BinaryAI: Binary Software Composition Analysis via Intelligent Binary Source Code Matching

Ling Jiang[*], Junwen An[*], Huihui Huang[*], Qiyi Tang[†], Sen Nie[†], Shi Wu[†], Yuqun Zhang[*]

*Southern University of Science and Technology

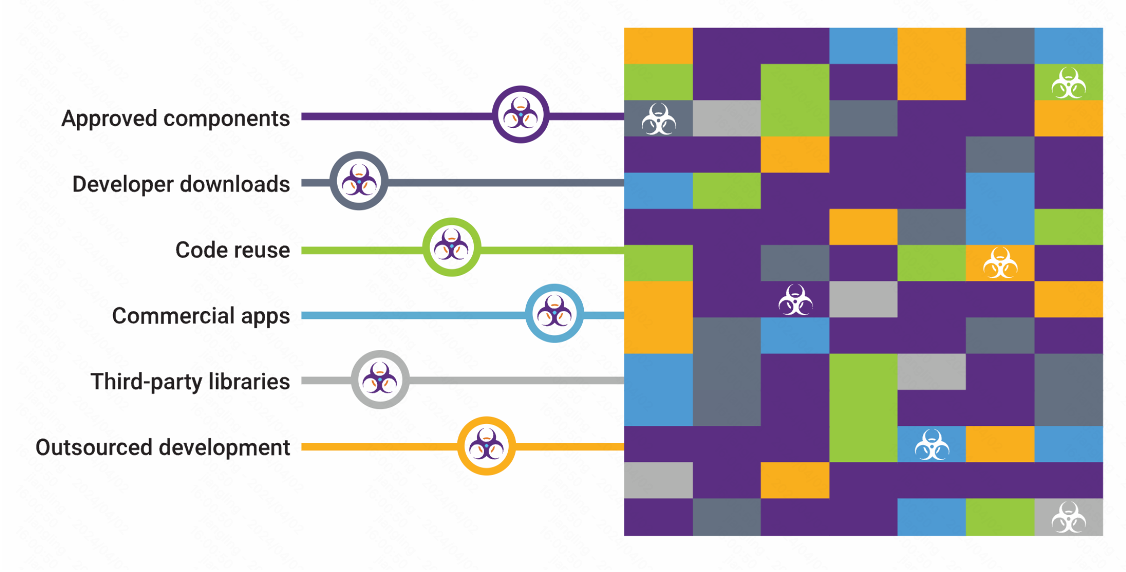
†Tencent Security Keen Lab

SUSTech Southern University of Science and Technology

科恩实验室 KEEN SECURITY LAB

Software Composition Analysis (SCA)

- Identifying open-source third-party libraries (TPLs) contained in software artifacts via **Code Clone Detection** with TPL dataset, integrated into modern DevSecOps
- Tracking potential license violations or 1-day security risks introduced by TPLs for the defense of supply-chain attacks
- E.g. SSHD backdoor in xz/liblzma-v5.6.0/5.6.1, CVE-2024-3094



Binary Software Composition Analysis

Binary-to-Binary SCA

- TPLs in the SCA database are stored in **binary format** built from source packages
- Existing techniques (e.g., LibDB, ModX) apply binary code similarity analysis (BCSA), where **deep neural network models** are integrated to embed **binary functions** for measuring code similarity
- **Limitations:** poor scalability of TPL dataset due to intricacies of automatic compilation (100 in ModX vs. 10K+ in Source SCA)

Binary-to-Source SCA

- TPL dataset consists of large-scale crawled open-source C/C++ **source projects**
- Existing techniques (e.g., OSSPolice, B2SFinder) select **basic syntactic features** that remain consistent after compilation (e.g., **string literals**) to match source code
- **Limitations:** ineffective binary source code matching based on basic syntactic features due to substantial disparities introduced by compilation

🤔 Can we employ fine-grained **function-level features** to include high-level semantic information in binary-to-source SCA?

Overview of BinaryAI

We propose **BinaryAI** to perform **function-level** binary source code matching for **binary-to-source SCA**, available as a **SaaS product.**

Basic Info

Interactive analysis

Binary-to-Source Function Matching

bbe34331e5068d7dc5b990fbef10002358b4ef8e07ab92c0d5620ed60fc36b30

example.strip

File size	Upload timestamp	Last analyzed	Type
858.2 KB	2023-02-01 19:10:20	2024-02-20 16:42:35	application/x-sharedlib

Composition | Details | ASCII string | Checksec | Properties

**Basic Info*

Components (Total items: 5)

