DIPLOMATIST: What Do Cross-language Dependencies Reflect Software Ecosystem Health?

Fanyi Meng
Shenyang University of Technology
Shenyang, China
mengfanyineu@163.com

Ying Wang*
Northeastern University
Shenyang, China
wangying@swc.neu.edu.cn

Chun Yong Chong
Monash University Malaysia
Subang Jaya, Malaysia
chong.chunyong@monash.edu

Hai Yu
Northeastern University
Shenyang, China
yuhai@mail.neu.edu.cn

Zhiliang Zhu
Northeastern University
Shenyang, China
zhuzhiliang_neu@163.com

***Abstract*—In large-scale software development, multilingual projects, those involving multiple interacting programming languages, have become increasingly common in both industry and the open-source community. Research indicates that cross-language dependencies in these projects can increase the likelihood of risks, such as functionality defects and security vulnerabilities. While most existing studies focus on cross-language dependencies between host languages and specific guest languages (e.g., C/C++), interactions between host languages and a broader range of guest languages, as well as the broader impact of such dependencies on software ecosystems, remain underexplored.**

To address the above limitations, in this paper, we develop a technique, DIPLOMATIST, to identify and analyze cross-language dependencies between host languages, such as Java, and guest languages, including JavaScript, Python, Ruby, PHP, and C/C++. DIPLOMATIST automatically analyzes *cross-language invocation APIs* and constructs a large-scale knowledge repository to standardize code features for identifying library versions across various guest languages, enabling host languages to trace the guest language libraries they invoke. Evaluation shows that DIPLOMATIST achieved an average precision of 88.9% and a recall of 91.5% on a high-quality benchmark, indicating its high accuracy in detecting cross-language dependencies. Using DIPLOMATIST, we identified 435,258 Java libraries that indirectly or transitively depend on libraries from other ecosystems. DIPLOMATIST provides a list of cross-language pivotal libraries that contribute to preserving the long-term health and sustainability of software ecosystems. Moreover, we conduct a case study to examine the impact of the risks introduced due to cross-language dependencies on programming language ecosystems, by analyzing a full-picture of the cross-language dependency graph. Our findings show that fragile projects or libraries can propagate security issues across ecosystems via these dependencies, impacting 13,739 downstream projects in the *Maven* ecosystem. We utilized DIPLOMATIST to provide remediation suggestions to relevant project developers. Issue reports of some subjects have been confirmed by developers.

***Index Terms*—Cross-language Dependency, Software Ecosystem, Empirical Study.**

I. INTRODUCTION

Several studies [1]–[10] have shown that software projects involving multiple interacting programming languages, known as *polyglot projects*, are becoming increasingly widespread in both industry and the open-source community. Polyglot software development enables developers to capitalize on the distinct advantages of various programming languages, improving both productivity and flexibility [9], [11], [12]. To facilitate interaction across different programming languages, foreign function interfaces (FFIs), such as *CFFI* [13] and *JNI* [14], provide robust bridging mechanisms that allow a host language to access code in guest languages (e.g., C/C++ native code). For other programming languages as guest languages, such as JavaScript and Python, there is direct interfacing or interaction between host and guest languages via application programming interfaces (APIs) [8]. We refer to these APIs as ***cross-language invocation APIs***.

In polyglot projects, Java is prevalently used as the host language, invoking other programming languages to address specific tasks and enhance efficiency in software development. Our investigation shows that, as of January 1, 2024, 6.8% and 64.3% of polyglot *Maven* projects directly or transitively use libraries implemented in guest languages (such as Python, JavaScript, Ruby, PHP, and C/C++). Research has shown that cross-language dependencies can increase the likelihood of risks in polyglot projects, such as functionality defects [15] and security vulnerabilities [16]. Therefore, prior study [16] performed cross-language dependency analysis between Java or Python, and C/C++, assisting developers in identifying issues caused by cross-language dependencies. ***However, they primarily focus on cross-language dependency analysis between host languages and specific guest languages, such as C/C++. The impact of cross-language dependencies between other different programming languages on the health of their software ecosystems has not yet been explored.*** To fill the knowledge gap, we selected Java as the host language and Python, JavaScript, Ruby, PHP, and C/C++ as guest languages due to their popularity, and conducted an empirical study on potential risks introduced by cross-language dependencies posed to programming language ecosystems.

In this study, we developed a technique, DIPLOMATIST,

*Corresponding author.

2643-1572/25/$31.00 ©2025 IEEE
DOI 10.1109/ASE63991.2025.00158

which identifies cross-language dependencies between different programming languages. The development of DIPLOMATIST required tackling the following two challenges: *(1) Lack of configuration files for guest languages, which makes it difficult to identify the guest language libraries used by the host language.* Our investigation shows that, among the 41,694 polyglot projects analyzed, 33.9% (9,808/28,935) lack JavaScript configuration files, 66.2% (1,774/2,681) lack Python, 35.7% (634/1,778) lack Ruby, and 39.8% (319/801) lack PHP files. *(2) Lack of essential static analysis tools impedes accurate detection of cross-language dependencies between host languages (e.g., Java) and guest languages (e.g., JavaScript, Python, Ruby, or PHP).* At present, no automated technique can (i) extract cross-language invocation APIs with high precision or (ii) pinpoint the concrete guest-language source files referenced by the host code, because each guest language introduces its own syntactic rules and binding mechanisms. To fill this gap, DIPLOMATIST analyzes cross-language invocation APIs (see Section 3.2) and constructs a knowledge repository for the library version of each guest language to trace the guest language libraries invoked by host languages (e.g., Java). We briefly introduce the proposed approach below.

First, we evaluate the effectiveness of DIPLOMATIST by exploring the following research question.

- **RQ1 (Effectiveness of DIPLOMATIST):** ***How effective is DIPLOMATIST in identifying cross-language dependencies?***

To address RQ1, we constructed a high-quality benchmark to assess DIPLOMATIST's ability to identify cross-language dependencies. Our results show that DIPLOMATIST achieved an average precision of 88.9% and a recall of 91.5%, demonstrating high accuracy in identifying cross-language dependency relationships. Building on this validated technique, we leveraged DIPLOMATIST to investigate the landscape of cross-language dependencies across different ecosystems.

- **RQ2 (Scale of Cross-language Libraries):** ***What is the scale of Java libraries that depend on libraries from other programming language ecosystems?***
- **RQ3 (Depth of Cross-language Dependencies):** ***What is the depth of transitive dependencies for cross-language libraries?***

The above empirical study was performed based on the data collected from the open-source discovery service *Libraries.io* [17]. Among the analyzed 612,249 Java libraries, we observed that only 41,694 (6.8%) directly depend on libraries from other programming language ecosystems, whereas a majority of 393,564 (64.3%) transitively depend on libraries from other ecosystems. Most guest language libraries invoked by host languages (Java) are concentrated within layers 2 to 8 of the dependency tree. While developers are generally aware of issues introduced by direct cross-language dependencies in their projects, they may difficult to recognize the risks posed by transitive cross-language dependencies. Then, we utilized DIPLOMATIST to examine the impact of cross-language dependencies on different programming language ecosystems, with the aim to explore the following two research questions:

- **RQ4 (Impacts of Pivotal Libraries):** ***How do pivotal libraries influence software development and maintenance?***
- **RQ5 (Impact of Fragile Libraries):** ***How do fragile libraries influence software development and maintenance?***

Due to cross-language dependencies, installing pivotal Java libraries—those with a significant impact across entire ecosystems (see **Definition 4**)—may require developers to account for up to 734 libraries across various ecosystems. Fragile libraries introduce security risks across different ecosystems (see **Definition 5**), affecting a total of 13,739 downstream projects in the *Maven* ecosystem. It is thus necessary to employ DIPLOMATIST to continuously capture fragile libraries and suggest proper remediation strategies based on impact analysis. We employed DIPLOMATIST to provide remediation suggestions to relevant project developers. Issue reports of some subjects have been confirmed by developers. Overall, DIPLOMATIST provides an identified list of cross-language pivotal or fragile libraries that are critical to maintaining the long-term health and sustainability of software ecosystems.

In summary, this paper makes three major contributions:

- ***An approach for automatically identifying cross-language dependencies.*** We developed a technique for cross-language dependency analysis, DIPLOMATIST, to automatically and reliably uncovers dependency links between Java and five guest ecosystems—JavaScript, Python, Ruby, PHP, and C/C++.
- ***An empirical study on the impacts of cross-language dependencies.*** Leveraging DIPLOMATIST, we conducted the empirical study to examine the impact of issues introduced by cross-language dependencies on various programming language ecosystems, aiming to raise developers' awareness of potential risks.
- ***A list of cross-language pivotal libraries.*** We maintain a list of cross-language pivotal libraries most frequently pulled into Java projects, providing maintainers and security auditors with a watch-list of high-leverage components.
- ***A reproduction package.*** We provided a reproduction package, including datasets, the tool, and raw experimental data, at (***https://diplomatist-dependencies.github.io/***) to facilitate future research.

