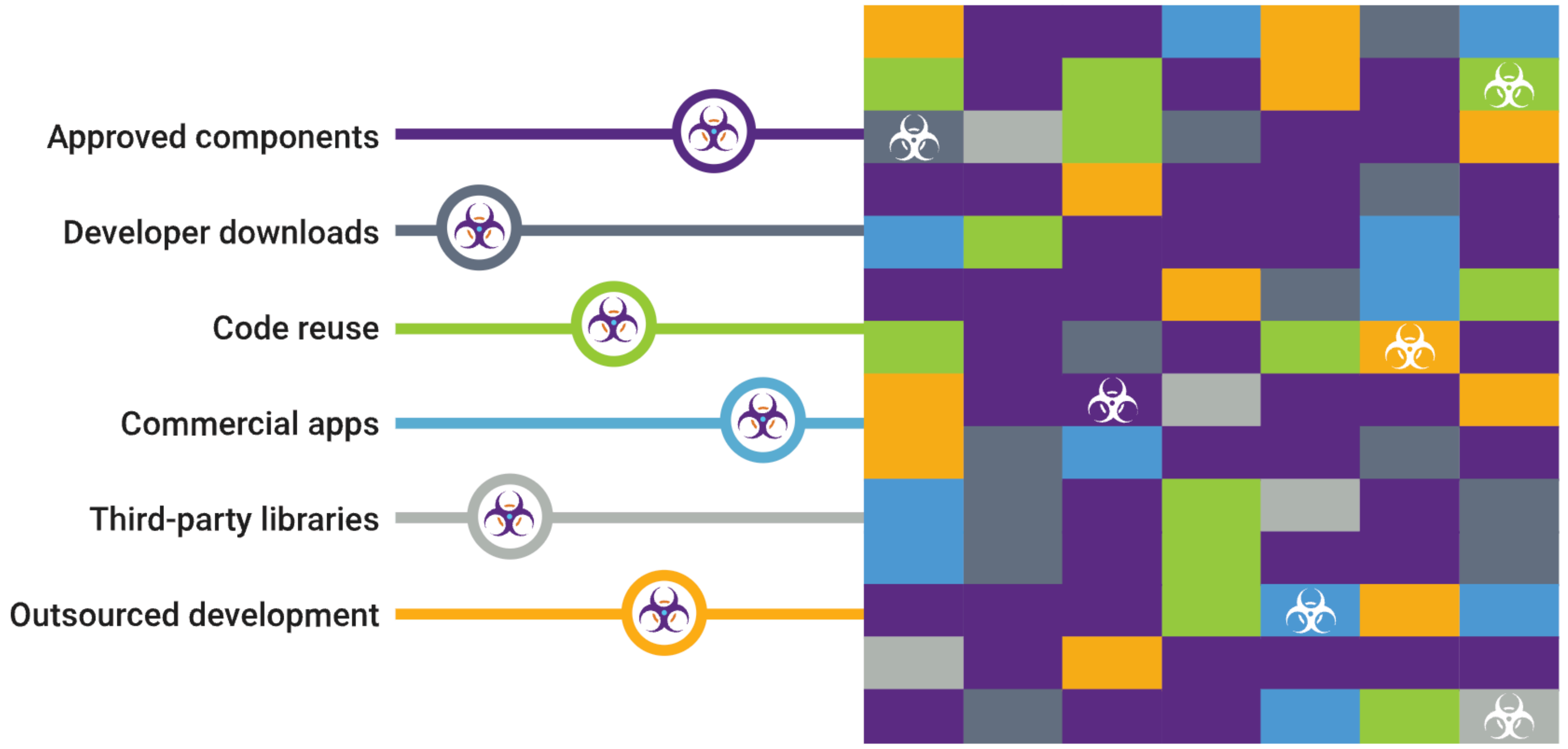
Third-Party Library Dependency for Large-Scale SCA in the C/C++ Ecosystem: How Far Are We?

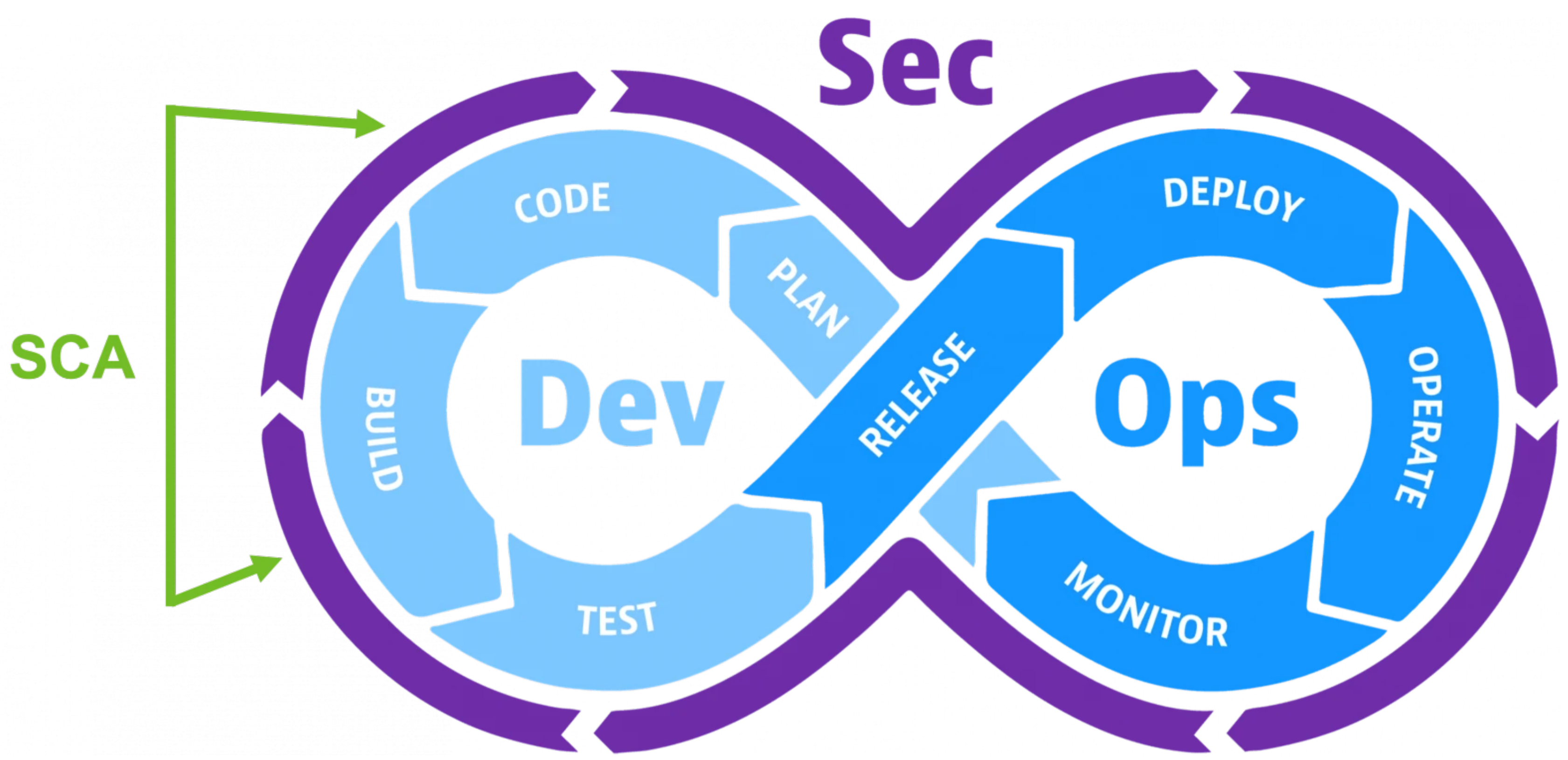
Ling Jiang*, Hengchen Yuan*, Qiyi Tang†, Sen Nie†, Shi Wu†, Yuqun Zhang*