Professional

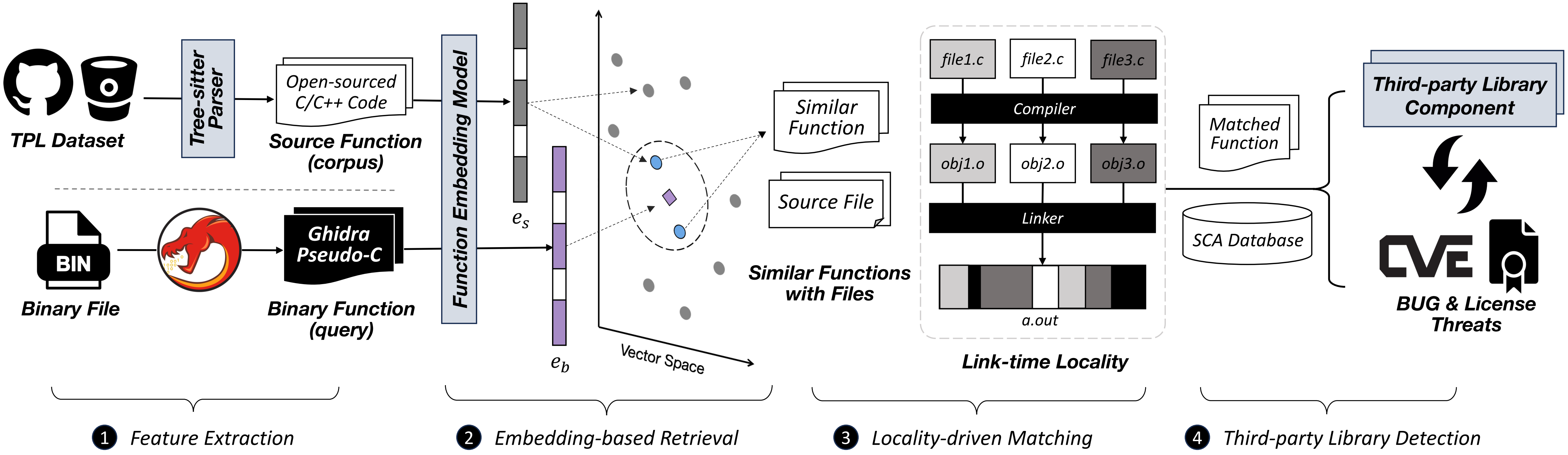
Third-party Libraries

Component name	Component version	Summary	Source code URL
cjson	v1.5.0	Ultralightweight JSON parser in ANSI C	https://github.com/DaveGamble/cJSON/tree/v1.5.0
libsodium	1.0.18-RELEASE	A modern, portable, easy to use crypto library.	https://github.com/jedisct1/libsodium/tree/1.0.18-RELEASE
mbed_tls	mbedtls-2.23.0	An open source, portable, easy to use, readable and fle...	https://github.com/ARMmbed/mbedtls/tree/mbedtls-2.23.0
paho.mqtt.embedded-c	v1.1.0	Paho MQTT C client library for embedded systems. Pah...	https://github.com/eclipse/paho.mqtt.embedded-c/tree/v1.1.0
st-device-sdk-c	v1.7.0	SmartThings SDK for Direct Connected Devices for C	https://github.com/SmartThingsCommunity/st-device-sdk-c/tr...

Overview of BinaryAI

Feature Extraction

- Source Function: **56M+** unique C/C++ functions from **12K+** open-source TPL projects† across all versions
- Binary Function: real-time decompilation with Ghidra to generate C-like pseudo-code representation



† All TPLs BinaryAI can detect with inverted index stored in SCA database by 2024.3.
We deploy continuous supplementation of new TPLs and source functions.

Embedding-based Function Retrieval

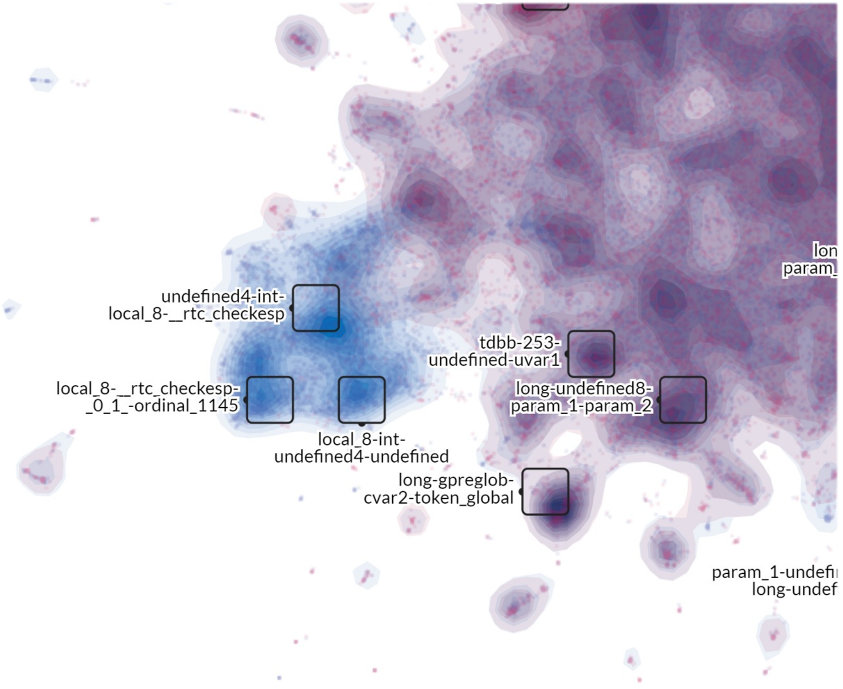
- Code representation learning for aligning binary and source functions in a single vector space
- Identify similar **token-based syntactic features** across different code formats (i.e., binary-to-source) based on the decoder-only autoregressive language model
- **Contrastive learning** with labeled binary source function pairs to further pre-train the base model acting as the function encoder to generate embeddings†

```
char* FUN_00153d7b(void) {
  char* pcVar1;
  pcVar1 = getenv("GOOGLE_LOG_DIR");
  if (((pcVar1 == (char*)0x0) ||
       (*pcVar1 == '\0')) &&
      ((pcVar1 = getenv("TEST_TMPDIR"),
        pcVar1 == (char*)0x0 ||
      (*pcVar1 == '\0')))) {
    pcVar1 = "";
  }
  return pcVar1;
}
```

