BINENHANCE: An Enhancement Framework Based on External Environment Semantics for Binary Code Search

Yongpan Wang

frankile@sjtu.edu.cn

Co-authors: Hong Li, Xiaojie Zhu, Siyuan Li, Chaopeng Dong, Shouguo Yang, Kangyuan Qin

中国科学院信息工程研究所
INSTITUTE OF INFORMATION ENGINEERING,CAS

中国科学院大学
University of Chinese Academy of Sciences

KAUST

CONTENTS

1. Background

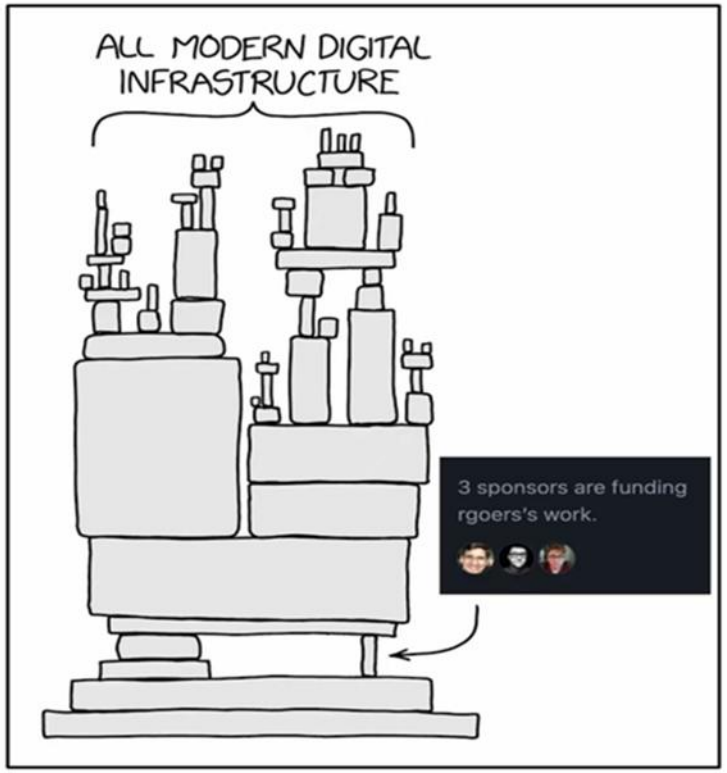
- Software development often reuses open-source code to reduce costs.
  - 96% of software uses open-source code, with 89% relying on versions over four years [1].
  - 84% of software has at least one known vulnerability, and 200+ new open-source vulnerabilities are found every day [1].
- The extensive workload of code auditing and the complexity of recursi ve code reuse results in substantial delays in vulnerability patching.
  - some software systems experiencing an average delay of 352 days [2].

[1] “Synopsys risk 2023 analysis open report.” source security and https://www.synopsys. com/software-integrity/resources/analyst-reports/ open-source-security-risk-analysis.html, 2024.
[2] C. Dong, S. Li, S. Yang, Y. Xiao, Y. Wang, H. Li, Z. Li, and L. Sun, “Libvdiff: Library version difference guided oss version identification in binaries,” in 2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE). IEEE Computer Society, 2023, pp. 780–791.

1. Background

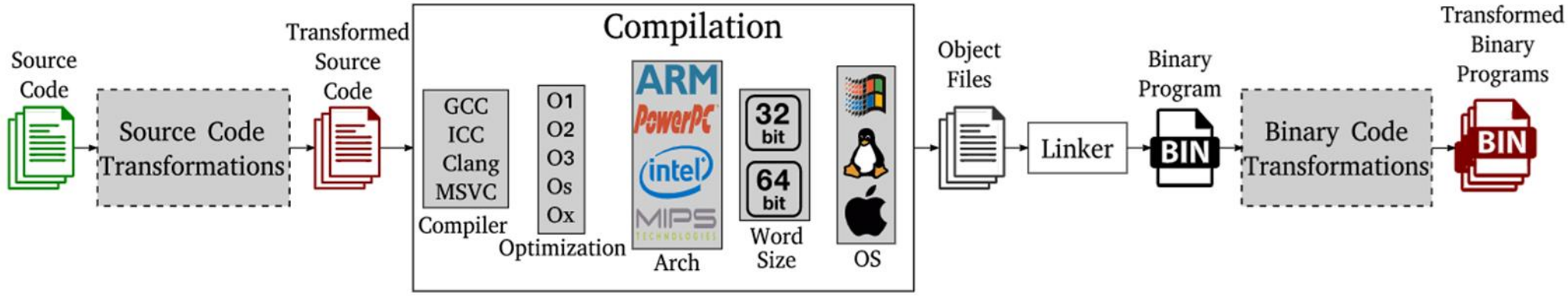
- Exploiting 1-day vulnerabilities in reused code has become a highly effective, low-cost, and large-scale attack method for attackers, posing severe risks.

- **binary code search has been proven to be a powerful method for automating the detection of insecure software components.**



1. Background

- Binary code search entails the meticulous analysis of numerous binary codes to identify the most similar ones.
- However, the syntactic structure of binary code can vary dramatically due to different compiler settings.
  - Binary codes with similar syntactic structures may have different semantics.

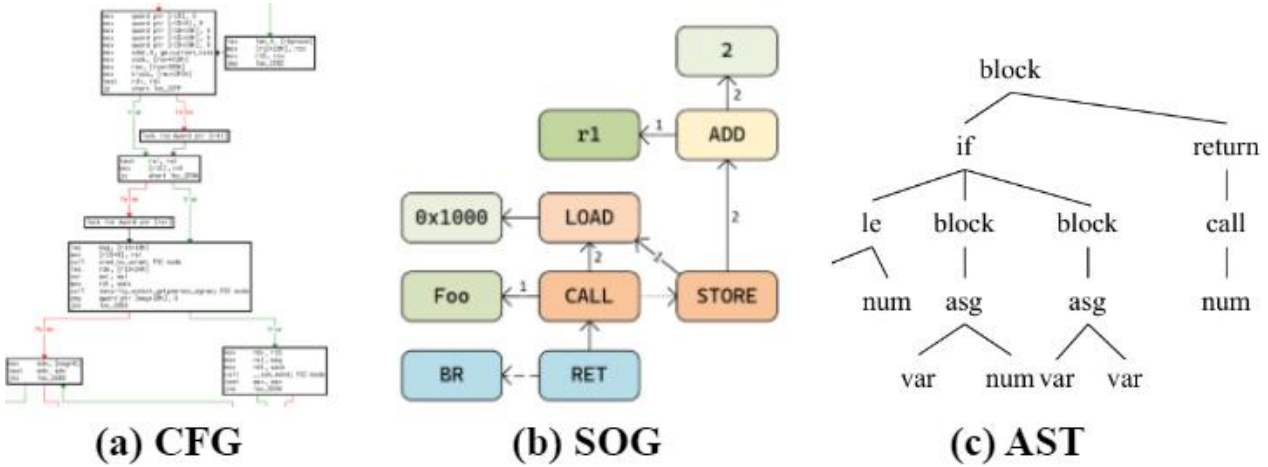


C/C++ source-to-binary compilation process[1]

[1] Haq, Irfan ul and Juan Caballero. “A Survey of Binary Code Similarity.” ACM Computing Surveys (CSUR) 54 (2019): 1 - 38.Haq, Irfan ul and Juan Caballero. “A Survey of Binary Code Similarity.” ACM Computing Surveys (CSUR) 54 (2019): 1 - 38.