*Southern University of Science and Technology
†Tencent Security Keen Lab

Software Composition Analysis (SCA)

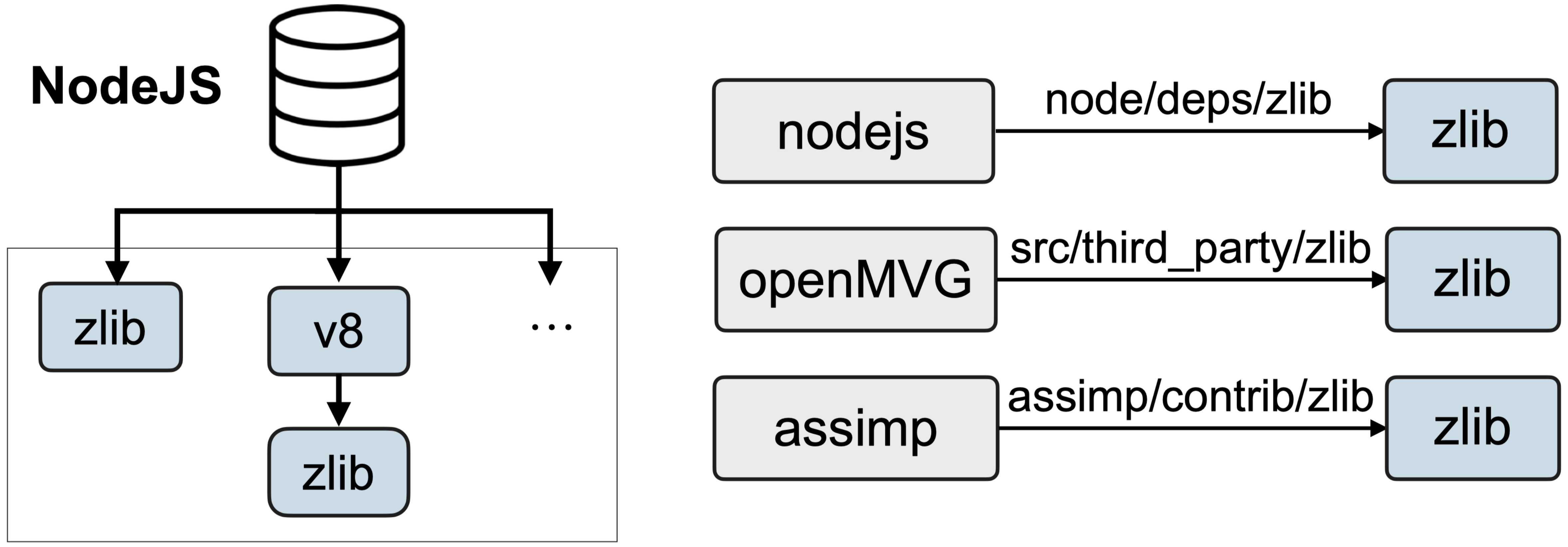
- Identifying and managing the open-source third-party libraries (TPLs) contained in softwares
- Relying on SCA, developers can effectively track potential threats introduced to softwares (e.g., vulnerability propagation, license violation)
- Existing SCA techniques advance component identification by matching features between target software and collected TPLs based on their similarities