Binary Function (Ghidra Pseudo-C)

```
static const char* DefaultLogDir() {
  const char* env;
  env = getenv("GOOGLE_LOG_DIR");
  if (env != NULL && env[0] != '\0') {
    return env;
  }
  env = getenv("TEST_TMPDIR");
  if (env != NULL && env[0] != '\0') {
    return env;
  }
  return "";
}
```

Source Function (C/C++)



† Embeddings for all 56M source functions are derived offline and stored to the vector database.

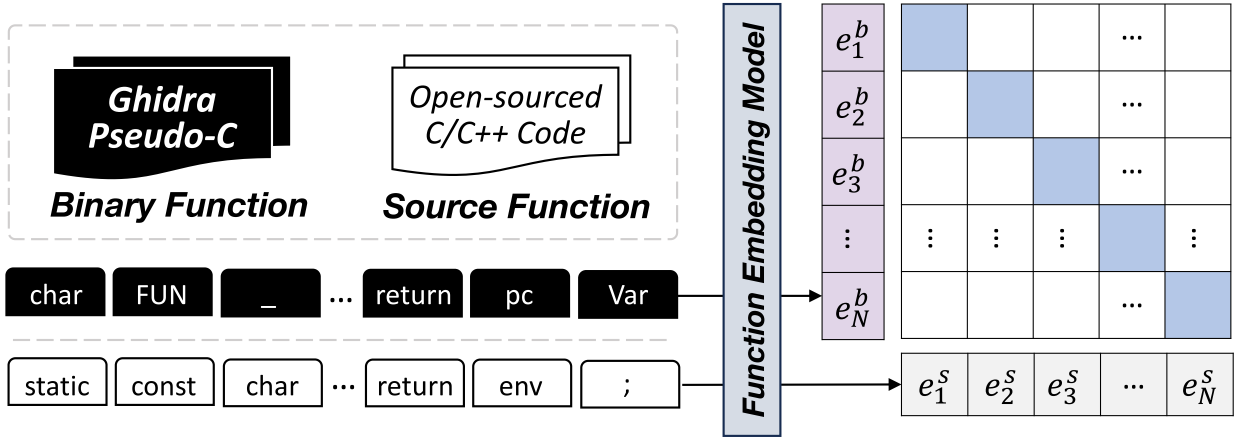
Embedding-based Function Retrieval

Model Training For BinaryAI

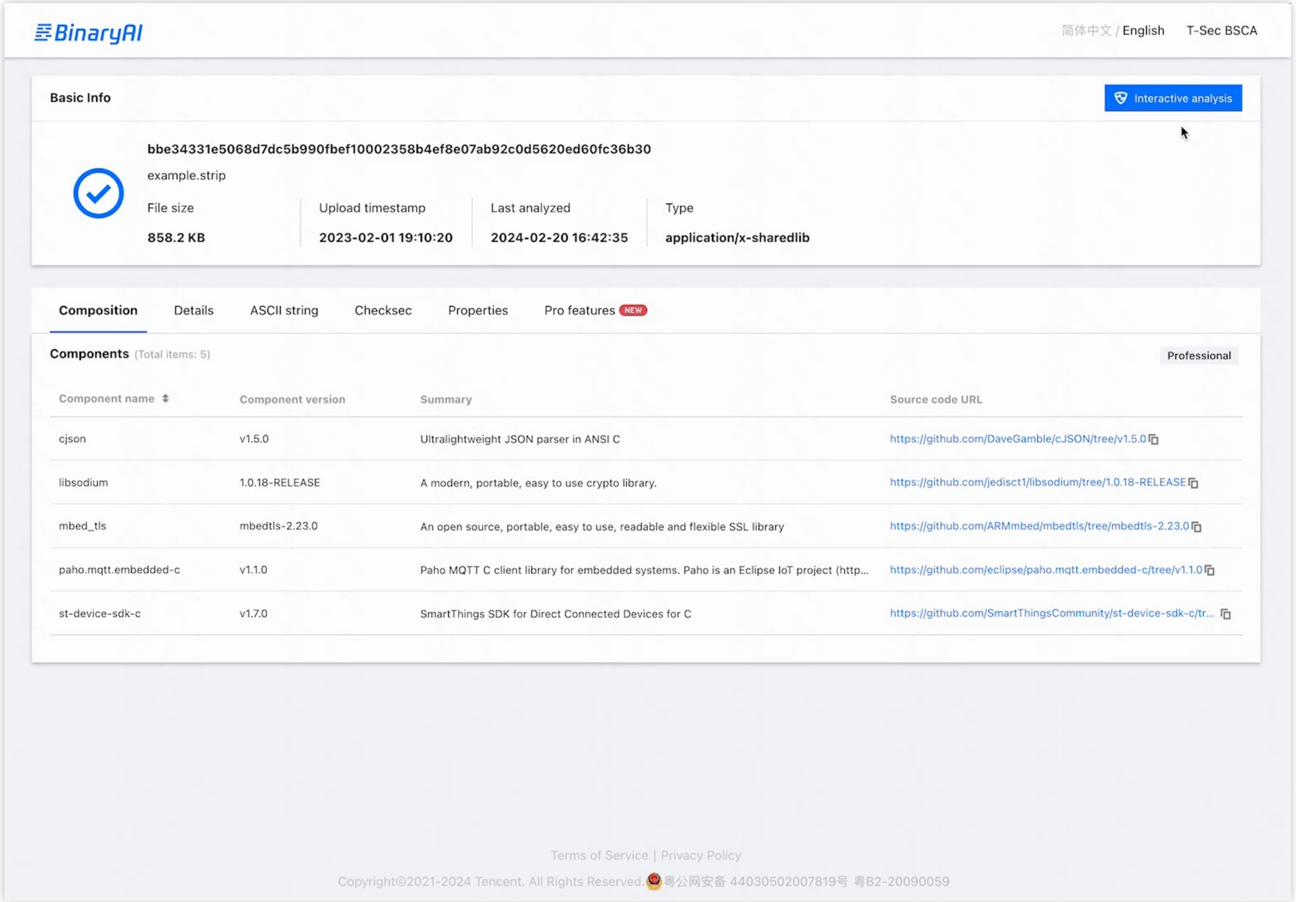
- **Base model:** OPT ⇨ BLOOM ⇨ **Pythia (410M)**
- **Training dataset:** **10M+** function pairs with an average of around **500** tokens per function
- **Contrastive representation learning :**
  - Extend the InfoNCE loss function of **CLIP**
  - Apply Momentum Contrast (MoCo) method
  - Enlarge in-batch negative samples