2. Motivation

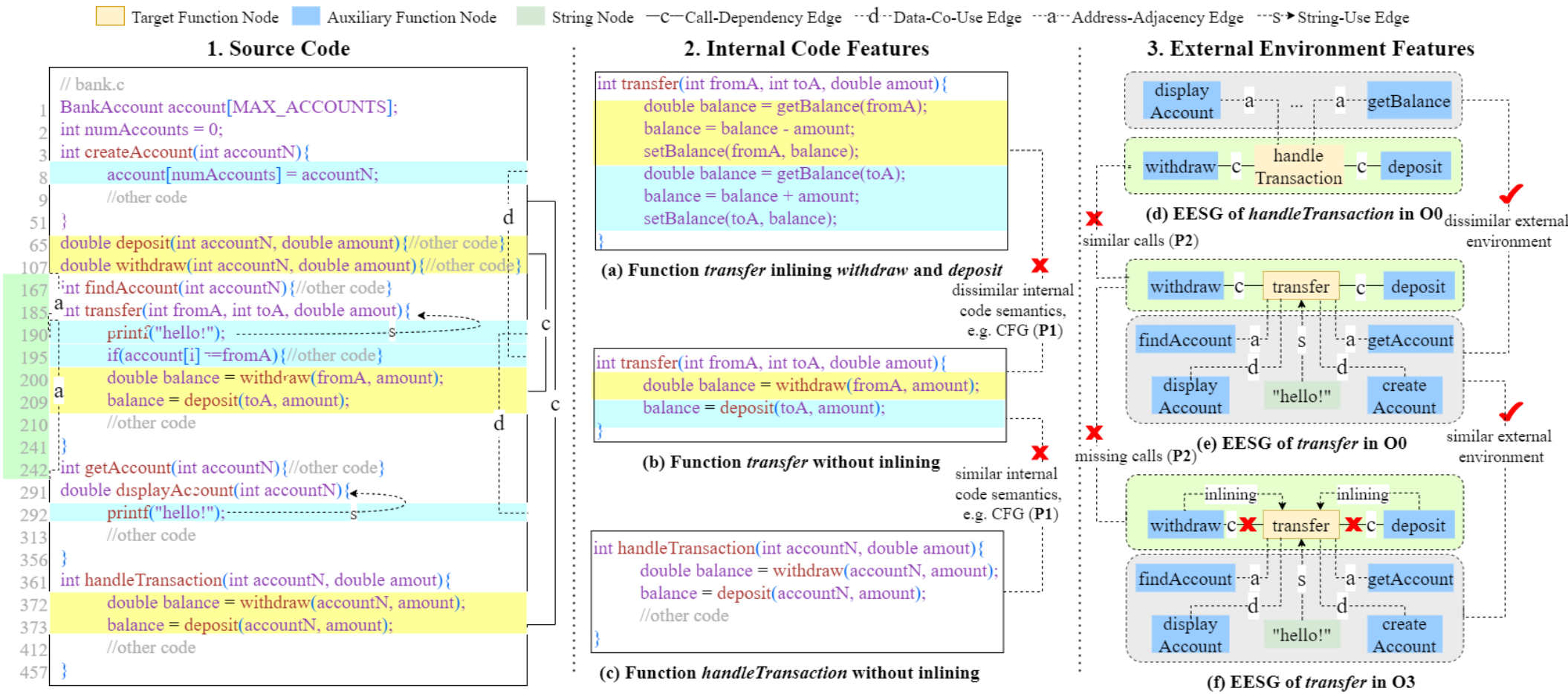
- Based on different perspectives, we classify them into two categories.
  - **Internal code semantics:** focuses on the function itself and is derived from both the binary code embedded within the function or from its derivatives.



(a) CFG (b) SOG (c) AST

  - **External environment semantics:** focuses on inter-function and is inferred from other supplementary functions (in the code segment) or and data (in the data segment).
  - In this paper, we present Call-Dependency, Data-Co-Use, Address-Adjacency, and String-Use as novel external environment semantics in our work.