II. PRELIMINARIES

To set the stage for our study, we define concepts used throughout this paper to explore the impact of cross-language dependencies on various programming language ecosystems.

Definition 1. Cross-language Dependency Graph ($CLDG$): The $CLDG$ is a directed graph $G = (V, E)$, where

- $V = V_H \cup V_G$ is a set of nodes, consisting of Java projects and libraries (V_H) as the host language, and libraries from other programming language ecosystems (V_G) such as JavaScript, Python, Ruby, and PHP, as well as C/C++ which are considered as the guest languages.
- $E = E_C \cup E_H \cup E_G$ is a set of directed edges. The distinct types of edges are described as follows: (❶) $E_C =$

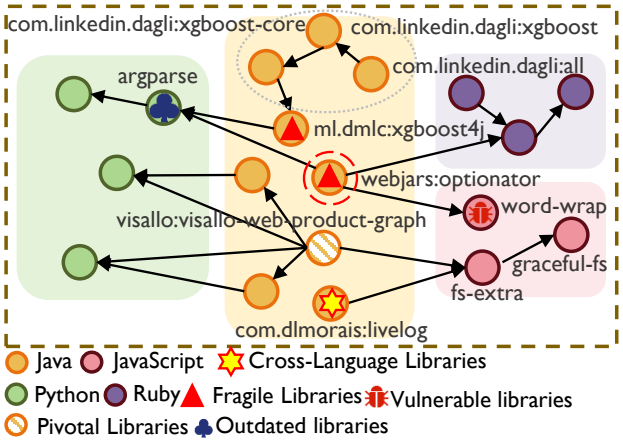

Fig. 1: Illustrative Examples of Libraries in the Cross-Language Dependency Graph (CLDG)

$\{(h_i, g_j) \,|h_i \in V_H, g_j \in V_G\}$ represents the dependency relationships between the projects implemented in host language and the libraries implemented in guest languages. (❷) $E_H = \{(h_i, h_s) \,|h_i, h_s \in V_H \wedge i \neq s\}$ represents the dependency relationships among the libraries implemented in host language. (❸) $E_G = \{(g_j, g_t) \,|g_j, g_t \in V_G \wedge j \neq t\}$ represents the dependency relationships among libraries that implement in the same guest language.

Definition 2. Cross-language Libraries (CL): $CL = \{h_i \in V_H | \exists e = (h_i, g_j) \in E_C\}$ refers to the Java libraries that depend on libraries implemented in guest languages.

As shown in Figure 1, the cross-language library `com.dlmorais:livelog 0.08` depends on the JavaScript library `fs-extra 0.30.0` via executing the JavaScript file `file.js`.

Definition 3. Cross-language Dependencies (CLD): CLD denotes dependencies from Java libraries to libraries implemented in guest languages.

Definition 4. Pivotal Libraries (PL): PL are cross-language libraries that serve as important hubs connecting libraries implemented in different guest languages and significantly influence various programming language ecosystems.

We utilize the widely used HITS algorithm [18] to quantify the significance of cross-language libraries and advise developers to prioritize maintaining pivotal libraries within them. Our work constructs a Cross-Language Dependency Graph (CLDG) that represents relationships between Java and guest languages, including JavaScript, C/C++, Python, Ruby, and PHP. CLDG comprises 435,258 Java, 223,692 JavaScript, 131,796 C/C++, 8,204 Python, 6,692 Ruby, and 3,503 PHP libraries. Considering the scale of the graph, computing node importance across the entire structure is computationally expensive. Therefore, we adopt the HITS algorithm, which efficiently operates on focused subgraphs [19]. HITS algorithm starts from a set of hub nodes—in our case, cross-language libraries—and iteratively propagates influence through their outgoing edges. This design makes it well-suited for subgraphs rooted in specific nodes and significantly reduces computational overhead. In contrast, algorithms like PageRank and Betweenness Centrality require a global view of the graph—either through random walks from all nodes or shortest paths between all pairs—making them less appropriate when analysis is confined to localized subgraphs. The cross-language library `visallo:visallo-web-product-graph 4.0.0` was identified as a pivotal library using the HITS algorithm, with a high out-degree of 734 and a hub score of 0.567, as shown in Figure 1. This is notable given the simplified representation of its dependencies in the figure.

Definition 5. Fragile Libraries (FL): FL represents Java projects that directly depend on libraries implemented in guest languages that exhibits at least one of the following quality deficits:

- *Outdated libraries:* Libraries that have not been maintained by developers for more than N years. Based on the approach defined by Huang et al. [20], DIPLOMATIST sets N = 3 and collects the last updated date of each library in the *Maven Central Repository* during analysis.
- *Vulnerable libraries:* Libraries contain the vulnerabilities that have been disclosed in vulnerability databases. DIPLOMATIST collects this information from the *GitHub Advisory DB* [21] and the *Snyk Vulnerability DB* [22].

By identifying fragile libraries, we can raise developers' attention to issues caused by cross-language dependencies. As shown in Figure 1, the cross-language library `webjars:optionator 0.9.1` is identified as a fragile library because it depends on the JavaScript library `word-wrap 1.2.3` via executing the JavaScript file `help.js`, which contains high-severity level vulnerability `#CVE-2023-26115`.

Definition 6. Dependency Reach of Fragile Libraries (DR): DR represents the set of all libraries that directly or transitively depend on the fragile library.

Figure 1 shows that the dependency reach of the fragile library `ml.dmlc:xgboost4j 0.90` includes Java libraries `com.linkedin.dagli:xgboost-core 15.0.0-beta9`, `com.linkedin.dagli:all 15.0.0-beta9`, and `com.linkedin.dagli:xgboost 15.0.0-beta9`.

III. DIPLOMATIST APPROACH

In this paper, we developed a technique, DIPLOMATIST, to analyze the impact of cross-language dependencies on various programming language ecosystems. We identify relationships between host languages, such as Java, and guest languages, including JavaScript, Python, Ruby, PHP, and C/C++. DIPLOMATIST comprises two stages: guest language configuration files parsing, and cross-language invocation APIs analysis.

DIPLOMATIST first parses the guest language configuration files (such as *package.json* for JavaScript, *setup.py*, *setup.cfg*, *pyproject.toml*, and *requirements.txt* for Python, *Gemfile* for Ruby, or *composer.json* for PHP) within Java project archives to identify cross-language dependencies. If these configuration files are unavailable, DIPLOMATIST converts Java bytecode to Jimple code. It then identifies publicly accessible APIs in the host language (Java) that load guest language files or codes, enabling it to trace the executed guest language files from

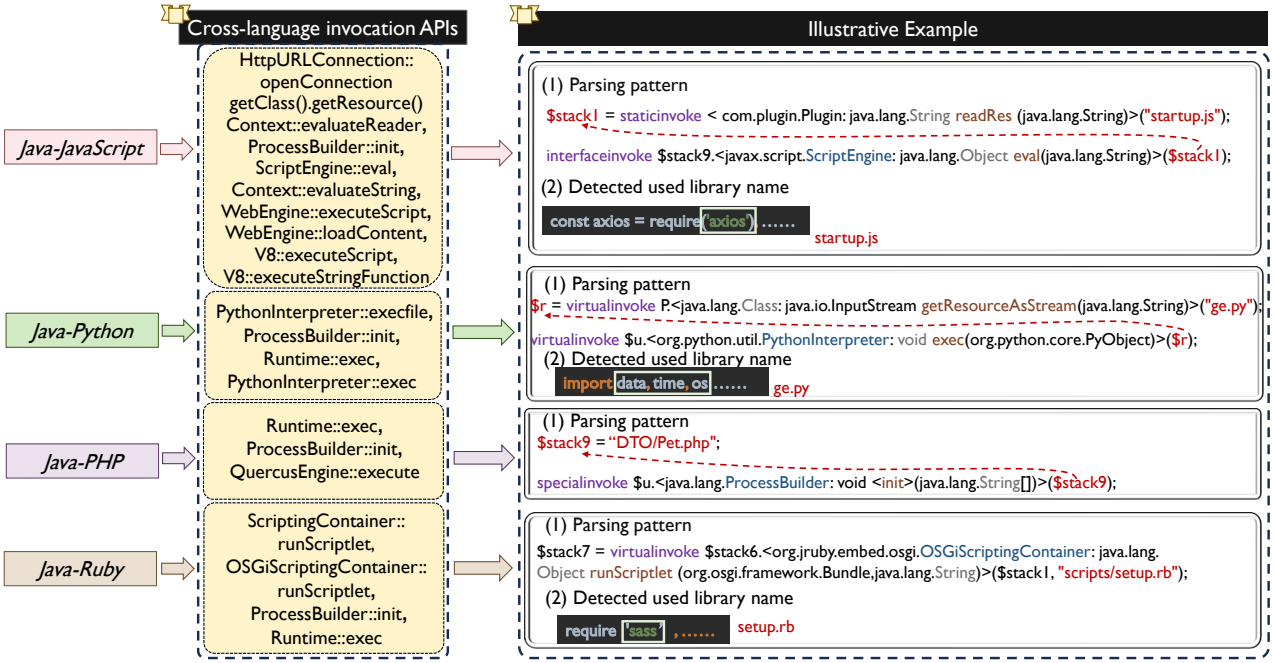

Fig. 2: DIPLOMATIST's parsing patterns for various languages

the Jimple representation. These publicly accessible APIs are sourced from official API documentation and reference materials [23]. We refer to these publicly accessible APIs as ***cross-language invocation APIs***. DIPLOMATIST subsequently identifies cross-language dependencies based on the constructed code features repository for library versions of each guest language, and the detected guest language code loading by host language. Finally, it constructs a cross-language dependency graph to analyze the impact of these dependencies on various programming language ecosystems.

