Loupe: End-to-End Learning of Loop Unrolling Heuristics for Abstract Interpretation

Maykel Mattar[1,2], Michele Alberti[1], Valentin Perrelle[1], Salah Sadou[2]

1. *Université Paris-Saclay, CEA, List, Palaiseau, France*
2. *Université Bretagne Sud, IRISA, Vannes, France*

ASE 2025

Agenda

Present an overview of the work

01
General context

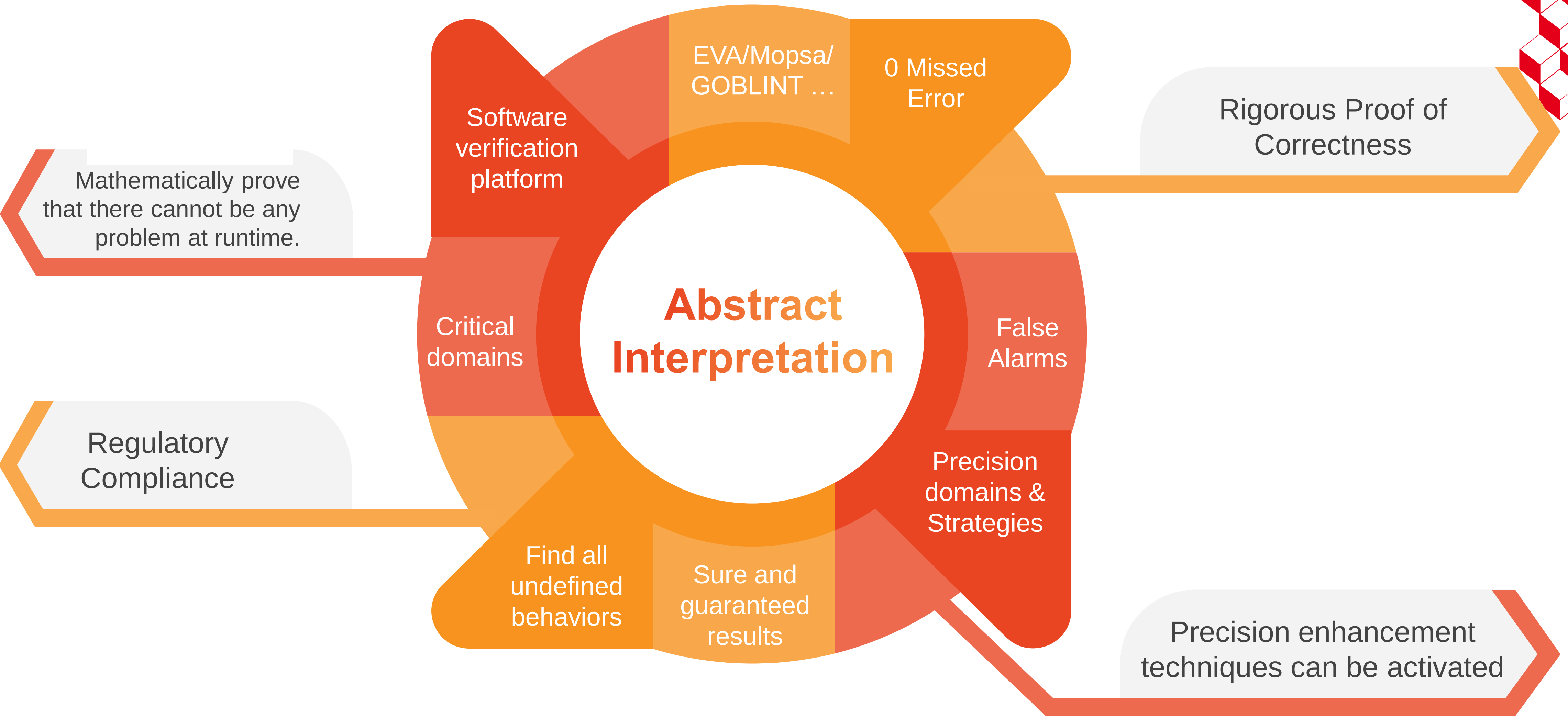
Present a brief introduction about the subject.

02
Our Approach

03
Results

Present and discuss the results

1. General context



Software verification platform
EVA/Mopsa/ GOBLINT …
0 Missed Error
Critical domains
Abstract Interpretation
False Alarms
Find all undefined behaviors
Sure and guaranteed results
Precision domains & Strategies
Mathematically prove that there cannot be any problem at runtime.
Rigorous Proof of Correctness
Regulatory Compliance
Precision enhancement techniques can be activated

“With great configurability comes great complexity.

REQUIRES EXTENSIVE USER EXPERTISE

HANDCRAFTED HEURISTICS ARE NOT SCALABLE

TRADE-OFF: COMPUTATIONAL RESOURCES AND FALSE POSITIVES

Example

When Eva is **not** requested unroll the loop:

- ***a*** and ***b*** are in the range $[1, 2^{31} - 1]$
- Overflow alarm for the operation ***a += b***.
- The interval abstraction of ***a*** and ***b*** fails to capture the relationship between these variables and ***i***;
- Eva may not find a precise invariant before considering the entire positive range of ***32-bit integers***

```
#include <stdio.h>

int fib(int n) {
    int a = 1, b = 1;

    for (int i = 3; i <= n; i++) {
        int tmp = a;
        a += b;
        b = tmp;
    }
    return a;
}

void main() {

    for (int i = 1, n; i <= 10; i++) {
        printf("Enter a number <= 30: ");
        scanf("%d", &n);
        If (n > 0 && n <= 30) {
            printf("fib(%d)=%d\n", n, fib(n));
            break;
        }
    }
}
```

Example

When Eva is requested to unroll the loop:

- The loop in line.4 is analyzed on iteration basis
- ***a and b*** range is precisely approximated.
- The alarms disappear

While loop unrolling can sometimes accelerate the analysis, it often increases computational cost significantly, particularly for nested loops.

```c
1.  #include <stdio.h>

2.  int fib(int n) {
3.      int a = 1, b = 1;
4.      //@ loop unroll 100;
5.      for (int i = 3; i <= n; i++) {
6.          int tmp = a;
7.          a += b;
8.          b = tmp;
9.      }
10.     return a;
11. }

12. void main() {
13.     //@ loop unroll 0;
14.     for (int i = 1, n; i <= 10; i++) {
15.         printf("Enter a number <= 30: ");
16.         scanf("%d", &n);
17.         If (n > 0 && n <= 30) {
18.             printf("fib(%d)=%d\n", n, fib(n));
19.             break;
20.         }
21.     }
22. }
```

Parameterization Approaches

Approach Type	Representative Works / Tools	Key Idea / Technique	Limitations
Hard Coded Heuristics	*eva-auto-loop-unroll*	According to the user-provided factor, it attempts to unroll all loops that can be unrolled.	Factor-driven; over-unroll simple loops and overlook complex ones; Time consuming
Automated Tuning (Algorithmic Search)	PARF [1], TAILOR [2]	Iterative execution with parameter optimization (local search, probabilistic refinement)	High computation cost; program-specific; no generalization
Machine Learning	Feature-based [3,4]	Learn heuristics from handcrafted program features	Requires feature engineering; not generalizable; expert bias
	Automatic Feature Generation – Sparrow [5,6]	Attempt to learn or generate features automatically	Limited to specific strategies; poor generality

1. Mansur et al., Automatically Tailoring Abstract Interpretation to Custom Usage Scenarios, CAV 2021.
2. Wang et al., Parf: Adaptive Parameter Refining for Abstract Interpretation, ASE 2024.
3. Oh et al., Learning a Strategy for Adapting a Program Analysis via Bayesian Optimisation, SIGPLAN Not., 2015.
4. Jeong et al., Data-Driven Context-Sensitivity for Points-to Analysis, Proc. ACM Program. Lang., OOPSLA 2017.
5. Jeon et al., Learning Graph-Based Heuristics for Pointer Analysis without Handcrafting Features, Proc. ACM Program. Lang., OOPSLA 2020.
6. Chae et al., Automatically Generating Features for Learning Program Analysis Heuristics for C-Like Languages, Proc. ACM Program. Lang., OOPSLA 2017.