2. Motivation

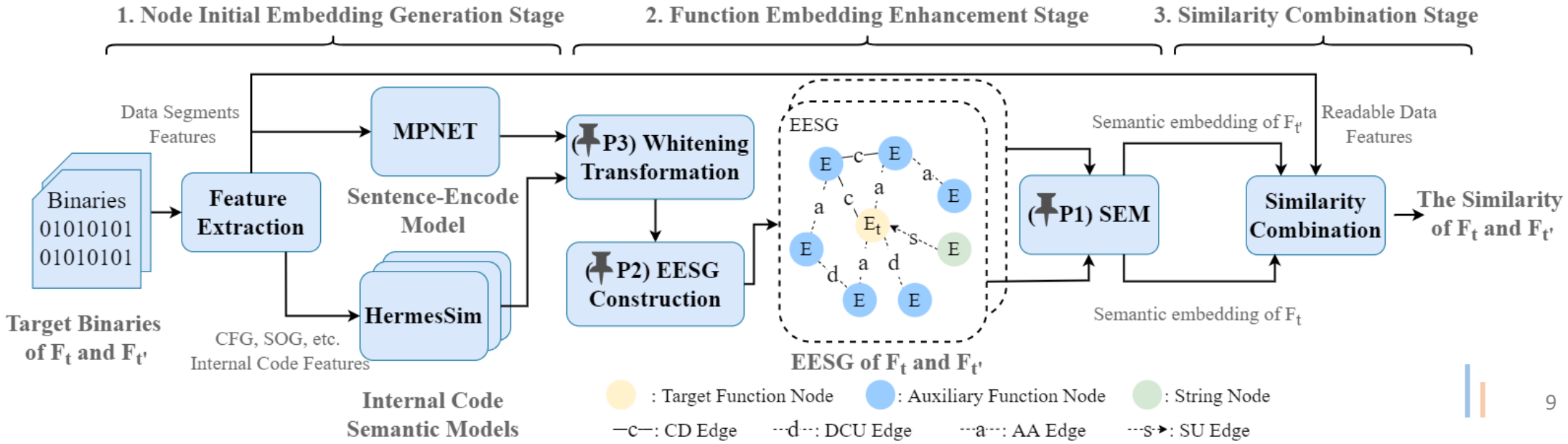


2. Motivation

- Problem 1 (**P1**): internal code semantics of functions may exhibit subs-tanti al variations due to different compilation settings, encompassing factors like function inlining and splitting.
  - compiler-caused function inlining can reach up to 70%
- Problem 2 (**P2**): exclusive reliance on function call graphs (CG) for assistan ce is insufficient for addressing complex real-world scenarios.
  - Missing calls
  - Similar calls and so on
- Problem 3 (**P3**): current solutions exhibit limited scalability and struggle to cope with large-scale function search tasks.
  - 768 dimension of function embeddings and large costs of retrain

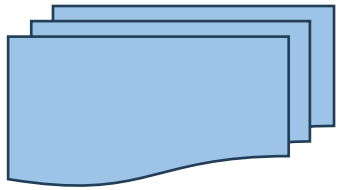
3. Design

- To solve these problems, we propose a general enhancement framework named BinEnhance for binary code search.
  - Node Initial Embedding Generation Stage
  - Function Embedding Enhancement Stage
  - Similarity Combination Stage

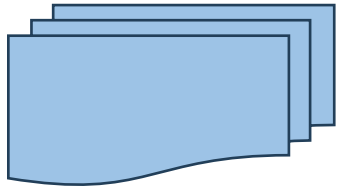


3. Design

- Node Initial Embedding Generation Stage
  - Function node embeddings: generated by internal code semantics model
  - String node embeddings: generated by sentence-transformer model MPNET



Function Node
Initial Embeddings



String Node
Initial Embeddings

Algorithm 1: Whitening Transformation Method

Input: Node Embeddings X Target Dimension d_t
Output: Whitening Embeddings E

Function *Whitening_Transformation*$(X,\ d_t)$**:**

$\mu = \frac{1}{n}\sum_{i=1}^{n} X_i;$

$Cov = \frac{1}{n-1}\sum_{i=1}^{n} (X_i - \mu)(X_i - \mu)^T;$

$V, D, V^T = SVD(Cov);$

$W = VD^{-\frac{1}{2}}V^T[:, : d_t];$

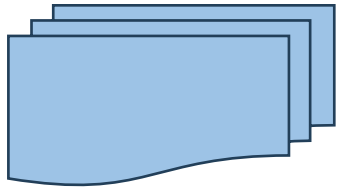
for $i = 1$ *to* n **do**

$E_i = (X_i - \mu)W;$

$E_i = \frac{E_i}{\sqrt{\sum_{j=1}^{d_t} E_{i,j}^2}};$

return E;

end



All Node
Initial Embeddings

3. Design

- Function Embedding Enhancement Stage
  - External Environment Semantic Graph (EESG) Construction
    - Node
      - Function Node
      - String Node
    - Edge
      - Call-Dependency (CD) Edge
      - Data-Co-Use (DCU) Edge
      - Address-Adjacency (AA) Edge
      - String-Use (SU) Edge

- ◆ CD Edge: two functions exist call/called relations
- ◆ DCU Edge: two functions share/use same data
- ◆ AA Edge: two functions are positionally adjacent
- ◆ SU Edge: function node uses a specific string node

3. Design

- Function Embedding Enhancement Stage
  - External Environment Semantic Graph (EESG) Construction Algorithm

Algorithm 2: EESG Construction Algorithm

Input: Feature Sets s_e, Function Sets s_u, Target Function f_t and Max Depth md
Output: the EESG of f_t

```
1  eesg, ns, es ← NULL;
2  Function EESG_Construct(s_e, s_u, f_t, md):
3      nss ← f_t;
4      eesg.addNode(f_t, type = function);
5      ns.add(f_t);
6      for d = 0 to md do
7          nes ← NULL;
8          for n ∈ nss do
9              cd, dcu, aa, su ← GetExternal(s_u, s_e, n);
10             nes.add(BuildEdges(n, cd, 0, 1, function));
11             nes.add(BuildEdges(n, aa, 2, 3, function));
12             nes.add(BuildEdges(n, dcu, 4, 4, function));
13             BuildBiEdges(n, su, 5, 5, string);
14         nss ← nes
15     return eesg;
16 end
```

```
17 Function BuildEdges(src, des_f, r_1, r_2, node_type):
18     n ← NULL;
19     for des ∈ des_f do
20         e ← (src, des, type = r_1);
21         if des ∉ ns then
22             essg.addNode(des, type = node_type);
23             n.add(des);
24             ns.add(des);
25         if e ∉ es then
26             if node_type == function then
27                 eesg.addEdge(src, des, type = r_1);
28                 es.add(src, des, type = r_1);
29             eesg.addEdge(des, src, type = r_2);
30             es.add(des, src, type = r_2);
31     return n;
32 end
```

