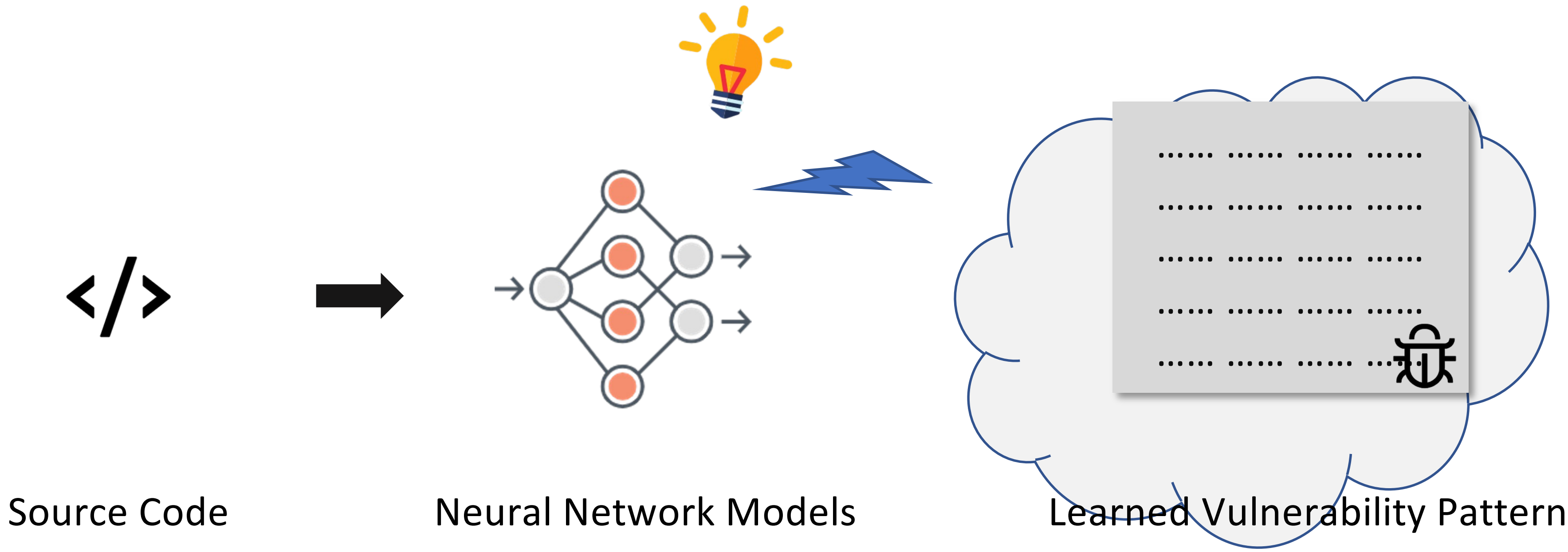
Some Static Vulnerability Detectors

Buffer Overflow
Memory Leak
Null Dereference
......
......

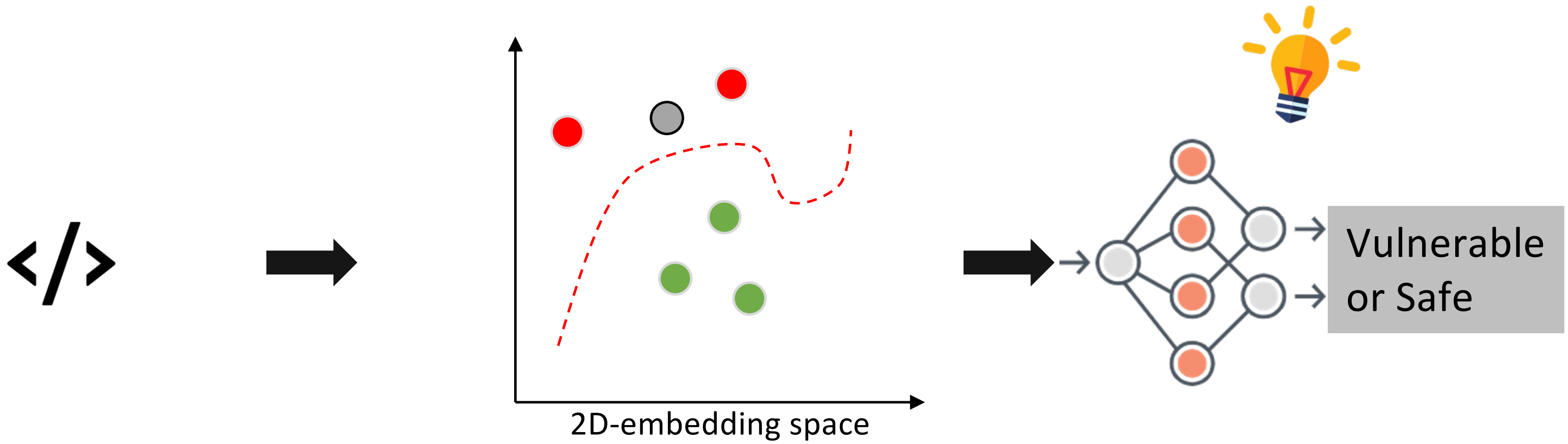
User-Defined Specifications

1. Rely heavily on **user-defined rules and domain knowledge**.
2. Have difficulty in finding **a wider range of vulnerabilities** (e.g., naming issues and incorrect business logic)

Learning-based Vulnerability Detector



Learning-based Vulnerability Detector



Source Code

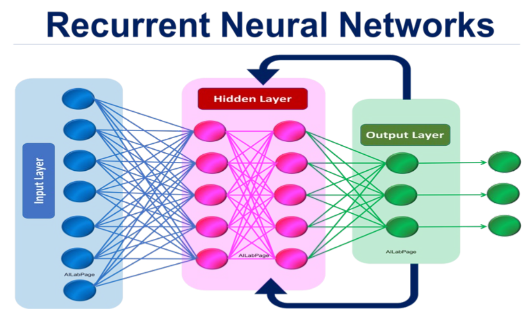
Code Embedding

Prediction

Code Embedding

Structure-unaware embedding

</> → int | main | (| ... →

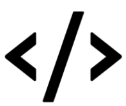


Source Code   Lexical Tokens   Natural Language Processing

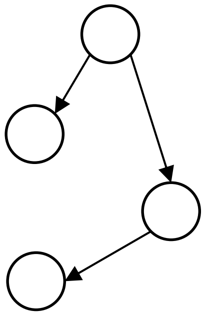
[1] Zhen Li, Deqing Zou, Shouhuai Xu, Xinyu Ou, Hai Jin, Sujuan Wang, Zhijun Deng, and Yuyi Zhong. 2018. VulDeePecker: A Deep Learning-Based System for Vulnerability Detection. NDSS (2018). https://doi.org/10.14722/ndss.2018.23158
[2] Z. Li, D. Zou, S. Xu, H. Jin, Y. Zhu, and Z. Chen. 2021. SySeVR: A Framework for Using Deep Learning to Detect Software Vulnerabilities. (2021), 1–1. https://doi.org/10.1109/TDSC.2021.3051525

Code Embedding

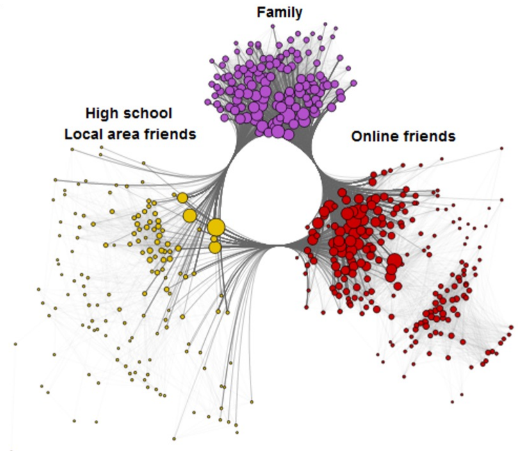
Structure-aware embedding



Source Code



Program Dependence Graphs



Graph Neural Network

[3] Xiao Cheng, Haoyu Wang, Jiayi Hua, Guoai Xu, and Yulei Sui. 2021. DeepWukong: Statically Detecting Software Vulnerabilities Using Deep Graph Neural Network. ACM Trans. Softw. Eng. Methodol. 30, 3, Article 38 (2021), 33 pages. https://doi.org/10.1145/3436877
[4] Yi Li, Shaohua Wang, and Tien N. Nguyen. 2021. Vulnerability Detection with Fine-Grained Interpretations (FSE '21). ACM, 292–303. https://doi.org/10.1145/3468264.3468597

Limitations

- Existing models are still Insufficient for precise bug detection, because the objective of these models is to **produce classification results** rather than **comprehending the semantics of vulnerabilities**, e.g., pinpointing **bug triggering paths**, which are essential for static bug detection.

