Semantic Analysis for Deduplication of Security Findings in DevOps Security Tool Reports

Abdullah Gulraiz, 22.08.2022, Master Thesis Final Presentation

Chair of Software Engineering for Business Information Systems (sebis)
Faculty of Informatics
Technische Universität München
wwwmatthes.in.tum.de

Outline

Motivation

Research Questions

1. What are duplicate security tool findings in DevSecOps and how can they be deduplicated?
2. How can we use Natural Language Processing (NLP) to deduplicate security tool findings?
3. How do base NLP Semantic Similarity methods perform when applied to security tool reports corpus?

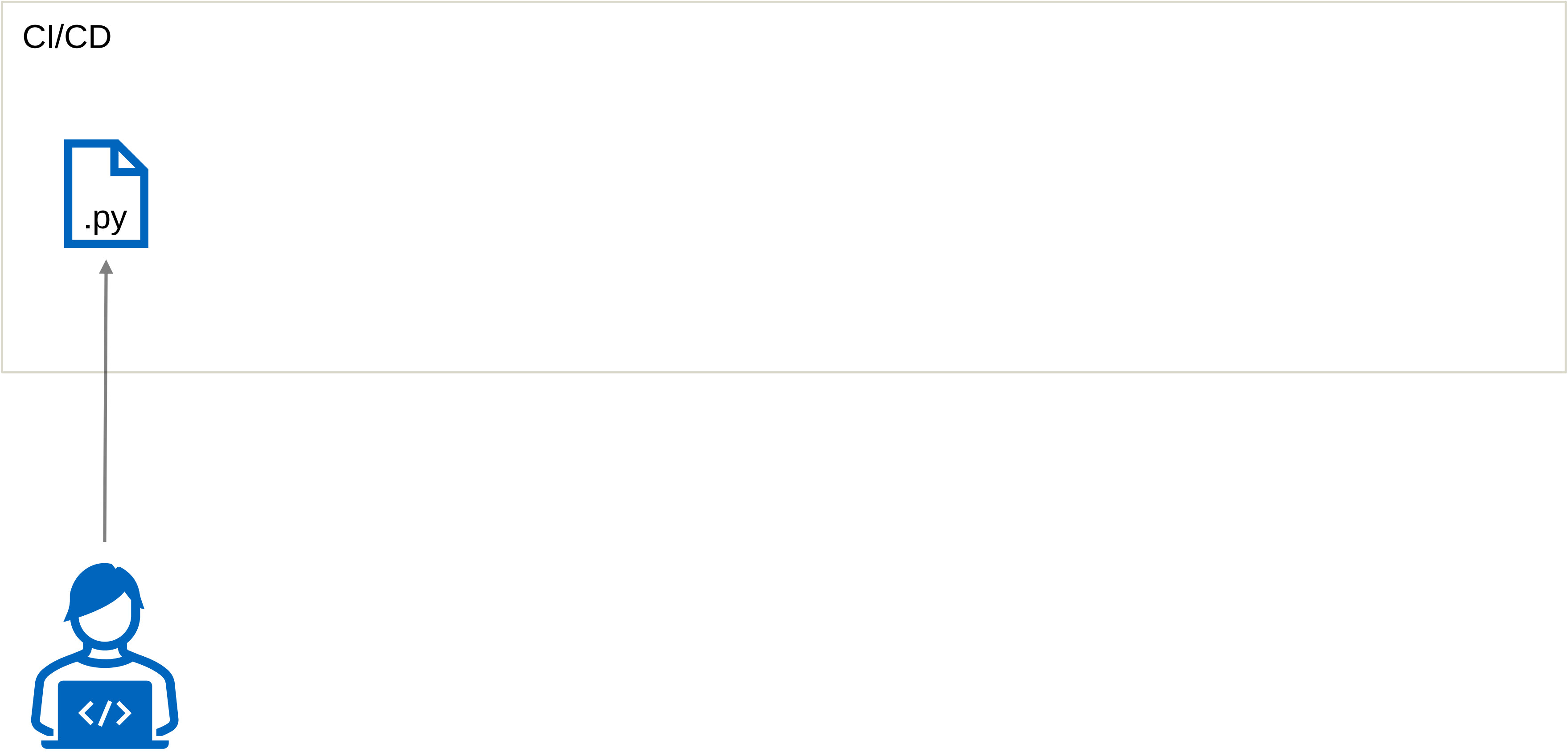
Conclusion

Cybersecurity concepts

- **DevSecOps:** An expansion of DevOps that integrates security controls and processes.
- **Application Security Testing (AST):** Process of scanning an application to identify security vulnerabilities and weaknesses.
- **Static-AST (SAST):** Scanning of code and application artifacts when they are not running.
- **Dynamic-AST (DAST):** Scanning of code and application artifacts in running state.