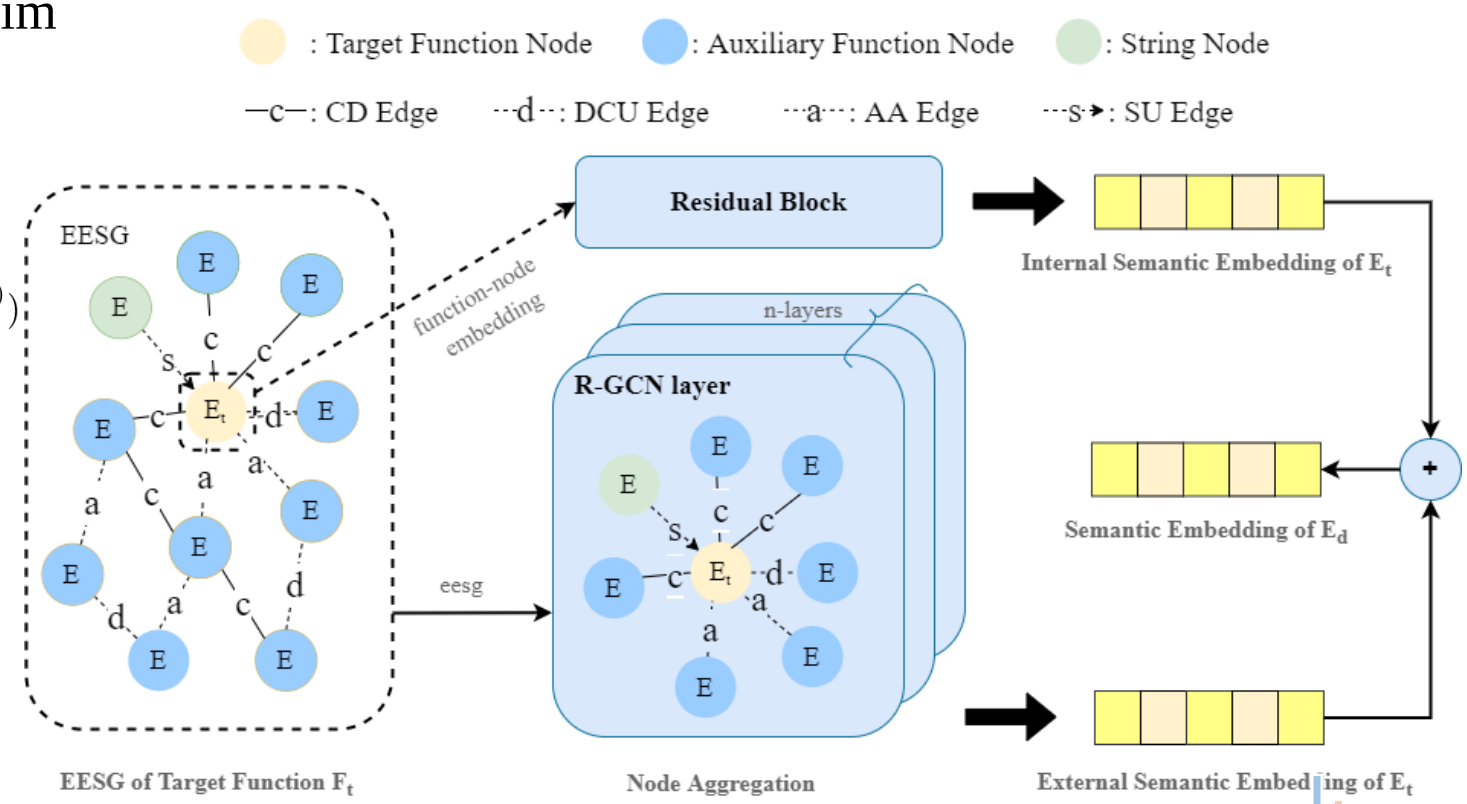
3. Design

- Function Embedding Enhancement Stage
  - Semantic Enhancement Model (SEM)
    - For example: HermesSim
  - RGCN layer:

$$E_i^{(l+1)} = LeakyRelu(\sum_{r \in R} \sum_{j \in N_i^r} \frac{1}{c_{i,r}} W_r^{(l)} h_j^{(l)})$$

  - SEM:

$$E_t = RGCN(F_t) + Resisual\_block(HermesSim(F_t))$$



3. Design

- Similarity Combination Stage
  - Sim_{cos}: Semantic Embedding Cosine Similarity of two functions

$$Sim_{cos} = Cosine\ (Embed_1, Embed_2)$$

  - Sim_{data}: Jaccard Similarity of readable data features ($F_{m/n}$) of two functions

$$Sim_{data} = \frac{|F_m \cap F_n|}{|F_m \cup F_n|}$$

$$Sim = Tanh(FFN(Concat(Sim_{cos}, Sim_{data})))$$

- Loss function

$$Loss = \frac{1}{m}\sum_{m}\frac{1}{2}(1 - Sim_p)^2 + \frac{1}{2}(1 + Sim_n)^2$$

4. Evaluation

- Benchmarks
  - Dataset D1: from Asteria, 2,751,667 functions
  - Dataset D2_norm: from BinKit, 1,654,864functions, normal compilation
  - Dataset D2_noinline: from BinKit, 1,991,864functions, inlining is prohibited
  - Dataset D3_firmware: 37 firmware images from 8 vendors

TABLE I: The details of three public datasets and one firmware dataset

Dataset	architectures	options	Functions	Applications	Projects	Source
D1	ARM, X86, X64	O0-O3	2,751,667	RQ1, RQ2	260	Asteria
D2_norm	ARM, MIPS, X86	O0-O3	1,654,805	RQ1, RQ2, RQ3, RQ4, RQ5	51	BinKit
D2_noinline	ARM, MIPS, X86	O0-O3	1,991,864	RQ3	51	BinKit
D3_firmware	ARM, MIPS, X86	UNKNOWN	6,817(binaries)	RQ6	37	Real-world

- Baselines: Gemini, Asm2vec, Asteria, TREX, HermesSim
- Metric: Mean Average Precision (MAP)

4. Evaluation

- Research questions
  - RQ1: How much could be improved when BINENHANCE is applied to baseli nes (including HermesSim, Asteria, Asm2vec, TREX, and Gemini)?
  - RQ2: Is BINENHANCE robust against different compiler optimization option s and architectures?
  - RQ3: Does BINENHANCE effectively alleviate the impact of function inlinin g in binary code search?
  - RQ4: What is the contribution of each part in BINENHANCE?
  - RQ5: Does BINENHANCE improve the efficiency of baselines?
  - RQ6: What is the performance of BINENHANCE in detecting 1-day vulnerabi lities in real-world firmware?

4. Evaluation

- RQ1: different function pool size (2, 16, 32, …, 8192, 10000)

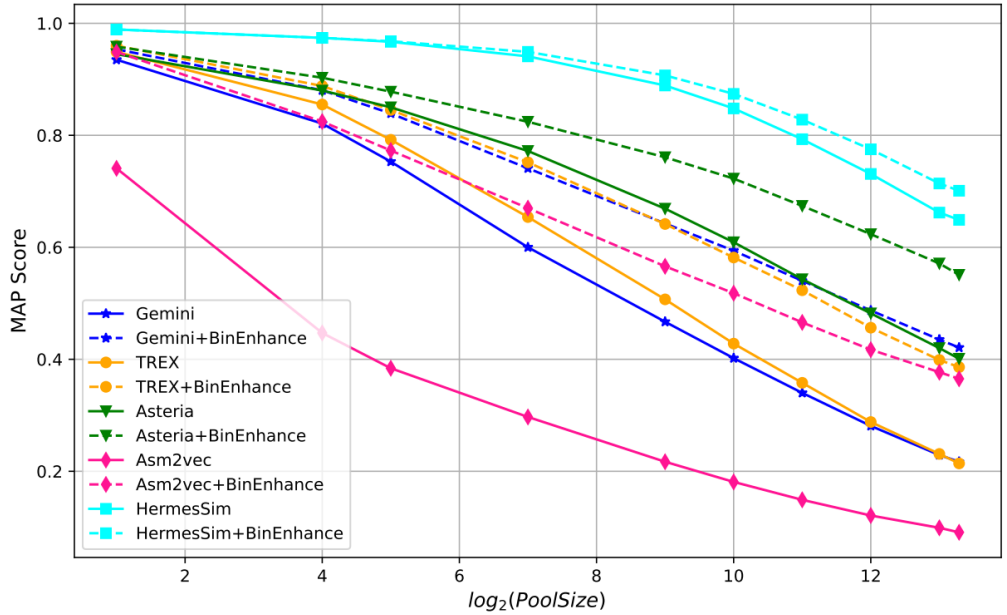
TABLE III: MAP scores of different methods of binary code search tasks in different function pool sizes.