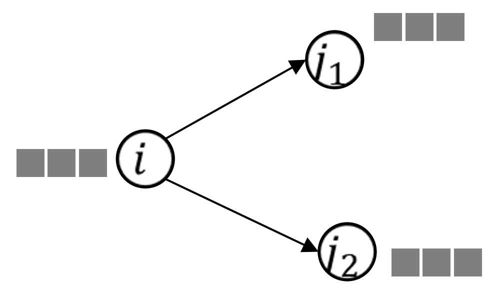
Limitations

GNN: Path-unaware Message-passing

GNN: **all pair-wise** message passing

$$x_i' = W_1 x_i + W_2 \sum_{j \in N(i)} e_{j,i} \cdot x_j$$

x_i: feature vector of node i
x_i': updated feature vector of node i
$N(i)$: neighbors of node i

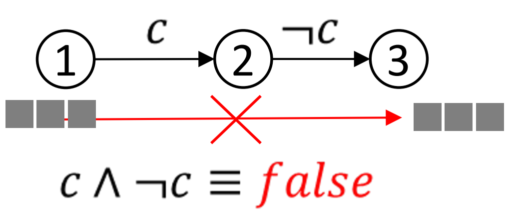


Message passing

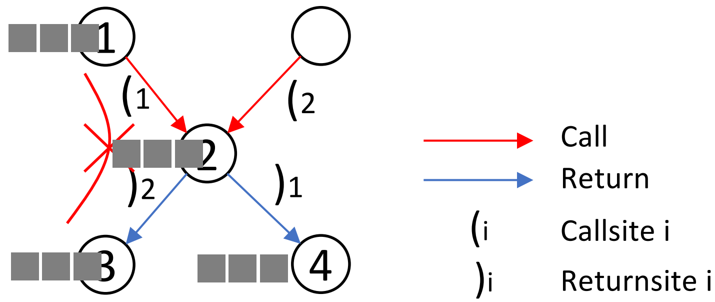
Limitations

GNN: Path-unaware Message-passing

GNN **does not distinguish feasible/infeasible program dependence paths**.



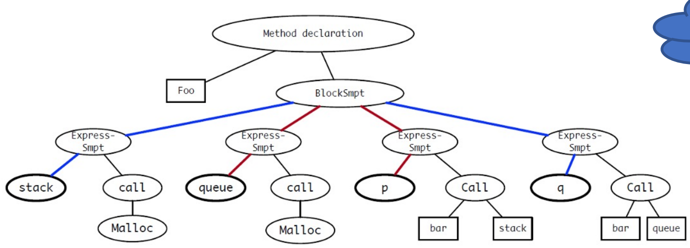
Path-insensitive



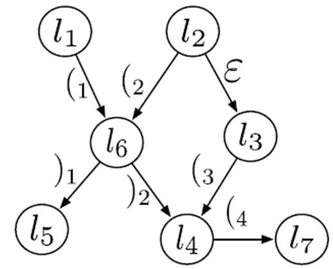
Context-insensitive

Path-based Code Embedding

- The detection approach needs to work on **a precise learning model** that **can preserve value-flow paths** such that we can check the feasibility.



Bag of paths



Abstract Syntax Tree

Value-Flow Graph

1. Aim at **code classification and summarization**.
2. Not suitable for path-based vulnerability detection due to **potentially unbounded number of paths**.

[5] Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. 2019. Code2vec: Learning Distributed Representations of Code. 3, POPL, Article 40 (Jan. 2019), 29 pages. https://doi.org/10.1145/3290353

[6] Yulei Sui, Xiao Cheng, Guanqin Zhang, and HaoyuWang. 2020. Flow2Vec: Value-Flow-Based Precise Code Embedding. Proc. ACM Program. Lang. 4, OOPSLA, Article 233 (Nov. 2020), 27 pages. https://doi.org/10.1145/3428301

Path-based Code Embedding

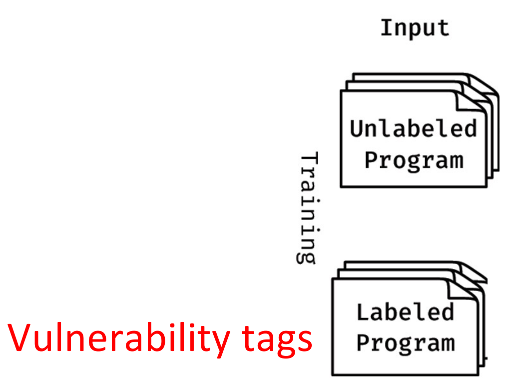
- Path embedding model
  - Preserve the **in-depth semantics of paths**
- Path selection strategy
  - Preserve **individual feasible paths** with **discriminative features**

The Aim of This Work

- ContraFlow: a **path-sensitive** code embedding approach which uses self-supervised **contrastive learning** to pinpoint vulnerabilities based on **value-flow paths**.

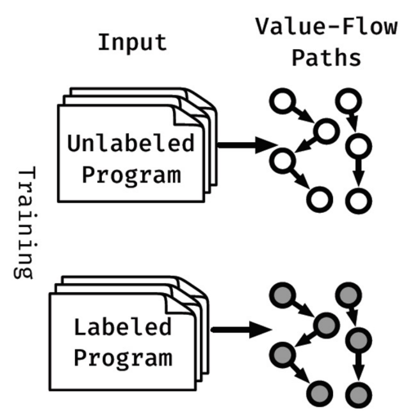
The Aim of This Work

- ContraFlow: a **path-sensitive** code embedding approach which uses self-supervised **contrastive learning** to pinpoint vulnerabilities based on **value-flow paths**.



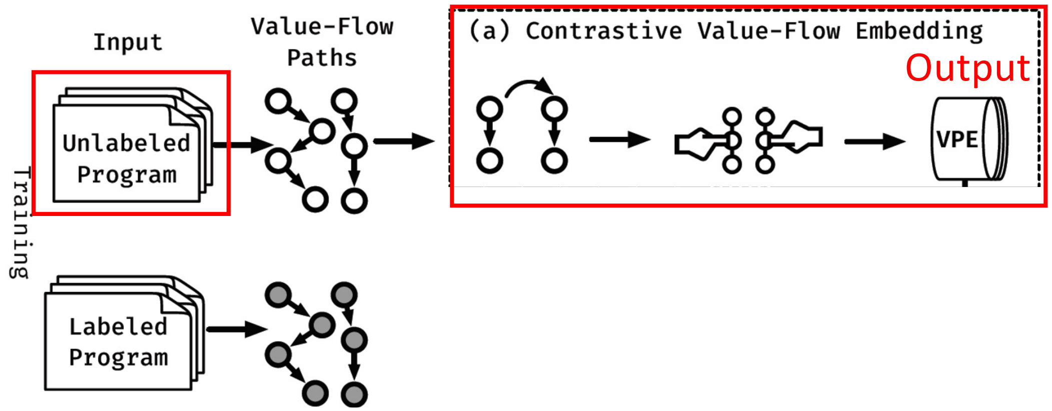
The Aim of This Work

- ContraFlow: a **path-sensitive** code embedding approach which uses self-supervised **contrastive learning** to pinpoint vulnerabilities based on **value-flow paths**.



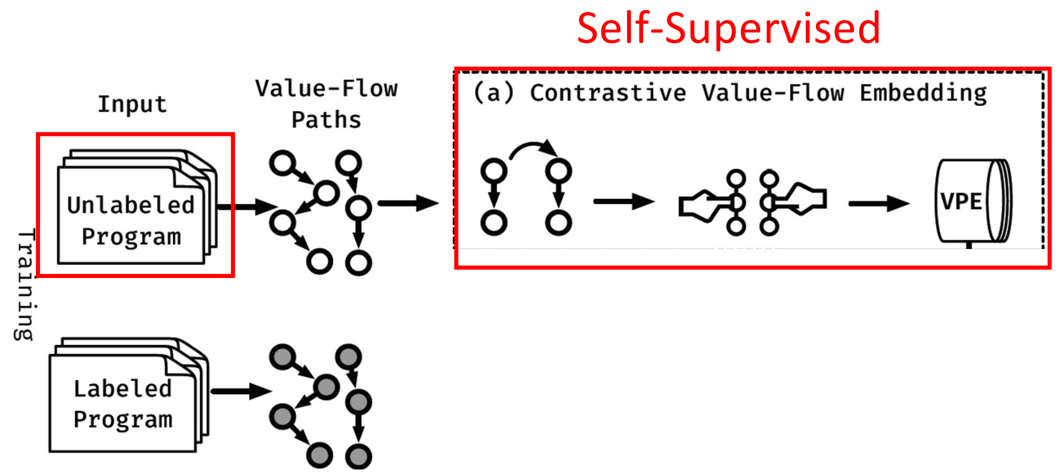
The Aim of This Work

- ContraFlow: a **path-sensitive** code embedding approach which uses self-supervised **contrastive learning** to pinpoint vulnerabilities based on **value-flow paths**.



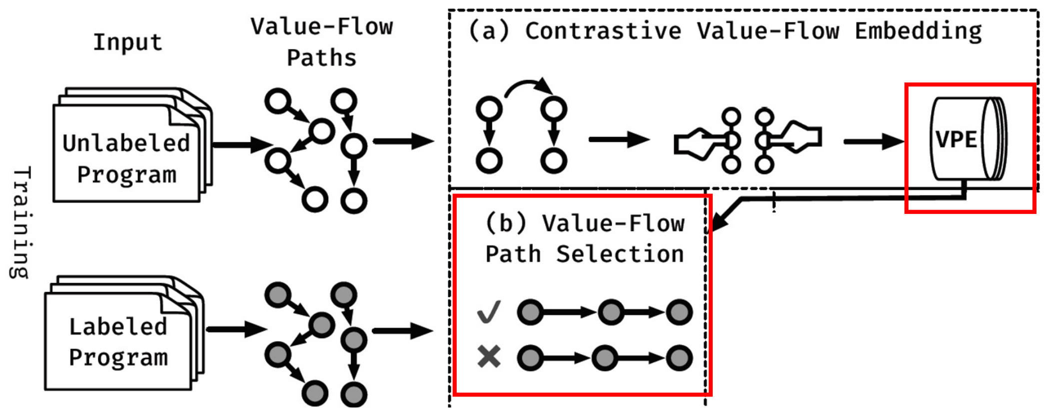
The Aim of This Work

- ContraFlow: a **path-sensitive** code embedding approach which uses self-supervised **contrastive learning** to pinpoint vulnerabilities based on **value-flow paths**.