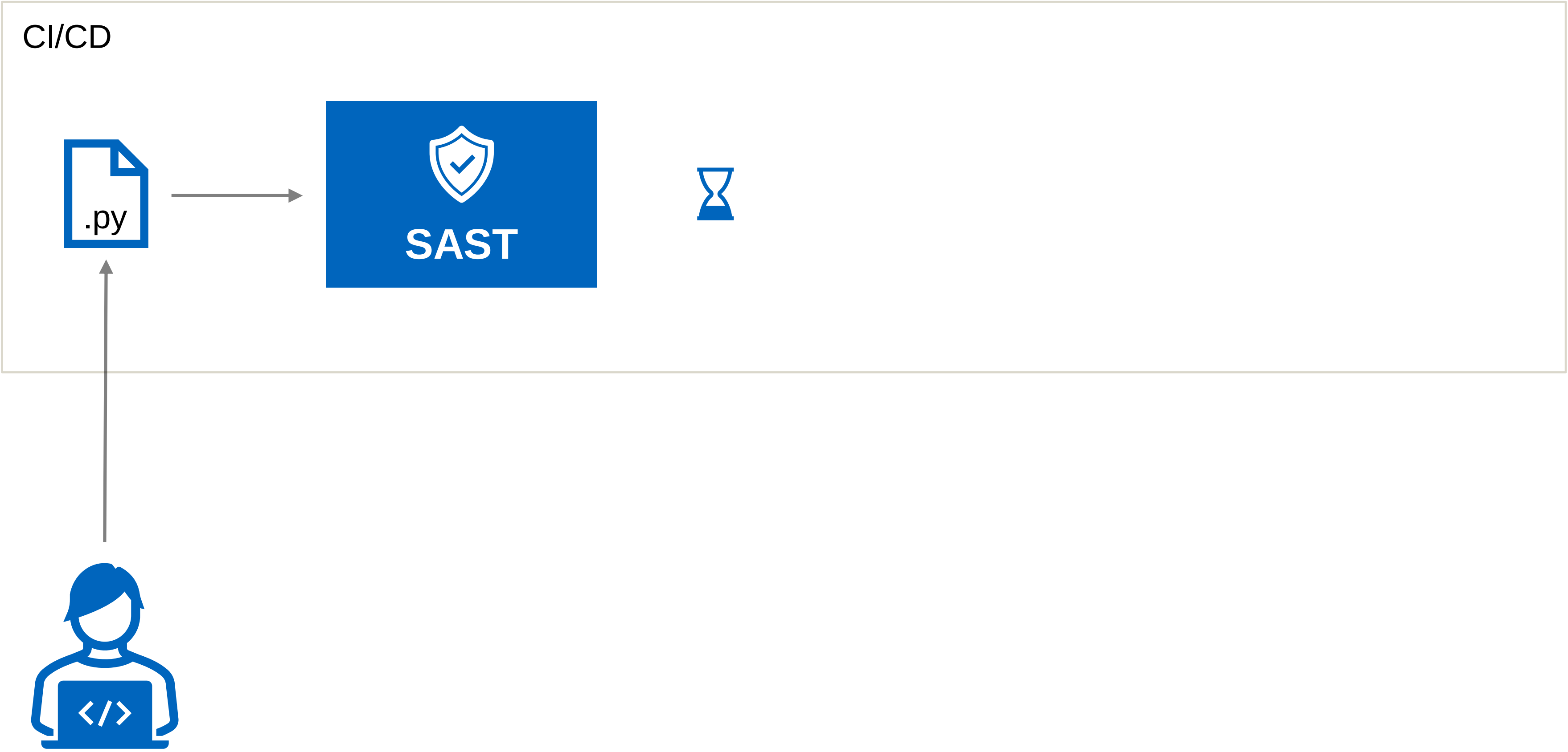
Motivation

Application security testing



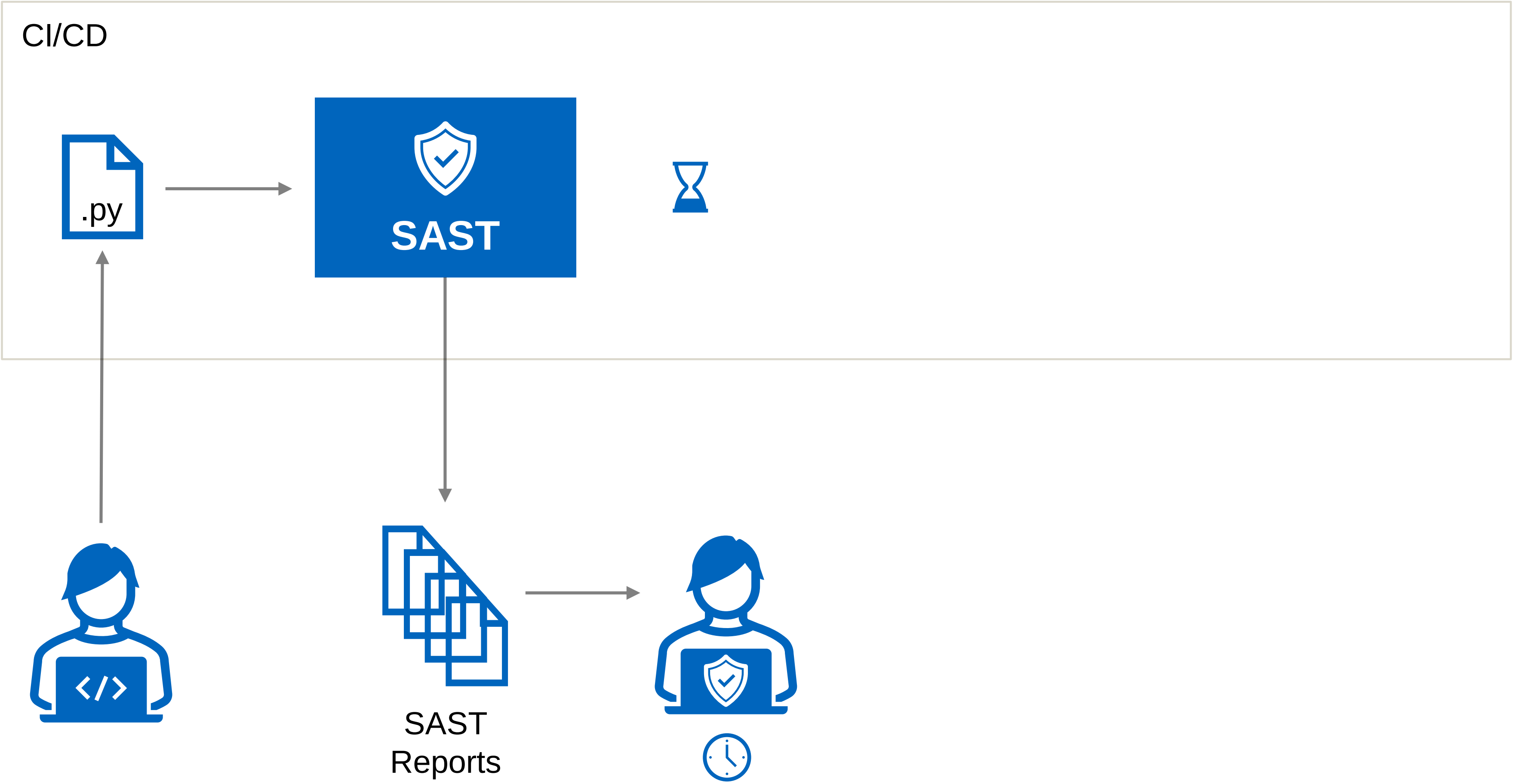
Motivation

Application security testing



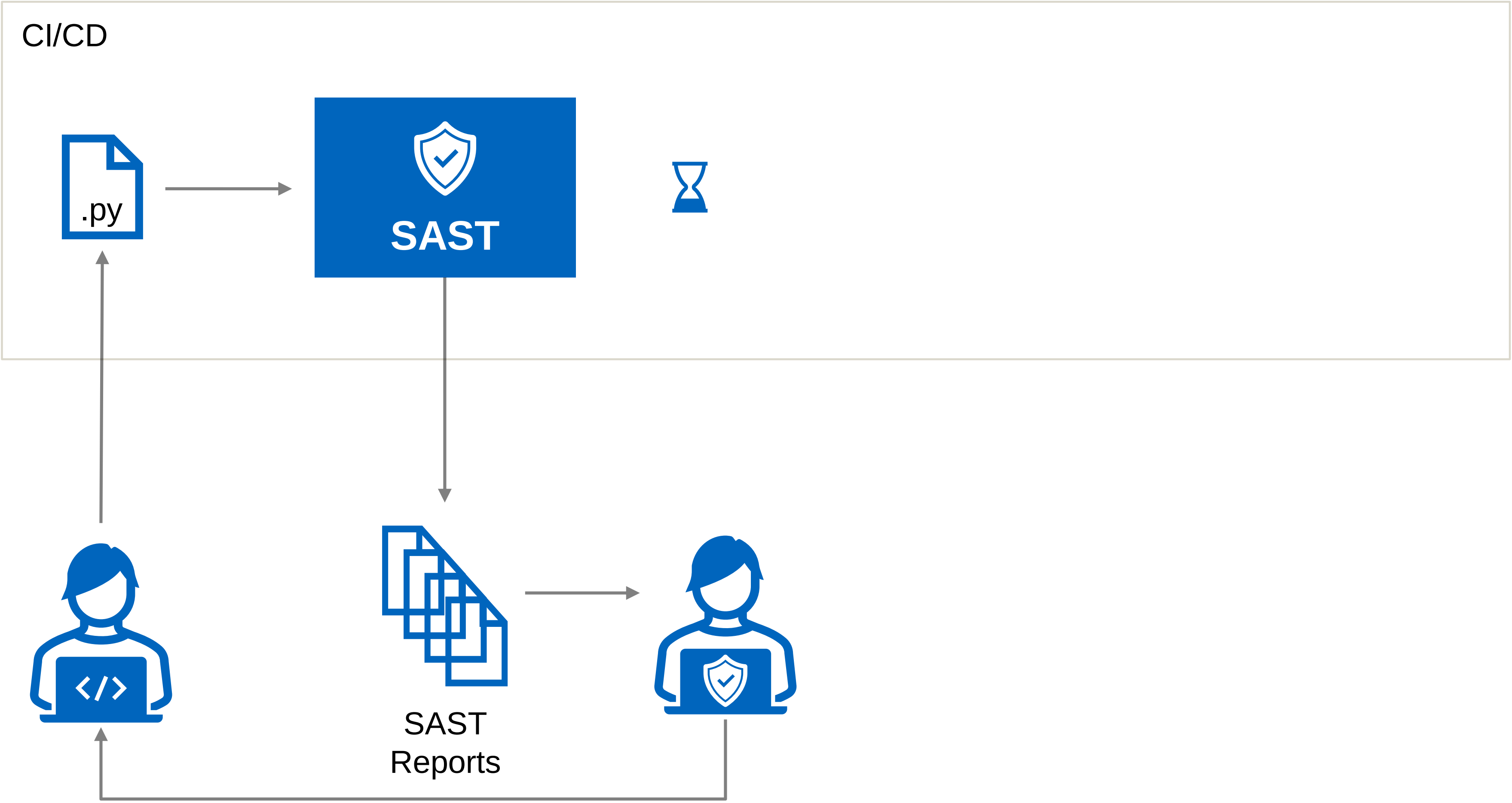
Motivation

Application security testing



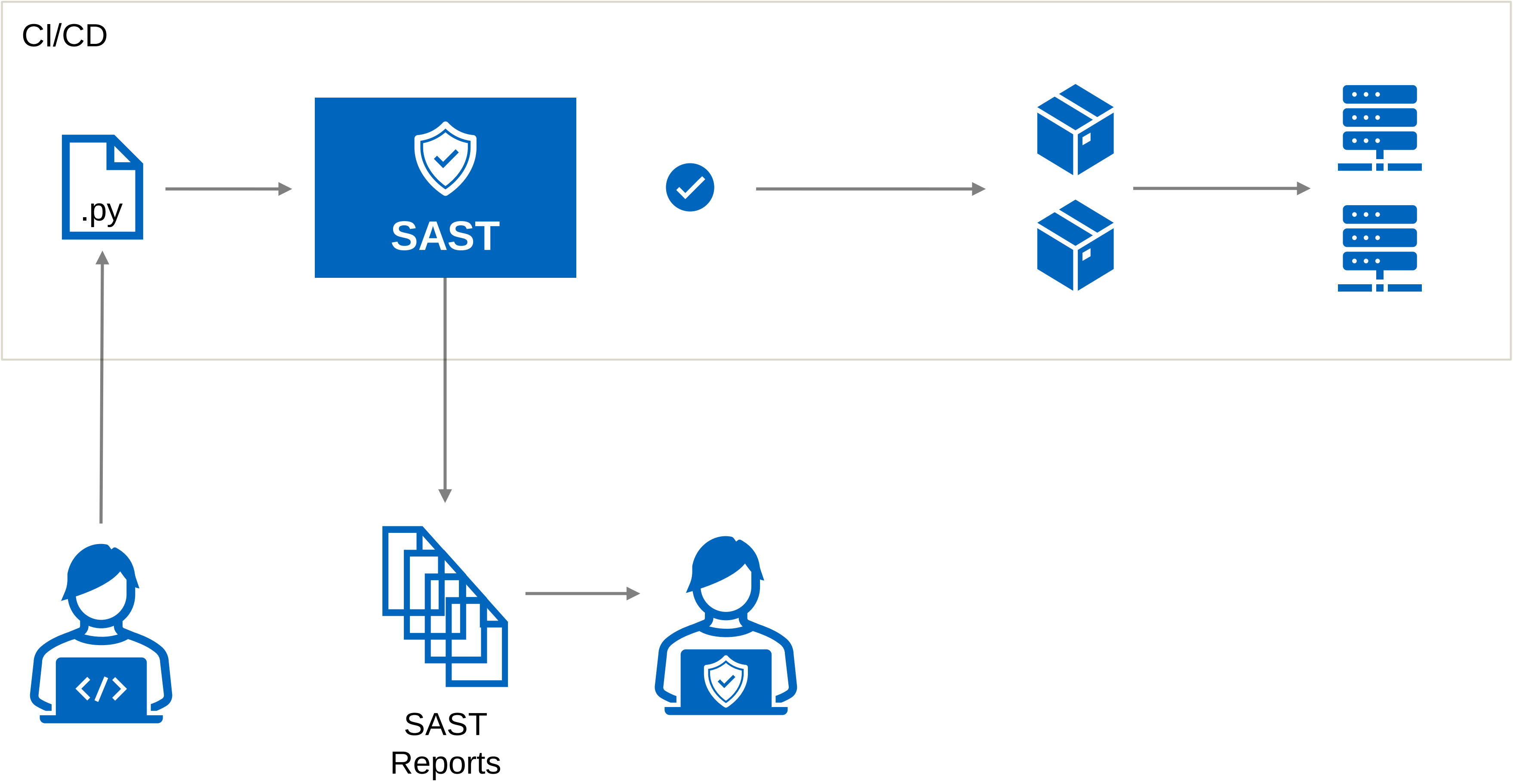
Motivation

Application security testing



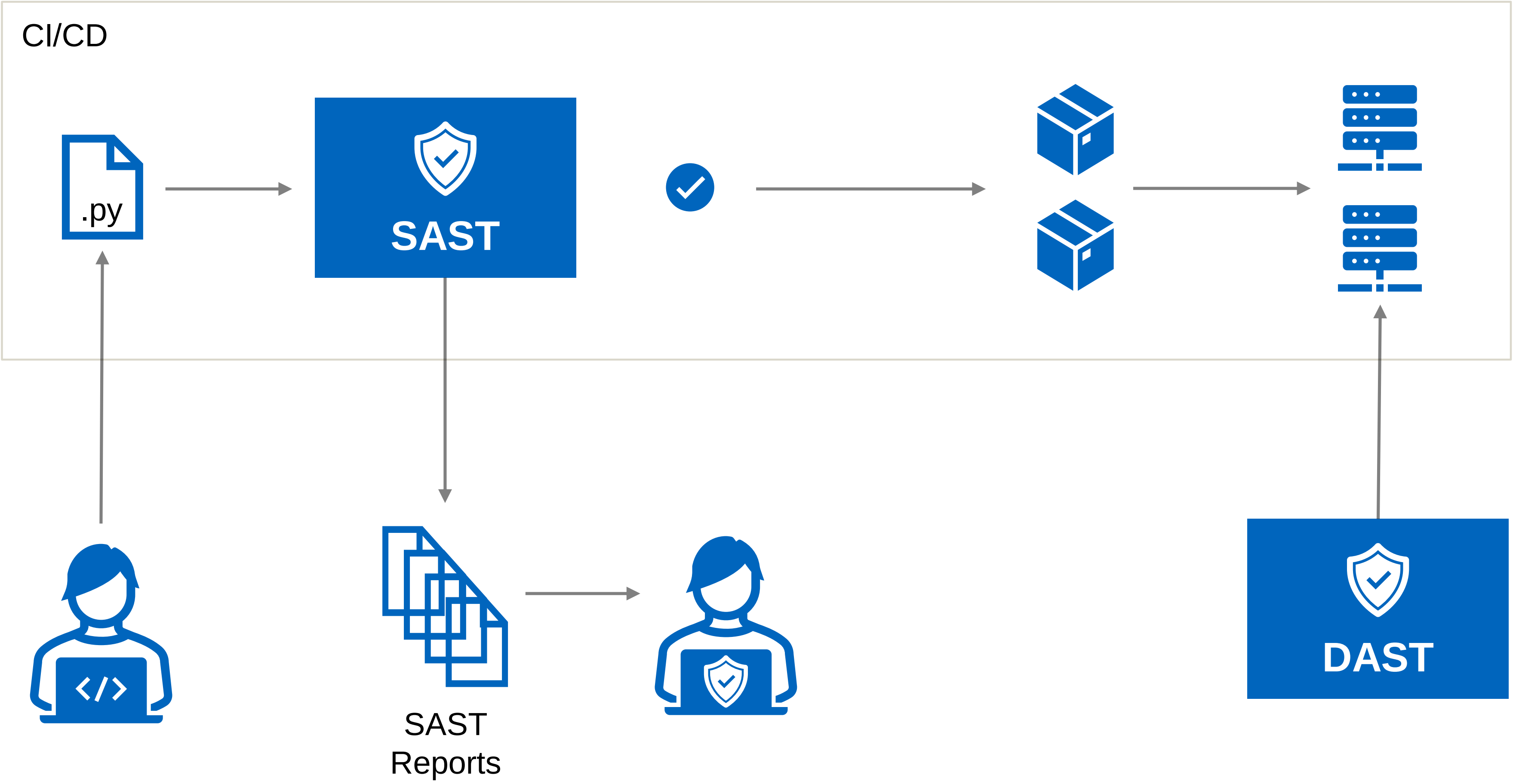
Motivation

Application security testing



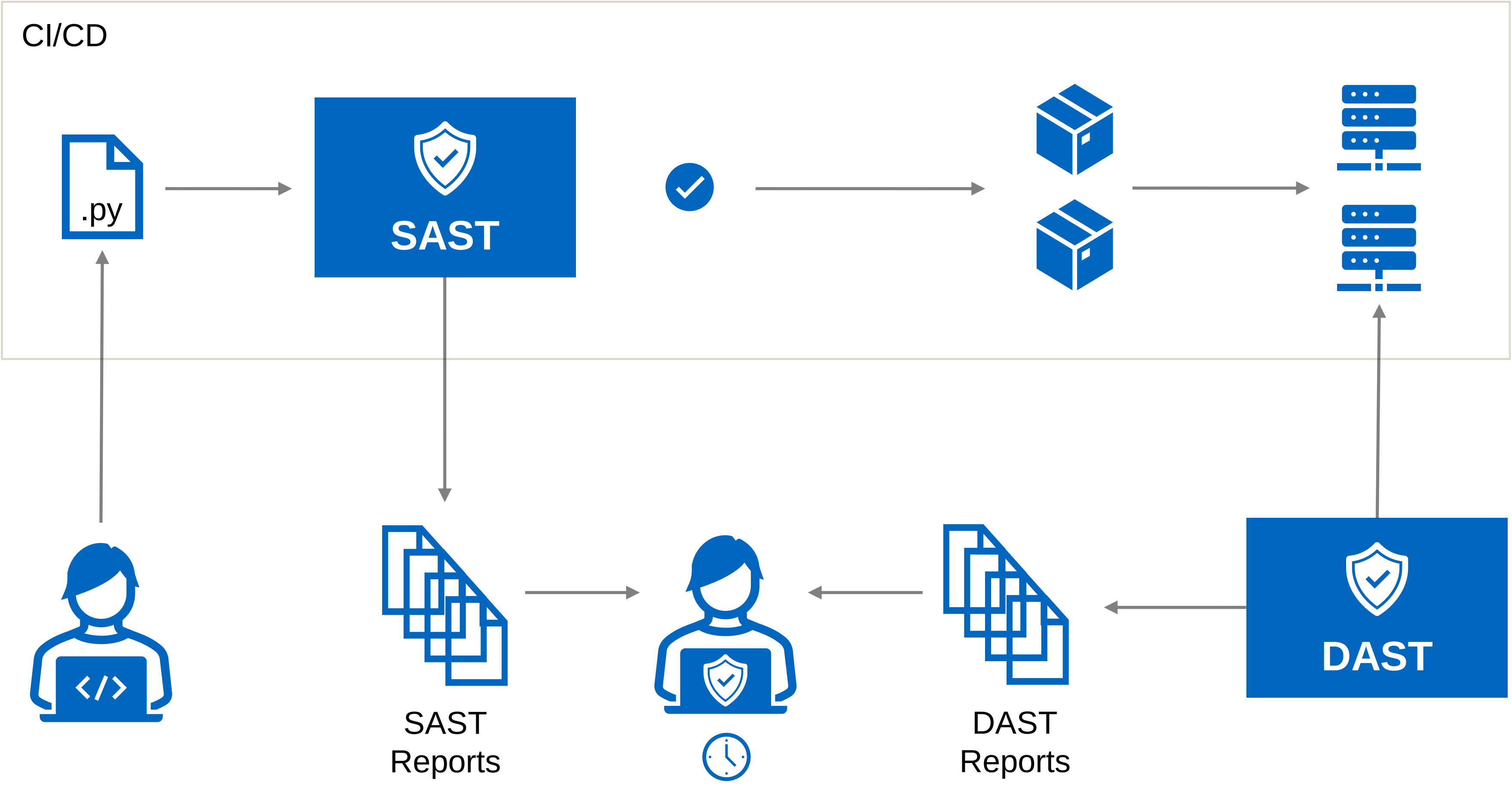
Motivation

Application security testing



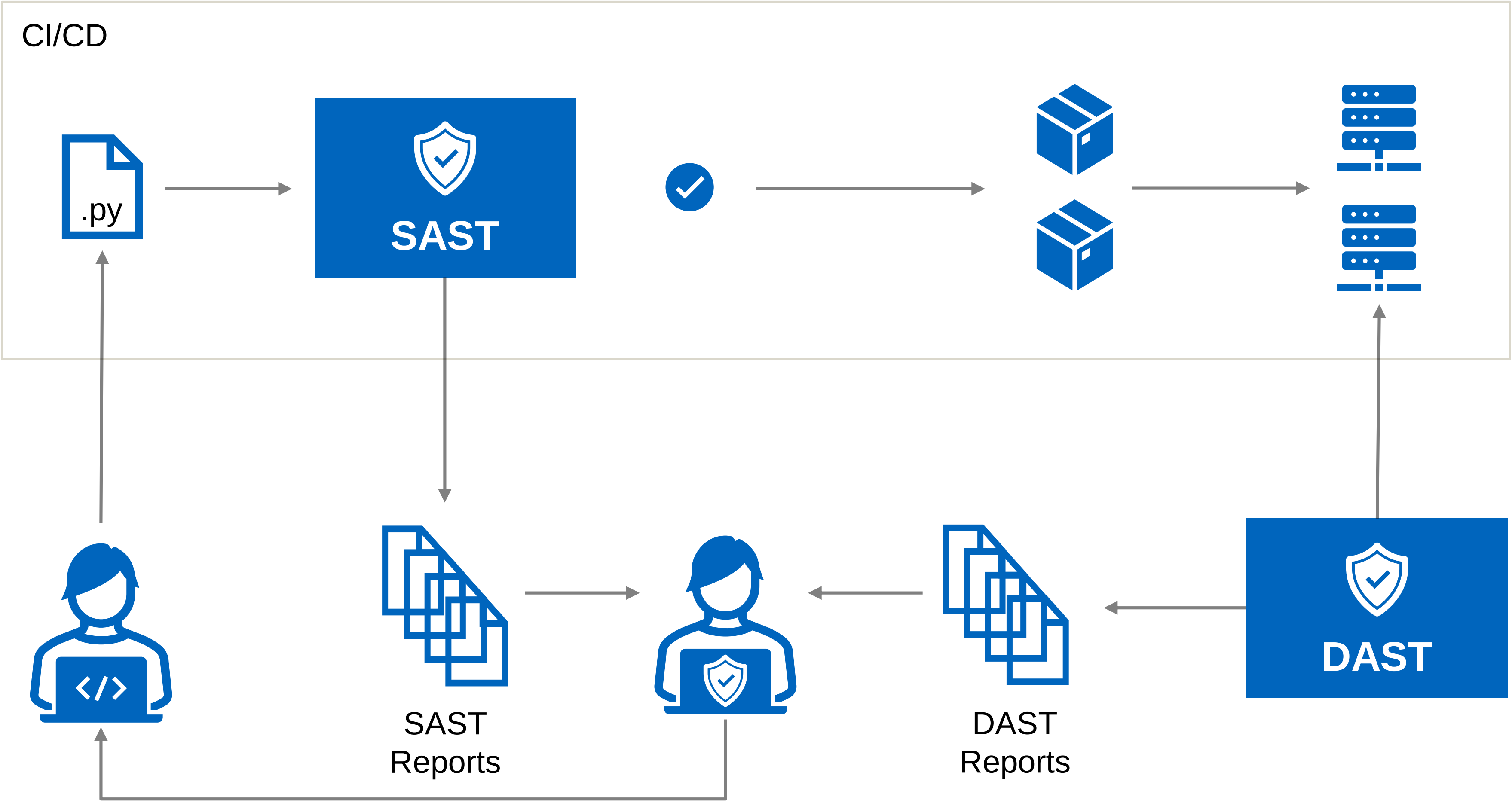
Motivation

Application security testing



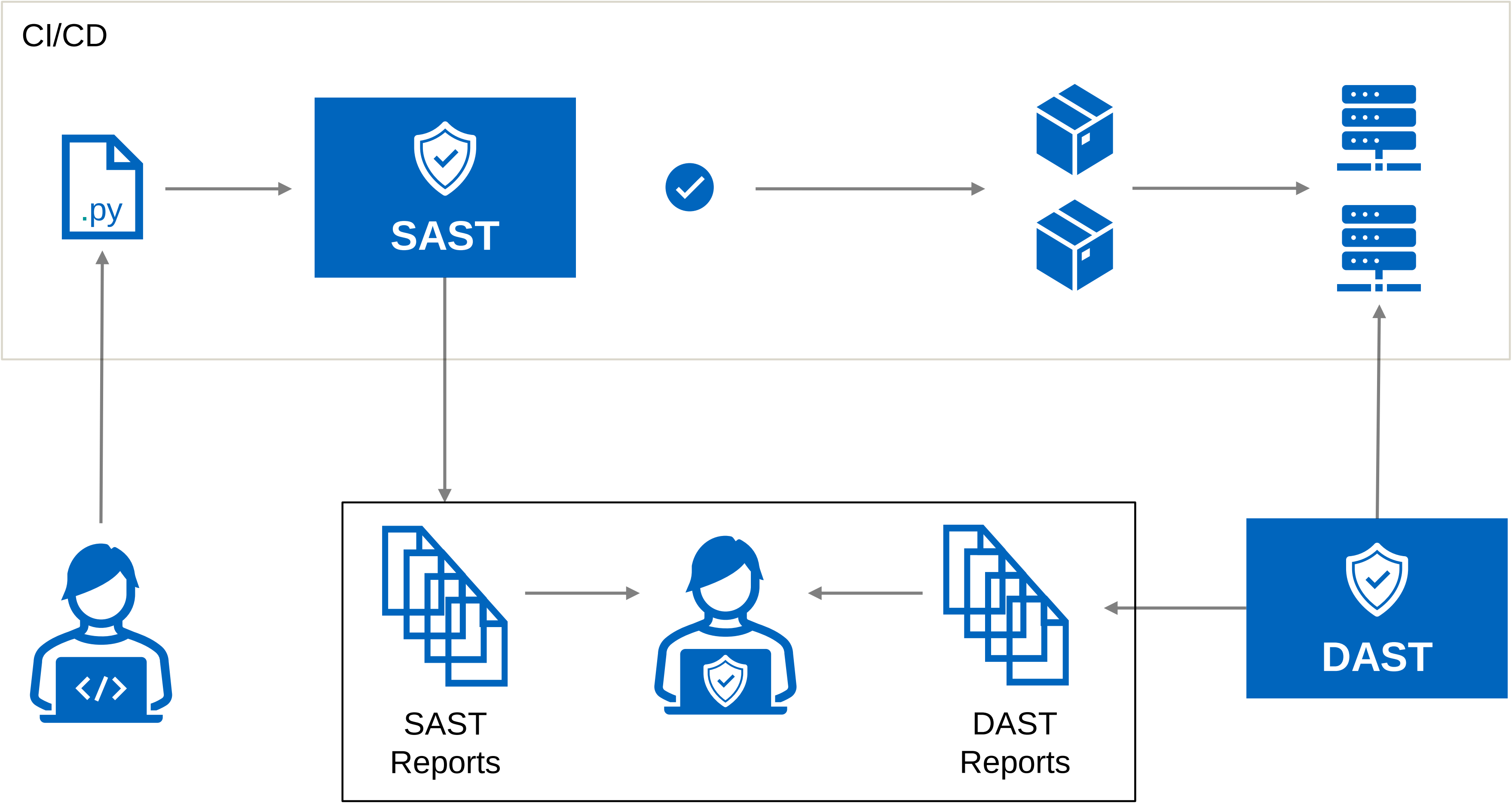
Motivation

Application security testing



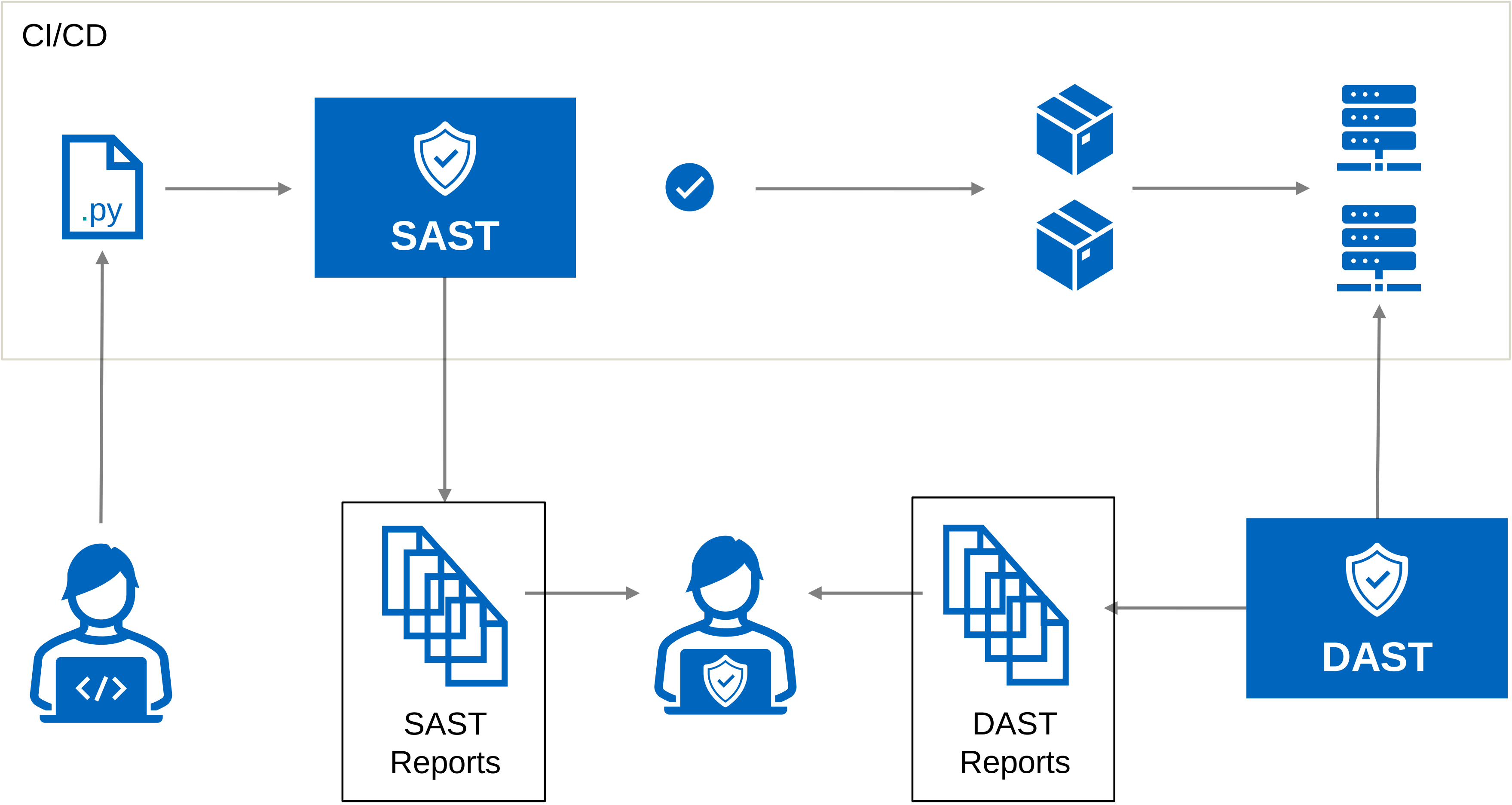
Motivation

Application security testing



Motivation

Application security testing



Motivation

Security findings examples


```
{
  "name": "SQL Injection",
  "description": "\nDue to the requirement for dynamic content of today's web
applications, many\nrely on a database backend to store data that will be
called upon and processed\nby the web application (or other programs).\nWeb