$$L_{bin(src)} = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{\exp\left(\mathrm{sim}(e_i^{b(s)}, e_i^{s(b)})/\tau\right)}{\sum_{j=1}^{N}\exp\left(\mathrm{sim}(e_i^{b(s)}, e_j^{s(b)})/\tau\right)}$$

$$L_{CLIP} = (L_{bin} + L_{src})/2$$



Binary Source Code Matching: Are we there yet?



*Directly retrieve the **top-1** most similar source function as the matching result?*

- A significant presence of source functions with **subtle modifications** in the large-scale TPLs
- Token-based syntactic feature captured by embedding model is **insufficient** for accurate matching

```c
int sodium_is_zero(
  const unsigned char *n,
  const size_t nlen
){
  size_t i;
  unsigned char d = 0U;
  for (i = 0U; i < nlen; i++) {
  d |= n[i];
  }
  return 1 & ((d - 1) >> 8);
}
```

Top-1, score=0.8477

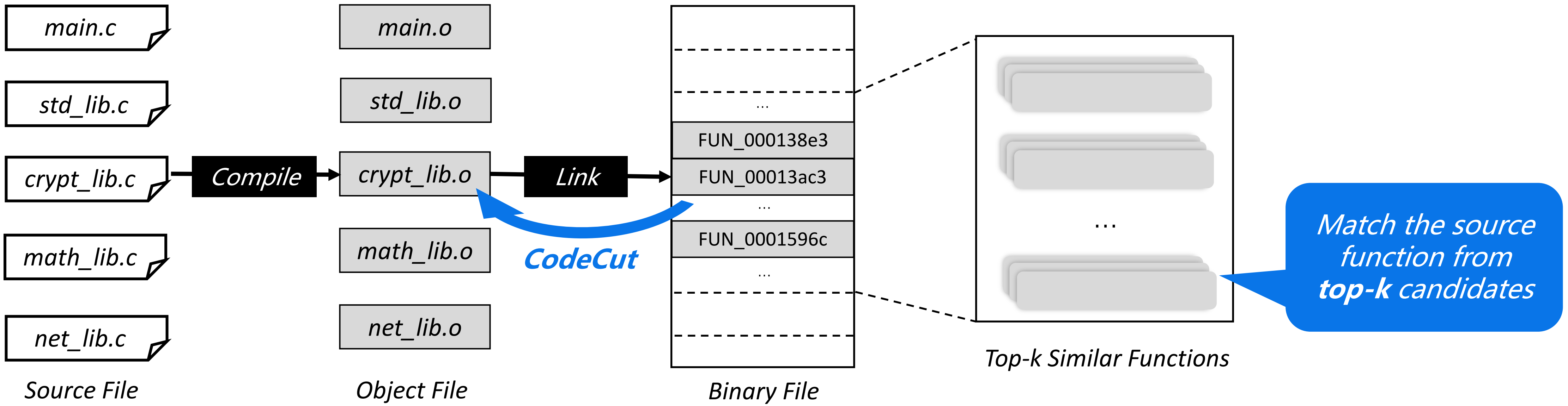
```c
int sodium_is_zero(
  const unsigned char *n,
  const size_t nlen
){
  size_t i;
  volatile unsigned char d = 0U;
  for (i = 0U; i < nlen; i++) {
  d |= n[i];
  }
  return 1 & ((d - 1) >> 8);
}
```