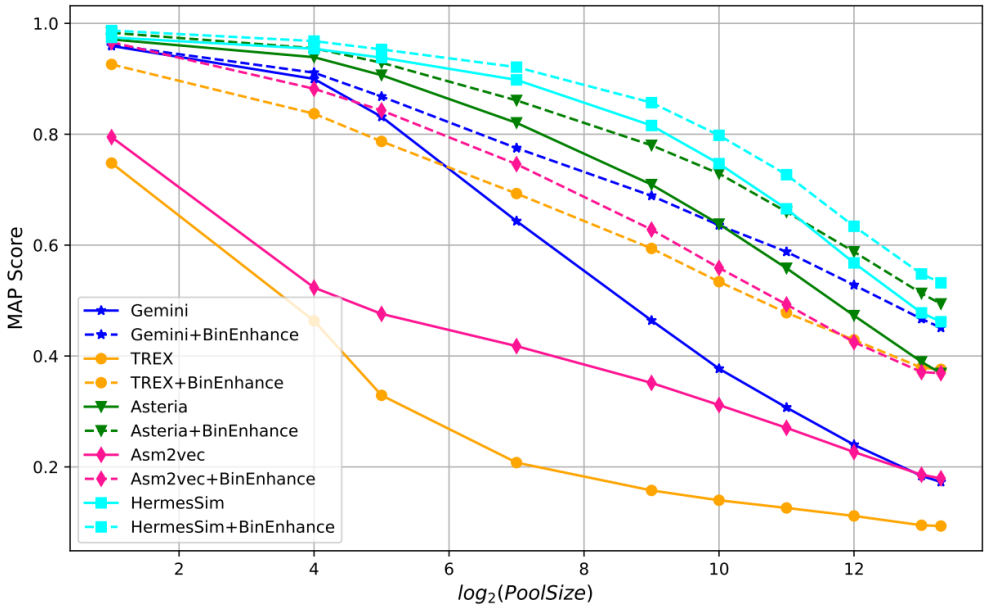
Models	*D*1						*D2_norm*					
	P=2	**P=128**	**P=1024**	**P=4096**	**P=10000**	**Avg.**	**P=2**	**P=128**	**P=1024**	**P=4096**	**P=10000**	**Avg.**
Gemini	93.5	60.0	40.2	28.1	21.7	48.7	95.9	64.3	37.7	24.0	17.3	47.8
Gemini+BINENHANCE	**95.4**	**74.1**	**59.4**	**48.7**	**42.1**	**63.9**	**96.1**	**77.5**	**63.6**	**52.8**	**45.1**	**67.0**
TREX	94.9	65.4	42.8	28.8	21.4	50.7	74.8	20.8	14.0	11.2	9.3	26.0
TREX+BINENHANCE	**96.0**	**75.2**	**58.2**	**45.6**	**38.6**	**62.7**	**92.6**	**69.3**	**53.4**	**42.9**	**37.6**	**59.2**
Asm2vec*	74.1	29.7	18.1	12.1	9.1	28.6	79.5	41.8	31.2	22.7	17.9	38.6
Asm2vec*+BINENHANCE	**94.9**	**67.0**	**51.8**	**41.7**	**36.5**	**58.4**	**96.6**	**74.6**	**55.9**	**42.5**	**36.9**	**61.3**
Asteria	94.5	77.2	60.9	48.2	40.1	64.2	97.1	82.1	63.8	47.3	36.9	65.4
Asteria+BINENHANCE	**95.8**	**82.4**	**72.3**	**62.3**	**55.1**	**73.6**	**98.3**	**86.1**	**72.9**	**58.8**	**49.4**	**73.1**
HermesSim	98.9	94.1	84.8	73.1	64.9	83.2	97.5	89.8	74.7	56.8	46.2	73.0
HermesSim+BINENHANCE	**98.9**	**94.9**	**87.4**	**77.5**	**70.1**	**85.8**	**98.7**	**92.1**	**79.8**	**63.4**	**53.2**	**77.4**

∗ Asm2vec only support X86 architecture.

4. Evaluation

- Answer to RQ1
  - BINENHANCE demonstrates significant improvement across all the baselines, evidenced by the average increment in MAP scores on the two public dataset (16.1%, from 53.6% to 69.7%). Furthermore, its improvement for each baselin e is positively correlated with the size of the function pool.





4. Evaluation

- RQ2: Cross-architecture and Cross optimization option

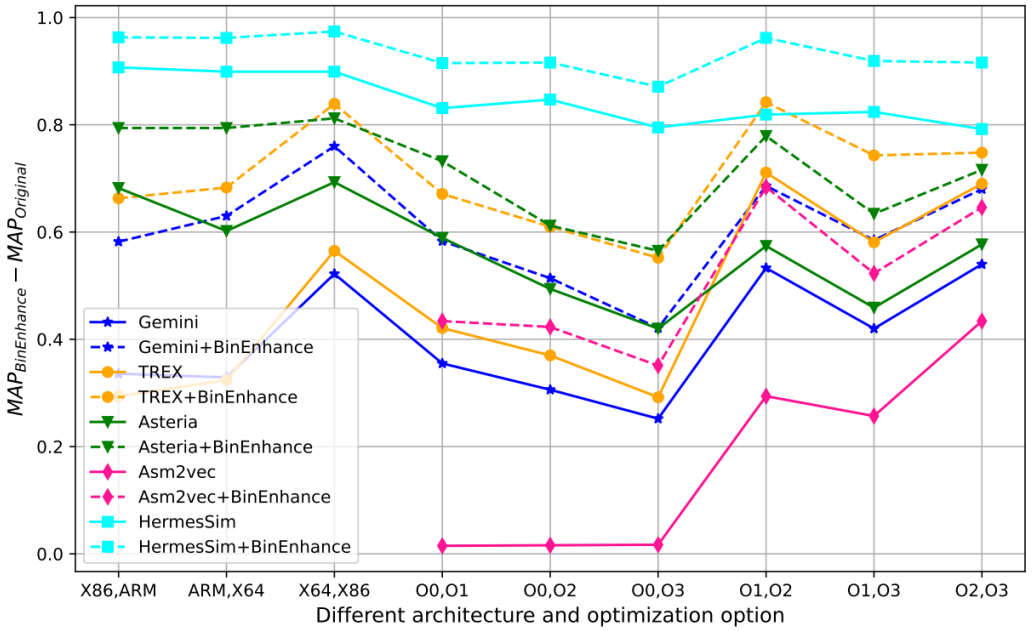
TABLE IV: MAP scores of different methods in different optimization options and architectures ($D1$).