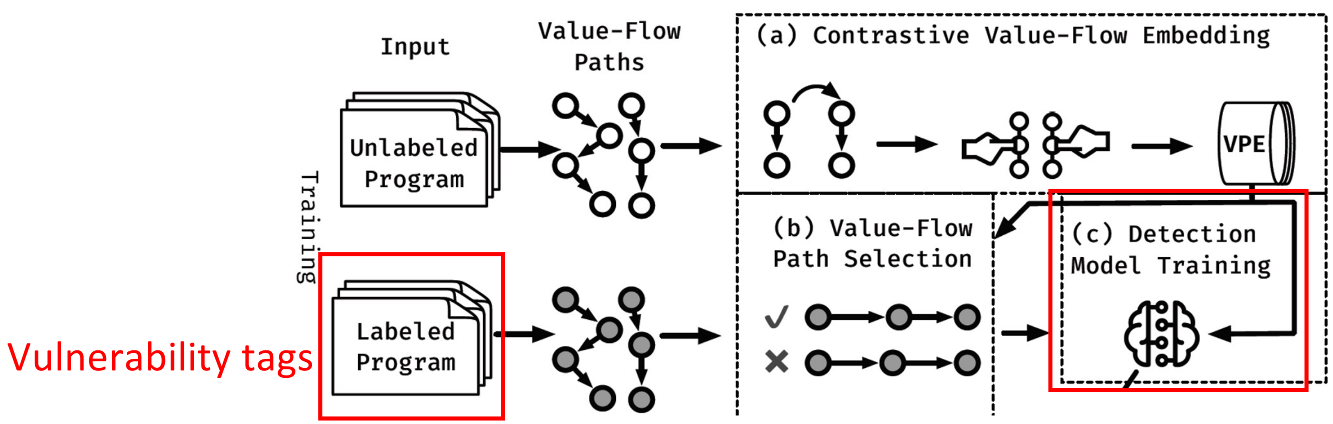
The Aim of This Work

- ContraFlow: a **path-sensitive** code embedding approach which uses self-supervised **contrastive learning** to pinpoint vulnerabilities based on **value-flow paths**.



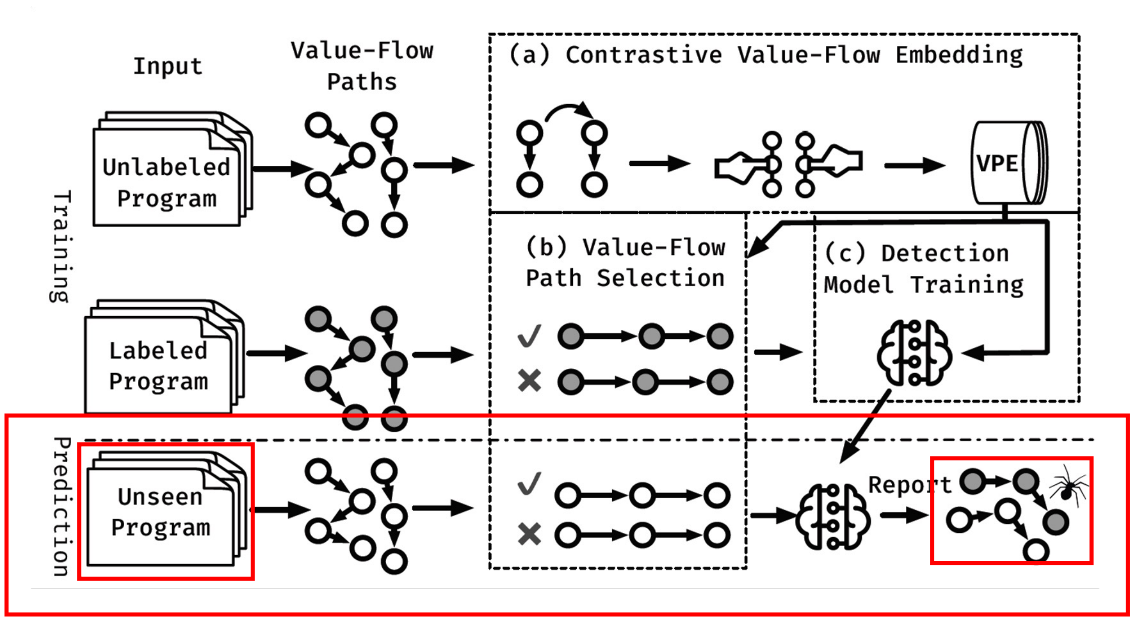
The Aim of This Work

- ContraFlow: a **path-sensitive** code embedding approach which uses self-supervised **contrastive learning** to pinpoint vulnerabilities based on **value-flow paths**.



The Aim of This Work

- ContraFlow: a **path-sensitive** code embedding approach which uses self-supervised **contrastive learning** to pinpoint vulnerabilities based on **value-flow paths**.



Motivating Example

(a) Contrastive Value-Flow Embedding

Source Code

```
void msg_q(){
    Inf hd = log_kits("head");
    Inf tl = log_kits("tail");
    ...
    if(FLG){
        rebuild_list(&hd);
        ...
    }else{
        rebuild_list(&tl);
        ...
    }
    if(FLG){
        set_status(&hd,&tl);
    }else{
        log_status(&hd, &tl);
    }
}
```

Motivating Example

(a) Contrastive Value-Flow Embedding

Source Code

```
void msg_q(){
    Inf hd = log_kits("head");
    Inf tl = log_kits("tail");
    ...
    if(FLG){
        rebuild_list(&hd);
        ...
    }else{
        rebuild_list(&tl);
        ...
    }
    if(FLG){
        set_status(&hd,&tl);
    }else{
        log_status(&hd, &tl);
    }
}
```

API misuse: log_kits → rebuild_list → set_status

Can cause unexpected behavior

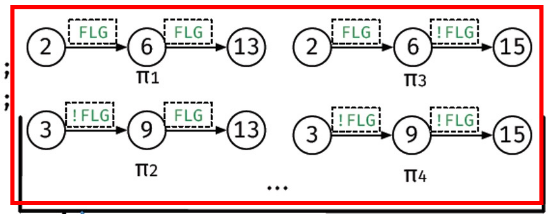
UNSW SYDNEY

Motivating Example

(a) Contrastive Value-Flow Embedding

Source Code **(a) Contrastive Value-Flow Embedding**

```
void msg_q(){
    Inf hd = log_kits("head");
    Inf tl = log_kits("tail");
    ...
    if(FLG){
        rebuild_list(&hd);
        ...
    }else{
        rebuild_list(&tl);
        ...
    }
    if(FLG){
        set_status(&hd,&tl);
    }else{
        log_status(&hd, &tl);
    }
}
```



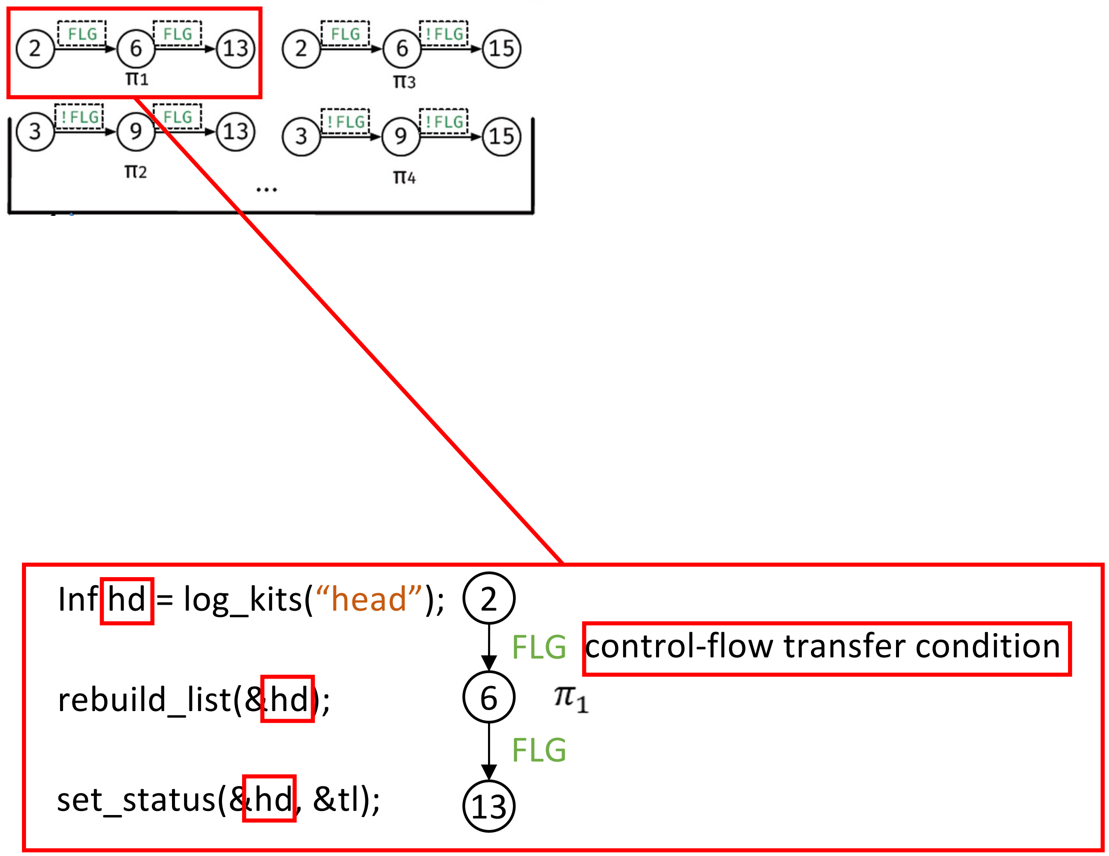
Motivating Example

(a) Contrastive Value-Flow Embedding

Source Code

```
void msg_q(){
    Inf hd = log_kits("head");
    Inf tl = log_kits("tail");
    ...
    if(FLG){
        rebuild_list(&hd);
        ...
    }else{
        rebuild_list(&tl);
        ...
    }
    if(FLG){
        set_status(&hd,&tl);
    }else{
        log_status(&hd, &tl);
    }
}
```