A. Guest Language Configuration Files Parsing

Developers in the *Maven* ecosystem are required to publish project archives to central repositories. Polyglot *Maven* project archives typically contain binaries, Java configuration files, and guest language configuration files, which include all necessary cross-language dependencies, such as libraries implemented in guest languages.

- ***Guest language configuration files extraction.*** DIPLOMATIST extracts guest language configuration files from Java archives, such as `*.jar` or `*.aar`. We focus on guest language configuration files, which include *package.json* for JavaScript, *setup.py*, *setup.cfg*, *pyproject.toml*, and *requirements.txt* for Python, *Gemfile* for Ruby, and *composer.json* for PHP.
- ***Guest language configuration files parsing.*** We analyze the extracted configuration files based on each guest language's unique structure, syntax, and semantics to identify cross-language dependencies and determine version information for these libraries.

B. Cross-language Invocation APIs Analysis

Only a small fraction of polyglot projects provide configuration files for guest languages. Among the 41,694 polyglot projects obtained (see Table I), 66.1% (19,127/28,935), 66.2% (1,774/2,681), 64.3% (1,144/1,778), and 60.2% (482/801) lack JavaScript, Python, Ruby, and PHP configuration files, respectively. When guest language configuration files are unavailable, we can identify cross-language dependencies by analyzing the cross-language invocation APIs in the host language (Java). These APIs load and execute guest language files or codes, allowing us to trace the executed files and identify the libraries implemented in the guest languages. To address the absence of configuration files, DIPLOMATIST performs cross-language dependency analysis in two steps:

Step 1: Converting Java byte code to Jimple code. Since the source code of the most collected Java libraries is unavailable, we use the Soot framework [24] to convert Java bytecode into the intermediate representation Jimple [25]. This enables us to identify cross-language invocation APIs in the Jimple files. DIPLOMATIST uses distinct parsing patterns to analyze cross-language dependencies between Java and guest languages.

Step 2: Analyzing Cross-Language Invocation APIs. While Java depends on JavaScript, Python, PHP, and Ruby, DIPLOMATIST identifies cross-language invocations APIs within the Jimple files of the analyzed project. Furthermore, it parses the arguments of the cross-language invocation APIs to locate the guest language files or code executed by the host language. Finally, DIPLOMATIST extracts libraries implemented in guest languages by analyzing the files or code loaded through cross-language invocation APIs. Figure 2 illustrates the cross-

language invocation APIs for different guest languages and the corresponding parsing patterns for each language.

- ***Java → JavaScript, Java → Python, Java → Ruby, and Java → PHP:*** We first identify the names of guest language libraries invoked by Java. Then, for JavaScript, Python, PHP, and Ruby, we detect their versions using feature databases.
  a) Statement Identification. We analyze JavaScript (`*.js`), Python (`*.py`), and Ruby (`*.rb`) source files to extract external library usage. Specifically, we extract "`import`" and "`require`" statements from JavaScript, "`import`" and "`from...import`" statements from Python, and "`require`" statements from Ruby. To distinguish external dependencies from local modules, we exclude statements that reference relative paths. In contrast to the other languages, PHP does not provide a dedicated import statement, so we construct the knowledge repository for PHP directly.
  b) Extracting key features of guest language library versions. Using Abstract Syntax Tree (AST) analysis, we extract features from the identified guest language libraries to support version identification, including:
  - *Key APIs/functions*: Core functions provided by a library, defining its main functionalities.
  - *Exported symbols*: Elements (like functions or classes) that a library makes available externally.
  - *Unique identifiers*: Version-specific markers embedded within the library's source code.

  c) Constructing a knowledge repository to Standardize Code Features. DIPLOMATIST constructs a comprehensive knowledge repository by collecting library version metadata for each guest language from the open-source discovery service *Libraries.io* [17]. Each version of a library is represented as a feature vector derived from the three types of information described above. To encode these features, we adopt Word2Vec [26], a widely used pre-trained model developed by *Google*, to generate word vector representations for libraries in each guest language. This metadata is represented as a 2-tuple (l, f), where:
  - l denotes the library name along with its version number;
  - f represents the vector capturing the extracted features from the library's source code.

  d) Examining library version through feature similarity. We extract the feature vector f_1 from the guest language file under analysis and iteratively compare it against the feature vectors f_2 of libraries stored in our knowledge repository. To quantify the similarity between features, we adopt cosine similarity [27], defined as follows:

$$cosine\,(f_1, f_2) = \frac{f_1 \cdot f_2}{\|f_1\|\,\|f_2\|} \tag{1}$$

  DIPLOMATIST determines the version of a guest language library by matching it against entries in the knowledge repository when the cosine similarity score exceeds 0.8, following the threshold established in prior research [28].
- ***Java → C/C++***: We first use the regular expression "`lib.+.so(.[0-9]+){0,3}`" to extract C/C++ binary files (e.g., `libavfilter.so.7`) from Java archives. We then utilize BINARYAI [29] to determine the corresponding C/C++ library versions and identify cross-language dependencies between Java and C/C++.

Java projects frequently execute HTML pages that load external JavaScript from Content Delivery Network (CDN) URLs via `<script>` tags. A CDN is a network of servers that is geographically dispersed to enable faster web performance [30]. To recover these cross-language links, we begin by scanning Java source and bytecode for invocation APIs such as `HttpURLConnection::openConnection()` and `getClass().getResource()`. By identifying these APIs, we can locate the executed HTML. Through the CDN, JavaScript libraries are linked in the HTML `<script>` tag using the "`src`" attribute, such as `<script src="https://code.jquery.com/jquery-3.6.0.min.js "></script>`. We can examine the "`src`" attribute to obtain the guest language libraries used and their versions.

After obtaining the cross-language dependencies, we gather dependency relationships among libraries implemented in the same language using the open-source discovery service *Libraries.io* to construct the cross-language dependency graph. Upon constructing the graph, we can analyze the impact of cross-language dependencies on various programming language ecosystems.

IV. EMPIRICAL STUDY AND EVALUATION

This section evaluates the effectiveness of DIPLOMATIST and leverages it to explore the landscape of cross-language dependencies across different ecosystems.

A. Data Collection

1) Libraries in Host Languages Collection: At the time of carrying out our experiment, we collected Java projects/libraries' metadata using the open-source discovery service *Libraries.io* [17] based on the snapshot of the *Maven* ecosystem dated January 1, 2024. In this study, we deployed DIPLOMATIST to iteratively analyze 612,249 Java projects from the *Maven* ecosystem, all of which were the latest released versions as of that date. We aim to identify cross-language dependencies between Java libraries and libraries from other programming language ecosystems, such as JavaScript, Python, Ruby, PHP, and C/C++. We focus on these ecosystems for our study because they are widely used and significant library central repositories.

2) Libraries in Guest Languages Collection: To construct the cross-language dependency graph, DIPLOMATIST identifies library versions in guest languages that cross-language libraries depend on by analyzing both code files and configuration files specific to each guest language. Table I summarizes the statistics of libraries implemented in guest language ecosystems detected from these projects, including 31,956 JavaScript, 1,172 Python, 956 Ruby, and 612 PHP libraries.