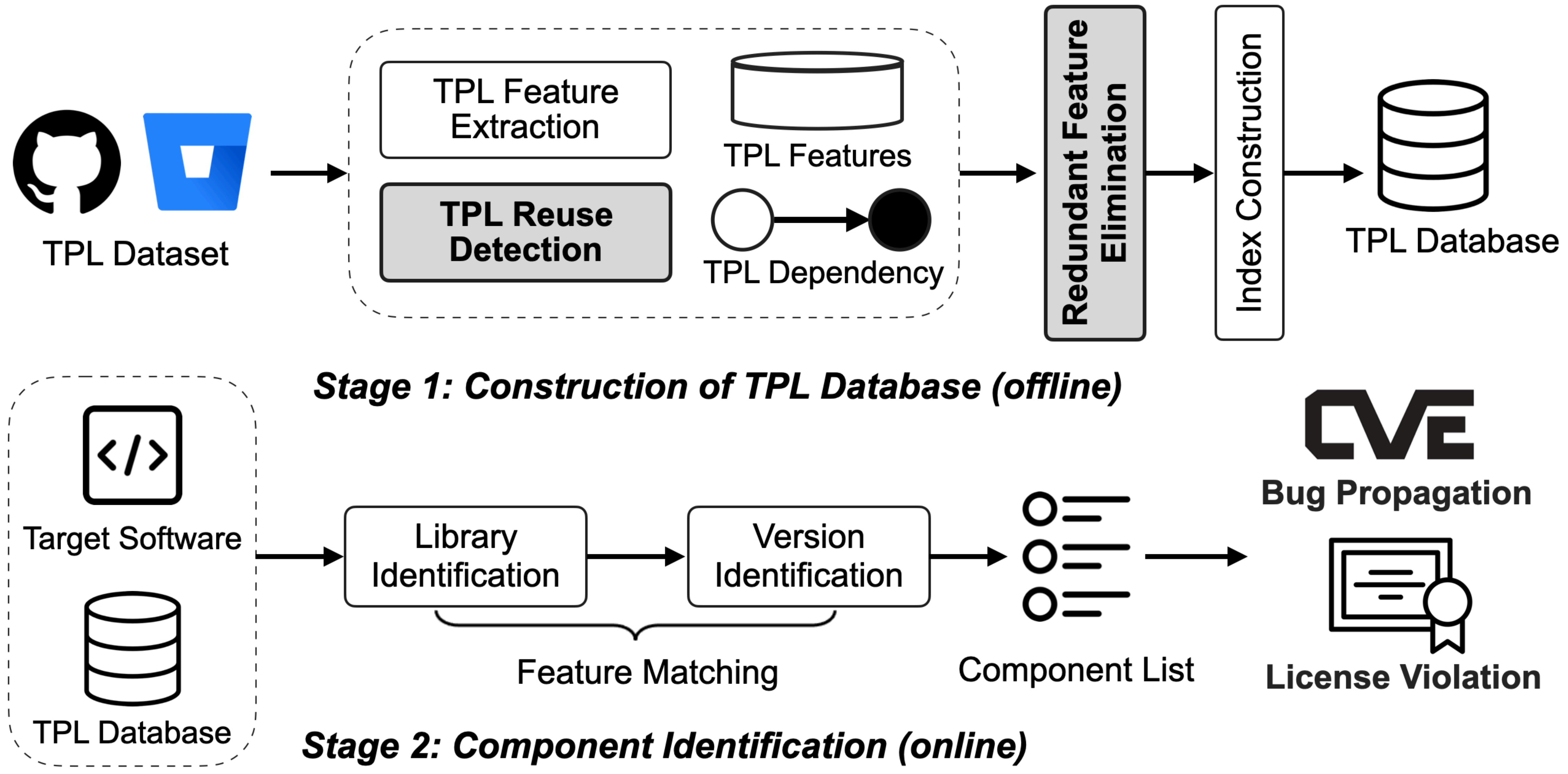
Internal Code Clone

- One TPL depends on other TPLs via code reuse within large-scale dataset
- Causing inevitable **feature duplication** across collected TPLs
- Compromising SCA by incurring false positives during feature matching



Example of internal code clone

Centris: C/C++ Function-level SCA



The workflow of Centris

Centris (ICSE'21)

- Performing TPL reuse detection to derive **TPL dependency** while extracting TPL code features
- Eliminating redundant features based on TPL dependency
- Constructing TPL database after elimination for online component identification

TPL Reuse Detection

- Utilizing **birth time** of duplicate functions between TPLs to help identify reused functions
- Recalling the dependency between TPLs when the ratio of reused functions surpasses a **preset threshold**

Challenges

- Lack of evaluation for the accuracy of the derived TPL dependencies in Centris.
- Evaluation of how TPL dependency impacts the SCA is limited in Centris.
- To validate generalized contribution of TPL dependency to other evaluation setups.

RQ1: *How does Centris perform in* ***TPL reuse detection*** *and* ***SCA****?*

- Accuracy of derived TPL dependencies
- Impact on the downstream binary-level SCA

RQ2: *What are the* ***major factors*** *that impact the performance of Centris?*

- Effectiveness of function birth time
- Threshold-based recall

Dataset

1) Ground-truth TPL dependencies

- Total **10,241** TPLs in dataset
- Manually labeled **2,150** TPL dependencies of top 1K mostly reused TPLs

2) Ground-truth SCA results

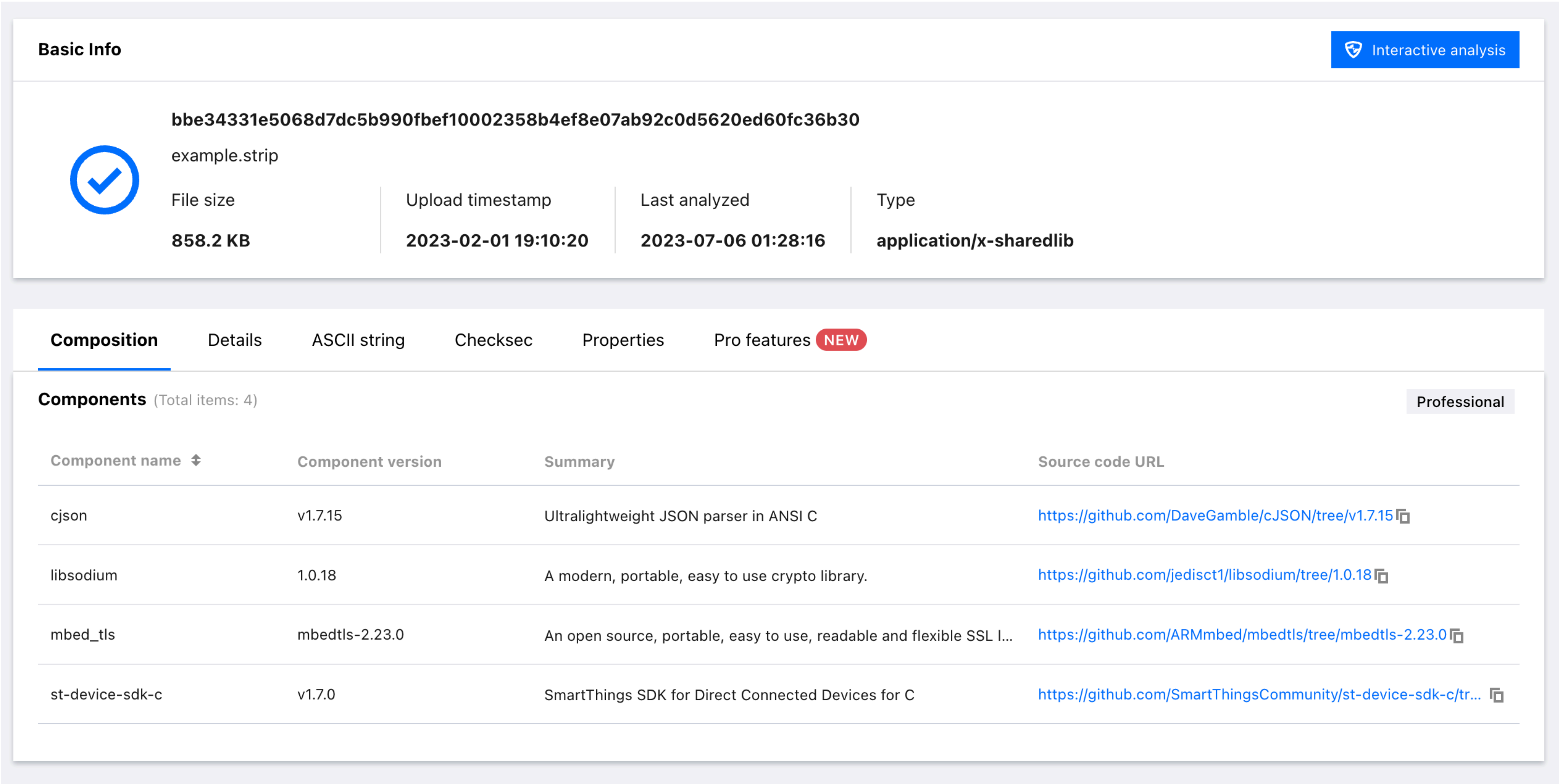
- Total **128** binary files compiled with 75 C/C++ open-source software projects
- Parsing DWARF to derive contained components