Top-2, score=0.8475
(Ground Truth)

Locality-driven Matching

- **Insight:** Link-time localities (i.e., relative virtual address) of binary functions compiled from the same source file are almost rendered continuous in the address space of the binary file

- **Basic workflow:**
  1. **CodeCut:** cut intervals with continuous binary functions to recover boundaries of object files
  2. Identify source files compiled into the binary file
  3. Match the source functions in the files as the result

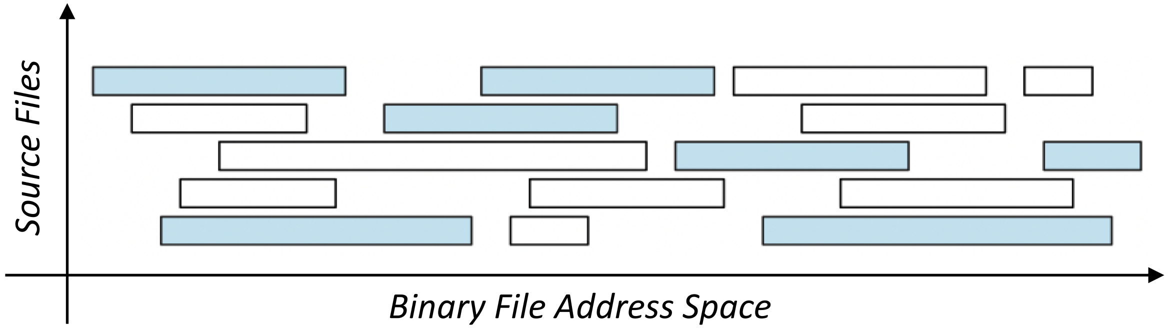


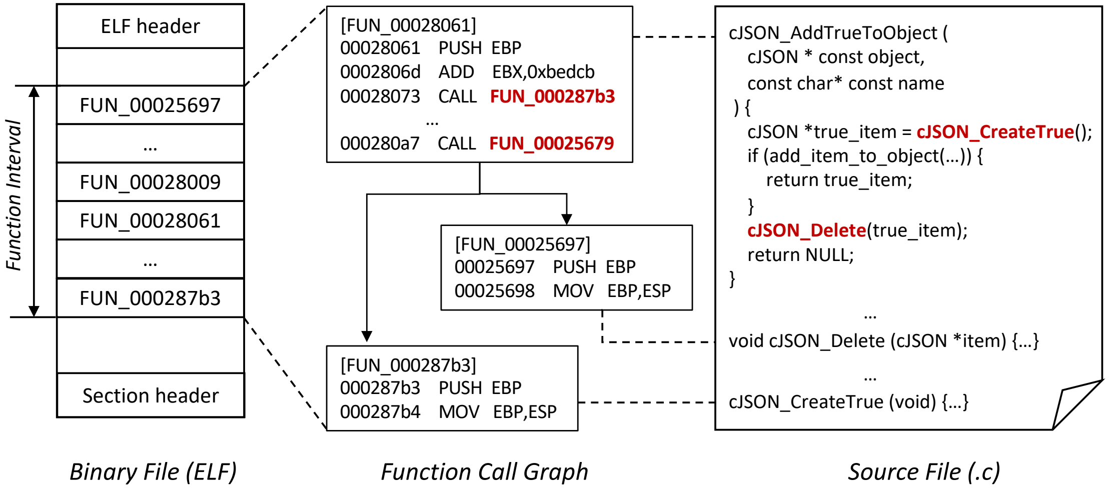
Locality-driven Matching

Algorithm

1. Convert top-k retrieved functions to index mapping from source files to binary source function pairs
2. **Interval covering problem:** source files compiled into binary should have longer continuous functions
3. Tackle the problem greedily by prioritizing longer intervals (i.e., files) that can cover more functions
4. Utilize function call graph to facilitate binary source function matching within selected files