B. RQ1: Effectiveness of DIPLOMATIST

Study Methodology. To evaluate DIPLOMATIST's capability in detecting cross-language dependencies, we constructed a benchmark consisting of real cross-language dependencies

TABLE I: Empirical results of RQ2

Results	*Java → JavaScript*	*Java → C/C++*	*Java → Python*	*Java → Ruby*	*Java → PHP*
#Cross-Language Libraries (Direct)			41,694/612,249 6.8%		
#Java Libraries Directly Depending on Libraires from Other Ecosystems	28,935/41,694 69.4%	22,433/41,694 53.8%	2,681/41,694 6.4%	1,778/41,694 4.3%	801/41,694 1.9%
#Direct Dependencies	464,054	142,110	9,367	25,065	6,389
#Libraries of guest languages (Direct) (/# All Libraries Released in Ecosystems)	31,956/4,976,772 0.6%	38,828/829,242 4.7%	1,172/557,370 0.2%	956/177,605 0.5%	612/450,432 0.1%
#Cross-Language Libraries (Transitive)			393,564/612,249 64.3%		
#Java Libraries Transitively Depending on Libraires from Other Ecosystems	198,918/393,564 50.5%	194,646/393,564 49.5%	104,170/393,564 26.5%	182,262/393,564 46.3%	3,652/393,564 0.9%
#Transitive Dependencies	984,161	496,615	208,342	318,956	7,508
#Libraries of guest languages (Transitive) (/# All Libraries Released in Ecosystems)	191,736/4,976,772 3.9%	92,968/829,242 11.2%	7,032/557,370 1.3%	5,736/177,605 3.2%	2,891/450,432 0.6%

All Libraries Released in Ecosystems: The total counts for *npm*, *PyPI*, *RubyGems*, and *Packagist* libraries were obtained from *Libraries.io* as of January 1, 2024. The total counts for C/C++ libraries were derived from existing work with Insight [16], which collects metadata from *Ubuntu* [31], *Debian* [32], and *Arch Linux* [33].

documented in configuration files. The benchmark dataset construction process involves two steps:

- *Collecting Polyglot Maven Projects:* Our investigation revealed that only a small fraction of polyglot projects include configuration files for guest languages (see Section 3.2). Thus, we located a set of projects on `GitHub` that use Java as the host language and determined whether their code repositories contain dependency configuration files from guest languages. The configuration files include *package.json* for JavaScript, *setup.py setup.cfg*, *pyproject.toml*, and *requirements.txt* for Python, *Gemfile* for Ruby, or *composer.json* for PHP. The benchmark collection satisfies the following criteria: (1) Our initial filtering targeted popular polyglot projects on GitHub that use Java as the host language. This step resulted in a list of 12,056 projects matching this criterion.We then examined their code repositories to determine whether they contained dependency configuration files for guest languages, yielding 3,631 polyglot projects. (2) To minimize the likelihood of including unused dependencies or overlooking those actually used in practice, we selected projects whose configuration files had been updated at least five times in the past year, yielding 58 projects. Table II shows the statistics of our benchmark. We obtained 58 polyglot projects, including 23, 34, 25, and 31 projects containing JavaScript, Python, Ruby, and PHP configuration files, involving 1,087, 371, 362, and 327 dependencies, respectively.
- *Identifying Cross-Language Dependencies:* We analyze configuration files based on the unique structure, syntax, and semantics of each guest language to identify cross-language dependencies as the benchmark. For the detection of dependencies between Java and C/C++, we rely on prior work [16]. Therefore, we focus on evaluating the effectiveness of dependencies between Java and JavaScript, Python, Ruby, and PHP.

Specifically, we evaluate the effectiveness of Diplomatist by comparing its library version detection based on feature databases with benchmark results.

Evaluation Metrics. Leveraging our constructed benchmark, we evaluate using six metrics: (1) ***True Positive (TP)***: the cross-language dependencies identified by Diplomatist are recorded in the benchmark; (2) ***False Positive (FP)***: the cross-language dependencies identified by Diplomatist are not recorded in the benchmark; (3) ***False Negative (FN)***: the cross-language dependencies are recorded in the benchmark but not identified by Diplomatist. Based on the above three metrics, we can obtain the *Precison*, *Recall* and *F-measure* as follows: (a) ***Precision*** = *TP* / (*TP* + *FP*); (b) ***Recall***: *TP* / (*TP* + *FN*); (c) ***F-measure***: 2 × *Precision* × *Recall* / (*Precision* + *Recall*). *Precision* evaluates whether Diplomatist can identify cross-language dependencies precisely. *Recall* evaluates the capability of Diplomatist in detecting all the cross-language dependencies. *F-measure* combines the *Precision* and *Recall* [34].

TABLE II: Statistics of our constructed benchmark

	JavaScript	*Python*	*Ruby*	*PHP*
#Polyglot Maven Projects	23	34	25	31
#Cross-language dependencies (Direct)	1,087	371	362	327

TABLE III: Effectiveness of analyzing different types of dependencies

Metrics \ *Types*	*Java → JavaScript*	*Java → Python*	*Java → Ruby*	*Java → PHP*
TP	994	336	331	303
FP	109	49	43	35
FN	93	35	31	24
Precision	90.1%	87.3%	88.5%	89.6%
Recall	91.4%	90.6%	91.4%	92.7%
F-measure	90.8%	88.9%	89.9%	91.1%

Results. Table III presents the experimental results for RQ1. Diplomatist identifies 2,200 cross-language dependencies with an average *Precision* of 88.9%, *Recall* of 91.5%, and *F-measure* of 90.2% for all types of dependencies. As shown in Table III, our findings indicate that Diplomatist successfully identified 994 out of 1,087 (91.4%) cross-language dependencies between Java and JavaScript, 336 out of 371 (90.6%) between Java and Python, 331 out of 362 (91.4%) between Java and Ruby, and 303 out of 327 (92.7%) between Java and PHP. Based on the results, we can conclude that Diplomatist can effectively identify different types of cross-language dependencies.

FP Analysis. In cross-language dependency analysis, Diplomatist sometimes misidentifies library versions implemented in guest languages due to limited feature information. For example, Diplomatist incorrectly identified

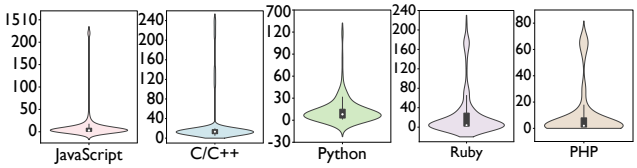

Fig. 3: Number of guest languages libraries in each polyglot project

version 0.18.1 of the JavaScript library `thrift` as version 0.19.0, which is a cross-language dependency of the project `org.webjars.bower:thrift`. The JavaScript file `parse.js` (invoked by this polyglot project) uses the library thrift and contains minimal code, resulting in insufficient data for feature extraction and incorrect version identification.

FN Analysis. DIPLOMATIST produces 183 (*FNs*) for all types of dependencies because it parses the arguments of the cross-language invocation APIs and performs analysis to trace the data flow but fails to identify the data source. This failure occurs due to ambiguous argument parsing. For example, DIPLOMATIST is unable to parse string arguments that involve string arrays. In addition, the missing of other language files in the `*.jar` or `*.aar` further hinders accurate parsing of cross-language dependencies.

> ***Answer to RQ1:*** *DIPLOMATIST achieves an average Precision of 88.9% and a Recall of 91.5%, achieving high detection rates in identifying cross-language dependency relationships.*

C. RQ2: Scale of Cross-language Libraries

Study Methodology. For a large-scale empirical study, we used DIPLOMATIST to analyze 612,249 projects/libraries to identify cross-language libraries. Specifically, we gathered the statistics on these libraries that directly or transitively depend on libraries from other programming language ecosystems.

Results. Table I summarizes the statistics of our investigation results. By analyzing 612,249 Java projects/libraries, we identified 41,694 (6.8%) and 393,564 (64.3%) cross-language libraries that directly or transitively depend on libraries in the other programming language ecosystems. Most Java projects (69.4% and 53.8%, respectively) directly depend on JavaScript for web application development and on C/C++ libraries for high-performance computing tasks. A small fraction of Java projects (4.3%) directly depend on Ruby libraries. However, a significant portion (46.3%) of Java projects transitively depend on Ruby libraries through projects that use Ruby frameworks. For example, we observed that a popular project, `Selenium` [35] (30.1k stars on `GitHub`), utilizes Ruby libraries to facilitate browser automation tasks and execute test scripts, supporting test-driven development. Currently, there are 4,629 downstream projects that utilize `Selenium` for web browser automation. Some projects may introduce libraries of guest languages into numerous Maven ecosystem projects through transitive cross-language dependencies, which should raise developers' awareness of potential risks such as compatibility issues, security vulnerabilities, and maintenance challenges.

Upon further inspection of the polyglot projects, we identified a total of 73,524 guest language libraries used within the Maven ecosystem as of January 1, 2024. The distribution plot in Figure 3 illustrates the number of guest language libraries within each polyglot Maven project. Our analysis reveals that the number of these libraries ranges from 1 to 1,510 for JavaScript, 1 to 242 for C/C++, 1 to 700 for Python, 1 to 226 for Ruby, and 1 to 91 for PHP. For instance, a popular project, `liferay-portal` [36] (2.1k stars on `GitHub`), depends on 252 JavaScript libraries to manage batch processing within the Liferay Portal framework, particularly for data manipulation and UI enhancements. Besides, another project, `common-domain-model` [37] (121 stars on `GitHub`) depends on 700 Python libraries to establish a standardized common domain model for financial services, facilitating effective data sharing and processing across different systems. Most polyglot *Maven* projects tend to introduce multiple JavaScript or Python libraries, making them more susceptible to increasing their attack surface through cross-language dependency chains.