3. Our Approach

Key Objectives

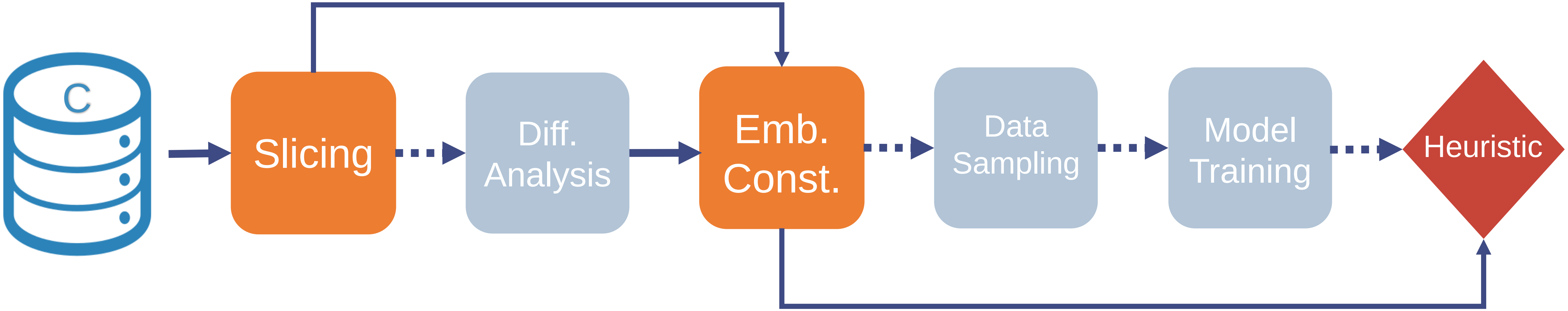
Use loop code; End-to-End learned heuristic

Auto-generate labeled data

Select the appropriate learning methods

Keep the approach modular and adaptable

Loupe: End-to-End Learning of Loop Unrolling Heuristics for Abstract Interpretation



C Datasets

AnghaBench

TRAIN/EVAL/TEST

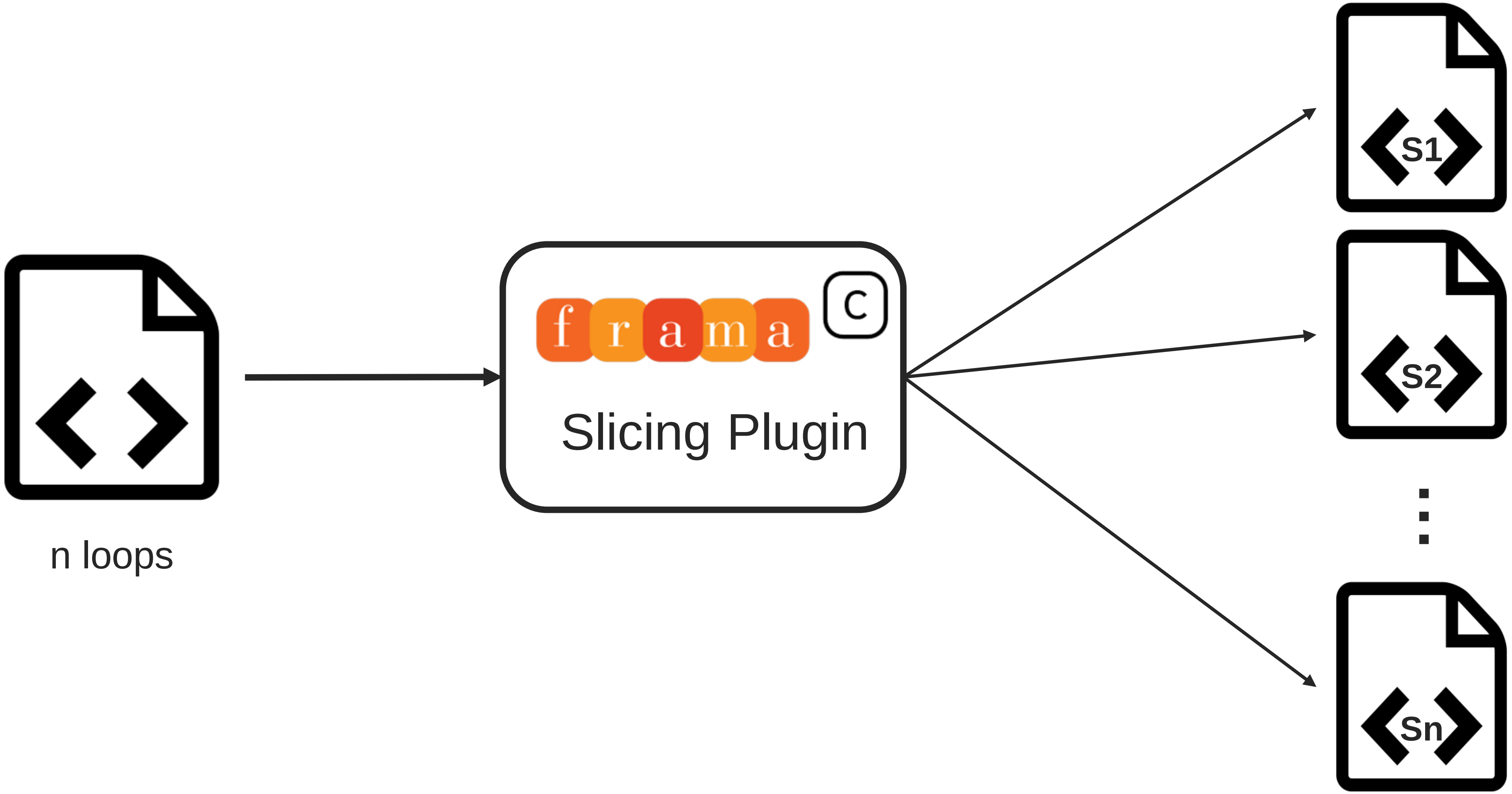
- Large-Scale Dataset >1M file
- Self-Contained Files
- Direct Processing
- Diverse Code Samples