(a) Contrastive Value-Flow Embedding

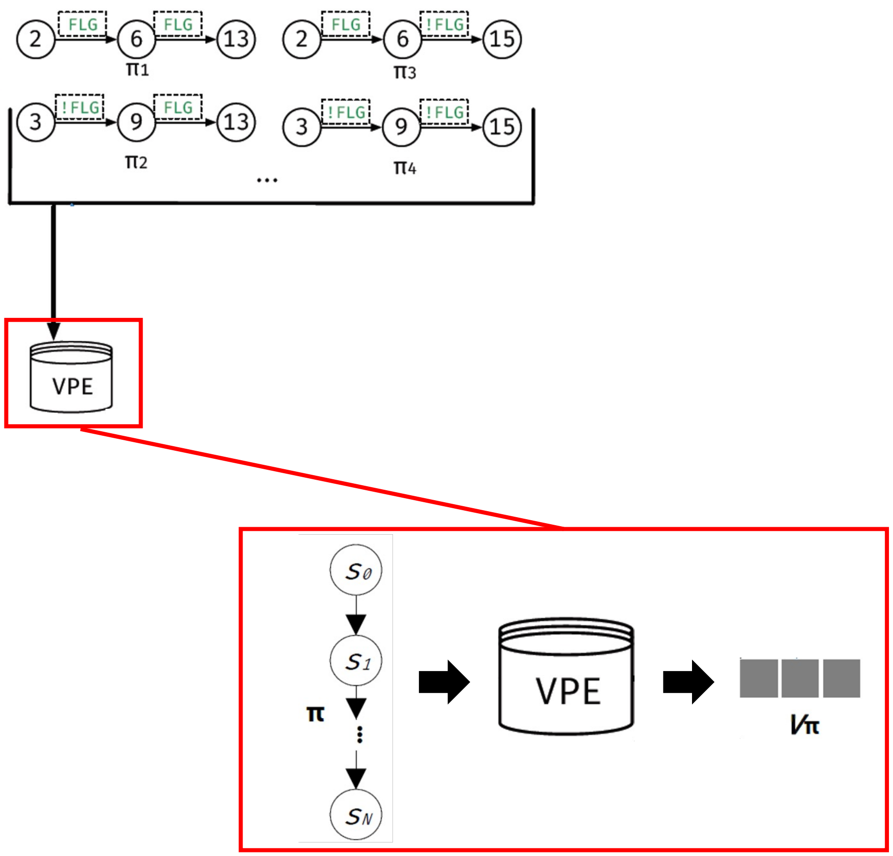


Motivating Example

(a) Contrastive Value-Flow Embedding

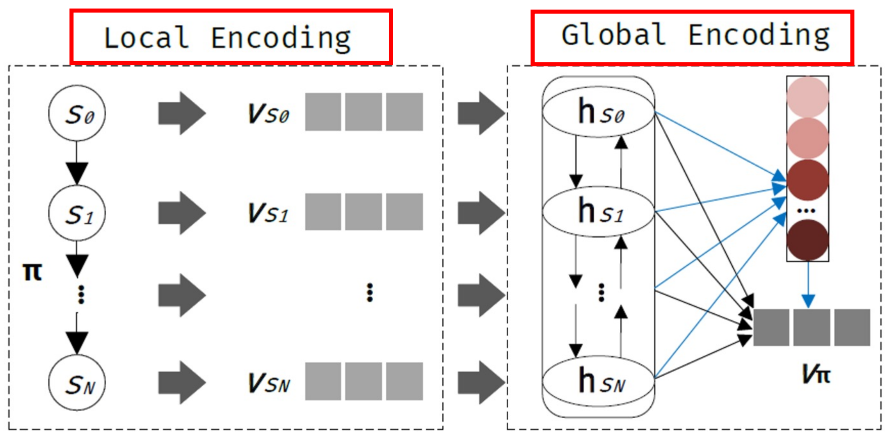
Source Code **(a) Contrastive Value-Flow Embedding**

```
void msg_q(){
    Inf hd = log_kits("head");
    Inf tl = log_kits("tail");
    ...
    if(FLG){
        rebuild_list(&hd);
        ...
    }else{
        rebuild_list(&tl);
        ...
    }
    if(FLG){
        set_status(&hd,&tl);
    }else{
        log_status(&hd, &tl);
    }
}
```



Motivating Example

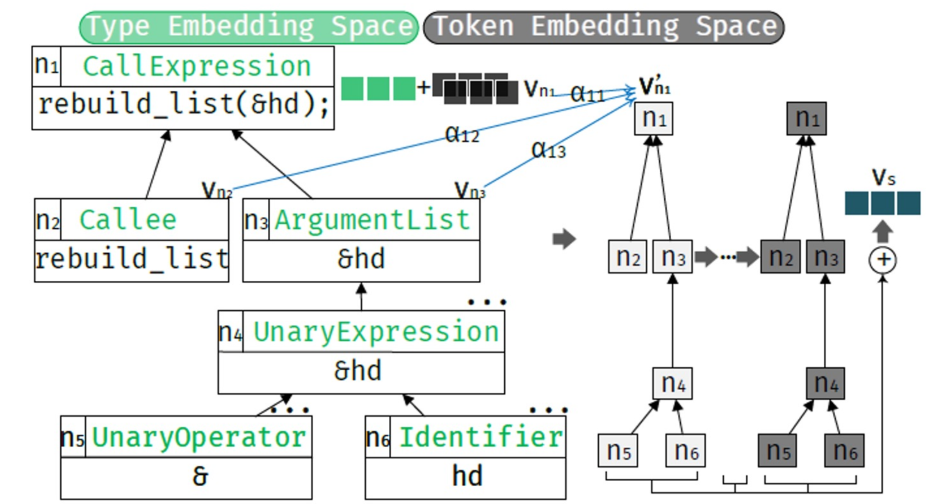
(a) Contrastive Value-Flow Embedding



Value-Flow Path Encoder (VPE)