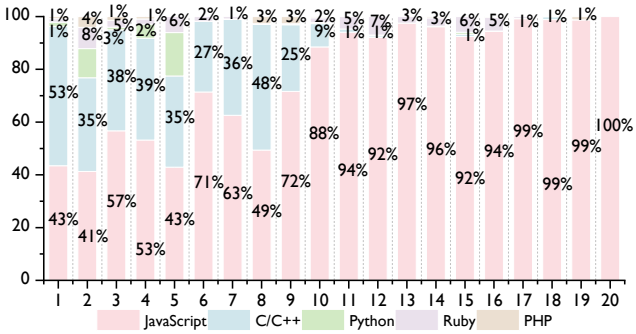

Fig. 4: Depth of cross-language dependencies

> ***Answer to RQ2:*** *Among the analyzed 612,249 Java libraries, only 41,694 (6.8%) directly depend on libraries from other programming language ecosystems, whereas a majority of 393,564 (64.3%) transitively depend on libraries from other programming language ecosystems.*

D. RQ3: Depth of Cross-language Dependencies

Study Methodology. Based on the cross-language dependency graph, we use DIPLOMATIST to calculate the depth of cross-language dependencies. The depth of cross-language dependencies denotes the length of the longest directed path from the root project in the Maven ecosystem, through its transitive dependencies, to a library implemented in a guest language. By analyzing the depth of cross-language dependencies, we can identify potential risks within programming language ecosystems due to issues introduced by numerous transitive dependencies.

Results. Figure 4 illustrates the dependency depth of Java projects on libraries from other programming language ecosystems, with levels ranging from 1 to 20. Most Java projects depend on libraries implemented in the five analyzed guest languages, with a dependency tree depth that typically does not exceed 9 levels. Most Java projects have cross-language dependency depths of three. Specifically, at a cross-language dependency depth of three, 57% (7,893) of Java projects depend on JavaScript libraries, 38% (5,325) on C/C++ libraries, 1% (143)

on Python libraries, 5% (365) on Ruby libraries, and 3% (194) on PHP libraries, as illustrated in Figure 4. It's notable that a small number of Java projects (358) exhibit a cross-language dependency depth of up to 20 for JavaScript libraries. This depth indicates a highly complex cross-language dependency chain, likely involving a combination of various JavaScript libraries, and frameworks integrated into the polyglot project. Frameworks such as `Vue.js`, `React`, and `Angular` frequently depend on numerous underlying JavaScript libraries, which can lead to extensive and complex dependency chains.

Developers may difficult to be aware of the potential risks introduced by transitive cross-language dependencies. For example, the project `datahub` [38] directly depends on the Python library `azure-identity` (<1.16.1), which is affected by the security vulnerability `CVE-2024-35255`. The library `azure-identity` itself depends on the Python library `msal`. In this project's configuration file, `msal` is pinned to version 1.22.0. However, to address the security vulnerability in `azure-identity`, the updated version is incompatible with the current version (1.22.0) of library `msal`. Therefore, it's necessary to update the library `msal` to version 1.24.0 or higher to prevent conflicts, as discussed in this project's issue [39]. Moreover, the Java project `cohorte-herald` depends on the Python library `Requests`, which in turn requires the library `ssl` for handling secure connections. However, since the `ssl` library isn't available on Arduino Yun (a microcontroller board designed for IoT applications), `Requests` must be replaced with a custom library, as discussed in the project's issue [40]. These cases demonstrate that limited visibility into the full dependency chain can prevent developers from identifying critical issues. A much larger set of cross-language libraries, often buried several layers deep, may go undetected, leaving projects vulnerable to security risks and maintenance burdens that are difficult to trace back to their source.

> ***Answer to RQ3:*** *In Java ecosystems, direct cross-language links are relatively rare, but the transitive footprint is large. Guest language libraries invoked from Java typically appear at depths 2 through 8 of the dependency tree, rather than at the root level.*
> ***Implication:*** *Direct cross-language calls are usually visible to developers. However, the much larger set of cross-language libraries hidden several layers down often escapes notice, leaving projects exposed to vulnerabilities or maintenance burdens that are difficult to trace back to their source.*

E. RQ4: Impacts of Pivotal Libraries

Study Methodology. We used the widely used HITS algorithm [18], applying complex network theory to the constructed cross-language dependency graph, to quantify the importance of nodes in identifying pivotal libraries. Specifically, we aim to investigate the influence of pivotal libraries on various programming language ecosystems.

Results. DIPLOMATIST identified 304 pivotal libraries using the HITS algorithm. Table IV lists the statistics of the top 20 pivotal libraries. The columns labeled "Out (direct)" and "Out (transitive)" indicate the number of libraries implemented in guest languages that these pivotal libraries directly and transitively depend on. The columns labeled "In (direct)" and "In (transitive)" indicate the number of Maven ecosystem libraries that directly and transitively depend on these pivotal libraries. On average, each pivotal library directly depends on 96 guest language libraries and transitively on 203 guest language libraries. The pivotal library can be influenced by up to approximately 1,517 libraries from guest languages due to transitive cross-language dependencies.

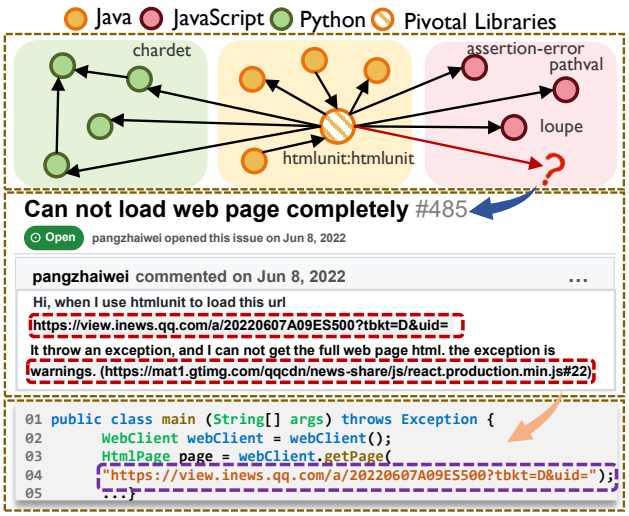

Fig. 5: Issue caused by cross-language dependencies

TABLE IV: Top 20 pivotal libraries

Pivotal Libraries	*Version*	*Out (direct)*	*In (direct)*	*Out (transitive)*	*In (transitive)*
aetr-core_2.12	0.23	977	0	1517	0
visallo-web-product-graph	4.0.0	734	0	1,268	1
com.arianegraphql:server	0.3.0	128	1	133	0
visallo-web-product-map	4.0.0	595	1	675	0
monsoon-api-bin	2.3.1	519	0	520	0
monsoon-api	2.3.1	512	1	513	0
monsoon-prometheus	2.3.1	513	0	514	0
monsoon-verify	2.3.1	519	0	520	0
visallo-web-table	4.0.0	502	1	502	0
visallo-ingest-cloud-s3	4.0.0	475	1	483	0
visallo-core	4.0.0	311	23	367	23
htmlunit	1.1.2	277	23	325	28
webjars.babel:babel	6.26.3	123	0	134	2
stormpath-stormpath-widget	0.01	111	0	125	0
date-fns	2.28.0	102	1	114	0
bs-recipes	1.3.4	99	2	110	3
systemjs-plugin-babel	0.0.25	99	0	109	0
facebook-regenerator	0.10.0	98	0	110	0
babel-cli	6.5.1	87	0	116	0
vue2-google-maps	0.8.12	83	0	93	0

The experiment results of all pivotal libraries can be available at:
https://diplomatist-dependencies.github.io/

When utilizing pivotal libraries, developers should pay attention to the extensive use of guest language libraries, as this may increase the potential risk. If pivotal libraries are involved with issues, they may significantly impact a large number of libraries from the Maven ecosystem due to cross-language dependencies. For example, as shown in Figure 5, the pivotal library `htmlunit` [41] was unable to load the JavaScript file "https://mat1.gtimg.com/qqcdn/news-

share/js/react.production.min.js" on the web page, resulting in an execution failure [42]. As a result, it failed to identify libraries implemented in guest languages that pivotal libraries or projects depend on. Moreover, this failure poses a potential threat to its 28 downstream projects. It is thus necessary to deploy DIPLOMATIST to continuously capture the impact of cross-language dependencies in pivotal libraries and raise widespread concern among developers about these pivotal libraries.