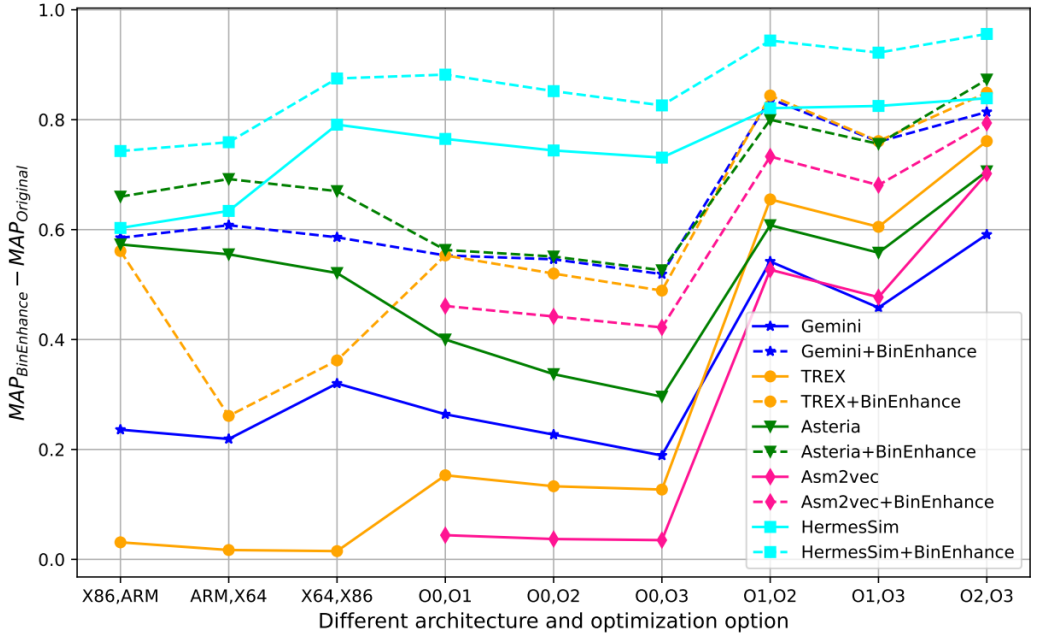
Models	Cross-architecture			Cross-optimization option					
	ARM,X86	ARM,X64	X64,X86	O0,O1	O0,O2	O0,O3	O1,O2	O1,O3	O2,O3
Gemini	33.6	32.9	52.2	35.5	30.6	25.2	53.3	42.0	54.0
Gemini+BINENHANCE	**58.2**	**63.0**	**76.0**	**58.3**	**51.4**	**42.1**	**68.6**	**58.5**	**68.0**
TREX	29.4	32.4	56.5	42.1	37.0	29.2	71.1	58.1	69.0
TREX+BINENHANCE	**66.3**	**68.3**	**83.9**	**67.1**	**61.0**	**55.2**	**84.2**	**74.3**	**74.8**
Asm2vec*	-	-	-	1.5	1.6	1.7	29.4	25.7	43.4
Asm2vec*+BINENHANCE	-	-	-	**43.4**	**42.3**	**35.1**	**68.4**	**52.3**	**64.6**
Asteria	68.2	60.2	69.3	58.9	49.4	42.0	57.4	45.9	57.7
Asteria+BINENHANCE	**79.4**	**79.4**	**81.2**	**73.2**	**61.2**	**56.5**	**77.9**	**63.4**	**71.6**
HermesSim	90.7	89.9	89.9	83.1	84.7	79.5	81.9	82.4	79.2
HermesSim+BINENHANCE	**96.3**	**96.2**	**97.4**	**91.5**	**91.6**	**87.1**	**96.2**	**91.9**	**91.6**

∗ Asm2vec do not support cross-architecture.

4. Evaluation

- Answer to RQ2
  - BINENHANCE stably enhances baselines across cross-architecture and cross-optimization option tasks, without succumbing to significant performance dips under various compilation settings.





4. Evaluation

- RQ3: D2_norm and D2_noinline

TABLE VI: MAP scores of different methods of binary code search tasks in inlining and non-inlining compilation settings.

Models	Normal-to-Normal						Normal-to-Noinline					
	P=2	P=128	P=1024	P=4096	P=10000	Avg.	P=2	P=128	P=1024	P=4096	P=10000	Avg.
Gemini	90.6	57.5	38.0	27.6	21.7	47.1	89.6	50.3	28.4	18.9	14.0	40.2
Gemini+BINENHANCE	**93.6**	**73.6**	**62.7**	**56.1**	**51.3**	**67.5**	**92.3**	**69.6**	**57.1**	**49.0**	**44.9**	**62.6**
TREX	73.0	24.3	16.3	13.4	11.9	27.8	71.9	22.1	12.7	9.6	8.1	24.9
TREX+BINENHANCE	**92.8**	**70.4**	**60.4**	**53.0**	**48.7**	**65.1**	**91.0**	**66.7**	**52.8**	**44.0**	**41.4**	**59.2**
Asm2vec*	86.7	40.7	30.2	23.8	19.3	40.1	11.2	6.0	3.0	2.3	1.7	4.8
Asm2vec*+BINENHANCE	**95.0**	**72.9**	**60.3**	**50.5**	**46.6**	**65.1**	**92.9**	**62.6**	**47.7**	**39.1**	**37.0**	**55.9**
Asteria	94.9	71.9	56.8	47.2	39.2	62.0	54.7	29.2	21.3	17.5	14.1	27.4
Asteria+BINENHANCE	**97.1**	**77.7**	**67.8**	**59.2**	**53.4**	**71.0**	**59.1**	**49.2**	**38.2**	**31.3**	**27.1**	**41.0**
HermesSim	93.9	85.3	74.5	62.7	53.3	73.9	92.7	84.5	72.0	57.6	48.6	71.1
HermesSim+BINENHANCE	**97.9**	**87.9**	**79.1**	**69.6**	**61.7**	**79.2**	**97.6**	**87.1**	**75.9**	**64.5**	**56.4**	**76.3**

∗ Asm2vec only support X86 architecture.