Open Source Case Studies

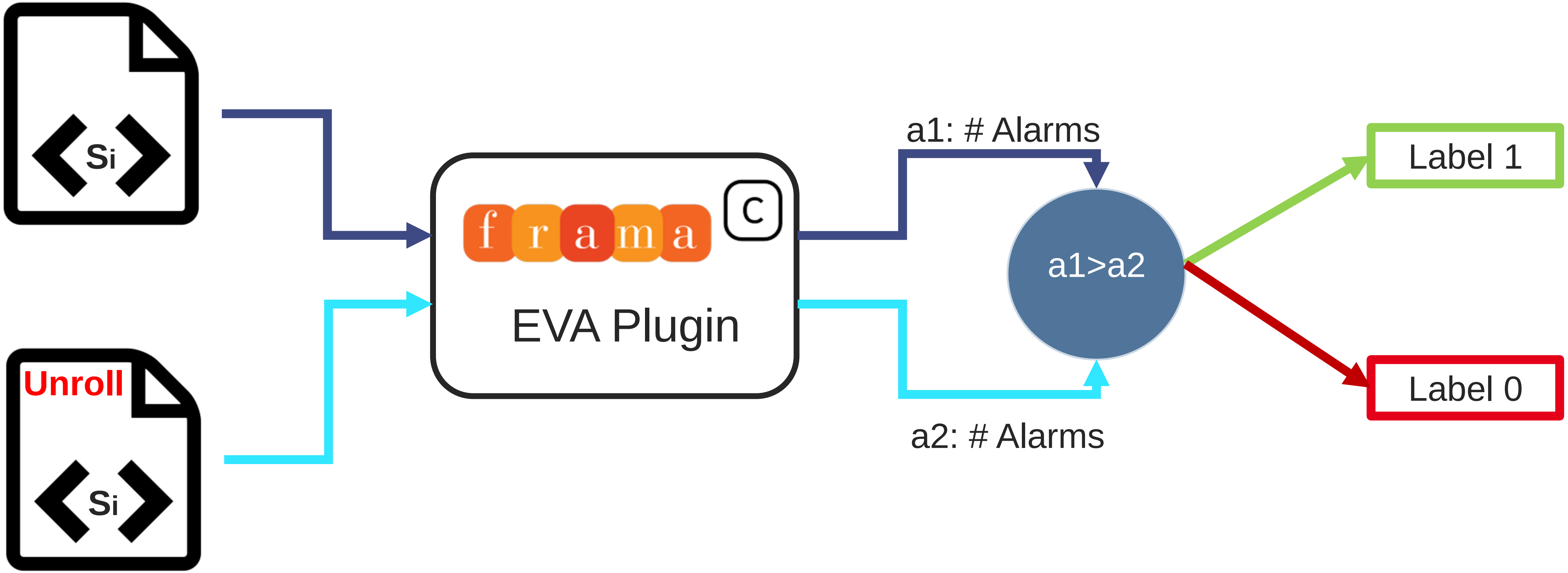
REAL-WORD AND PERFORMANCE EVALUATION

- Real-world C projects in their original form.
- Dependencies & complexity as found in actual software.
- Adapted for static analysis with Eva.
- Partially annotated.