applications retrieve data from the database by using Structured Query
Language\n(SQL) queries.\n\nTo meet demands of many developers, database
servers (such as MSSQL, MySQL,\nOracle etc.) have additional built-in
functionality that can allow extensive\ncontrol of the database and
interaction with the host operating system itself.\n\nAn SQL injection occurs
when a value originating from the client's request is used\nwithin a SQL
query without prior sanitisation. This could allow cyber-criminals\nto
execute arbitrary SQL code and steal data or use the additional
functionality\nof the database server to take control of more server
components.\n\nThe successful exploitation of a SQL injection can be
devastating to an\norganisation and is one of the most commonly exploited web
application vulnerabilities.\n\nThis injection was detected as Arachni was
able to cause the server to respond to\nthe request with a database related
error.\n",
  "severity": "high",
  "remedy_guidance": "\nThe only proven method to prevent against SQL
injection attacks while still\nmaintaining full application functionality is
to use parameterized queries\n(also known as prepared statements).\nWhen
utilising this method of querying the database, any value supplied by
the\nclient will be handled as a string value rather than part of the SQL
query.\n\nAdditionally, when utilising parameterized queries, the database
engine will\nautomatically check to make sure the string being used matches
that of the column.\nFor example, the database engine will check that the
user supplied input is an\ninteger if the database column is configured to
contain integers.\n",
…
}
```


Arachni (DAST)


```
{
  "feed": "vulnerabilities",
  "feed_group": "github:npm",
  "fix": "5.0.1",
  "nvd_data": [
    {
      "cvss_v2": {…},
       "cvss_v3": {…},
       "id": "CVE-2021-3807"
    }
  ],
  "package": "ansi-regex-4.1.0",
  "package_cpe": "None",
  "package_cpe23": "None",
  "package_name": "ansi-regex",
  "package_path": "/juice-shop/node_modules/cliui/node_modules/ansi-
regex/package.json",
  "package_type": "npm",
  "package_version": "4.1.0",
  "severity": "Medium",
  "url": "https://github.com/advisories/GHSA-93q8-gq69-wqmw",
  "vendor_data": [],
  "vuln": "GHSA-93q8-gq69-wqmw",
  "will_not_fix": false,