Motivating Example

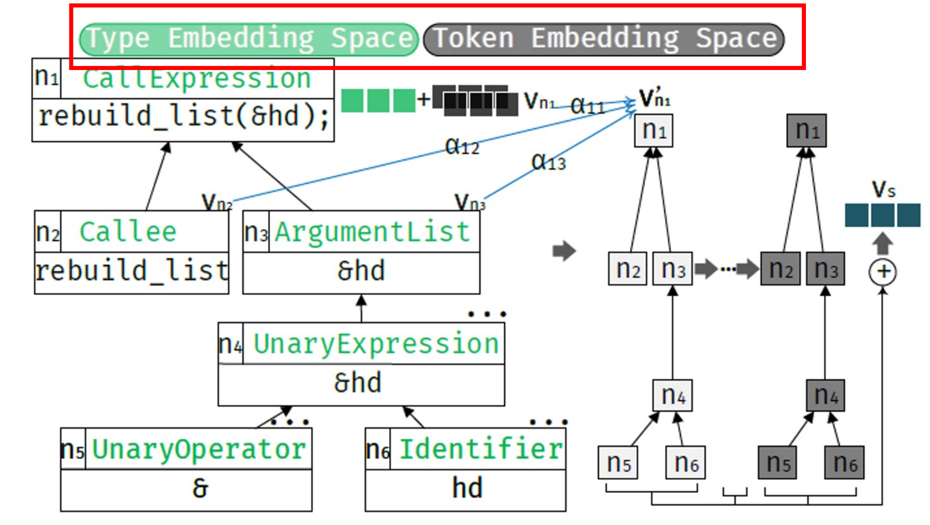
(a) Contrastive Value-Flow Embedding



Local Encoding

Motivating Example

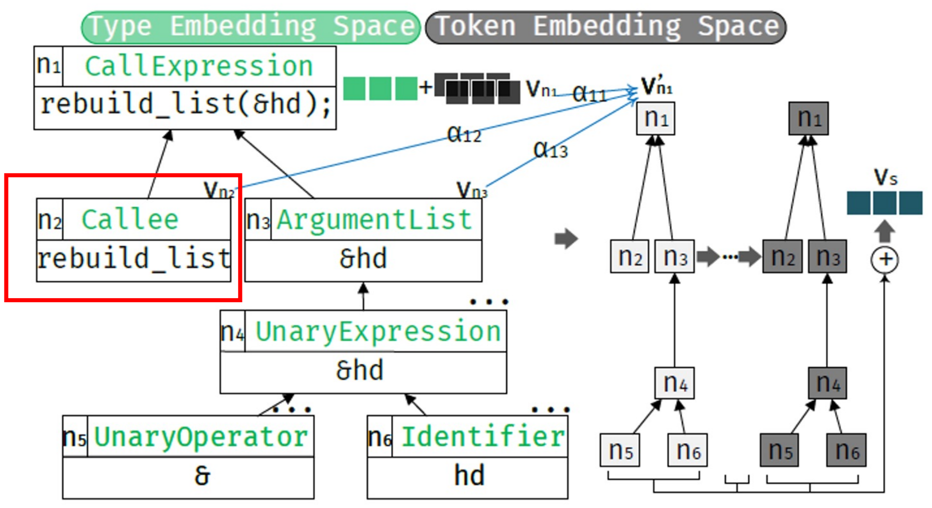
(a) Contrastive Value-Flow Embedding



Local Encoding

Motivating Example

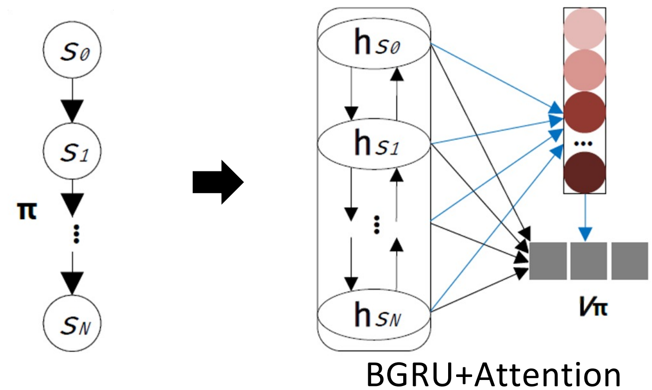
(a) Contrastive Value-Flow Embedding



Local Encoding

Motivating Example

(a) Contrastive Value-Flow Embedding



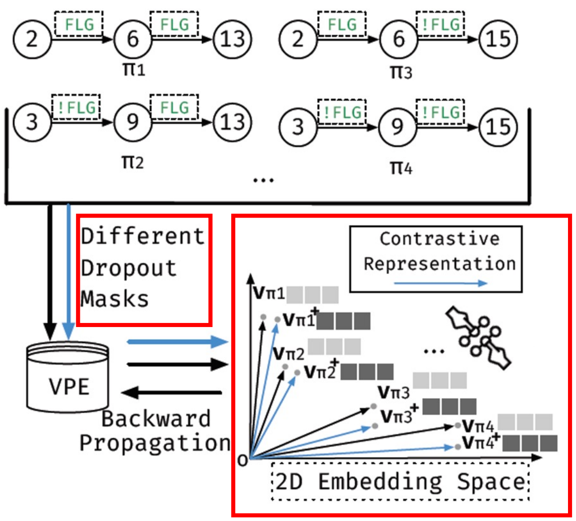
Global encoding

Motivating Example

(a) Contrastive Value-Flow Embedding

Source Code **(a) Contrastive Value-Flow Embedding**

```
void msg_q(){
    Inf hd = log_kits("head");
    Inf tl = log_kits("tail");
    ...
    if(FLG){
        rebuild_list(&hd);
        ...
    }else{
        rebuild_list(&tl);
        ...
    }
    if(FLG){
        set_status(&hd,&tl);
    }else{
        log_status(&hd, &tl);
    }
}
```

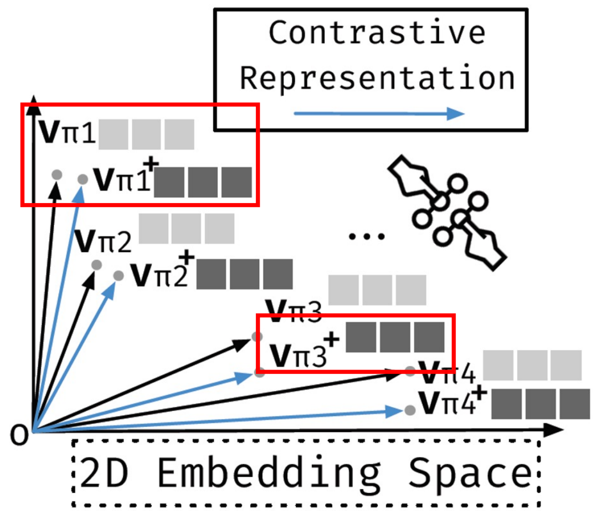


Motivating Example

(a) Contrastive Value-Flow Embedding

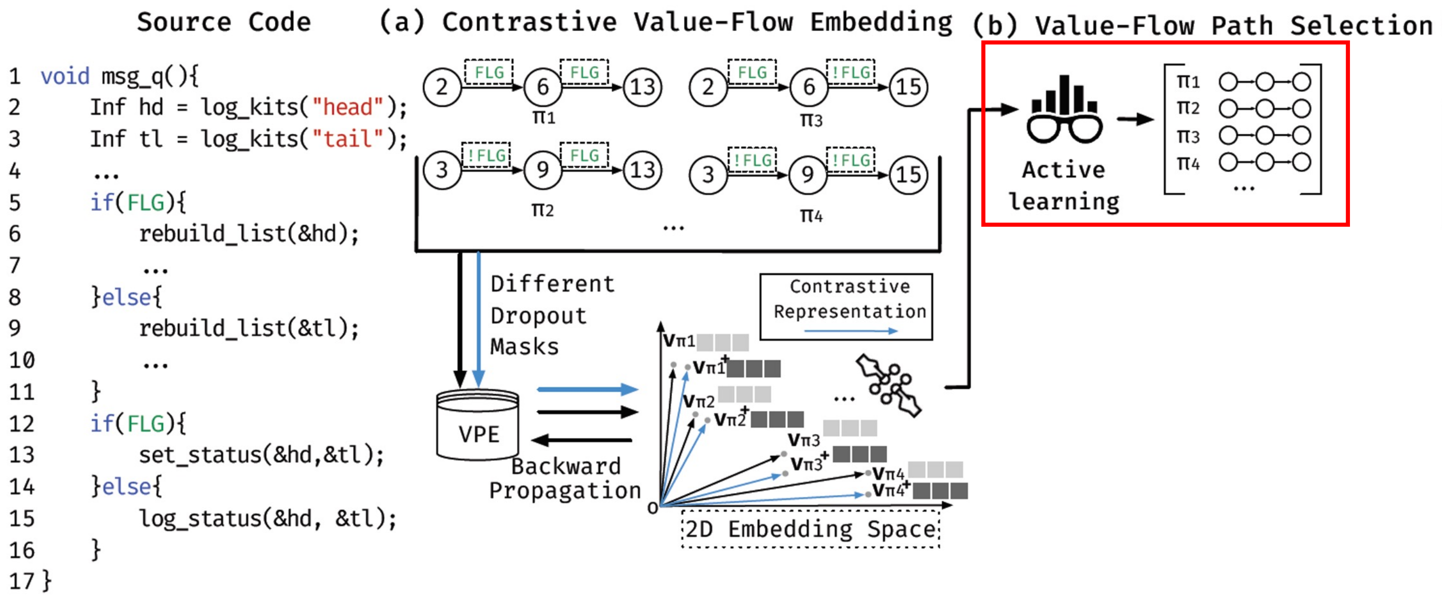
$$sim(\mathbf{v}_{\pi_i}, \mathbf{v}_{\pi_j}) = \frac{\mathbf{v}_{\pi_i}^{\top}\mathbf{v}_{\pi_j}}{||\mathbf{v}_{\pi_i}|| \cdot ||\mathbf{v}_{\pi_j}||} \quad loss(\pi_i) = -log\frac{exp(sim(\mathbf{v}_{\pi_i}, \mathbf{v}_{\pi_i}^{+}))}{\sum_{k=1}^{B} exp(sim(\mathbf{v}_{\pi_i}, \mathbf{v}_{\pi_k}^{+}))} \quad \mathcal{L} = \frac{1}{B}\sum_{i=1}^{B} loss(\pi_i)$$