Slicing

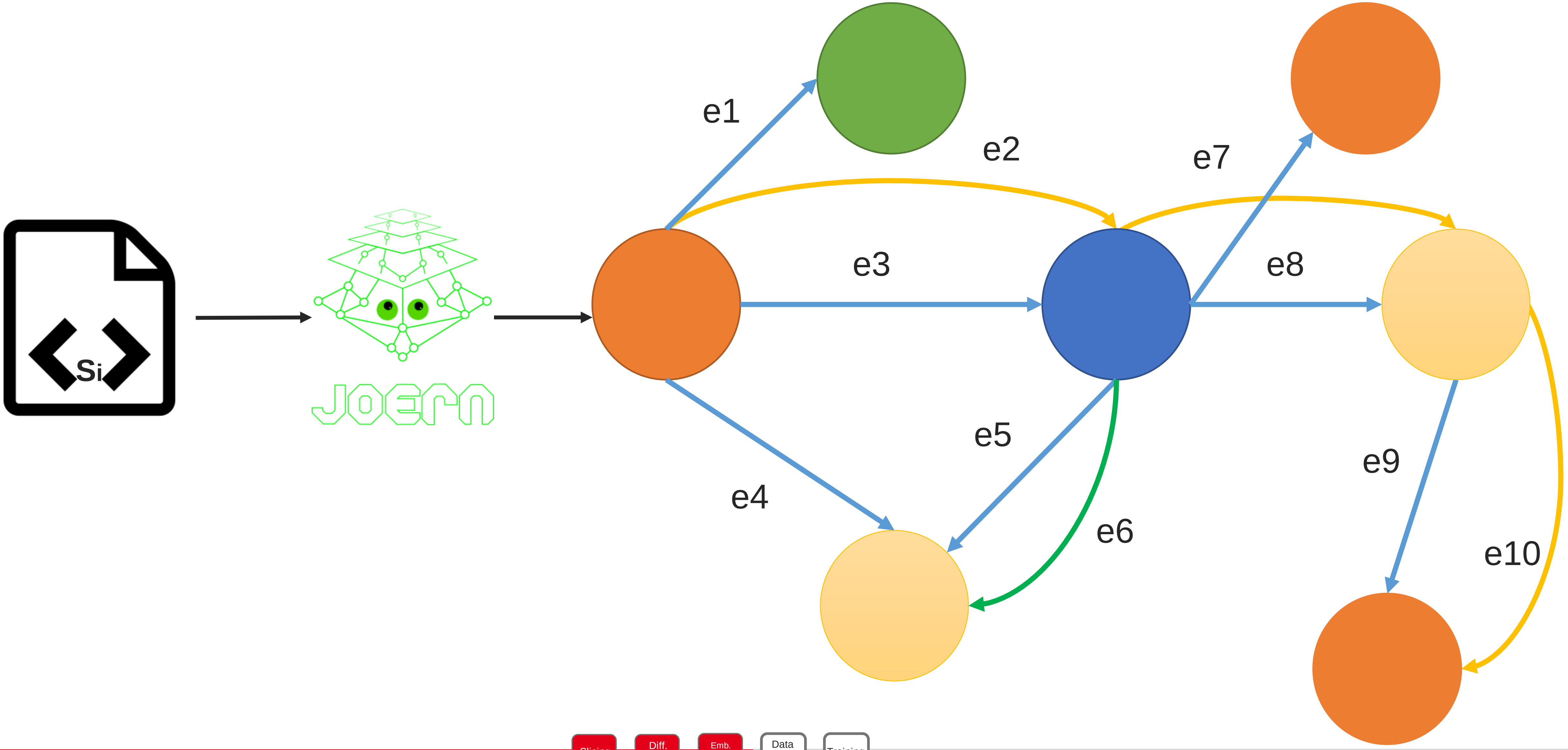


Differential Analysis

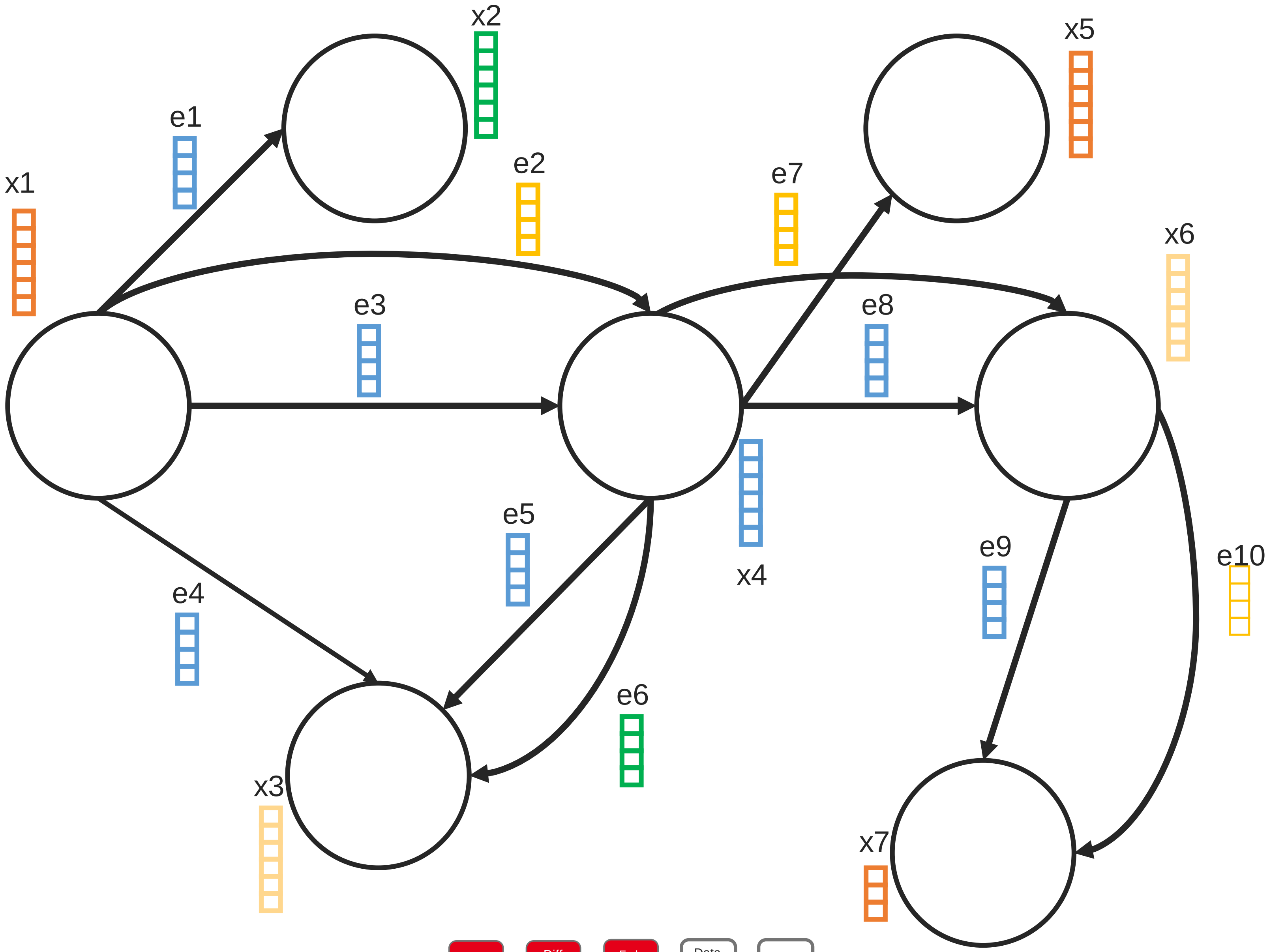


Embedding Construction

$G_{CPG} = (V, E, \lambda, \mu)$



G(s) = (X, EdgeAttributes, EdgeIndex)



Slicing
Diff. Analysis
Emb. Construct
Data Sampling
Training

Data Sampling

*Since loop unrolling is rarely required, the resulting dataset exhibits a significant imbalance, with ratio around **1:10** of positive to negative cases.*