  "scraped_description": "ansi-regex is vulnerable to Inefficient Regular
Expression Complexity"
}
```


Anchore (SAST)

Motivation

Problem description

- Multiple security tools produce thousands of findings, including *duplicates* which
  - represent the same underlying security flaw
  - are time wasting for cybersecurity professionals to read through
  - make the management of security findings challenging

- What if we could identify and semantically group these *duplicate* findings?
  - So they can be reported and resolved once
  - Via Natural Language Processing (NLP) to stay invariant to lexical differences in machine-generated textual information from different security tools

Outline

Motivation

Research Questions

1. What are duplicate security tool findings in DevSecOps and how can they be deduplicated?
2. How can we use Natural Language Processing (NLP) to deduplicate security tool findings?
3. How do base NLP Semantic Similarity methods perform when applied to security tool reports corpus?

Conclusion

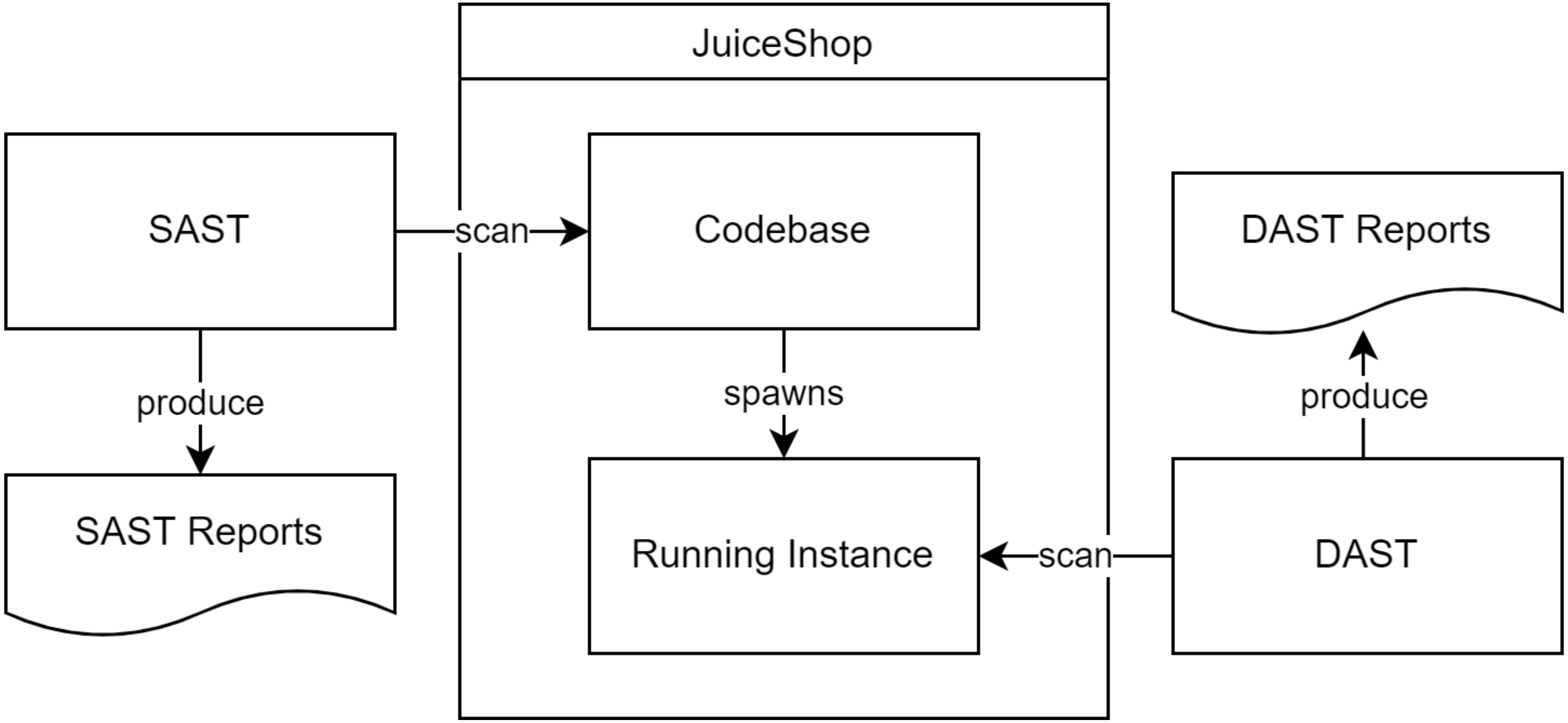
What are duplicate security tool findings in DevSecOps and how can they be deduplicated?