Answer to RQ4: *When installing pivotal libraries, a developer may need to pay attention to the usage of up to 734 libraries from different ecosystems due to cross-language dependencies. Such pivotal libraries with extensive cross-language dependencies represent a potential Achilles' heel in an ecosystem: issues with build failures, or dynamic loading problems in just one of these libraries can significantly impact a large number of libraries across multiple software ecosystems.*
Implication: *Continuous monitoring of pivotal libraries across ecosystems is crucial for fostering healthy open-source communities. This proactive approach helps mitigate risks, ensure compatibility, and maintain the stability and reliability of software projects that rely on these libraries.*

F. RQ5: Impacts of Fragile Libraries

Study Methodology. In this evaluation, we explore the impact of fragile libraries (see **Definition 5**) across ecosystems. In other words, we measure the dependency reach (i.e., the number of downstream projects) of the identified fragile libraries. Specifically, we investigate the influences of issues introduced by cross-language dependencies on ecosystems. As of January 1, 2024, DIPLOMATIST detected 101 fragile libraries or projects. Additionally, we used DIPLOMATIST to provide remediation suggestions to relevant project developers, reporting only those that met two specific criteria: (1) their code repositories had commit records within the past year (non-outdated), and (2) their dependency reach was greater than five. Based on these criteria, we selected eight subjects for further case study. Figure 6 illustrates the cross-language dependency graph of these projects and highlights issues introduced through cross-language dependencies.

Results. In Table V, we use the dependency reach to estimate the impact of issues introduced by cross-language dependencies across different ecosystems. Our case study findings indicate that these subjects propagate security issues from other programming language ecosystems into the Maven ecosystem by cross-language dependencies, affecting a total of 13,739 downstream projects. Moreover, on average, these projects received 5,787 stars, indicating substantial attention from numerous developers. The high likelihood of reuse in popular projects may lead to issues spreading to a large amount of projects simultaneously. Consequently, it is essential to continuously detect fragile libraries as well as suggest proper remediation strategies with impact analysis.

As shown in Table 5, three issue reports have been confirmed by developers, while the other issues are still pending likely due to the inactive maintenance of the projects. For instance, the polyglot *Maven* project `dbt-dataops-starrocks`, which depends on the vulnerable Python library `pymysql` (version $\geq$ 1.1.0, `CVE-2024-36039`). `dbt-dataops-starrocks` is a popular project (Stars: 8.9K) and is directly or transitively used by 34 *Maven* projects. Moreover, the vulnerability (`#CVE-2024-36039`) with a high severity level has a significant impact on downstream projects. As a result, in issue #52450 [43], the developer acknowledged and commented that "*I think you can pull a request to bump it*". Similarly, the project `apache.dolphinscheduler:dolphinscheduler-spi` also depends on a vulnerable Python library, as shown in Figure 6. In issue #16742 [44], the developer regarded this as a serious security risk for their project and asked me to contribute a pull request to address the vulnerability.

Project `io.springfox:springfox-swagger-ui` has two cross-language dependencies on the JavaScript libraries `@babel/traverse` (version `^7.2.3`) and `semver` (version `^5.3.0`), which are affected by vulnerabilities `#CVE-2023-45133` and `#CVE-2022-25883`, respectively. This project impacts 7,039 downstream projects, including 9 pivotal libraries or projects. Additionally, the project `org.scala-lang:scala-compile` affects a substantial number of (6,340) downstream projects. Projects `org.webjars.npm:micromatch`, `springframework:spring-cloud-dataflow-core`, `com.aventstack:extentreports`, and `org.apache.unomi:unomi-api` all depend on a vulnerable JavaScript library and introduce security issues in the *Maven* ecosystem. In fact, the fixed versions of fragile projects could be adopted by 13,379 downstream projects, which effectively decreases the attack surface within the *Maven* ecosystem.

Answer to RQ5: *We observed that fragile libraries or projects can introduce security issues from other software ecosystems via cross-language dependencies, thereby impacting the substantial number of downstream projects in the Maven ecosystem.*
Implication: *Continuously detecting fragile libraries and providing remediation strategies based on impact analysis can mitigate the risks cross-language dependencies pose to downstream projects.*

V. DISCUSSION

Limitations. (1) DIPLOMATIST may fail to detect cross-language dependencies when invocation targets are dynamically generated at runtime through string concatenation or similar techniques, as such behavior is inherently invisible to static analysis. (2) DIPLOMATIST leverages a constructed feature database to identify the versions of guest language libraries invoked by Java projects. However, due to the distinct code structures of guest languages such as JavaScript, Python, Ruby, and PHP, we rely on a limited set of features when constructing the database, which may lead to misidentifications in cross-language dependency analysis.

Threats to Validity. (1) ***Internal validity.*** One possible threat may be affected by the identification of guest language libraries. We employ a pre-trained Word2Vec model to extract version-specific features and match them against a

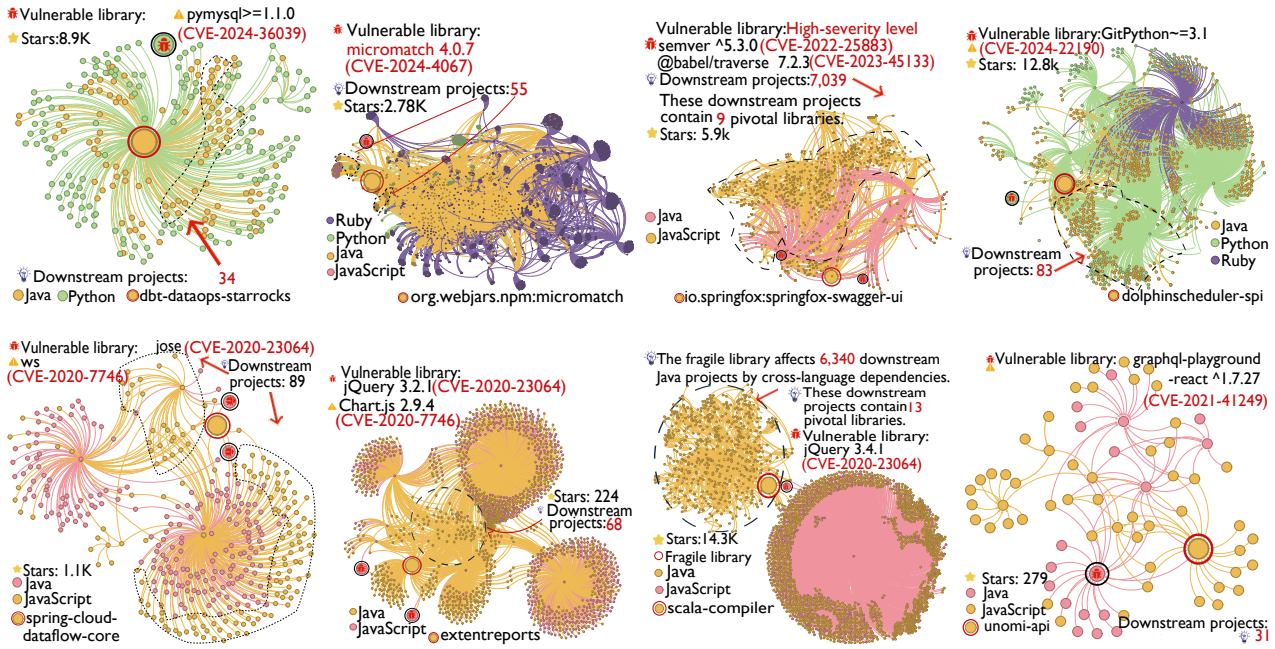

Fig. 6: The cross-language dependency graph of selected subjects

TABLE V: The dependency reach of fragile libraries

Fragile libraries	*Version*	*#Stars*	*Dependency Reach*	*Issue Introduced by Cross-language Dependencies*	*Issue ID*	*Status*
`dbt-dataops-starrocks`	1.4.1	8.9K	34	pymysql≥ 1.1.0 (CVE-2024-36039)	#52450	✓
`org.webjars.npm:micromatch`	4.0.5	2.78K	55	micromatch 4.0.7 (CVE-2024-4067)	#263	✓
`io.springfox:springfox-swagger-ui`	3.0.0	5.9K	7,039	@babel/traverse 7.2.3 (CVE-2023-45133) semver ^5.3.0 (CVE-2022-25883)	#4092	–
`apache.dolphinscheduler:dolphinscheduler-spi`	3.1.9	12.8K	83	GitPython (CVE-2024-22190)	#16742	✓
`springframework:spring-cloud-dataflow-core`	2.11.1	1.1K	89	ws ^8.8.1 (CVE-2024-37890) jose 4.9.3 (CVE-2024-28176)	#6027	–
`com.aventstack:extentreports`	5.1.1	224	68	jQuery 3.2.1 (CVE-2020-23064) Chart.js 2.7.2 (CVE-2020-7746)	#444	–
`org.scala-lang:scala-compiler`	2.13.2	14.3K	6,340	jQuery 3.4.1 (CVE-2020-23064)	#11974	–
`org.apache.unomi:unomi-api`	2.4.0	279	31	graphql-playground-react ^1.7.27 (CVE-2021-41249)	#UNOMI-860	–

✓Confirmed issues; – Pending issues; More detailed information of issue reports is on ***https://diplomatist-dependencies.github.io/***

curated knowledge repository. However, minor version changes often leave exported symbols unchanged, possibly leading to misidentification. To reduce this risk, we limit similarity matching to a narrow candidate set. Another potential threat lies in the completeness of our cross-language API set. To mitigate this, we collected widely used APIs from public documentation and references. Diplomatist achieves high accuracy in identifying cross-language dependencies (precision: 88.9%, recall: 91.5%, F1-score: 90.2%). (2) ***External validity.*** The external threat to validity concerns the generalizability of our study results. Our experiments focus solely on open-source polyglot projects and do not cover industrial projects. To mitigate this threat, we conducted a large-scale study collecting projects of varying scales and complexities to comprehensively explore the impact of cross-language dependencies across multiple ecosystems. (3) ***Construct validity.*** We use the dependencies specified in configuration files as our benchmark. However, these files may contain unused dependencies or overlook those actually used in practice. To mitigate this risk and obtain high-quality dependencies, we selected actively maintained projects whose configuration files had been updated at least five times within the past year.