01

NAÏVE METHODS

- Undersampling
- Oversampling
- α-undersampling
- α-oversampling

02

DURING TRAINING

- Weight Balancing
- Focal Loss

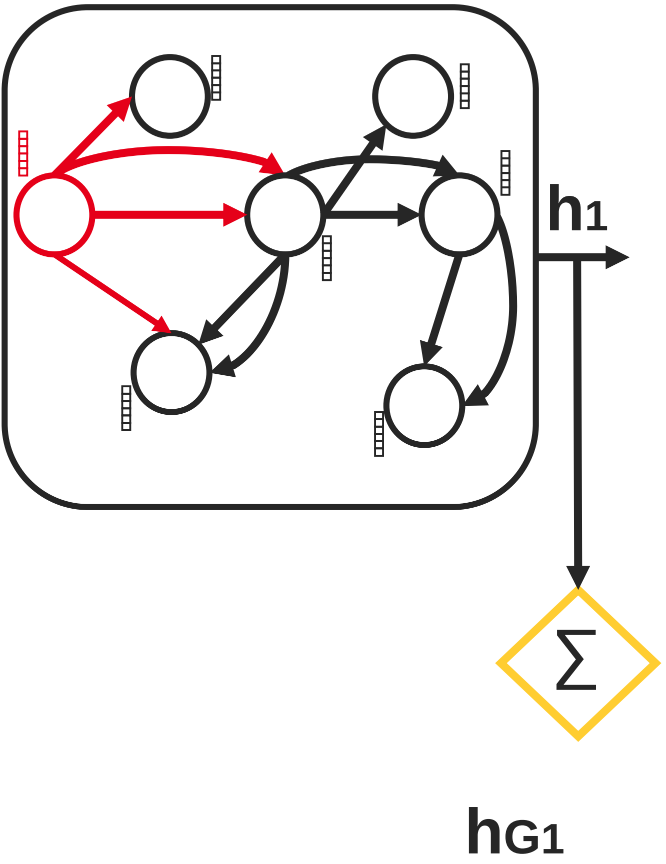
03

GENERATION

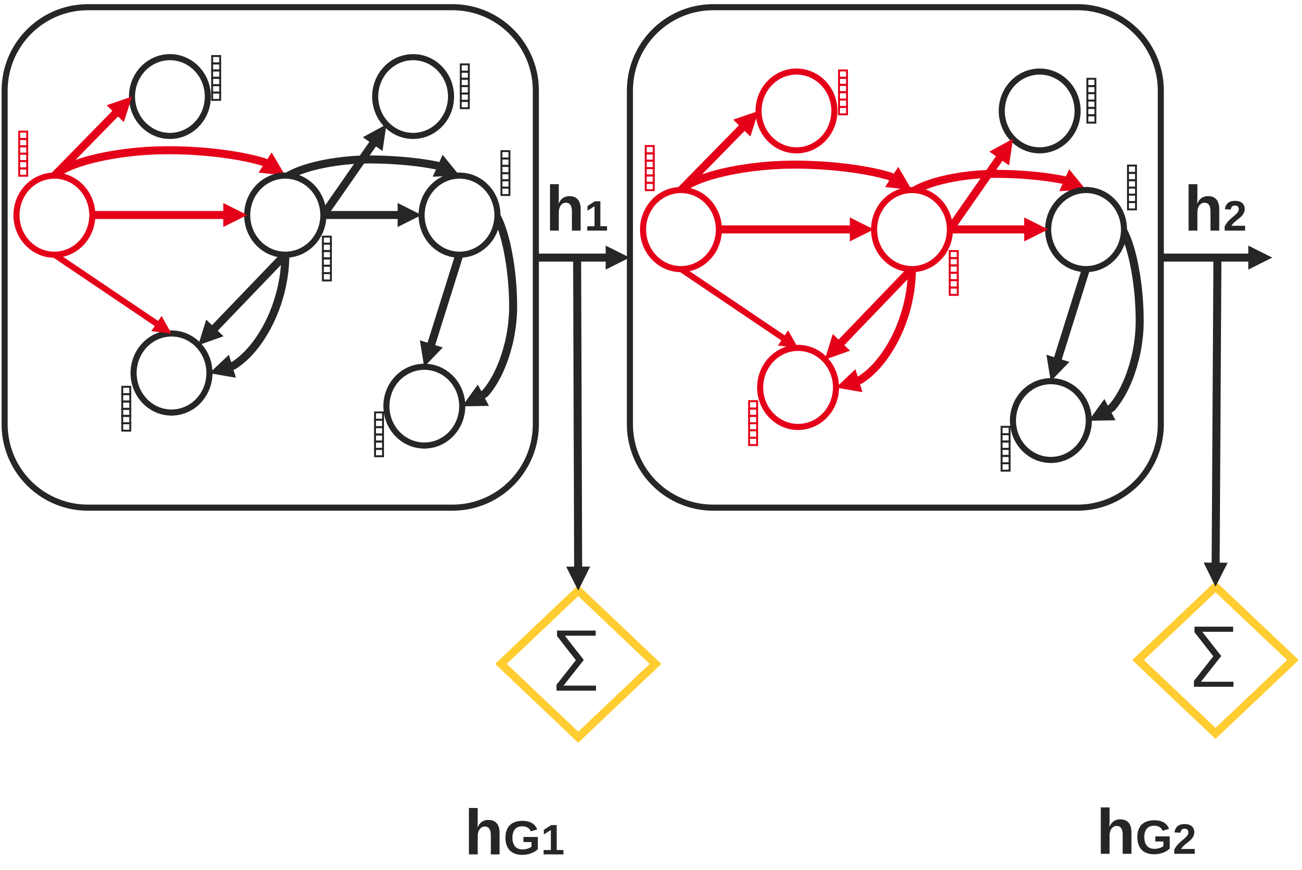
- SMOTE
- SMOTE+TOMEK

Slicing | Diff. Analysis | Emb. Construct | Data Sampling | Training

Model Training