- It is necessary to establish when two findings count as a duplicates
- Duplicate security findings could appear due to:
  - Overlapping use-cases of security tools
    - To improve software coverage, cater for edge-cases, and avoid black spots
  - Presence of the same vulnerability in multiple locations in software
  - Multiple vulnerabilities having a similar solution
- To confirm this, we scan *JuiceShop* and analyze the findings from generated reports
  - Open-source web app containing security flaws found in real-world applications
  - We use 7 SAST tools and 2 DAST tools

OWASP JuiceShop - https://owasp.org/www-project-juice-shop/

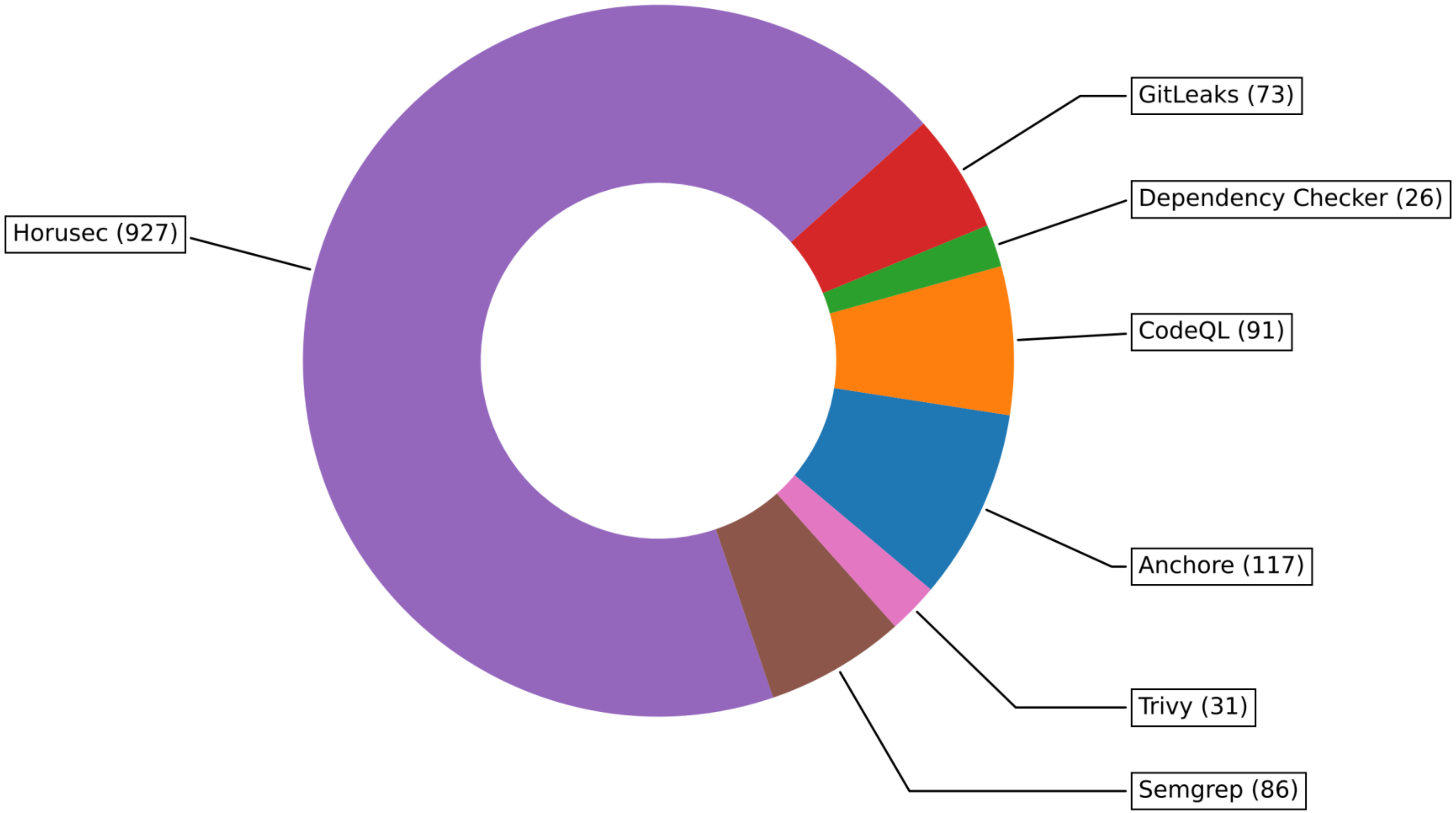
What are duplicate security tool findings in DevSecOps and how can they be deduplicated?

Scanning *JuiceShop*



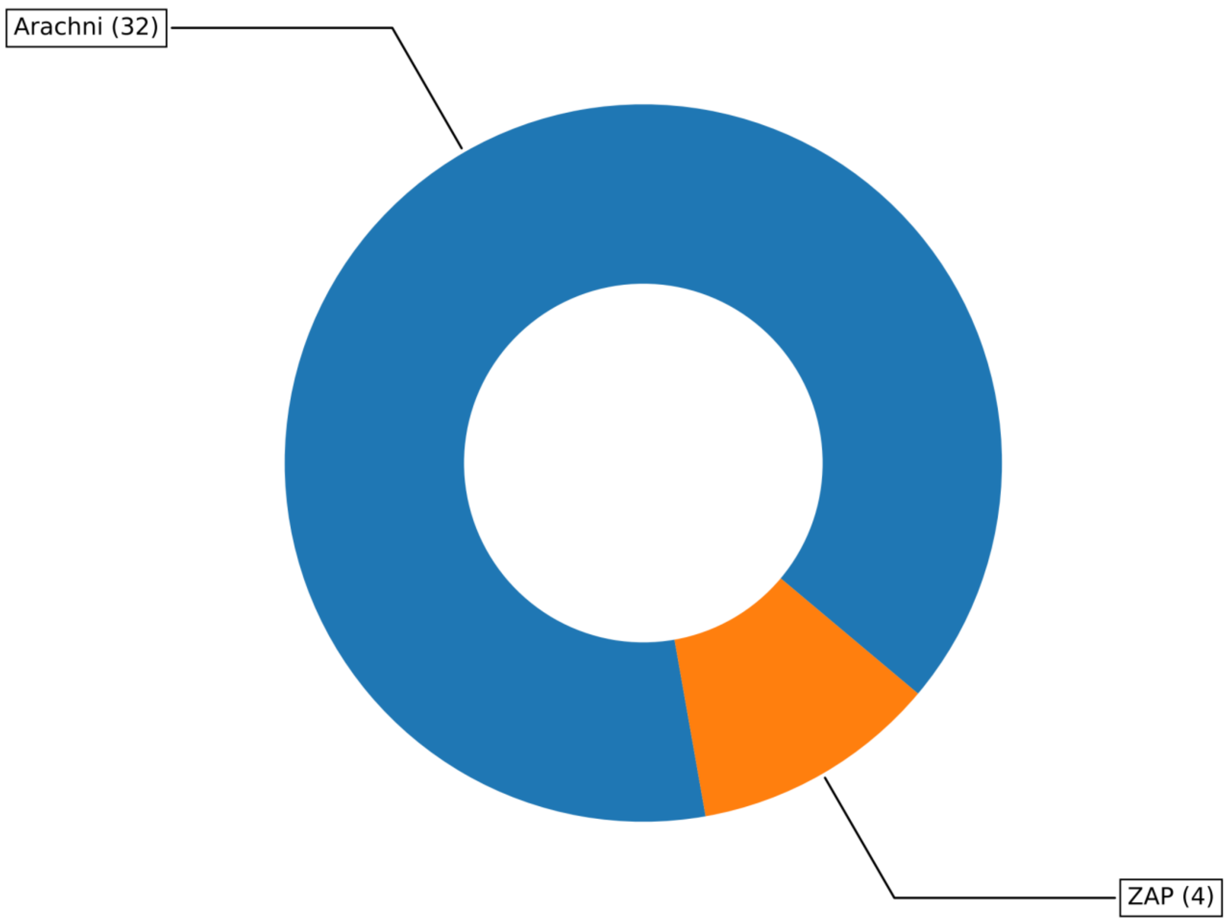
What are duplicate security tool findings in DevSecOps and how can they be deduplicated?

SAST tools findings distribution



What are duplicate security tool findings in DevSecOps and how can they be deduplicated?

DAST tools findings distribution



What are duplicate security tool findings in DevSecOps and how can they be deduplicated?