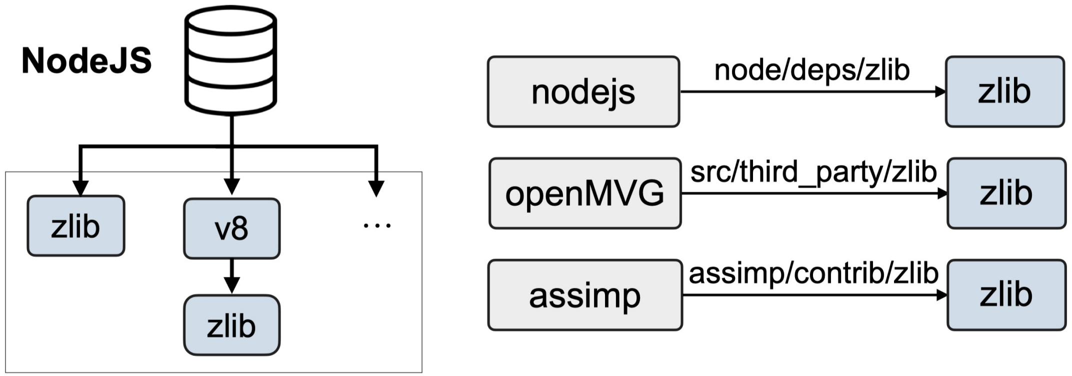
- **Syntactic features:** function embeddings
- **Semantic features:** link-time locality, function call graph





Third-party Library Detection

- Calculate the ratio of matched functions as the similarity between binary file and source code repository (i.e., TPL)
- Identify the TPL whose similarity exceeds a pre-defined threshold, along with potential security threats
- Alleviate the issue of **internal code clones** by integrating TPL dependency to filter invalid TPLs




```
Input: bin2src_match, tpl_dependency
Result: components
Function DetectComponents:
    tpl2func_match, components ← ∅
    for (bin_rva, src_func) ∈ bin2src_match do
        src_tpls ← retrieved TPLs containing src_func in SCA database
        filtered_tpls ← FilterByDependency(src_tpls, tpl_dependency)
        for tpl ∈ filtered_tpls do
            tpl2func_match[tpl].add(bin_rva)
    for (tpl, matched_funcs) ∈ tpl2func_match do
        if len(matched_funcs) / tpl.total_func_count > θ then
            components.add(tpl)
    return components

Function FilterByDependency(src_tpls, tpl_dependency):
    filtered_tpls ← src_tpls
    for tpl ∈ src_tpls do
        reused_tpls ← tpl_dependency[tpl]
        if reused_tpls and src_tpls have intersection then
            filtered_tpls.remove(tpl)
    return filtered_tpls
```

RQ1: Effectiveness of Function Embedding

- BinaryAI achieves **0.3407** MRR (Mean Reciprocal Rank) in contrast to **0.1769** of CodeCMR†, increasing recall@1 from **10.75%** to **22.54%** and recall@100 from **33.87%** to **56.60%**
- Traditional techniques (BinPro and B2SFinder) incur limited performance in matching source functions from a large-scale dataset (MRR<0.005, recall@100<10%)

Model	Objective	Validation Set of Model (query=32,296)					Binary SCA Test Set (query=23,529)				
		MRR	Count/Recall@1	Count/Recall@10	Count/Recall@50	Count/Recall@100	MRR	Count/Recall@1	Count/Recall@10	Count/Recall@50	Count/Recall@100
BinPro	N/A	0.0027	771 / 2.39	1,165 / 3.61	1,593 / 4.93	1,845 / 5.71	0.0036	612 / 2.60	944 / 4.01	1,262 / 5.36	1,507 / 6.40
B2SFinder	N/A	0.0042	945 / 2.93	1,717 / 5.32	2,108 / 6.53	2,436 / 7.54	0.0048	864 / 3.67	1,305 / 5.55	1,740 / 7.40	2,082 / 8.85
CodeCMR	Triplet	0.1431	3,195 / 9.89	6,543 / 20.26	7,827 / 24.24	8,347 / 25.85	0.2232	2,805 / 11.92	7,873 / 33.46	9,875 / 41.97	1,0561 / 44.89
CodeCMR	CLIP	0.2319	5,456 / 16.89	10,589 / 32.79	12,256 / 37.95	12,801 / 39.64	0.2820	3,638 / 15.46	9,889 / 42.03	12,510 / 53.17	13,319 / 56.61
BinaryAI	Triplet	0.2774	6,552 / 20.29	12,627 / 39.10	14,009 / 43.38	14,460 / 44.77	0.3539	4,692 / 19.94	12,113 / 51.48	14,650 / 62.26	15,395 / 65.43
BinaryAI	**CLIP**	**0.3006**	**7,235 / 22.40**	**13,465 / 41.69**	**14,682 / 45.46**	**15,020 / 46.51**	**0.3958**	**5,348 / 22.73**	**13,493 / 57.35**	**15,873 / 67.46**	**16,576 / 70.45**

Finding: *BinaryAI can be more effective than CodeCMR and other traditional techniques in terms of the embedding-based function retrieval with the usage of LLM and CLIP as the training objective.*