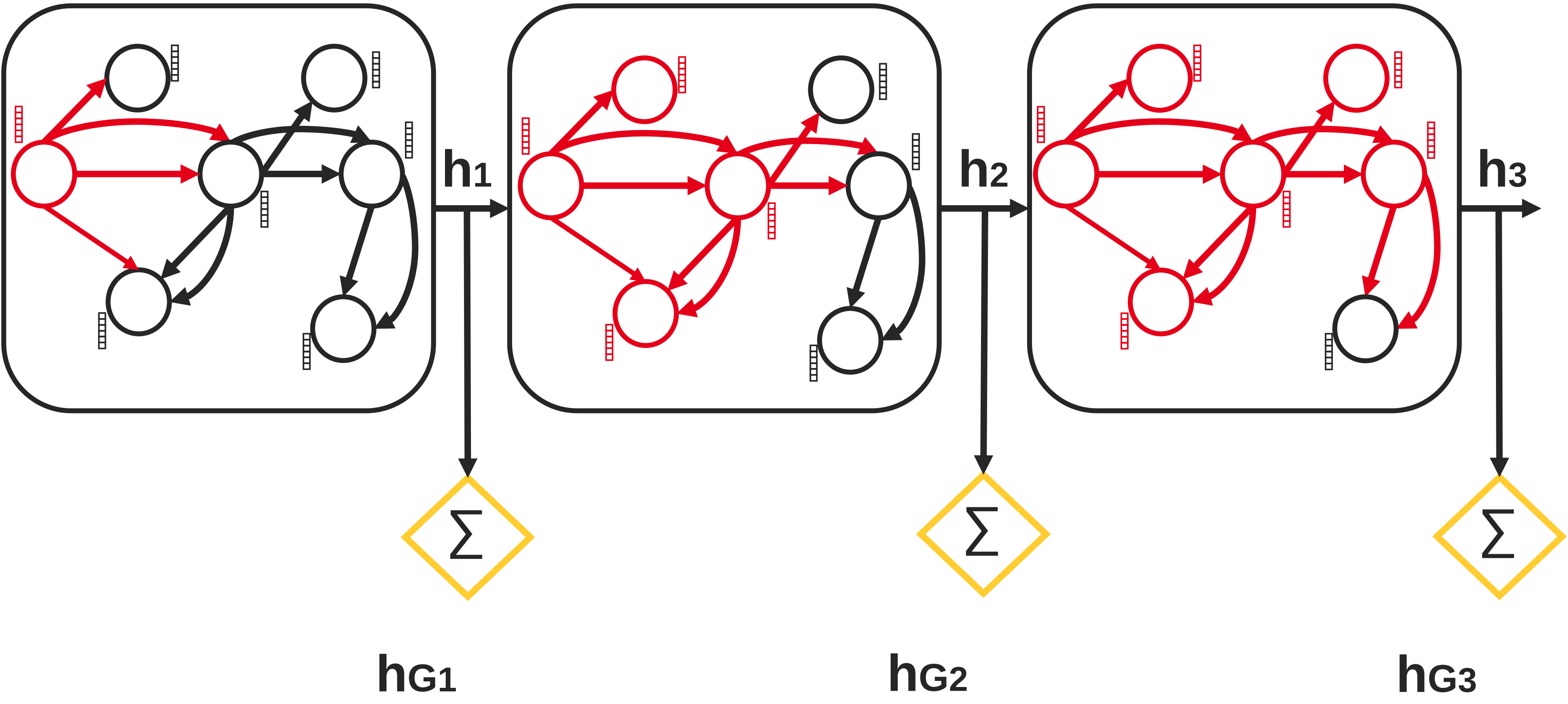


Model Training

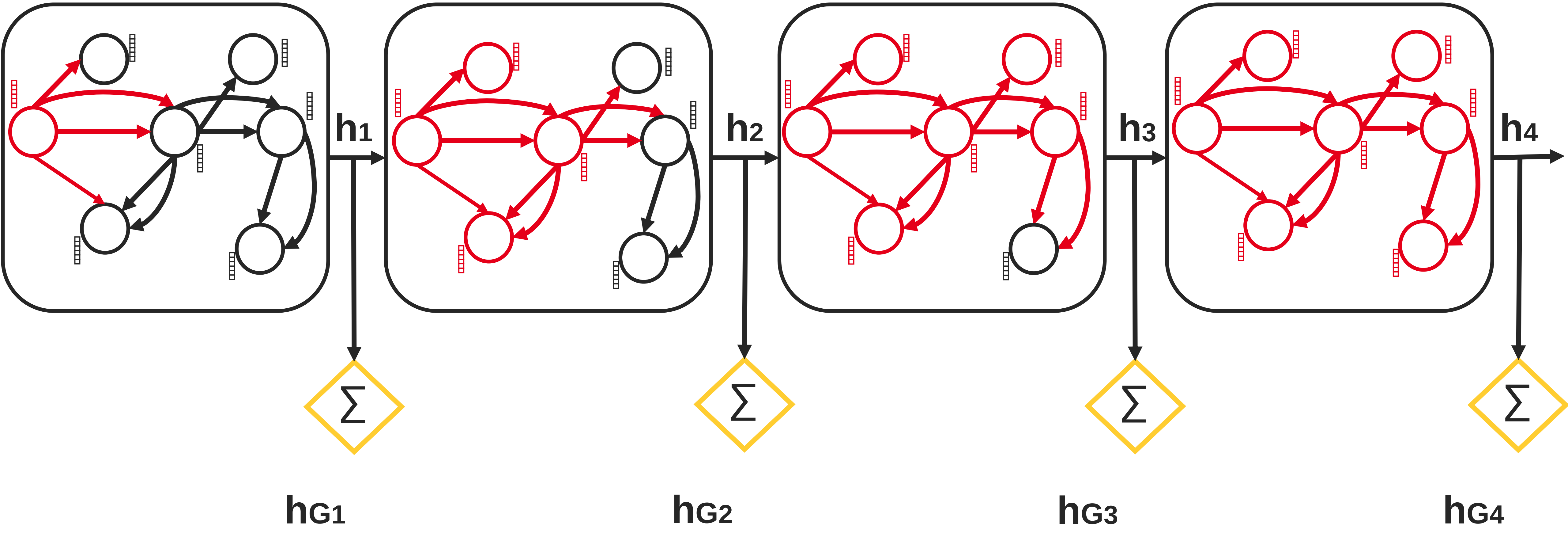


Slicing | Diff. Analysis | Emb. Construct | Data Sampling | Training

Model Training

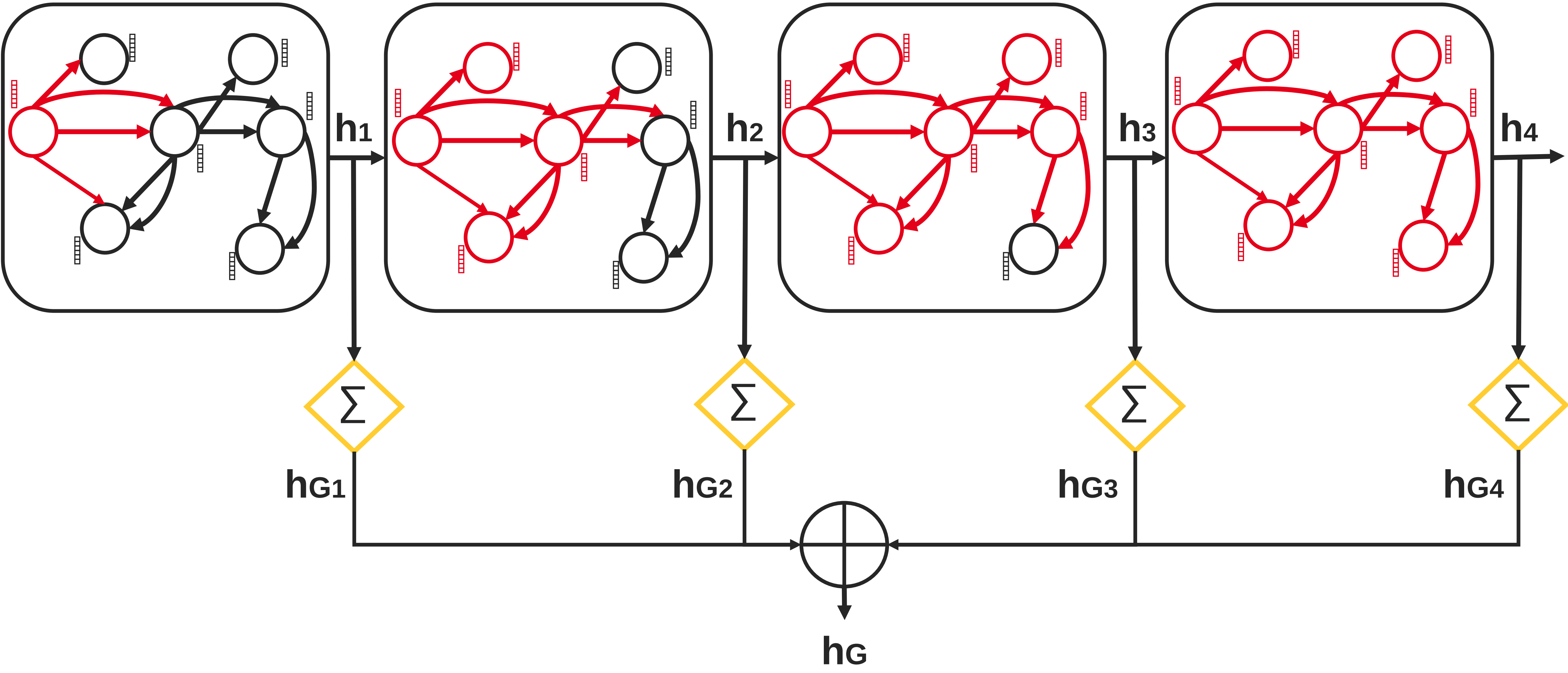


Model Training



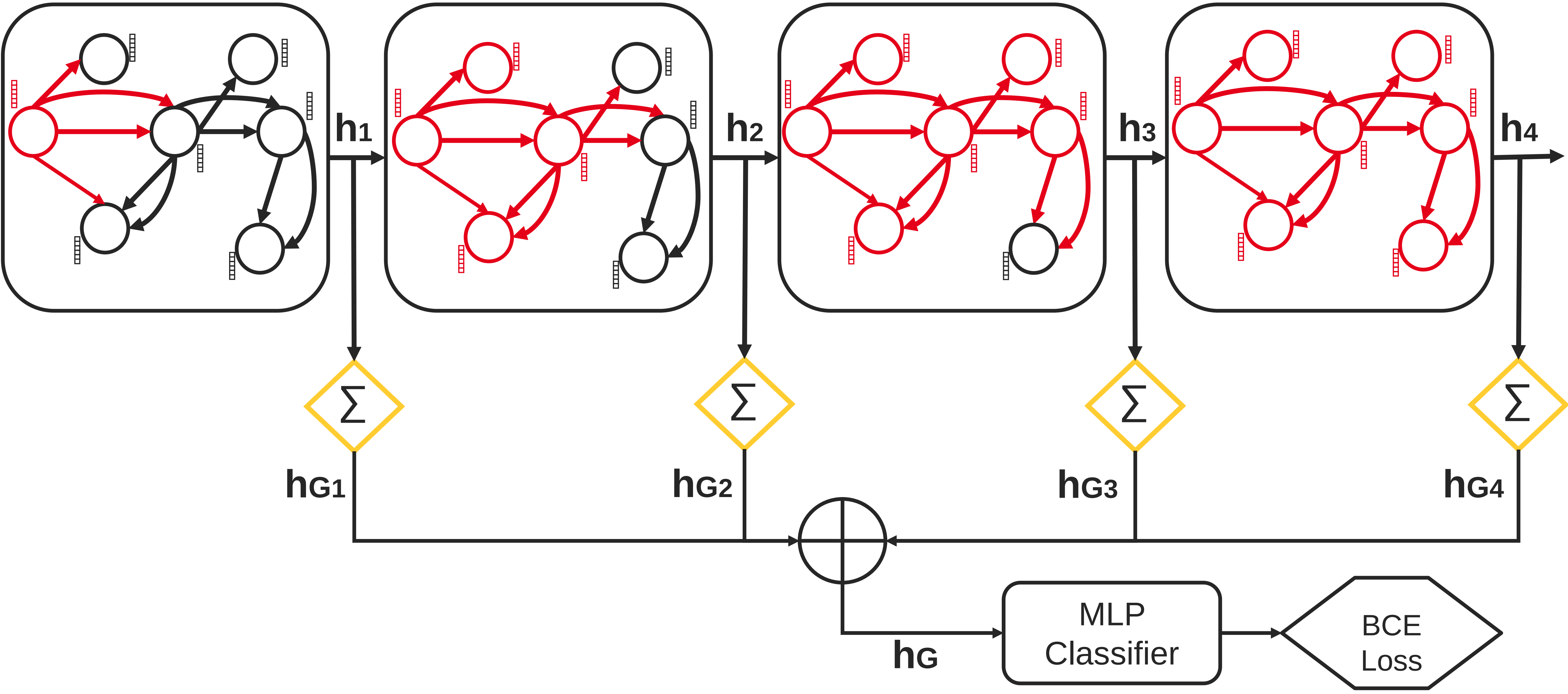
Slicing | Diff. Analysis | Emb. Construct | Data Sampling | Training