Software	Binary	Version	Sys/Arch†	#TPLs	Sample TPLs
terarkdb	db_bench	v1.3.6	arch linux/x86_64	15	bzip2, zlib, lz4, xxHash
ClickHouse	clickhouse	v22.1.2.2	macOS/arm64	61	libxml2, grpc, libexpat
TIC-80	tic80.exe	v0.90.1723	windows/x86_64	15	blip-buf, libpng, dirent
kvrocks	kvrocks	v2.0.5	ubuntu/i386	12	glibc, libevent, rocksdb
Tendis	tendisplus	v2.4.3	ubuntu/x86_64	15	glibc, rapidjson, snappy

† The system and architecture applied to compile the binary

Evaluation of SCA

- Adapting TPL dependency of Centris to binary-level SCA platform - **BinaryAI** developed by Tencent Security Keen Lab
- BinaryAI is now available at https://binaryai.net/



Accuracy of TPL dependencies

Default version (threshold=0.10):
Precision **35.71%** , Recall **49.44%**, F1 score **41.47%**

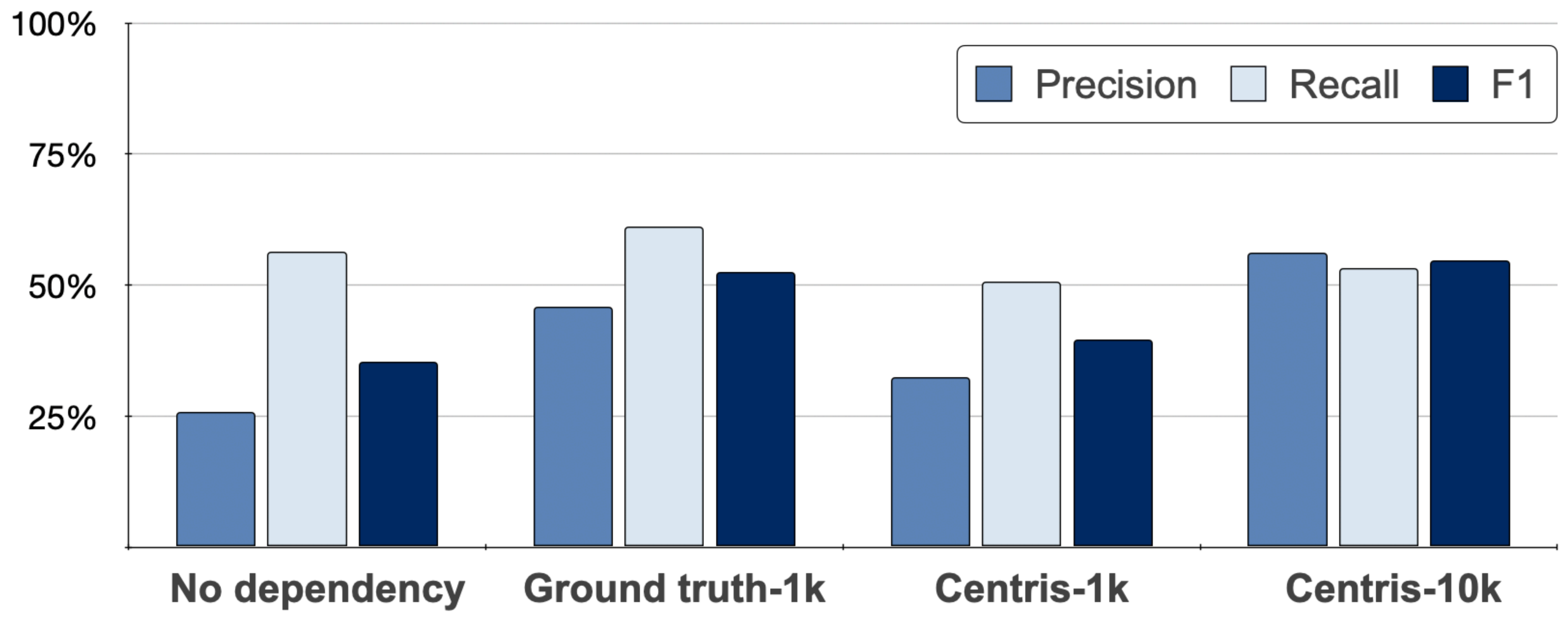
Threshold	Verification of TPL dependency						
	Total	#TP	#FP	#FN	Precision(%)	Recall(%)	F1(%)
0.75	59	37	22	2,113	62.71	1.72	3.35
0.50	261	159	102	1,991	60.92	7.40	13.20
0.20	1,611	662	949	1,488	41.09	30.79	35.20
0.15	2,118	839	1,279	1,311	39.61	39.02	39.31
0.10	2,977	1,063	1,914	1,087	35.71	49.44	41.47
0.05	4,349	1,300	3,049	850	29.89	60.47	40.01
0.01	8,447	1,491	6,956	659	17.65	69.35	28.14

Accuracy of TPL reuse detection by Centris

Finding: The accuracy of TPL dependencies derived by Centris may not well generalize to other datasets.

Impact on SCA

- Ground truth-1k increases precision from 25.76% to 45.90%
- Centris-1k increases precision to 32.44% while decreases recall from 56.34% to 50.71%



SCA results with TPL dependency

Finding: While the redundant feature elimination based on TPL dependencies can advance the SCA results, Centris may be limited in leveraging the power of TPL reuse detection.