† CodeCMR utilizes separate function encoders (DPCNN for source function and GNN for binary function)

RQ2: Accuracy of Binary Source Code Matching

Binary	#Label	BinaryAI	Exact Match			Fuzzy Match		
			#TP	P (%)	R (%)	#TP	P (%)	R (%)
controlblock	185	107	86	80.37	46.49	99	92.52	53.51
db_bench	359	253	209	82.61	58.22	239	94.47	66.57
dosbox_core	2,804	2,042	1,854	90.79	66.12	1,974	96.67	70.40
eth_sc	267	232	190	81.90	71.16	221	95.26	82.77
hyriseSystem	318	197	187	94.92	58.81	193	97.97	60.69
kvrocks	2,240	1,452	1,190	81.96	53.13	1,415	97.45	63.17
nano_node	1,604	939	752	80.09	46.88	923	98.30	57.54
pagespeed	6,430	3,442	2,683	77.95	41.73	3,305	96.02	51.40
prometheus	204	157	138	87.90	67.65	146	92.99	71.57
replay-sorcery	770	454	367	80.84	47.66	437	96.26	56.75
st-device-sdk	801	582	486	83.51	60.67	536	92.10	66.92
tendisplus	2,197	1,541	1,265	82.09	57.58	1,498	97.21	68.18
tic80	832	695	573	82.45	68.87	668	96.12	80.29
turbobench	762	270	203	75.19	26.64	243	90.00	31.89
yuzu-cmd	3,756	1,795	1,374	76.55	36.58	1,675	93.31	44.60
Total	**23,529**	**14,158**	**11,557**	**81.63**	**49.12**	**13,572**	**95.86**	**57.68**

Accuracy of locality-driven matching

- **Test set:** 15 stripped binary files with manually labeled binary-to-source function mappings
- **Result with top-10 retrieved functions:**
  - On average, the precision is **81.63%** for exact match and **95.86%** for fuzzy match†
  - In all binary files, the precision exceeds **75%** for exact match and **90%** for fuzzy match

Finding: *Locality-driven matching can effectively identify the exact source function from top-k retrieved results and such results generalize to different binary files.*

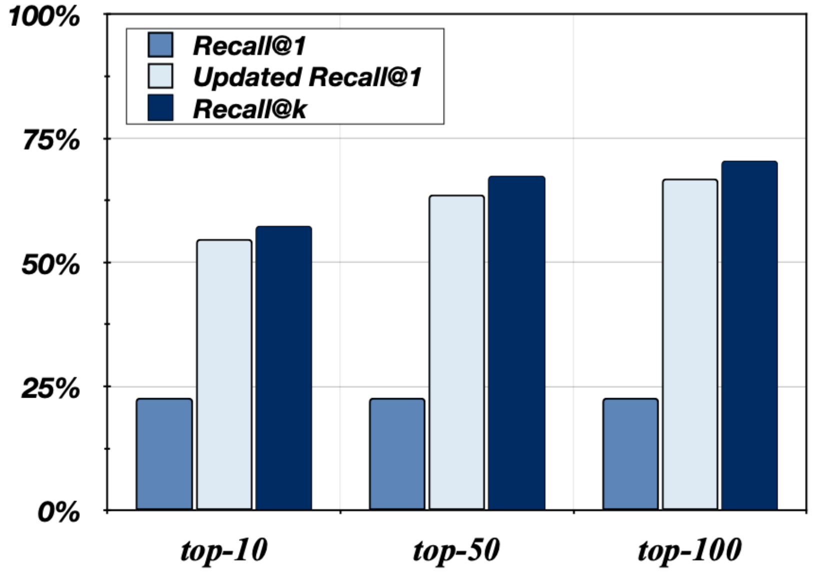
† Match ground truth after normalization, applicable for other downstream tasks (e.g., reverse engineering)

RQ2: Accuracy of Binary Source Code Matching

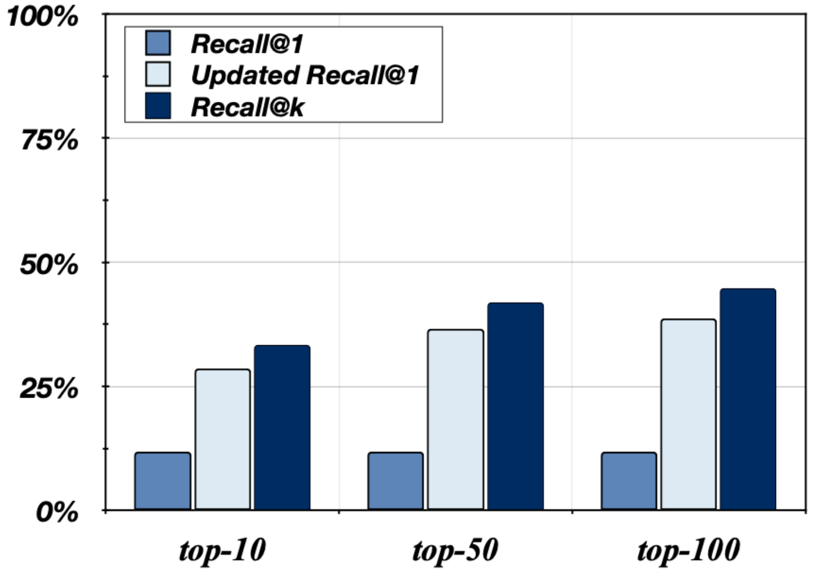
Contribution to binary source code matching