Contrastive Value-Flow Embedding Loss



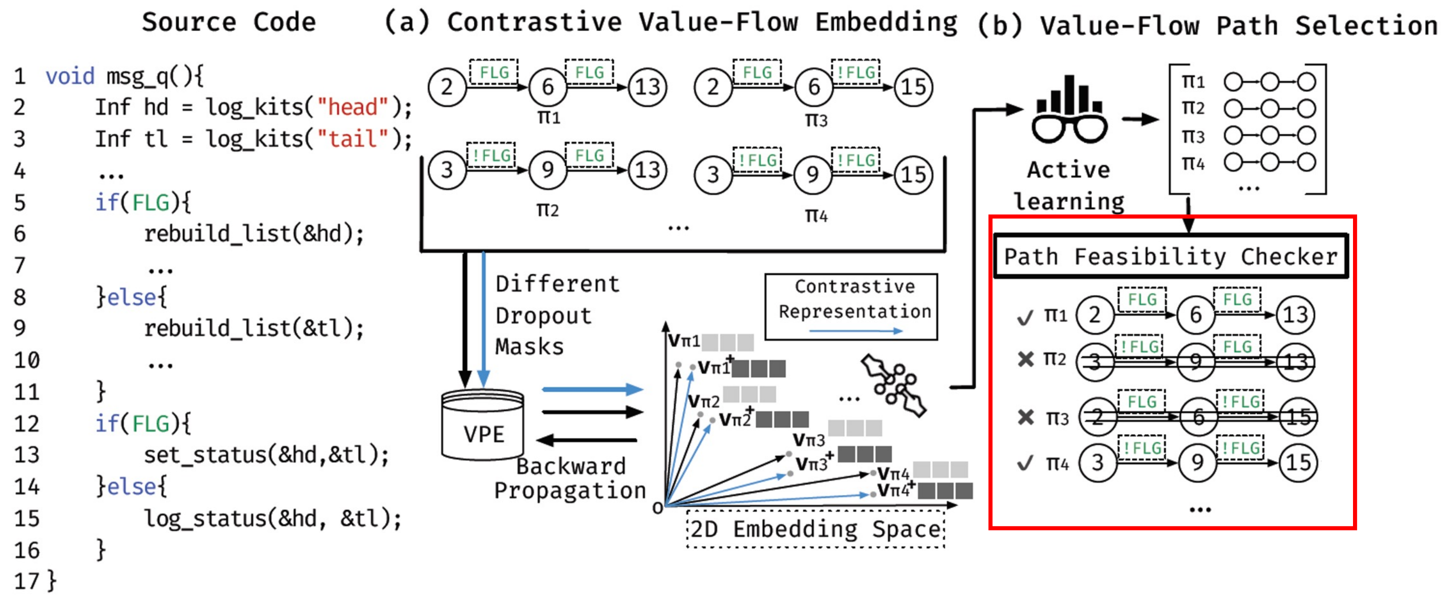
Motivating Example

(b) Value-Flow Path Selection



Motivating Example

(b) Value-Flow Path Selection

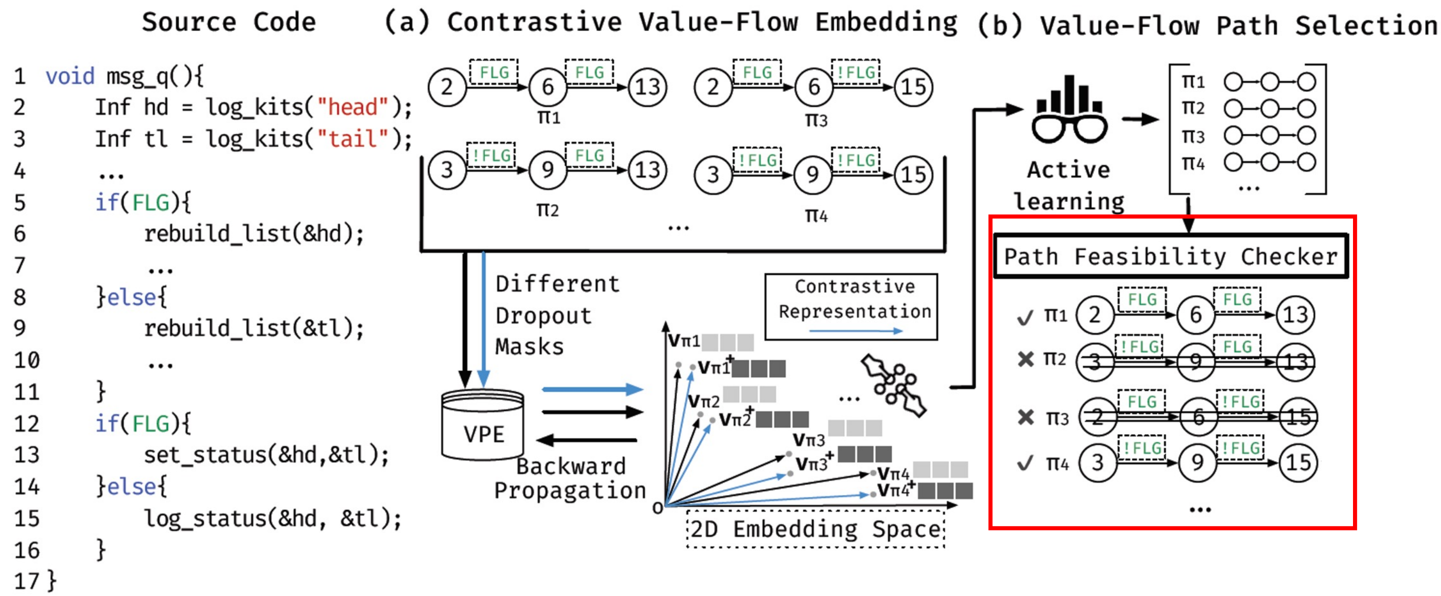


$$guard_v(\pi) = \bigwedge_{i=0}^{N-1} \bigvee_{p \in CP(s_i, s_{i+1})} \bigwedge_{e \in CE(p)} guard_e(e)$$

Value-Flow Guard

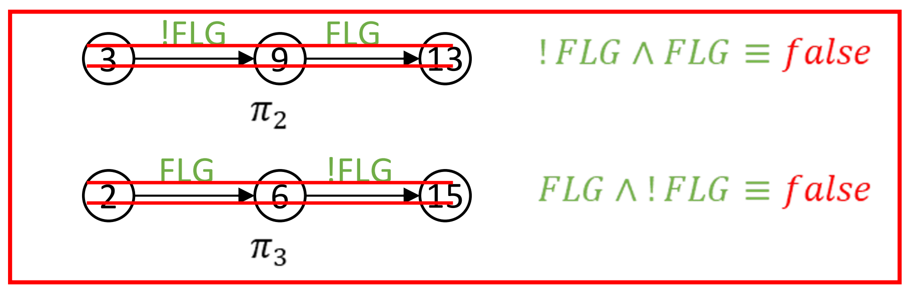
Motivating Example

(b) Value-Flow Path Selection



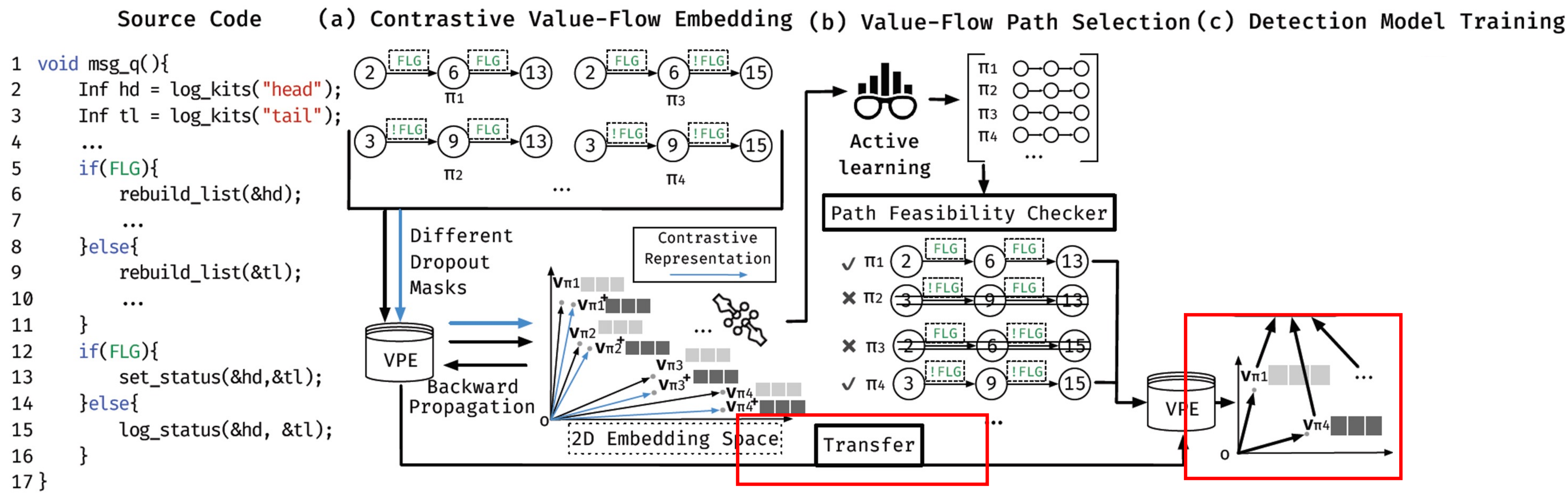
$$guard_v(\pi) = \bigwedge_{i=0}^{N-1} \bigvee_{p \in CP(s_i, s_{i+1})} \bigwedge_{e \in CE(p)} guard_e(e)$$