Similar SAST tool findings


```
{
  "VulnerabilityID": "CVE-2021-3807",
  "PkgName": "ansi-regex",
  "InstalledVersion": "4.1.0",
  "FixedVersion": "5.0.1, 6.0.1",
  "Layer": {…},
  "SeveritySource": "nvd",
  "PrimaryURL": "https://avd.aquasec.com/nvd/cve-2021-3807",
  "Title": "nodejs-ansi-regex: Regular expression denial of service (ReDoS)
matching ANSI escape codes",
  "Description": "ansi-regex is vulnerable to Inefficient Regular Expression
Complexity",
  "Severity": "HIGH",
  "CVSS": {
    "nvd": {…},
    "redhat": {…}
  },
  "References": [
    "https://app.snyk.io/vuln/SNYK-JS-ANSIREGEX-1583908",
    "https://github.com/advisories/GHSA-93q8-gq69-wqmw",
    "https://linux.oracle.com/cve/CVE-2021-3807.html",
    "https://linux.oracle.com/errata/ELSA-2022-0350.html",
    "https://nvd.nist.gov/vuln/detail/CVE-2021-3807"
    …
   ],
  "PublishedDate": "2021-09-17T07:15:00Z",
  "LastModifiedDate": "2021-10-19T13:11:00Z"
}
```


Trivy


```
{
  "feed": "vulnerabilities",
  "feed_group": "github:npm",
  "fix": "5.0.1",
  "nvd_data": [
    {
      "cvss_v2": {…},
       "cvss_v3": {…},
       "id": "CVE-2021-3807"
    }
  ],
  "package": "ansi-regex-4.1.0",
  "package_cpe": "None",
  "package_cpe23": "None",
  "package_name": "ansi-regex",
  "package_path": "/juice-shop/node_modules/cliui/node_modules/ansi-
regex/package.json",
  "package_type": "npm",
  "package_version": "4.1.0",
  "severity": "Medium",
  "url": "https://github.com/advisories/GHSA-93q8-gq69-wqmw",
  "vendor_data": [],
  "vuln": "GHSA-93q8-gq69-wqmw",
  "will_not_fix": false,
  "scraped_description": "ansi-regex is vulnerable to Inefficient Regular
Expression Complexity"
}
```


Anchore

What are duplicate security tool findings in DevSecOps and how can they be deduplicated?

Similar SAST tool findings


```
{
  "VulnerabilityID": "CVE-2021-3807",
  "PkgName": "ansi-regex",
  "InstalledVersion": "4.1.0",
  "FixedVersion": "5.0.1, 6.0.1",
  "Layer": {…},
  "SeveritySource": "nvd",
  "PrimaryURL": "https://avd.aquasec.com/nvd/cve-2021-3807",
  "Title": "nodejs-ansi-regex: Regular expression denial of service (ReDoS)
matching ANSI escape codes",
  "Description": "ansi-regex is vulnerable to Inefficient Regular Expression
Complexity",
  "Severity": "HIGH",
  "CVSS": {
    "nvd": {…},
    "redhat": {…}
  },
  "References": [
    "https://app.snyk.io/vuln/SNYK-JS-ANSIREGEX-1583908",
    "https://github.com/advisories/GHSA-93q8-gq69-wqmw",
    "https://linux.oracle.com/cve/CVE-2021-3807.html",
    "https://linux.oracle.com/errata/ELSA-2022-0350.html",
    "https://nvd.nist.gov/vuln/detail/CVE-2021-3807"
    …
   ],
  "PublishedDate": "2021-09-17T07:15:00Z",
  "LastModifiedDate": "2021-10-19T13:11:00Z"
}
```


location

Trivy


```
{
  "feed": "vulnerabilities",
  "feed_group": "github:npm",
  "fix": "5.0.1",
  "nvd_data": [
    {
      "cvss_v2": {…},
       "cvss_v3": {…},
       "id": "CVE-2021-3807"
    }
  ],
  "package": "ansi-regex-4.1.0",
  "package_cpe": "None",
  "package_cpe23": "None",
  "package_name": "ansi-regex",
  "package_path": "/juice-shop/node_modules/cliui/node_modules/ansi-
regex/package.json",
  "package_type": "npm",
  "package_version": "4.1.0",
  "severity": "Medium",
  "url": "https://github.com/advisories/GHSA-93q8-gq69-wqmw",
  "vendor_data": [],
  "vuln": "GHSA-93q8-gq69-wqmw",
  "will_not_fix": false,
  "scraped_description": "ansi-regex is vulnerable to Inefficient Regular
Expression Complexity"
}
```


location

Anchore

What are duplicate security tool findings in DevSecOps and how can they be deduplicated?

Similar SAST tool findings

```
{
  "VulnerabilityID": "CVE-2021-3807",   ← identifier
  "PkgName": "ansi-regex",
  "InstalledVersion": "4.1.0",
  "FixedVersion": "5.0.1, 6.0.1",
  "Layer": {…},
  "SeveritySource": "nvd",
  "PrimaryURL": "https://avd.aquasec.com/nvd/cve-2021-3807",
  "Title": "nodejs-ansi-regex: Regular expression denial of service (ReDoS)
matching ANSI escape codes",
  "Description": "ansi-regex is vulnerable to Inefficient Regular Expression
Complexity",
  "Severity": "HIGH",
  "CVSS": {
    "nvd": {…},
    "redhat": {…}
  },
  "References": [
    "https://app.snyk.io/vuln/SNYK-JS-ANSIREGEX-1583908",
    "https://github.com/advisories/GHSA-93q8-gq69-wqmw",
    "https://linux.oracle.com/cve/CVE-2021-3807.html",
    "https://linux.oracle.com/errata/ELSA-2022-0350.html",
    "https://nvd.nist.gov/vuln/detail/CVE-2021-3807"
    …
   ],
  "PublishedDate": "2021-09-17T07:15:00Z",
  "LastModifiedDate": "2021-10-19T13:11:00Z"
}
```

Trivy

```
{
  "feed": "vulnerabilities",
  "feed_group": "github:npm",
  "fix": "5.0.1",
  "nvd_data": [
    {
      "cvss_v2": {…},
       "cvss_v3": {…},
       "id": "CVE-2021-3807"
    }
  ],
  "package": "ansi-regex-4.1.0",
  "package_cpe": "None",
  "package_cpe23": "None",
  "package_name": "ansi-regex",
  "package_path": "/juice-shop/node_modules/cliui/node_modules/ansi-
regex/package.json",
  "package_type": "npm",
  "package_version": "4.1.0",
  "severity": "Medium",
  "url": "https://github.com/advisories/GHSA-93q8-gq69-wqmw",
  "vendor_data": [],
  "vuln": "GHSA-93q8-gq69-wqmw",   ← identifier
  "will_not_fix": false,
  "scraped_description": "ansi-regex is vulnerable to Inefficient Regular
Expression Complexity"
}
```

Anchore

What are duplicate security tool findings in DevSecOps and how can they be deduplicated?

Similar SAST tool findings


```
{
  "VulnerabilityID": "CVE-2021-3807",
  "PkgName": "ansi-regex",
  "InstalledVersion": "4.1.0",
  "FixedVersion": "5.0.1, 6.0.1",
  "Layer": {…},
  "SeveritySource": "nvd",
  "PrimaryURL": "https://avd.aquasec.com/nvd/cve-2021-3807",
  "Title": "nodejs-ansi-regex: Regular expression denial of service (ReDoS)
matching ANSI escape codes",
  "Description": "ansi-regex is vulnerable to Inefficient Regular Expression
Complexity",
  "Severity": "HIGH",
  "CVSS": {
    "nvd": {…},
    "redhat": {…}
  },
  "References": [
    "https://app.snyk.io/vuln/SNYK-JS-ANSIREGEX-1583908",
    "https://github.com/advisories/GHSA-93q8-gq69-wqmw",
    "https://linux.oracle.com/cve/CVE-2021-3807.html",
    "https://linux.oracle.com/errata/ELSA-2022-0350.html",
    "https://nvd.nist.gov/vuln/detail/CVE-2021-3807"
    …
   ],
  "PublishedDate": "2021-09-17T07:15:00Z",
  "LastModifiedDate": "2021-10-19T13:11:00Z"
}
```


problem

Trivy


```
{
  "feed": "vulnerabilities",
  "feed_group": "github:npm",
  "fix": "5.0.1",
  "nvd_data": [
    {
      "cvss_v2": {…},
       "cvss_v3": {…},
       "id": "CVE-2021-3807"
    }
  ],
  "package": "ansi-regex-4.1.0",
  "package_cpe": "None",
  "package_cpe23": "None",
  "package_name": "ansi-regex",
  "package_path": "/juice-shop/node_modules/cliui/node_modules/ansi-
regex/package.json",
  "package_type": "npm",
  "package_version": "4.1.0",
  "severity": "Medium",
  "url": "https://github.com/advisories/GHSA-93q8-gq69-wqmw",
  "vendor_data": [],
  "vuln": "GHSA-93q8-gq69-wqmw",
  "will_not_fix": false,
  "scraped_description": "ansi-regex is vulnerable to Inefficient Regular
Expression Complexity"
}
```


problem

Anchore

What are duplicate security tool findings in DevSecOps and how can they be deduplicated?

Deduplication strategies

Type	Description	Mitigation Strategy	Advantage	Disadvantage
Findings based	Same finding at the exact same place	Approach finding one by one	Unique findings	Same location often impossible to determine
Problem based	Same finding, regardless of the location	If you know how to solve a finding, do it all occurrences	Summarizes problems	Solution might be different, even though finding is equal
Solution based	Various findings, solved by the same action	Solve multiple findings with one change	Identify valuable actions	Highly error prone identification
Location based	Various findings, located at the same place	If you work on one location, solve everything there	Hotspot identification	Different phrases to refer to the same location possible

What are duplicate security tool findings in DevSecOps and how can they be deduplicated?

Deduplication strategies

Type	Description	Mitigation Strategy	Advantage	Disadvantage
Findings based	Same finding at the exact same place	Approach finding one by one	Unique findings	Same location often impossible to determine
Problem based	Same finding, regardless of the location	If you know how to solve a finding, do it all occurrences	Summarizes problems	Solution might be different, even though finding is equal
Solution based	Various findings, solved by the same action	Solve multiple findings with one change	Identify valuable actions	Highly error prone identification
Location based	Various findings, located at the same place	If you work on one location, solve everything there	Hotspot identification	Different phrases to refer to the same location possible

What are duplicate security tool findings in DevSecOps and how can they be deduplicated?