- **BinaryAI:** recall@1 from 22.73% to 54.70% with upper bound as 57.35% for top-10, and to 66.90% with upper bound as 70.45% for top-100

- **CodeCMR:** recall@1 from **11.92%** to **28.61%** with upper bound as **33.46%** for top-10, and to **38.76%** with upper bound as **44.89%** for top-100



BinaryAI



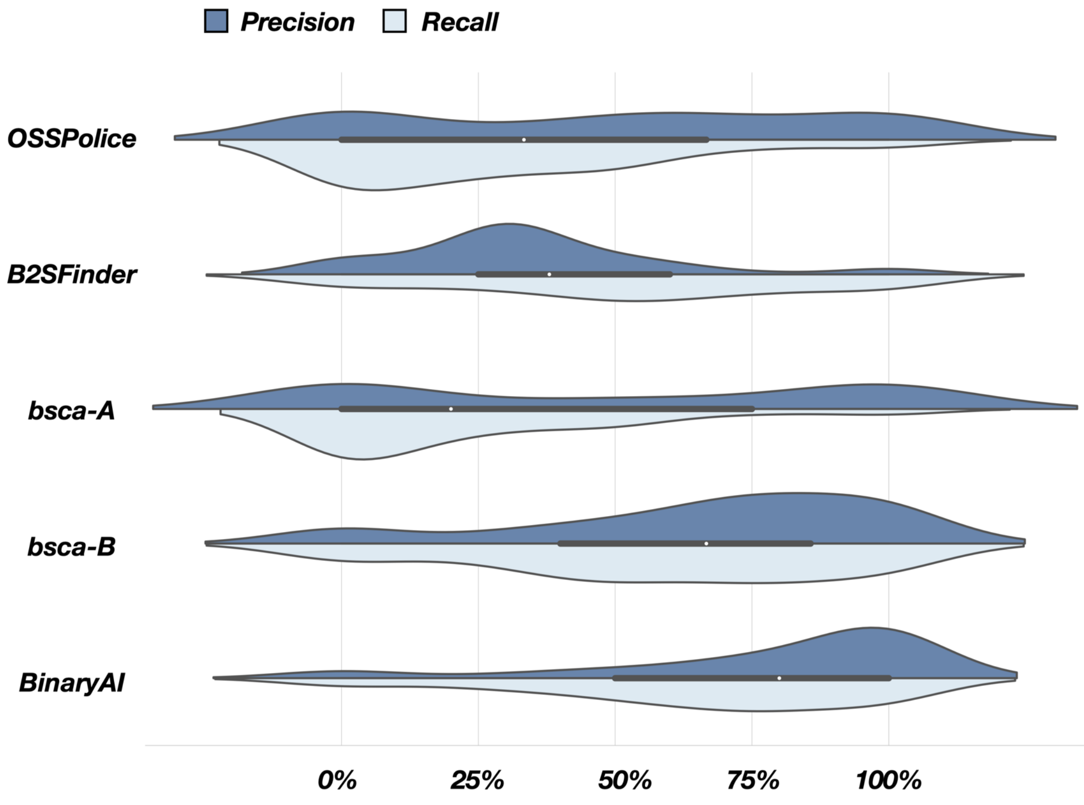
CodeCMR

RQ3: Accuracy of TPL Detection (BSCA)

- **Test Set:** 150 stripped binary files from 85 projects, labeled with 1,045 third-party components
- BinaryAI outperforms existing commercial BSCA tool with increased precision from **73.36%** to **85.84%** and recall from **59.81%** to **64.98%**

Tools	#TP	#FP	#FN	Precision	Recall	F1
OSSPolice	348	191	697	64.56	33.30	43.94
B2SFinder	574	1232	471	31.78	54.93	40.26
bsca-A	232	108	813	68.24	22.20	33.50
bsca-B	625	227	420	73.36	59.81	65.90
BinaryAI	**679**	**112**	**366**	**85.84**	**64.98**	**73.97**

Finding: *BinaryAI dominates the performance of TPL detection among the state-of-the-art binary SCA tools.*



Summarizing BinaryAI

- The first function-level binary-to-source SCA based on model, achieving **85.84%** precision and **64.98%** recall
- We propose two-phase binary source function matching to capture both syntactic and semantic code features
- BinaryAI contains **12K+** TPLs with **56M+** unique functions, and the model is trained with **10M+** function pairs
- **More features available:** IDA/Ghidra plugin, Binary diffing, Malware analysis, etc

binaryai

binaryai-sdk

@keen_lab