Value-Flow Guard



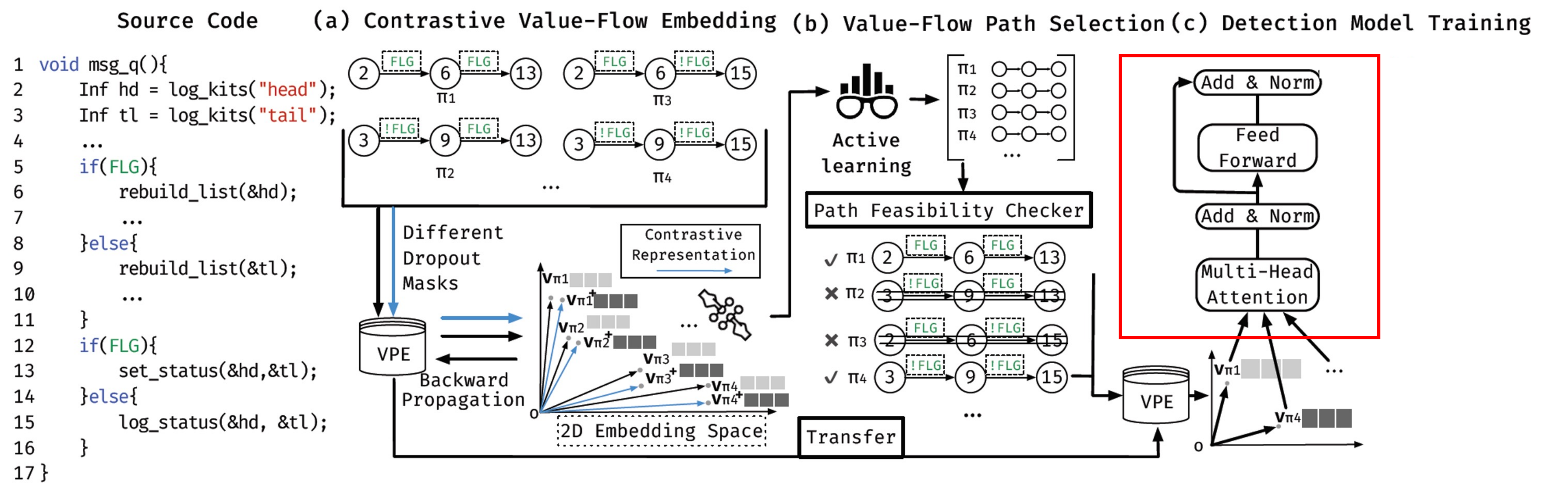
Motivating Example

(c) Detection Model Training



Motivating Example

(c) Detection Model Training



$$V' = [\mathbf{h}_1 || \ldots || \mathbf{h}_h]\mathbf{W}^o$$

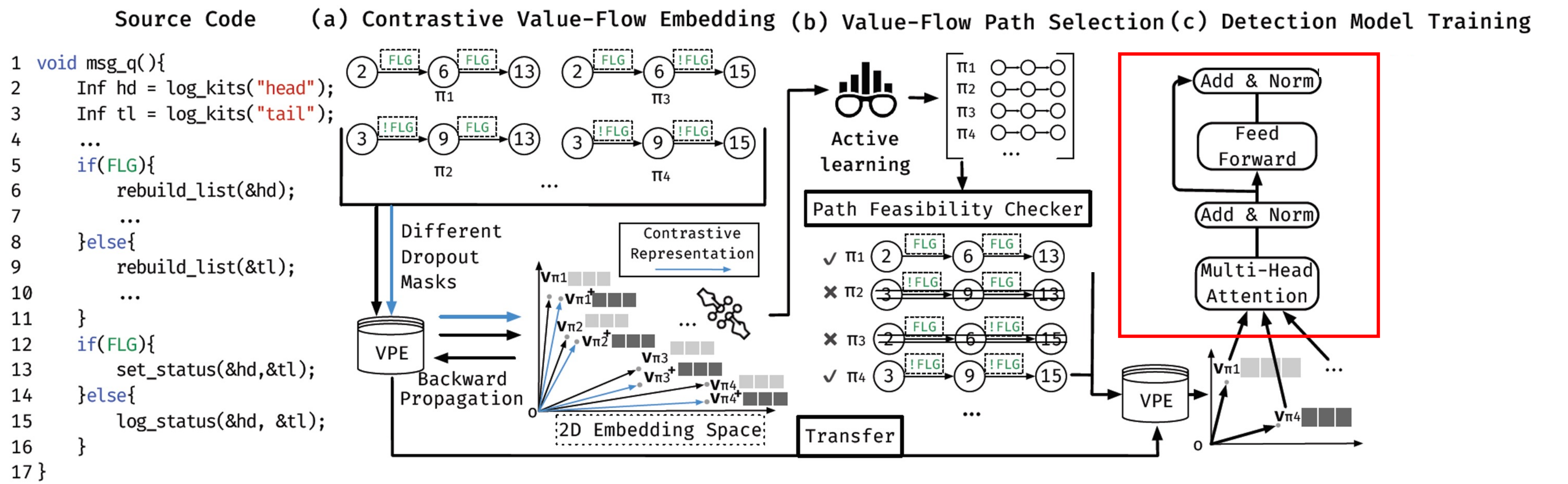
$$\mathbf{h}_i = Attn(\mathbf{V}\mathbf{W}_i^Q, \mathbf{V}\mathbf{W}_i^K)(\mathbf{V}\mathbf{W}_i^V)$$

$$Attn(\mathbf{Q}, \mathbf{K}) = softmax(norm(\mathbf{Q}\mathbf{K}^\top))$$

Multi-head self-attention

Motivating Example

(c) Detection Model Training

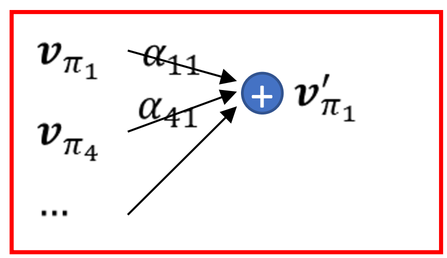


$$V' = [\mathbf{h}_1 || \ldots || \mathbf{h}_h]\mathbf{W}^o$$

$$\mathbf{h}_i = Attn(\mathbf{V}\mathbf{W}_i^Q, \mathbf{V}\mathbf{W}_i^K)(\mathbf{V}\mathbf{W}_i^V)$$

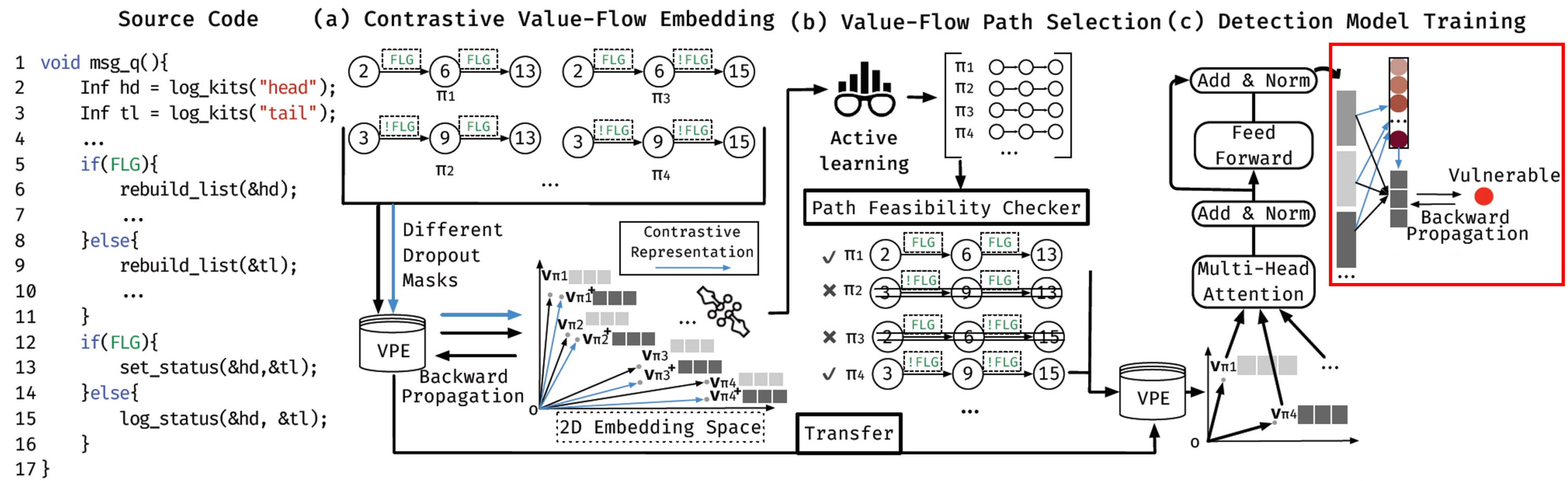
$$Attn(\mathbf{Q}, \mathbf{K}) = softmax(norm(\mathbf{Q}\mathbf{K}^\top))$$

Multi-head self-attention



Motivating Example

(c) Detection Model Training



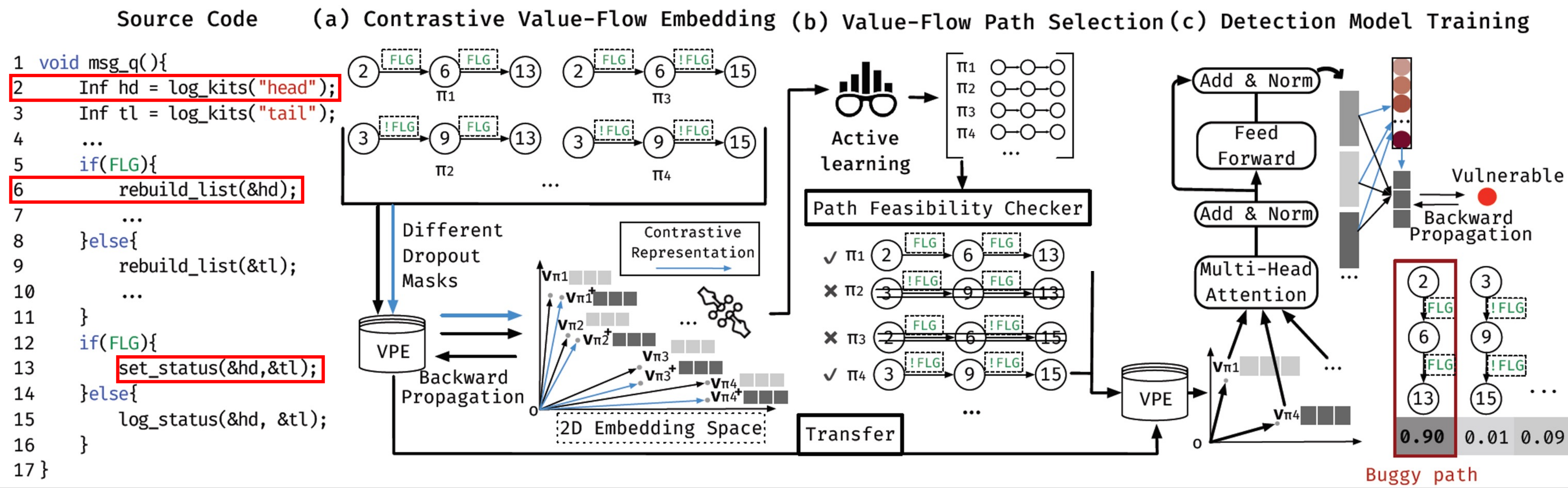
$$\alpha_i^c = \frac{exp(\mathbf{v}_{\pi_i}^\top \mathbf{a}_c)}{\sum_{j=1}^{N} exp(\mathbf{v}_{\pi_j}^\top \mathbf{a}_c)}$$

$$\mathbf{v}_c = \sum_{i=1}^{N} \alpha_i^c \cdot \mathbf{v}_{\pi_i}$$

soft attention

Motivating Example

(c) Detection Model Training



highest attention weights!

Experimental Evaluation

Benchmarks

OpenSSL Cryptography and SSL/TLS Toolkit

APACHE HTTP SERVER PROJECT

libav

FFmpeg

NGINX

TIFF

CVE® Common Vulnerabilities and Exposures

QEMU

……

288 open-sourced projects
30Million lines of code
275K programs

BUFFER_OVERRUN_L1
BUFFER_OVERRUN_L2
BUFFER_OVERRUN_L3
BUFFER_OVERRUN_S2
INTEGER_OVERFLOW_L1
INTEGER_OVERFLOW_L2
INTEGER_OVERFLOW_R2
MEMORY_LEAK
NULL_DEREFERENCE
RESOURCE_LEAK
UNINITIALIZED_VALUE
USE_AFTER_FREE
……

[7] Yunhui Zheng, Saurabh Pujar, Burn Lewis, Luca Buratti, Edward Epstein, Bo Yang, Jim Laredo, Alessandro Morari, and Zhong Su. 2021. D2A: A Dataset Built for AI Based Vulnerability Detection Methods Using Differential Analysis. In Proceedings of the ACM/IEEE 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP). ACM, New York, NY, USA.