Function birth time

- Earliest tag time of function introduced to the repository
- Error-prone with elusive reasons, e.g., repository migration
- Causing most of the false positives (1,336 out of 1,914)

- **Example:** all functions in *zlib* share birth time of 2011 from `commit-bcf78a2`, while they were actually first introduced in 1995 due to migration of repository

- **Discussion:** additional information (e.g., source directory) should be included to help identify the TPL dependency

Finding: The inaccurate function birth time significantly compromises the accuracy of Centris for TPL reuse detection.

```
/* deflate.c,v1.3 1995/04/10 16:03:45 */
#include "deflate.h"
...
int deflateInit (z_stream *strm, int level) {...}
int deflateReset (z_stream *strm) {...}
int deflate (z_stream *strm, int flush) {...}
int deflateEnd (z_stream *strm) {...}
int deflateCopy (z_stream *dest, z_stream *source) {...}
```

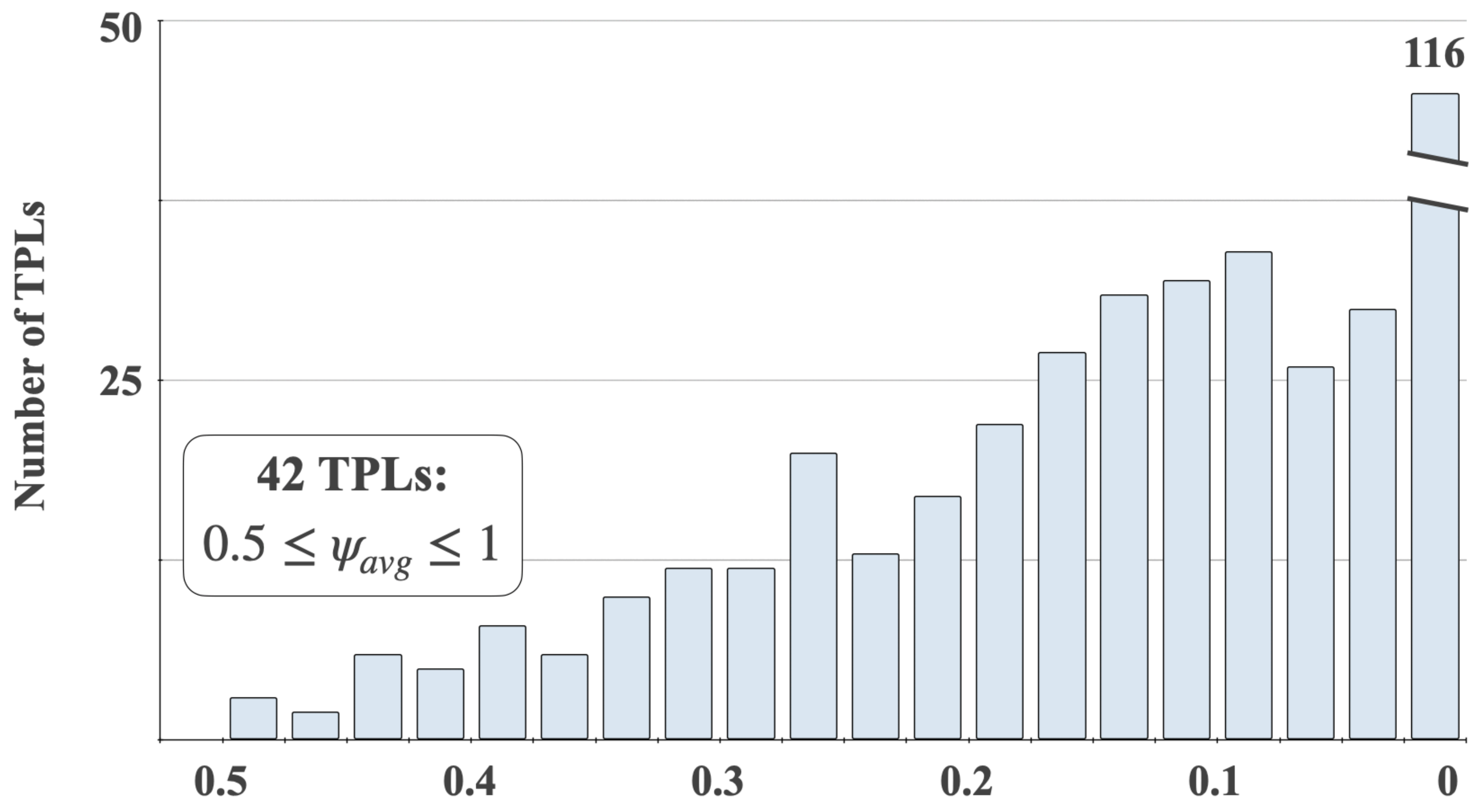
Code snippet in deflate.c of zlib

TPL Name	Source file directories	Birth time
rsync	zlib/deflate.c	1998-05-14 07:22:45
wxWidgets	src/zlib/deflate.c	1998-05-20 14:02:15
gdal	frmts/zlib/deflate.c	2001-09-15 21:50:31
mysql-server	zlib/deflate.c	2002-04-21 10:06:34
llvm-project	llvm/runtime/zlib/deflate.c	2004-03-19 21:59:23
CMake	Utilities/cmzlib/deflate.c	2006-04-18 20:40:40
libpng	deflate.c	2009-04-16 15:46:37
tigervnc	common/zlib/deflate.c	2009-04-30 11:41:03
libwebsockets	tmp/win32port/zlib/deflate.c	2011-04-24 07:12:38
zlib	deflate.c	2011-09-10 06:19:21