VI. Related Work

Cross-language Dependency Analysis. Prior research has proposed various techniques for analyzing cross-language dependencies, particularly between host languages (e.g., Java, Python) and native code (e.g., C/C++). Hu et al. [45] modeled the semantics of foreign function declarations to construct call graphs for Python and JavaScript invoking C/C++ code. Several works focused on the Java Native Interface (JNI): Fourtounis et al. [46] identified JNI usage by analyzing binaries; Lee et al. [47] statically analyzed polyglot projects using semantic summaries; and Park et al. [48] extended this to cases without source code via binary decompilation. Beyond static analysis, Li et al. [49] introduced POLYCRUISE, a dynamic analysis tool capturing information flows between Python and C, while POLYFUZZ [50] enables greybox fuzzing

for vulnerability discovery in polyglot projects. Xu et al. [16] proposed INSIGHT, a tool for detecting cross-language vulnerabilities (CLVs) in Java/Python and C projects. Other studies, such as Atlas [51] and NativeSummary [52], conduct cross-language analysis between native and Java libraries using static analysis techniques.

However, the above-mentioned techniques focus solely on cross-language dependency analysis between host languages and specific guest languages, such as C/C++. These techniques do not account for cross-language dependencies between host languages and other guest languages, such as JavaScript, Python, Ruby, and PHP. Our work aims to analyze dependency relationships between popular programming host languages, such as Java, and various guest languages, to construct cross-language dependency graphs that provide insights into dependency management, ecosystem health, and potential security risks in a polyglot environment.

Software Ecosystems. Existing studies in this area primarily analyze dependency and maintenance dynamics within individual language ecosystems. For instance, research on the *npm* ecosystem includes studies on contributor behavior [53], deprecation practices [54], vulnerability propagation [55], [56], and malicious package detection [57]. Other ecosystems have also been explored: Decan et al. [58], [59] compared seven software ecosystems. Zhang et al. [60] restored compatible and secure version ranges to automatically mitigate persistent vulnerabilities. Jiang et al. [61] improved library detection in C/C++ with TPLite; Valiev et al. [62] examined sustainability in Python projects; and Yin et al. [63], Zheng et al. [64], and Cui et al. [65] conducted comprehensive investigations into the health of the Rust ecosystem.

Compared with our study, existing works focused on a single ecosystem, using a systematic approach to investigate issues within that specific ecosystem. Instead, we conducted a large-scale study analyzing the impact of cross-language dependencies on multiple programming language ecosystems using the proposed CLDG.

VII. CONCLUSIONS AND FUTURE WORK

We introduced DIPLOMATIST, a novel approach that identifies and analyzes cross-language dependencies between host languages, such as Java, and guest languages, including JavaScript, Python, Ruby, PHP, and C/C++. The evaluation demonstrated the effectiveness of our tool in identifying cross-language dependencies and its associated risks. By making these blind spots visible, DIPLOMATIST equips maintainers and security auditors with actionable intelligence to harden their software supply chains. Future work will extend the analyser with finer-grained semantic fingerprints and support for additional language pairs, driving toward a comprehensive, multi-ecosystem dependency knowledge graph.

ACKNOWLEDGMENT

This work was supported by the National Key Research and Development Program of China (Grant No. 2024YFF0908000) and the China Postdoctoral Science Foundation (Grant No. 2024M750375).

REFERENCES

[1] W. Li, A. Marino, H. Yang, N. Meng, L. Li, and H. Cai, "How are multilingual systems constructed: Characterizing language use and selection in open-source multilingual software," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 3, pp. 1–46, 2024.

[2] H. Yang, Y. Nong, S. Wang, and H. Cai, "Multi-language software development: Issues, challenges, and solutions," *IEEE Transactions on Software Engineering*, vol. 50, no. 3, pp. 512–533, 2024.

[3] H. Yang, Y. Nong, T. Zhang, X. Luo, and H. Cai, "Learning to detect and localize multilingual bugs," *Proceedings of the ACM on Software Engineering*, vol. 1, no. FSE, pp. 2190–2213, 2024.

[4] P. Mayer, M. Kirsch, and M. A. Le, "On multi-language software development, cross-language links and accompanying tools: a survey of professional software developers," *Journal of Software Engineering Research and Development*, vol. 5, pp. 1–33, 2017.

[5] W. Li, N. Meng, L. Li, and H. Cai, "Understanding language selection in multi-language software projects on github," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings*. Madrid, ES: IEEE, 2021, pp. 256–257.

[6] J. Zhang, P. Nie, J. J. Li, and M. Gligoric, "Multilingual code co-evolution using large language models," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, San Francisco, CA, USA, 2023, pp. 695–707.

[7] S. Hong, T. Kwak, B. Lee, Y. Jeon, B. Ko, Y. Kim, and M. Kim, "Museum: Debugging real-world multilingual programs using mutation analysis," *Information and Software Technology*, vol. 82, pp. 80–95, 2017.

[8] W. Li, L. Li, and H. Cai, "Polyfax: a toolkit for characterizing multi-language software," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. Singapore, Singapore: Association for Computing Machinery, 2022, p. 1662–1666.

[9] M. Grichi, E. E. Eghan, and B. Adams, "On the impact of multi-language development in machine learning frameworks," in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. Adelaide, SA, Australia: IEEE, 2020, pp. 546–556.

[10] Z. Li, X. Qi, Q. Yu, P. Liang, R. Mo, and C. Yang, "Exploring multi-programming-language commits and their impacts on software quality: An empirical study on apache projects," *Journal of Systems and Software*, vol. 194, p. 111508, 2022.

[11] M. Abidi, M. S. Rahman, M. Openja, and F. Khomh, "Multi-language design smells: a backstage perspective," *Empirical Software Engineering*, vol. 27, no. 5, p. 116, 2022.

[12] P. Mayer and A. Bauer, "An empirical analysis of the utilization of multiple programming languages in open source projects," in *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*, Nanjing, China, 2015, pp. 1–10.

[13] "Cffi," https://cffi.readthedocs.io/en/latest/, 2024, accessed: 2024-08-01.

[14] "Java native interface(jni)," https://docs.oracle.com/en/java/javase/17/docs/specs/jni/index.html, 2024, accessed: 2024-08-01.

[15] M. Grichi, M. Abidi, F. Jaafar, E. E. Eghan, and B. Adams, "On the impact of interlanguage dependencies in multilanguage systems empirical case study on java native interface applications (jni)," *IEEE Transactions on Reliability*, vol. 70, no. 1, pp. 428–440, 2020.

[16] M. Xu, Y. Wang, S.-C. Cheung, H. Yu, and Z. Zhu, "Insight: Exploring cross-ecosystem vulnerability impacts," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. New York, NY, USA: ACM, 2022, pp. 1–13.

[17] "Libraries.io," https://libraries.io/, 2024, accessed: 2024-08-01.

[18] J. M. Kleinberg, "Authoritative sources in a hyperlinked environment," *Journal of the ACM*, vol. 46, no. 5, pp. 604–632, 1999.

[19] X. Li, M. K. Ng, and Y. Ye, "Har: hub, authority and relevance scores in multi-relational data for query search," in *Proceedings of the 2012 SIAM International Conference on Data Mining*. SIAM, 2012, pp. 141–152.

[20] K. Huang, B. Chen, C. Xu, Y. Wang, B. Shi, X. Peng, Y. Wu, and Y. Liu, "Characterizing usages, updates and risks of third-party libraries in java projects," *Empirical Software Engineering*, vol. 27, no. 4, p. 90, 2022.