[8] Jiahao Fan, Yi Li, Shaohua Wang, and Tien N. Nguyen. 2020. A C/C++ Code Vulnerability Dataset with Code Changes and CVE Summaries. In Proceedings of the 17th International Conference on Mining Software Repositories (MSR). ACM, 508–512. https://doi.org/10.1145/3379597.3387501

[9] YaQin Zhou, Shangqing Liu, Jingkai Siow, Xiaoning Du, and Yang Liu. 2019. Devign: Effective Vulnerability Identification by Learning Comprehensive Program Semantics via Graph Neural Networks. In Proceedings of the 33rd International Conference on Neural Information Processing Systems (NIPS '19). Curran Associates Inc. https://doi.org/10.5555/3454287.3455202

Experimental Evaluation

Benchmarks

Table 1: Labeled sample Distribution.

Dataset	granularity	# Vulnerable	# Safe	# Total
D2A	Method	21,396	2,194,592	2,215,988
	Slice	105,973	10,983,992	11,089,965
Fan	Method	8,456	142,853	151,309
	Slice	42,527	713,239	717,496
FQ	Method	8,923	9,845	18,768
	Slice	45,627	50,125	95,752
Total	Method	38,775	2,347,290	2,386,065
	Slice	194,127	11,747,356	11,903,213

Experimental Evaluation

Comparison with baselines

Table 2: Comparison of method- and slice-level approaches under informedness (IF), markedness (MK), F1 Score (F1), Precision (P) and Recall (R). ContraFlow-method/slice denotes the evaluation at method- and slice-level respectively.

Model Name	IF (%)	MK (%)	F1 (%)	P (%)	R (%)
VGDetector	31.1	29.3	56.7	52.6	61.4
Devign	30.1	28.8	58.7	54.6	63.4
Reveal	34.2	33.8	63.4	61.5	65.5
ContraFlow-method	**60.3**	**58.2**	**75.3**	**71.5**	**79.4**
VulDeePecker	17.3	17.3	52.3	52.2	52.4
SySeVR	24.3	24.2	55.0	54.5	55.4
DeepWukong	48.1	48.4	67.0	67.4	66.5
VulDeeLocator	38.4	38.1	62.0	61.4	62.5
IVDetect	37.4	37.3	64.1	64.0	64.6
ContraFlow-slice	**75.1**	**72.3**	**82.8**	**79.5**	**86.4**

Experimental Evaluation

Comparison with baselines

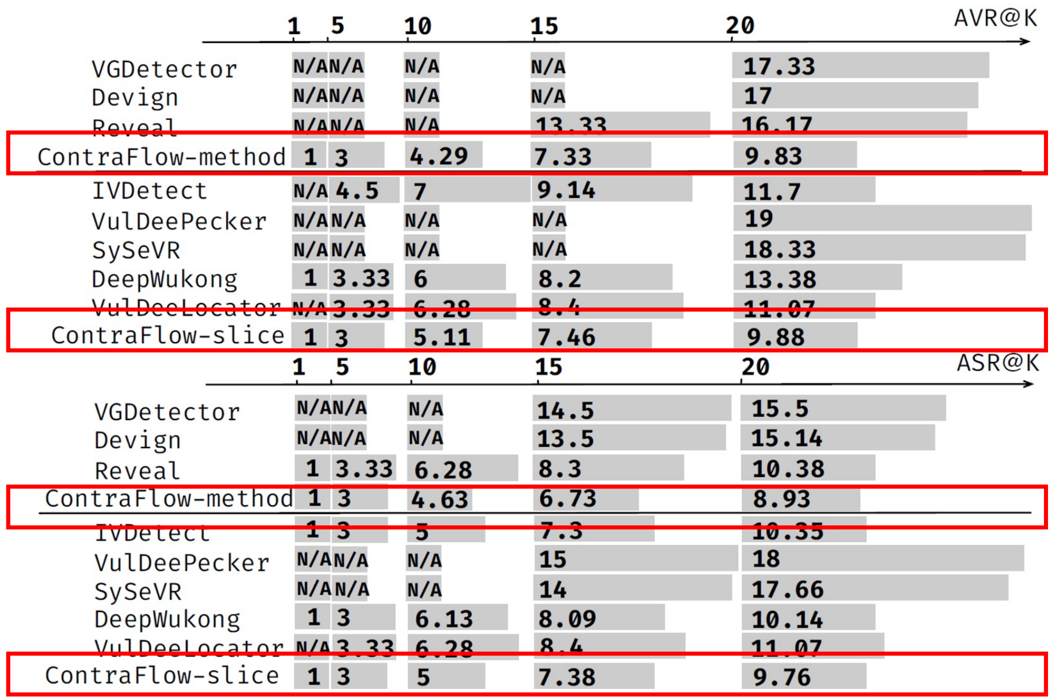


Figure 7: Comparison with IVDetect and VulDeeLocator under AVR@k (ASR@k) [48]. AVR@k (ASR@k) represents the average top-k ranking of the correctly predicted vulnerable (safe) samples. N/A means that there is no correctly predicted sample in the top-ranked list.

Experimental Evaluation

Comparison with baselines

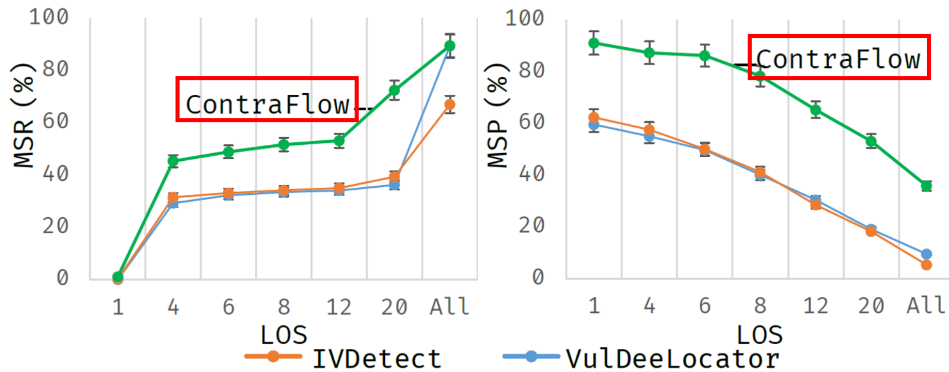


Figure 8: MSR and MSP under different LOSs.

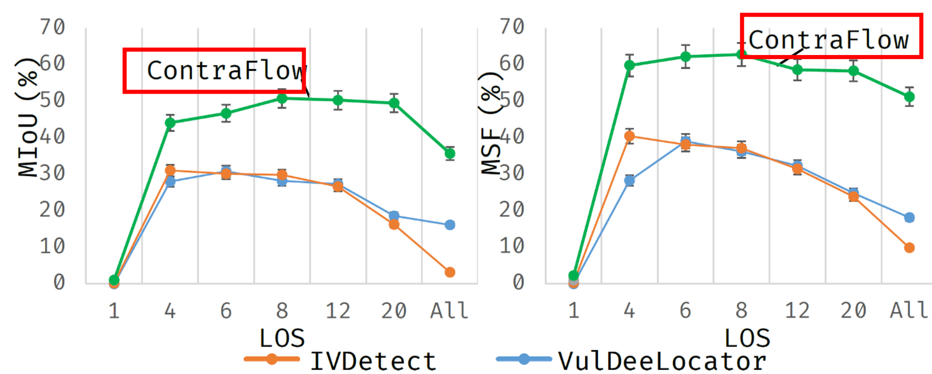


Figure 9: MIoU and MSF under different LOSs. MSF is the harmonic mean of MSP and MSR.

Experimental Evaluation

Comparison with baselines

Table 3: Comparison with IVDetect and VulDeeLocator under SA, MFR and MAR [48]. Statement Accuracy (SA) counts a correct detection if one labeled vulnerable statement is reported. MFR/MAR are the mean value of the first/average ranks of correctly detected statements.

Model Name	SA(%)				MFR	MAR
	1 LOS	4 LOS	6 LOS	12 LOS		
VulDeeLocator	1.3	46.7	50.2	54.4	6.9	10.5
IVDetect	2.1	55.5	59.7	63.5	6.8	9.5
ContraFlow	**15.1**	**73.9**	**78.2**	**84.1**	**2.1**	**5.7**

Experimental Evaluation

Ablation Analysis

Table 4: Ablation Analysis Results. ContraFlow-CodeBert/BLSTM/BGRU means ContraFlow with CodeBert/BLSTM/BGRU as the value-flow path encoder.

Model Name	IF (%)	MK (%)	F1 (%)	MIoU (%)	MAR
Non-contrastive	61.3	57.9	74.2	40.3	7.8
Random-sampling	63.2	59.6	75.0	42.9	7.1
Path-insensitive	49.3	47.2	68.6	33.2	9.8
ContraFlow-CodeBert	68.3	63.9	78.0	45.3	6.4
ContraFlow-BLSTM	56.3	54.4	73.2	42.3	7.5
ContraFlow-BGRU	58.3	56.2	74.2	43.1	6.9
ContraFlow	**75.1**	**72.3**	**82.8**	**50.9**	**5.7**

UNSW SYDNEY

Thanks!

Q&A