Model Training



Slicing
Diff. Analysis
Emb. Construct
Data Sampling
Training

Model Training



4. Results

Results

Family	Models Tested	Key Feature / Representation
Vector-based	XGBoost, SVM	Uses IR2Vec or handcrafted features (SPARROW) on linear code representations.
Code Language Models (CodeLLM)	GraphCodeBERT, CodeBERT	Transformer models pre-trained on code. Leverage pre-trained representations via fine-tuning.
Graph Neural Networks (GNN)	DGCNN, GAT, GIN, GINE	Programs represented CPG. Captures structural relationships in the code.

Results

TABLE I: Performance comparison of loop unrolling heuristic learning on preprocessed and real-world imbalanced datasets

Model	ANGHABENCH (Test Split)				OSCS			
	P / R	F1	F2	B-ACC	P / R	F1	F2	B-ACC
Random	8.4 / 50.3	14.3	25.1	50.1	11.5 / 48.2	18.6	29.5	49.2
XGBOOST- IR2VEC	28.3 / 33.5	30.7	32.3	62.7	54.9 / 14.7	23.2	17.2	56.3
SVM- SPARROW	8.2 / 88.6	15.0	29.9	49.2	7.2 / 41.3	12.3	21.3	35.1
GRAPHCODEBERT	20.4 / 81.8	32.7	51.1	76.6	23.3 / 80.7	36.1	54.0	66.2
DGCNN	24.3 / 79.3	37.4	54.8	78.7	21.7 / 50.8	26.6	34.70	58.3
GAT	22.8 / 73.6	35.6	52.2	76.7	21.8 / 68.7	31.1	51.27	66.7
GIN	11.1 / 77.1	34.8	52.1	76.9	23.1 / 49.1	33.4	48.95	67.0
GINE	26.8 / 81.6	**40.4**	**57.98**	**80.85**	26.9 / 79.1	**40.1**	**57.01**	**70.06**

Performance Benchmark

	Frama-C (RQ2)								
Project	$\bot$		$\top$		Alu-100		**Loupe (GINE)**		
	#	Time	#	Time	#	Time	#	Time	Alignment
ioccc									
mini-gmp									
genann									
papabench									
chrony									
kgflags									
libspng									
gnugo									
debie1									
basic-cwe-examples									
jsmn									
microstrain									
bench-moerman2018									
khash									
c-testsuite									
icpc									
solitaire									
line-following-robot									
safestringlib									
stmr									
powerwindow									
tweetnacl-usable									
qlz									
2048									
hiredis									
verisec									
c-utils									
Total**									

Performance Benchmark

	FRAMA-C (RQ2)								
Project	⊥		⊤		Alu-100		LOUPE (GINE)		
	#	Time	#	Time	#	Time	#	Time	Alignment
ioccc									
mini-gmp									
genann									
papabench									
chrony									
kgflags									
libspng									
gnugo									
debie1									
basic-cwe-examples									
jsmn									
microstrain									
bench-moerman2018									
khash									
c-testsuite									
icpc									
solitaire									
line-following-robot									
safestringlib									
stmr									
powerwindow									
tweetnacl-usable									
qlz									
2048									
hiredis									
verisec									
c-utils									
Total**									

⊥ **(Analyzer with No Unrolling):** The baseline configuration where no loops are unrolled.

⊤ **(Analyzer with All Unrolling):** The configuration where all loops in the program are fully unrolled.

 # ALU-100: Eva built in heuristic *-eva-auto-loop-unroll* with 100 as a factor

 # Loupe (GINE): Our approach with the best performing model

Performance Benchmark

	FRAMA-C (RQ2)								
Project	⊥		⊤		Alu-100		LOUPE (GINE)		
	#	Time	#	Time	#	Time	#	Time	Alignment
ioccc									
mini-gmp									
genann									
papabench									
chrony									
kgflags									
libspng									
gnugo									
debie1									
basic-cwe-examples									
jsmn									
microstrain									
bench-moerman2018									
khash									
c-testsuite									
icpc									
solitaire									
line-following-robot									
safestringlib									
stmr									
powerwindow									
tweetnacl-usable									
qlz									
2048									
hiredis									
verisec									
c-utils									
Total**									

 # Alarms: The number of alarms reported by the analyzer (Lower is better/more precise).

Time (s): The total analysis time in seconds (Lower is better).

Alignment: Loupe (GINE) predictions vs. expert annotations. Higher ratio (closer to 1/1) is better.

- **TO:** Timeout of 2700s
- **Bold values**: Best performing result w.r.t. metric
- **Underlined values**: Better than ⊥, but worse than ⊤
- ***:** Loupe (GINE) outperforms or matches Alu-100