Birth time of reused functions from zlib

Reuse ratio between TPLs

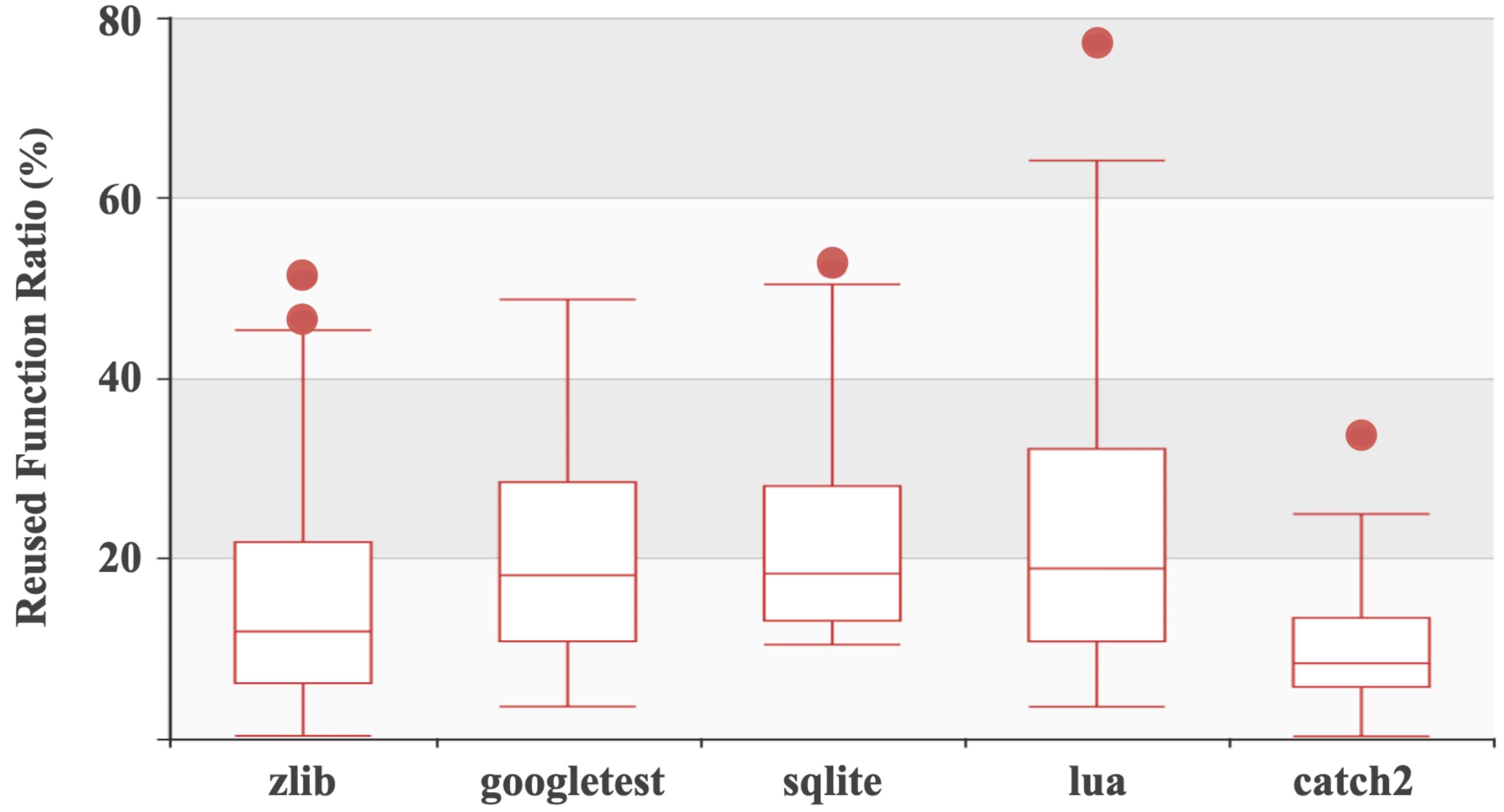
- Average reuse ratio enormously vary for different TPLs
- Reuse ratio can be distributed divergently among different dependencies for specific TPLs



Average reuse ratio

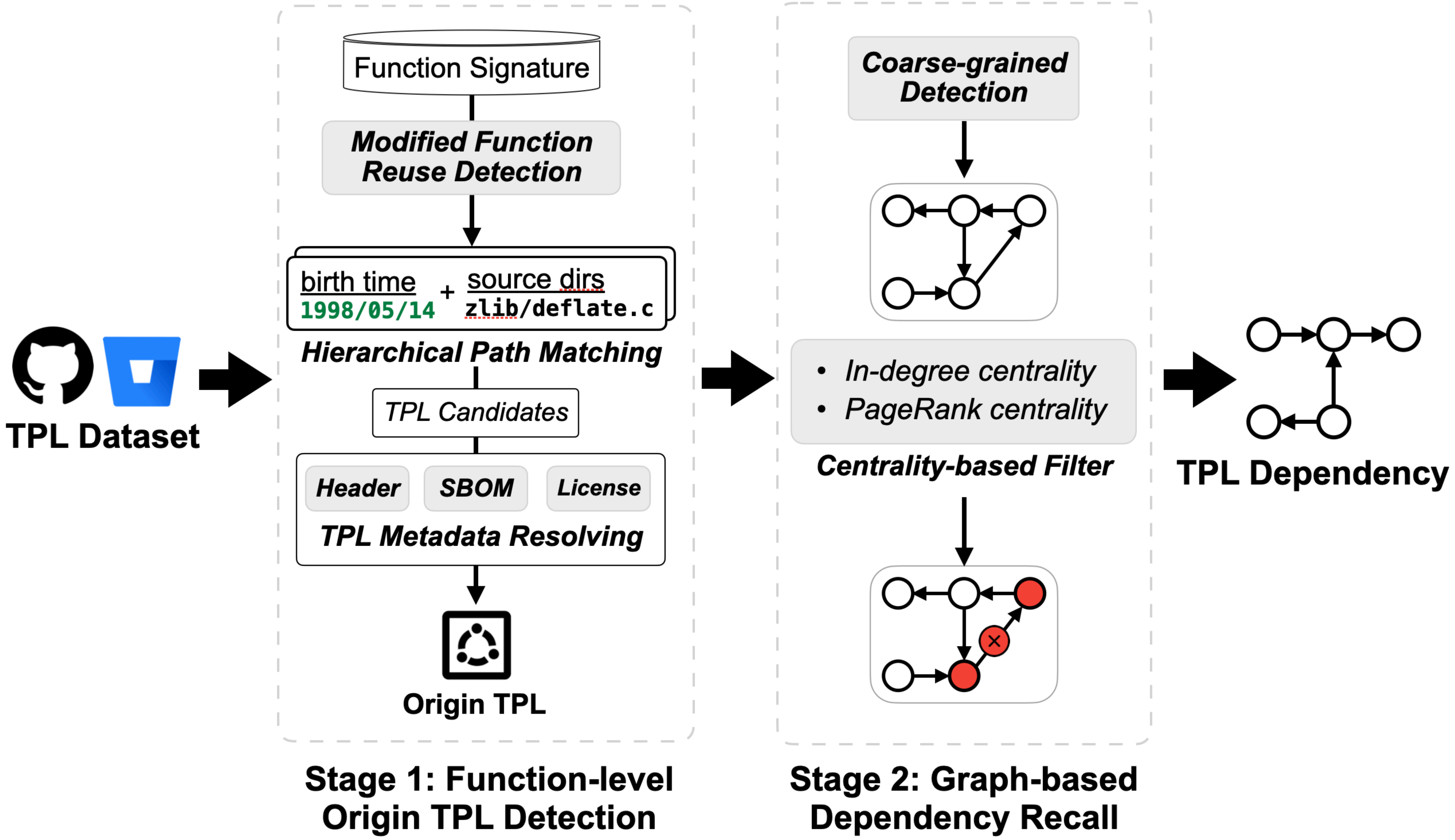
Discussion

Two-phase recall: 1) initializing dependencies in a coarse-grained manner 2) filtering invalid dependencies based on graph analysis with edges assigned reuse ratio as weight



TPL reuse distribution

Finding: The reuse ratio can be quite divergent for different TPL dependencies. Thus, a fixed threshold to denote the reuse ratio may not well generalize to different TPL dependencies.