4. Evaluation

- Answer to RQ3
  - Function inlining leads to a substantial performance loss in the binary code sea rch task, but BINENHANCE mitigates the decline and improves the baseline model's ability to cope with optimization strategies such as function inlining.

MAP Score Decline Ratio

$$\frac{(MAP_{NI} - MAP_{NO})}{MAP_{NI}}$$

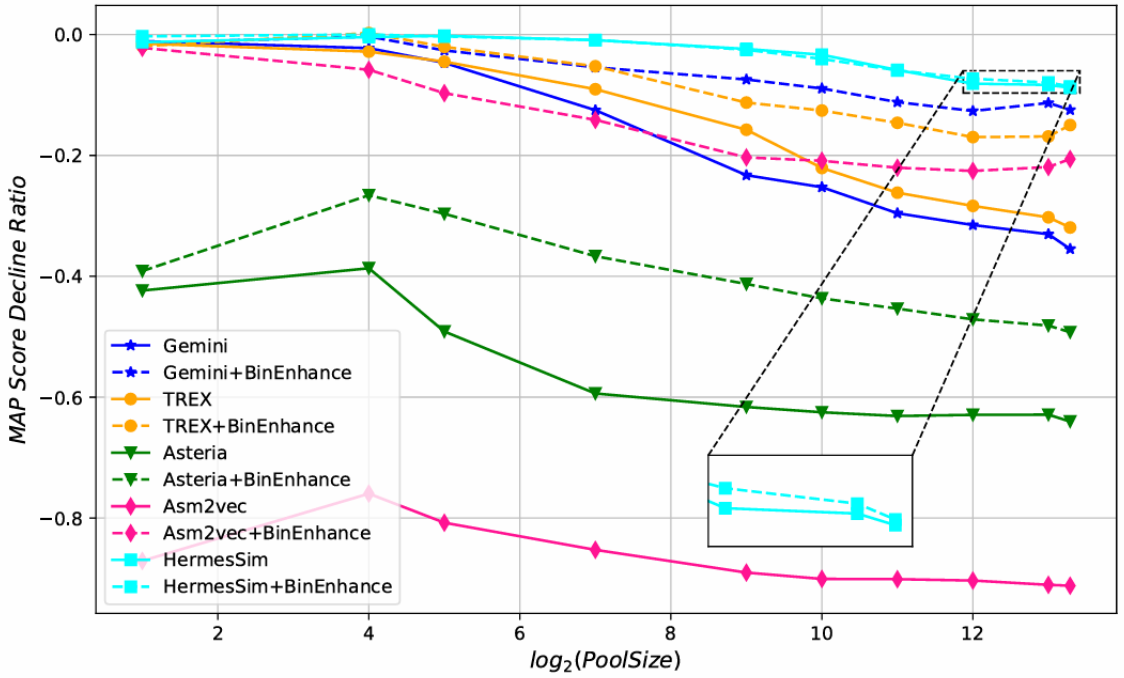


Fig. 9: MAP Score Decline Ratio in function inlining

4. Evaluation

- RQ4: Ablation Study

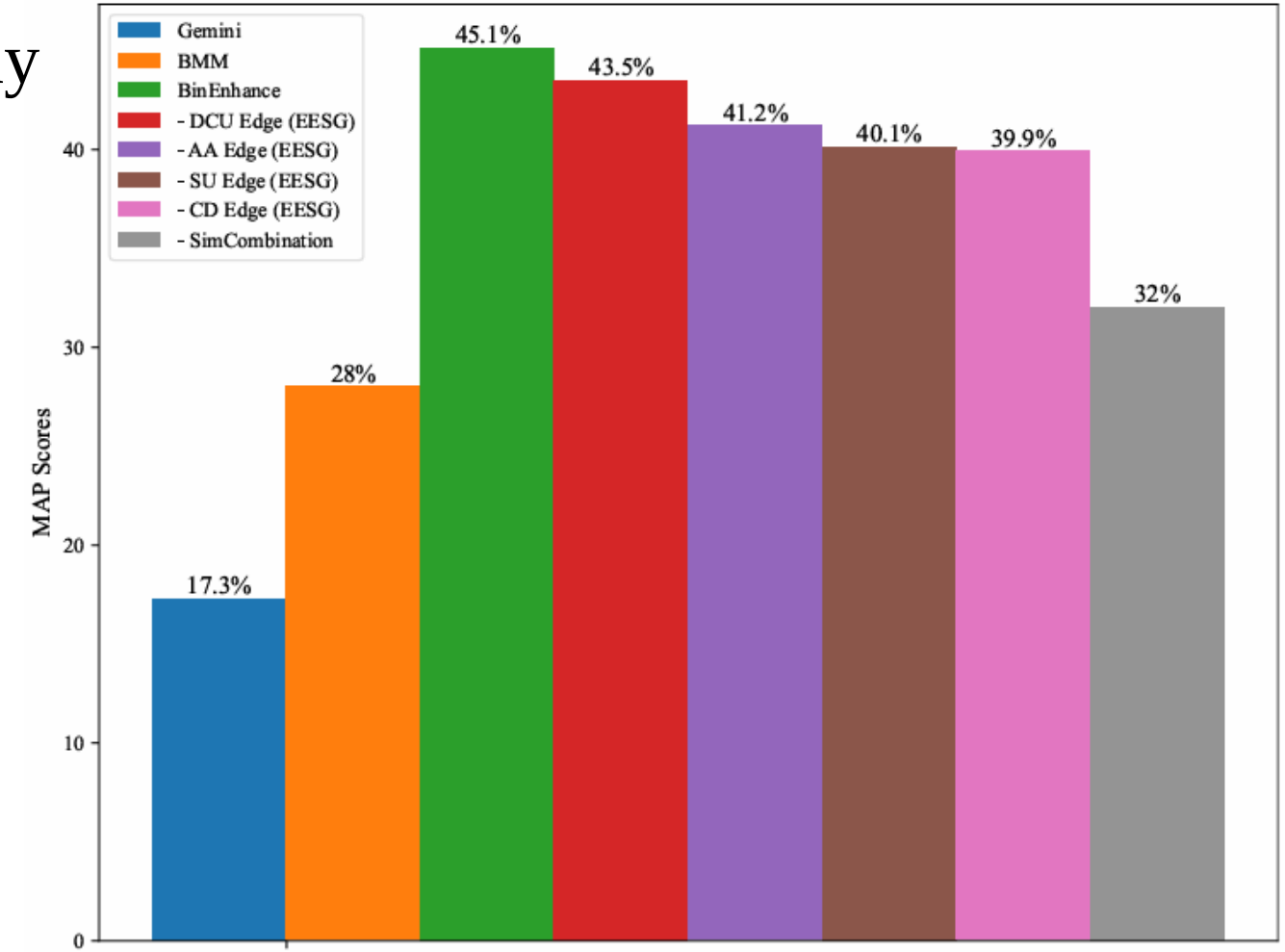


Fig. 10: MAP scores of different situations

4. Evaluation

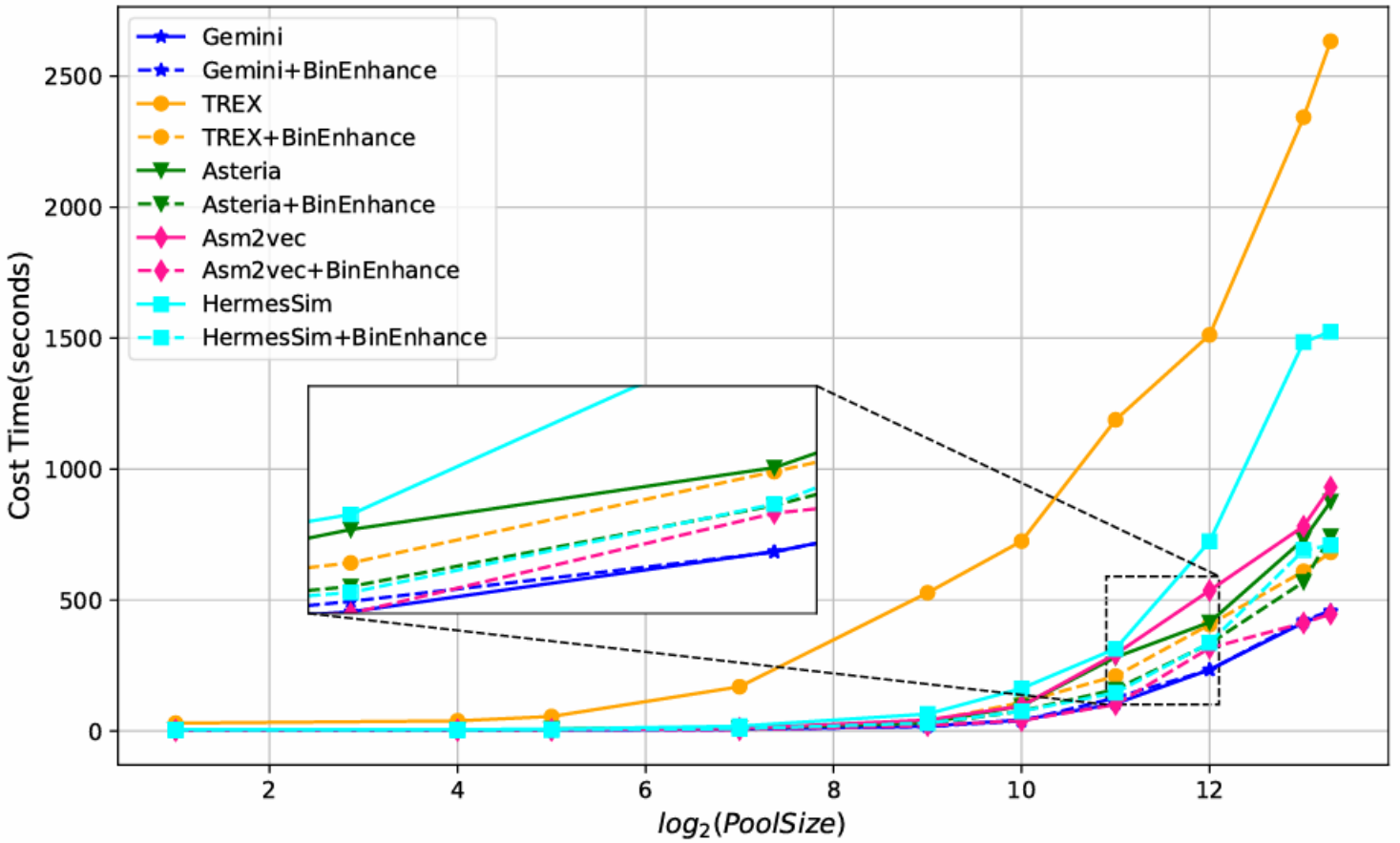
- Answer to RQ4
  - Each component of BINENHANCE plays a crucial role in the final result, and the absence of any one of them can lead to a degradation in the performance of the binary search task.

4. Evaluation

- RQ5: Efficiency Evaluation

TABLE II: Function Embedding dimensions

Model	Original model	+BINENHANCE
Gemini	64	64
Asm2vec	200	64
Asteria	150	128
TREX	768	128
HermesSim	384	128



4. Evaluation

- Answer to RQ5
  - The additional time cost of BINENHANCE is in training and generating functi on embeddings, but it significantly reduces the time for binary code search tas ks, resulting in an overall lower time cost compared to the original model.

4. Evaluation