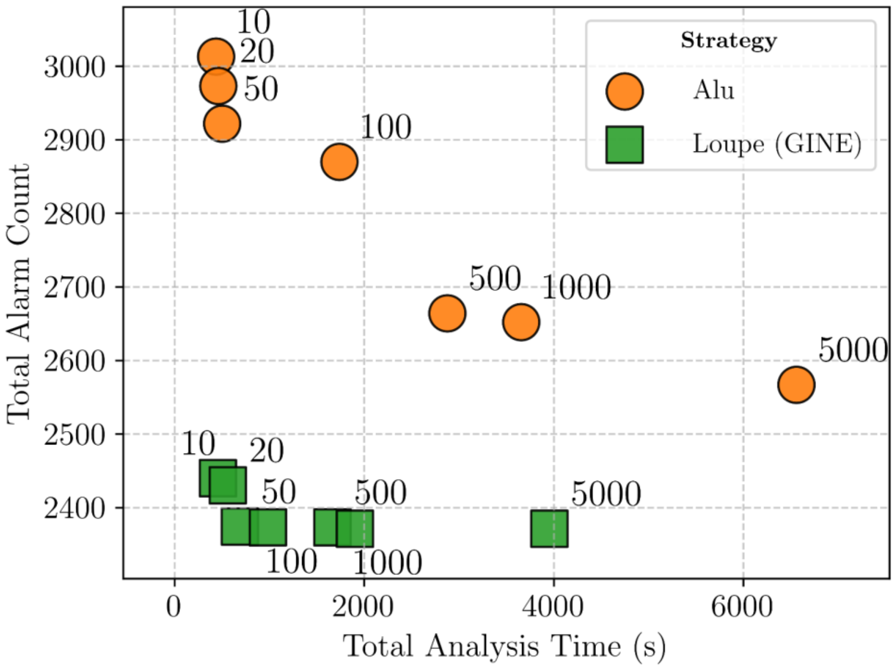
Performance Benchmark

	Frama-C (RQ2)								
Project	⊥		⊤		Alu-100		**Loupe (GINE)**		
	#	Time	#	Time	#	Time	#	Time	Alignment
ioccc	81	114.25	77	675.94	80	110.20	78*	127.56	
mini-gmp	64	1.88	64	**TO**	64	2.02	64*	1.45	
genann	235	3.38	136	58.76	232	4.18	169*	4.80	
papabench	40	5.15	40	54.92	40	5.15	40*	3.68	1/1
chrony		**TO**		**TO**		**TO**		**TO**	27/38
kgflags	4	4.69	4	976.12	4	4.80	4*	102.05	
libspng	237	11.17	233	44.59	237	11.58	237*	8.43	
gnugo	114	21.84	114	**TO**	114	**TO**	**100***	27.22	4/4
debie1	31	39.54	9	1158.30	20	81.53	11*	110.77	5/6
basic-cwe-examples	1	1.62	1	1.17	1	1.69	1*	0.70	
jsmn	68	20.83	2	12.54	68	21.14	**2***	66.35	
microstrain	1287	36.51	1287	**TO**	699	25.61	**700**	19.35	
bench-moerman2018	3	39.29	3	28.88	3	41.49	3*	34.74	
khash	1	0.13	0	0.05	1	0.12	1*	0.03	1/1
c-testsuite	0	34.32	0	**TO**	0	**TO**	0*	23.49	
icpc	1	3.17	1	7.72	1	3.10	1*	2.30	
solitaire	182	2.31	182	**TO**	173	4.17	**173***	18.46	6/6
line-following-robot	2	0.94	2	4.22	2	0.94	2*	0.64	
safestringlib	987	39.94	987	**TO**	845	69.19	**453***	267.48	
stmr	67	5.21	67	602.19	67	5.38	67*	4.94	
powerwindow	1	18.18	1	67.33	1	18.17	1*	14.27	
tweetnacl-usable	99	3.26	3	26.32	4	25.54	56	4.96	2/2
qlz	26	11.13	26	**TO**	26	39.02	26*	130.15	
2048	30	0.99	10	55.19	13	4.54	12*	1.18	5/5
hiredis	248	110.69	210	244.97	224	101.75	241	92.39	
verisec									
c-utils									
Total**	3809	3230.488	3459	28319,22	2919	8681,39	**2442***	3766,84	51/63 (81%)

- ✓ Matches approximately **67%** of the alarms reduced by ⊤
- ✓ **7.5x** faster than ⊤
- ✓ **5 TO** less than ⊤
- ✓ Reduce around **35%** of alarms compared to ⊥
- ✓ Outperforms or matches Alu-100 **87%** of the time with almost half the time.
- ✓ Aligns with **51 out of 63** manually labeled unrolling decisions (81%)

Loupe (GINE) VS Alu

Let's examine how the learned heuristic Loupe (GINE) stacks up against the built-in heuristic (Alu) across varying unrolling factors.

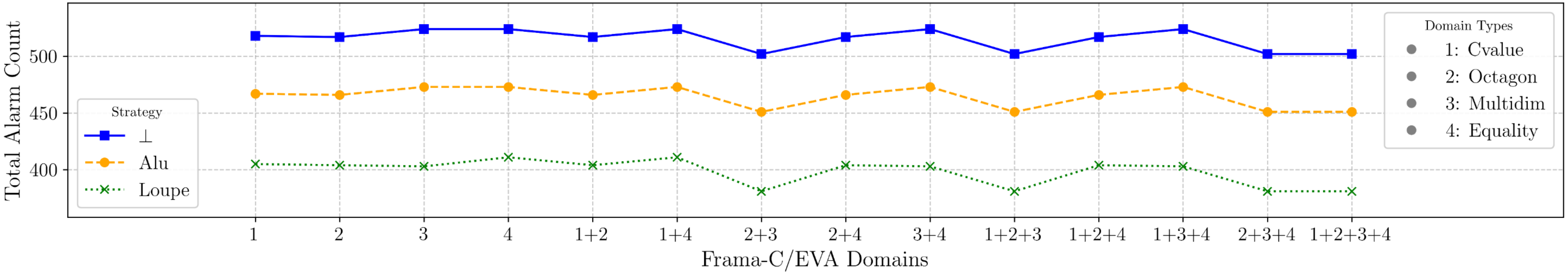


- ✓ Increasing precision (low alarms) costs a massive, **non-linear increase** in time for Alu
- ✓ The learned model achieves **consistent efficiency** while maintaining high precision.
- ✓ Loupe (GINE) **selectively** unrolls required loops.
- ✓ Loupe (GINE) provides the **superior balance**, beating the built-in heuristic even at its extreme settings.

Consistency w.r.t Abstract Domains

Let's examine the consistency of our approach and how its alarm reduction trends perform across different abstract domain settings

- ✓ Alarm reduction trends remain **uniform** and predictable
- ✓ Loupe (GINE) **consistently outperforms** the built-in Alu heuristic in total alarm count on all configurations
- ✓ The absence of anomalies confirms the approach's **robustness and scalability** across varied configurations.



Transferability

Mopsa (RQ3)							
⊥		⊤		Loupe (GINE)			
#	Time	#	Time	Frama-C #	Time	Mopsa #	Time
398	5.75	337	1877.12	386	338.83	383	489.98
15	1.60	15	1.82	15	7.90	15	1.67
5700	20.59	5700	58.65	5700	47.02	5700	43.74
3	0.04	3	0.14	3	0.03	3	0.04
42	1.18	40	2.22	**40**	1.00	**40**	1.00
64	0.36	61	1.12	64	0.25	64	0.24
392	21.32	213	18.04	249	16.21	245	15.98
19	0.34	19	**TO**	19	0.26	19	0.25
156	2.58	65	432.85	117	3.22	112	3.26
1	0.25	1	0.51	1	0.18	1	0.51
729	1.03	729	**TO**	**692**	194.98	**692**	567.24
55	0.42	55	46.74	55	0.32	55	0.35
11	0.28	11	0.62	11	0.26	11	0.21
142	3.68	68	9.16	106	3.78	**68**	8.76
1113	11.52	793	68.55	974	11.18	853	39.25
380	2.87	375	33.26	377	17.79	**375**	22.00
9220	73.88	8485	7950.86	8809	636.93	8636	1194.5

- ✓ Loupe (GINE) trained on Frama-C/EVA data **improved** performance when applied to MOPSA.
- ✓ Retraining Loupe (GINE) for MOPSA achieved **75%** of ⊤'s alarm reduction.
- ✓ The MOPSA-specific model was **6.5x** faster than the ⊤ strategy.

Merci

Maykel Mattar[1,2], Michele Alberti[1], Valentin Perrelle[1], Salah Sadou[2]

1. Université Paris-Saclay, CEA, List, Palaiseau, France
2. Université Bretagne Sud, IRISA, Vannes, France

maykel.mattar@cea.fr
maykel.mattar@univ-ubs.fr