[21] "Github advisory db," https://github.com/advisories, 2024, accessed: 2024-08-01.

[22] "Snyk vulnerability db," https://security.snyk.io/vuln/maven, 2024, accessed: 2024-08-01.

[23] D. Bosanac, *Scripting in Java: Languages, Frameworks, and Patterns*. Pearson Education, 2007.

[24] R. Vallée-Rai, P. Co, E. Gagnon, L. Hendren, P. Lam, and V. Sundaresan, "Soot: A java bytecode optimization framework," in *CASCON First Decade High Impact Papers*. Mississauga, Ontario, Canada: IBM Press, 2010, pp. 214–224.

[25] R. Vallee-Rai and L. J. Hendren, "Jimple: Simplifying java bytecode for analyses and transformations," Technical report, McGill University, Tech. Rep., 1998.

[26] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems*, vol. 26, 2013.

[27] A. Singhal *et al.*, "Modern information retrieval: A brief overview," *IEEE Data Eng. Bull.*, vol. 24, no. 4, pp. 35–43, 2001.

[28] T. D. Nguyen, A. T. Nguyen, H. D. Phan, and T. N. Nguyen, "Exploring api embedding for api usages and applications," in *2017 IEEE/ACM 39th International Conference on Software Engineering*. IEEE, 2017, pp. 438–449.

[29] L. Jiang, J. An, H. Huang, Q. Tang, S. Nie, S. Wu, and Y. Zhang, "Binaryai: binary software composition analysis via intelligent binary source code matching," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024, pp. 1–13.

[30] A. Vakali and G. Pallis, "Content delivery networks: Status and trends," *IEEE Internet Computing*, vol. 7, no. 6, pp. 68–74, 2003.

[31] "Ubuntu," https://ubuntu.com/, 2024, accessed: 2024-08-01.

[32] "Debian," https://www.debian.org/, 2024, accessed: 2024-08-01.

[33] "Arch linux," https://archlinux.org/, 2024, accessed: 2024-08-01.

[34] R. Wu, H. Zhang, S. Kim, and S.-C. Cheung, "Relink: recovering links between bugs and changes," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. Szeged, Hungary: ACM, 2011, pp. 15–25.

[35] "selenium," https://github.com/SeleniumHQ/selenium, 2024, accessed: 2024-08-01.

[36] "liferay/liferay-portal," https://github.com/liferay/liferay-portal, 2024, accessed: 2024-08-01.

[37] "finos/common-domain-model," https://github.com/finos/common-domain-model, 2024, accessed: 2024-08-01.

[38] "datahub-project/datahub," https://github.com/datahub-project/datahub, 2024, accessed: 2024-08-01.

[39] "datahub-project/datahub," https://github.com/datahub-project/datahub/issues/11103, 2024, accessed: 2024-08-01.

[40] "cohorte/cohorte-herald," https://github.com/cohorte/cohorte-herald/issues/8, 2024, accessed: 2024-08-01.

[41] "htmlunit," https://github.com/HtmlUnit/htmlunit, 2024, accessed: 2024-08-01.

[42] "Issue htmlunit#48," https://github.com/HtmlUnit/htmlunit/issues/485, 2024, accessed: 2024-08-01.

[43] "issue #52450 of starrocks," https://github.com/StarRocks/starrocks/issues/52450, 2024, accessed: 2024-10-31.

[44] "issue #16742 of dolphinscheduler," https://github.com/apache/dolphinscheduler/issues/16742, 2024, accessed: 2024-10-31.

[45] M. Hu, Q. Zhao, Y. Zhang, and Y. Xiong, "Cross-language call graph construction supporting different host languages," in *2023 IEEE International Conference on Software Analysis, Evolution and Reengineering*. Taipa, Macao: IEEE, 2023, pp. 155–166.

[46] G. Fourtounis, L. Triantafyllou, and Y. Smaragdakis, "Identifying java calls in native code via binary scanning," in *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*. Virtual Event, USA: ACM, 2020, pp. 388–400.

[47] S. Lee, H. Lee, and S. Ryu, "Broadening horizons of multilingual static analysis: Semantic summary extraction from c code for jni program analysis," in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*. Virtual Event, Australia: ACM, 2020, pp. 127–137.

[48] J. Park, S. Lee, J. Hong, and S. Ryu, "Static analysis of jni programs via binary decompilation," *IEEE Transactions on Software Engineering*, vol. 49, no. 5, pp. 3089–3105, 2023.

[49] W. Li, J. Ming, X. Luo, and H. Cai, "{PolyCruise}: A {Cross-Language} dynamic information flow analysis," in *31st USENIX Security Symposium (USENIX Security 22)*, Boston, MA, USA, 2022, pp. 2513–2530.

[50] W. Li, J. Ruan, G. Yi, L. Cheng, X. Luo, and H. Cai, "{PolyFuzz}: Holistic greybox fuzzing of {Multi-Language} systems," in *32nd USENIX Security Symposium (USENIX Security 23)*, Anaheim, CA, USA, 2023, pp. 1379–1396.

[51] H. Xiong, Q. Dai, R. Chang, M. Qiu, R. Wang, W. Shen, and Y. Zhou, "Atlas: Automating cross-language fuzzing on android closed-source libraries," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, Vienna, Austria, 2024, pp. 350–362.

[52] J. Wang and H. Wang, "Nativesummary: Summarizing native binary code for inter-language static analysis of android apps," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, Vienna, Austria, 2024, pp. 971–982.

[53] S. Wattanakriengkrai, D. Wang, R. G. Kula, C. Treude, P. Thongtanunam, T. Ishio, and K. Matsumoto, "Giving back: Contributions congruent to library dependency changes in a software ecosystem," *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 2566–2579, 2022.

[54] F. R. Cogo, G. A. Oliva, and A. E. Hassan, "Deprecation of packages and releases in software ecosystems: A case study on npm," *IEEE Transactions on Software Engineering*, vol. 48, no. 7, pp. 2208–2223, 2021.

[55] C. Liu, S. Chen, L. Fan, B. Chen, Y. Liu, and X. Peng, "Demystifying the vulnerability propagation and its evolution via dependency trees in the npm ecosystem," in *Proceedings of the 44th International Conference on Software Engineering*, Pittsburgh, Pennsylvani, 2022, pp. 672–684.

[56] Y. Wang, P. Sun, L. Pei, Y. Yu, C. Xu, S.-C. Cheung, H. Yu, and Z. Zhu, "Plumber: Boosting the propagation of vulnerability fixes in the npm ecosystem," *IEEE Transactions on Software Engineering*, vol. 49, no. 5, pp. 3155–3181, 2023.

[57] W. Liang, X. Ling, J. Wu, T. Luo, and Y. Wu, "A needle is an outlier in a haystack: Hunting malicious pypi packages with code clustering," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. Luxembourg, Luxembourg: IEEE, 2023, pp. 307–318.

[58] A. Decan, T. Mens, and M. Claes, "An empirical comparison of dependency issues in oss packaging ecosystems," in *2017 IEEE 24th international conference on software analysis, evolution and reengineering*. Klagenfurt, Austria: IEEE, 2017, pp. 2–12.

[59] A. Decan, T. Mens, and P. Grosjean, "An empirical comparison of dependency network evolution in seven software packaging ecosystems," *Empirical Software Engineering*, vol. 24, no. 1, pp. 381–416, 2019.

[60] L. Zhang, C. Liu, S. Chen, Z. Xu, L. Fan, L. Zhao, Y. Zhang, and Y. Liu, "Mitigating persistence of open-source vulnerabilities in maven ecosystem," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. Luxembourg, Luxembourg: IEEE, 2023, pp. 191–203.

[61] L. Jiang, H. Yuan, Q. Tang, S. Nie, S. Wu, and Y. Zhang, "Third-party library dependency for large-scale sca in the c/c++ ecosystem: How far are we?" in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, Seattle, WA, USA, 2023, pp. 1383–1395.

[62] M. Valiev, B. Vasilescu, and J. Herbsleb, "Ecosystem-level determinants of sustained activity in open-source projects: A case study of the pypi ecosystem," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, Lake Buena Vista, FL, USA, 2018, pp. 644–655.

[63] X. Yin, Y. Feng, Q. Shi, Z. Liu, H. Liu, and B. Xu, "Fries: Fuzzing rust library interactions via efficient ecosystem-guided target generation," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, Vienna, Austria, 2024, pp. 1137–1148.

[64] X. Zheng, Z. Wan, Y. Zhang, R. Chang, and D. Lo, "A closer look at the security risks in the rust ecosystem," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 2, pp. 1–30, 2023.

[65] M. Cui, S. Sun, H. Xu, and Y. Zhou, "Is unsafe an achilles' heel? a comprehensive study of safety requirements in unsafe rust programming," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, Lisbon, Portugal, 2024, pp. 1–13.