- RQ6: 1-day Vulnerability Search

TABLE VIII: Results of vulnerability search (*fail to recall*).

ID	CVE	Tot.	Gemini	Asteria	TREX	HermesSim	Ours
1	2014-4877	20	3	19	14	0	0
2	2016-8622	9	9	8	7	8	2
3	2016-6301	10	10	0	10	0	0
4	2016-8618	12	11	11	6	6	3
5	2018-19519	6	6	6	6	0	0
6	2018-1000301	1	0	0	0	0	0
7	2018-16230	7	4	4	6	3	3
8	2018-16452	20	1	19	19	2	1
9	2018-16451	3	3	2	3	2	2
10	2020-8306	20	1	18	18	4	2
11	2021-22924	3	2	3	1	2	2
12	2022-0778	5	5	5	5	0	0
	MAP	-	14.2	26.8	15.2	60.2	**67.9**

4. Evaluation

- Answer to RQ6
  - BINENHANCE identified 101 1-day vulnerabilities in 37 firmware images wit h 67.9% MAP scores, which is 12 more vulnerabilities detected than HermesSi m, and a MAP score of 7.7% higher.

5. Conclusion

- We propose a binary code search enhancement framework BINENHANCE, which enhances internal code semantic models with valuable external environment semantic information, thereby reducing the false positive and false negative.
  - Design an EESG to resolve Problem 2
  - Propose a SEM to resolve Problem 1
  - Use whitening transformations to resolve Problem 3
  - We implement prototype BINENHANCE
  - The evaluation shows the performance of BINENHANCE

Q & A

Yongpan Wang

Contact: frankile@sjtu.edu.cn

Co-authors: Hong Li, Xiaojie Zhu, Siyuan Li, Chaopeng Dong, Shouguo Yang, Kangyuan Qin