The framework of TPLite

Function-level Origin TPL Detection

- Modified Function Reuse Detection
- Hierarchical Path Matching
- TPL Metadata Resolving

Graph-based Dependency Recall

- Coarse-grained Detection
- Centrality-based Filter

Function-level Origin TPL Detection

Objective: Identify origin TPL for each function

1) Modified Function Reuse Detection

- Detecting similar functions due to modified reuse
- Based on existing token-based code clone detector

2) Hierarchical Path Matching

- Utilizing **hierarchical path features** to help identify the origin TPL since TPL reuse tends to retain structural similarity
- Still retaining TPL with earliest birth time as candidate

3) TPL Metadata Resolving

- Implementing parsers for header files, SBOM files and licenses to derive actual origin TPL out of candidates

Algorithm 1: Function-level Origin TPL Detection

```
Input: func ;                                              ▹ Source code of function
Result: origin_tpl
Function DetectOriginTPL:
    similar_funcs ← DetectModifiedReuse(func) ;           ▹ Token-based detection
    candidate_tpls ← set()
    for f_i in similar_funcs do
        tpls_f_i ← set of TPLs containing the function f_i
        source_dirs ← ExtractSourcePath(tpls_f_i, f_i) ;  ▹ Across all versions
        terms_count ← ∅
        for p_i in source_dirs do
            terms ← PathHierarchyTokenizer(p_i)
            terms_count.update(terms)
        end
        terms_sort ← Sort(terms_count) ;                  ▹ Sort terms by frequency
        for t_i in terms_sort do
            if t_i contains the name of TPL in tpl_f_i then
                tpl_path ← earliest TPL with the name in t_i
                candidate_tpls.add(tpl_path) ;            ▹ Based on source dirs
                break
            end
        end
        tpl_time ← TPL with the earliest birth time t(tpl, f_i) in tpls_f_i
        candidate_tpls.add(tpl_time) ;                    ▹ Based on birth time as Centris
    end
    origin_tpl ← ResolveMeta(candidate_tpls) ;            ▹ Header, SBOM, License
return
```

Graph-based Dependency Recall

Objective: Two-phase recall to improve accuracy

1) Coarse-grained Detection

- Dynamically deriving optimal reused ratios for different TPLs in a coarse-grained manner and retain all dependencies that satisfy

$$\frac{|\omega|}{|R|} > \delta \cdot \frac{|R|}{|R_{\varphi}|}$$

2) Centrality-based Filter

- Nodes with high eigenvector centrality but low degree centrality indicates unstable relationships
- Eliminating dependencies pointing to the TPLs whose ratio of **PageRank value to in-degree centrality** after normalization exceeds a preset value

Algorithm 2: Graph-based Dependency Recall

Input: TPL dataset *tpl_set*
Result: *tpl_dependencies*
Function *DependencyRecall*:
    tpl_dependencies $\leftarrow \emptyset$
    for $tpl_s, tpl_r \in tpl\_set \times tpl\_set$ **do**
        $\omega \leftarrow$ reused functions from tpl_s to tpl_r
        if $|\omega| > 0$ *and CoarseGrainedCheck*(tpl_s, tpl_r) **then**
            $R \leftarrow$ set of functions of tpl_r
            tpl_dependencies.add_edge(tpl_s, tpl_r, *weight*=$|\omega|/|R|$)
        end
    end
    CentralityFilter(tpl_dependencies)
return
Function *CentralityFilter(tpl_dependencies)*:
    $tpl_{\sigma}\_set \leftarrow \emptyset$
    centrality_indegree $\leftarrow$ *InDegreeCentrality(tpl_dependencies)*
    centrality_pagerank $\leftarrow$ *PageRank(tpl_dependencies)*
    for *tpl in tpl_dependencies.nodes* **do**
        if *Normalize(centrality_pagerank[tpl]) / Normalize(centrality_indegree[tpl])* $> \epsilon$ **then**
            tpl_{σ}_set*.add(tpl)*
        end
    end
    for tpl_s, tpl_r *in tpl_dependencies.edges* **do**
        if *in-degree*(tpl_s) $> \eta$ *and* $tpl_r \in tpl_{\sigma}\_set$ **then**
            tpl_dependencies.delete(tpl_s, tpl_r)
        end
    end
return

Evaluation of TPL dependency

- TPLite enhances precision (**35.71%** to **88.33%**) and recall (**49.44%** to **62.65%**) compared with Centris
- **FP analysis**: absence of TPLs in current dataset / **FN analysis**: black-box reuse injected during linking
- Ablation study with two variants: Centris_{otd} / Centris_{otd+cg}

Tool	Metrics of TPL dependency						
	Total	#TP	#FP	#FN	Precision(%)	Recall(%)	F1(%)
Centris	2,977	1,063	1,914	1,087	35.71	49.44	41.47
CCScanner	32,51	1,179	2,072	971	36.27	54.84	43.66
$Centris_{otd}$	1,622	1,230	392	920	75.83	57.21	65.22
$Centris_{otd+cg}$	1,828	1,352	476	798	73.96	62.88	67.97
TPLite	**1,525**	**1,347**	**178**	**803**	**88.33**	**62.65**	**73.31**

$Centris_{otd}$ = *Centris + function-level origin TPL detection*
$Centris_{otd+cg}$ = $Centris_{otd}$ + *coarse-grained detection*

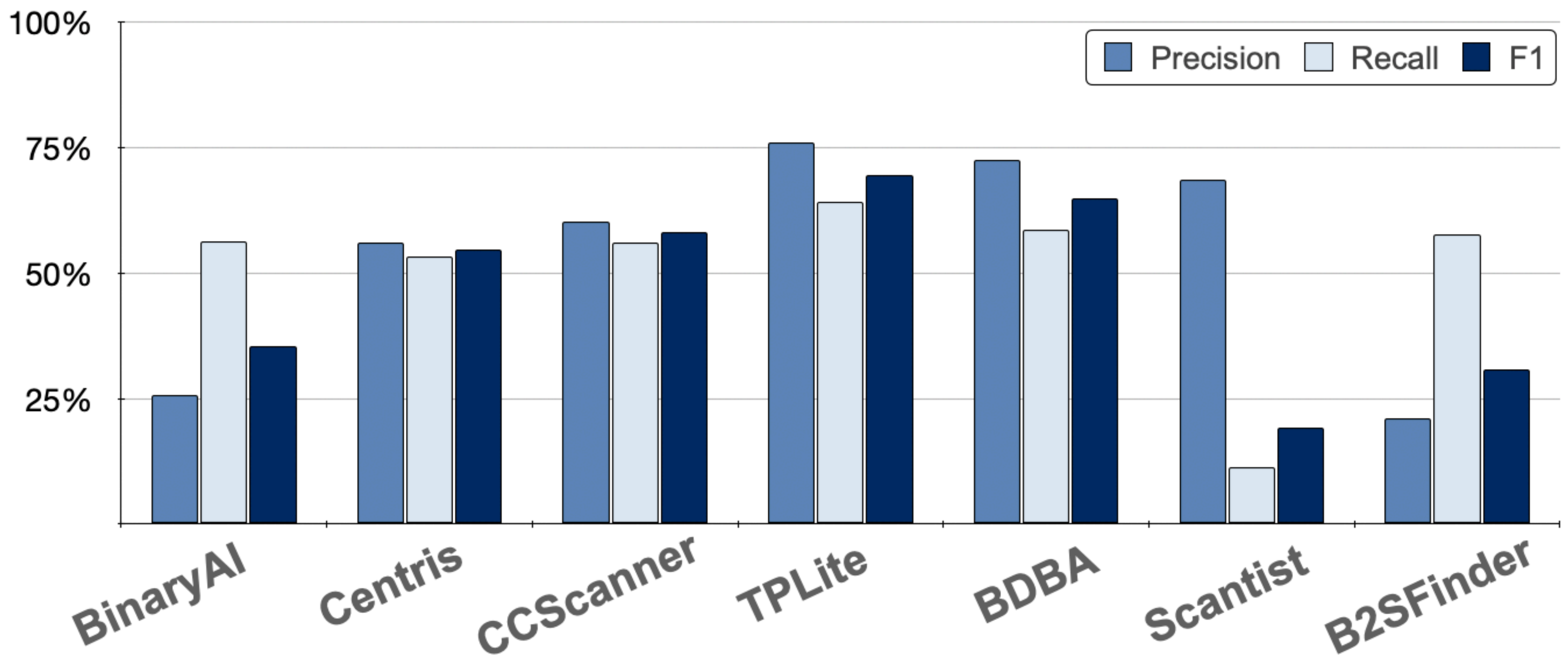
Subjects

- **B2SFinder**: state-of-the-art academic binary SCA tool
- **Black Duck Binary Analysis (BDBA)**: well-adopted commercial SCA product by Synopsys
- **Scantist**: commercial SCA platform including binary analysis

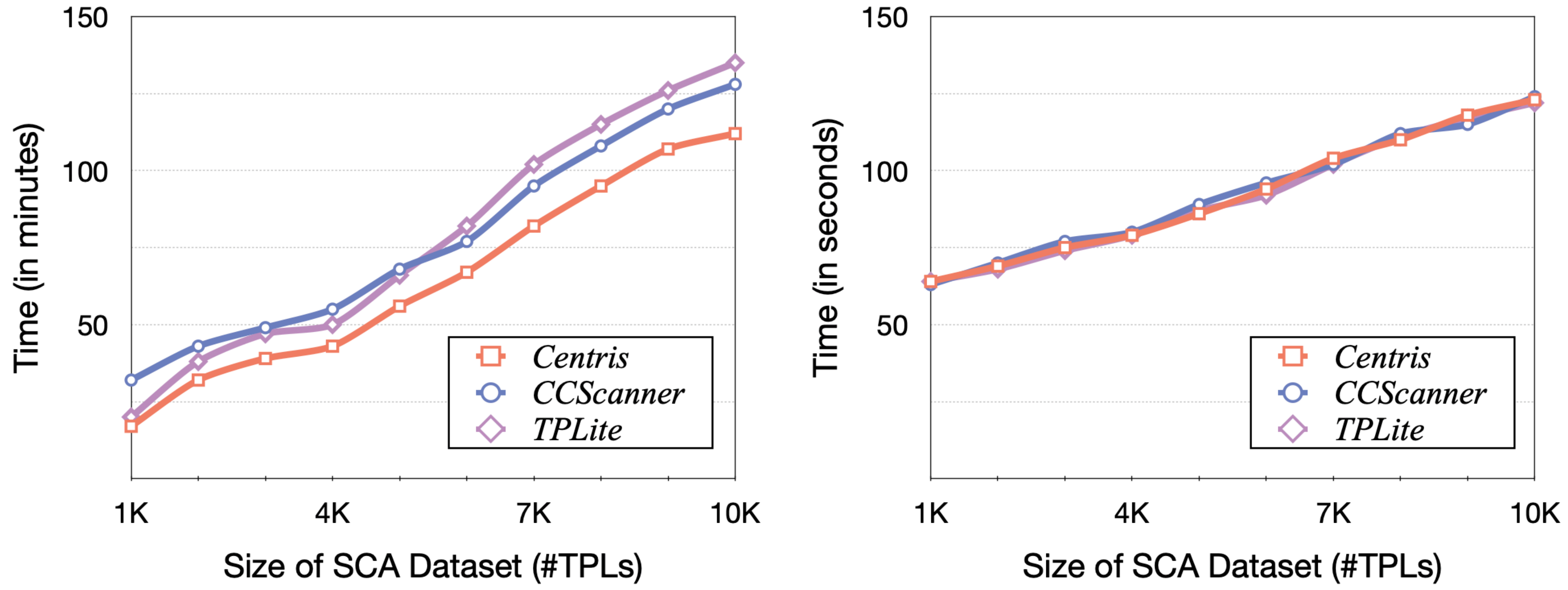
Evaluation of SCA

- TPLite increases precision (**25.76%** to **75.90%**) and recall (**56.34%** to **64.17%**) when integrated to BinaryAI
- TPLite dominates performance among binary SCA tools, and even outperforms BDBA (**75.90%** vs.**72.46%** precision, **64.17%** vs. **58.55%** recall)
- Additional overhead of TPLite is tolerable for both offline TPL reuse detection and online SCA

The SCA results



Time cost of SCA tools



(a) TPL reuse detection **(b) SCA**

Extensive Study

- Study of Centris, the state-of-the-art SCA technique in the C/C++ ecosystem
- Accuracy of TPL dependencies and impact on SCA may not well generalize to our evaluation dataset
- Function birth time and threshold-based recall can be the factors to degrade the performance

Technique Improvement

- We propose TPLite based on the study findings and we are the first to adapt the TPL dependencies to the binary-level SCA, both of them outperform the existing state-of-the-art techniques

- Open-sourced technique: https://github.com/Tricker-z/TPLite

Thanks

Q & A