Problem-based deduplication

- **DevSecOps:**
  - Show overview of problems to cybersecurity experts
  - Most promising and widely used
  - Easier for developers to identify existing problems

- **Natural Language Processing:**
  - Use features with most semantic data
  - Explore full potential of NLP application to security findings texts

Outline

Motivation

Research Questions

1. What are duplicate security tool findings in DevSecOps and how can they be deduplicated?
2. How can we use Natural Language Processing (NLP) to deduplicate security tool findings?
3. How do base NLP Semantic Similarity methods perform when applied to security tool reports corpus?

Conclusion

How can we use Natural Language Processing (NLP) to deduplicate security tool findings?

NLP concepts

- **Semantic (Textual) Similarity:** A task of NLP that measures the similarity between two texts based on their semantic meaning. Similarity is measured as a percentage.

- **Word embeddings:** Models that convert texts to numerical vectors.

- **Cosine similarity:** A distance measure for numerical vectors generated from word embeddings. Used to calculate semantic similarity.

How can we use Natural Language Processing (NLP) to deduplicate security tool findings?

Semantic similarity methods

- **Knowledge-based methods:**
  - Use an external ontology or knowledge source to score semantic similarity between two words.
  - Method used: compare words in sentences using WordNet

- **Corpus-based methods:**
  - Create a vector space of large body of text using distributional hypothesis. Form numerical vectors of texts using the vector space, and use a distance measure (e.g., *cosine similarity*) between them.
  - Method used: Latent Semantic Indexing (LSI)

- **Transformer-based methods:**
  - Neural networks that capture semantic context of sentences and form a latent space and can convert texts to numerical vectors. Trained on a large corpora of text and can be fine-tuned for a specific task.
  - Method used: Sentence BERT (SBERT)

D. Chandrasekaran and V. Mago. "Evolution of semantic similarity—a survey". In: ACM Computing Surveys (CSUR) 54.2 (2021), pp. 1–37.

How can we use Natural Language Processing (NLP) to deduplicate security tool findings?

Extracting finding features to form sentences

description feature

solution feature


```
{
  "name": "SQL Injection",
  "description": "\nDue to the requirement for dynamic content of today's web
applications, many\nrely on a database backend to store data that will be
called upon and processed\nby the web application (or other programs).\nWeb
applications retrieve data from the database by using Structured Query
Language\n(SQL) queries.\n\nTo meet demands of many developers, database
servers (such as MSSQL, MySQL,\nOracle etc.) have additional built-in
functionality that can allow extensive\ncontrol of the database and
interaction with the host operating system itself.\n\nAn SQL injection occurs
when a value originating from the client's request is used\nwithin a SQL
query without prior sanitisation. This could allow cyber-criminals\nto
execute arbitrary SQL code and steal data or use the additional
functionality\nof the database server to take control of more server
components.\n\nThe successful exploitation of a SQL injection can be
devastating to an\norganisation and is one of the most commonly exploited web
application vulnerabilities.\n\nThis injection was detected as Arachni was
able to cause the server to respond to\nthe request with a database related
error.\n",
  "severity": "high",
  "remedy_guidance": "\nThe only proven method to prevent against SQL
injection attacks while still\nmaintaining full application functionality is
to use parameterized queries\n(also known as prepared statements).\nWhen
utilising this method of querying the database, any value supplied by
the\nclient will be handled as a string value rather than part of the SQL
query.\n\nAdditionally, when utilising parameterized queries, the database
engine will\nautomatically check to make sure the string being used matches
that of the column.\nFor example, the database engine will check that the
user supplied input is an\ninteger if the database column is configured to
contain integers.\n",
…
}
```


Arachni (DAST)


```
{
  "feed": "vulnerabilities",
  "feed_group": "github:npm",
  "fix": "5.0.1",
  "nvd_data": [
    {
      "cvss_v2": {…},
      "cvss_v3": {…},
      "id": "CVE-2021-3807"
    }
  ],
  "package": "ansi-regex-4.1.0",
  "package_cpe": "None",
  "package_cpe23": "None",
  "package_name": "ansi-regex",
  "package_path": "/juice-shop/node_modules/cliui/node_modules/ansi-
regex/package.json",
  "package_type": "npm",
  "package_version": "4.1.0",
  "severity": "Medium",
  "url": "https://github.com/advisories/GHSA-93q8-gq69-wqmw",
  "vendor_data": [],
  "vuln": "GHSA-93q8-gq69-wqmw",
  "will_not_fix": false,
  "scraped_description": "ansi-regex is vulnerable to Inefficient Regular
Expression Complexity"
}
```


description feature

Anchore (SAST)

How can we use Natural Language Processing (NLP) to deduplicate security tool findings?

Extracting finding features to form sentences

Feature	Arachni	ZAP
Name	*name*	*name*
Description	*description*	*desc*
Solution	*remedy_guidance*	*solution*

Feature names for DAST tools

Tool	Property	Comment
Anchore	*url*	Scrape data from URL
CodeQL	*message*	Use sub-property *text*
Dependency Checker	*description*	-
GitLeaks	*Description, Message*	Concatenate text from both features
Horusec	*vulnerabilities*	Use sub-property *details*
Semgrep	*message*	-
Trivy	*Description*	-

Description feature names for SAST tools

How can we use Natural Language Processing (NLP) to deduplicate security tool findings?

DAST and SAST deduplication

We deduplicate DAST tools and SAST tools findings *separately* due to difference of

- perspective on product (looking at static code or the running application)
- available features
- verbosity of corresponding features

What are duplicate security tool findings in DevSecOps and how can they be deduplicated?

Similar SAST tool findings

```
{
  "VulnerabilityID": "CVE-2021-3807",
  "PkgName": "ansi-regex",
  "InstalledVersion": "4.1.0",
  "FixedVersion": "5.0.1, 6.0.1",
  "Layer": {…},
  "SeveritySource": "nvd",
  "PrimaryURL": "https://avd.aquasec.com/nvd/cve-2021-3807",
  "Title": "nodejs-ansi-regex: Regular expression denial of service (ReDoS)
matching ANSI escape codes",
  "Description": "ansi-regex is vulnerable to Inefficient Regular Expression
Complexity",
  "Severity": "HIGH",
  "CVSS": {
    "nvd": {…},
    "redhat": {…}
  },
  "References": [
    "https://app.snyk.io/vuln/SNYK-JS-ANSIREGEX-1583908",
    "https://github.com/advisories/GHSA-93q8-gq69-wqmw",
    "https://linux.oracle.com/cve/CVE-2021-3807.html",
    "https://linux.oracle.com/errata/ELSA-2022-0350.html",
    "https://nvd.nist.gov/vuln/detail/CVE-2021-3807"
    …
   ],
  "PublishedDate": "2021-09-17T07:15:00Z",
  "LastModifiedDate": "2021-10-19T13:11:00Z"
}
```

CVE-ID

Trivy

```
{
  "feed": "vulnerabilities",
  "feed_group": "github:npm",
  "fix": "5.0.1",
  "nvd_data": [
    {
      "cvss_v2": {…},
       "cvss_v3": {…},
       "id": "CVE-2021-3807"
    }
  ],
  "package": "ansi-regex-4.1.0",
  "package_cpe": "None",
  "package_cpe23": "None",
  "package_name": "ansi-regex",
  "package_path": "/juice-shop/node_modules/cliui/node_modules/ansi-
regex/package.json",
  "package_type": "npm",
  "package_version": "4.1.0",
  "severity": "Medium",
  "url": "https://github.com/advisories/GHSA-93q8-gq69-wqmw",
  "vendor_data": [],
  "vuln": "GHSA-93q8-gq69-wqmw",
  "will_not_fix": false,
  "scraped_description": "ansi-regex is vulnerable to Inefficient Regular
Expression Complexity"
}
```

CVE-ID

Anchore

How can we use Natural Language Processing (NLP) to deduplicate security tool findings?

Aggregating semantic content for SAST

Finding
CVE-ID: ABC-DEF-GHI Description: This is description for finding 1

Finding
CVE-ID: ABC-DEF-GHI Description: This is description for finding 2

This is description for finding 1. This is description for finding 2.

How can we use Natural Language Processing (NLP) to deduplicate security tool findings?

Aggregating semantic content for SAST

Finding
CVE-ID: ABC-DEF-GHI Description: This is description for finding 1. This is description for finding 2.

Finding
CVE-ID: ABC-DEF-GHI Description: This is description for finding 1. This is description for finding 2.

How can we use Natural Language Processing (NLP) to deduplicate security tool findings?

Applying semantic similarity

(Ref)

"ansi-regex is vulnerable to Inefficient Regular Expression Complexity."

"No Log Sensitive Information in console. The App logs information. Sensitive information should never be logged."

Semantic Similarity method → 0.013

Greater than similarity threshold e.g., 0.8?

Add to collection of similar findings

Skip

How can we use Natural Language Processing (NLP) to deduplicate security tool findings?

Dataset formation

- **Identified duplicate findings need comparing with ground-truth to evaluate performance**
  - Compare predicted collections to ground-truth collections
  - Need a dataset

- Inefficient to create a dataset by hand
  - *N* * *M* feature names for *N* security tools when inspecting *M* features to detect duplicates
  - Thousands of findings in SAST, verbose descriptions in DAST

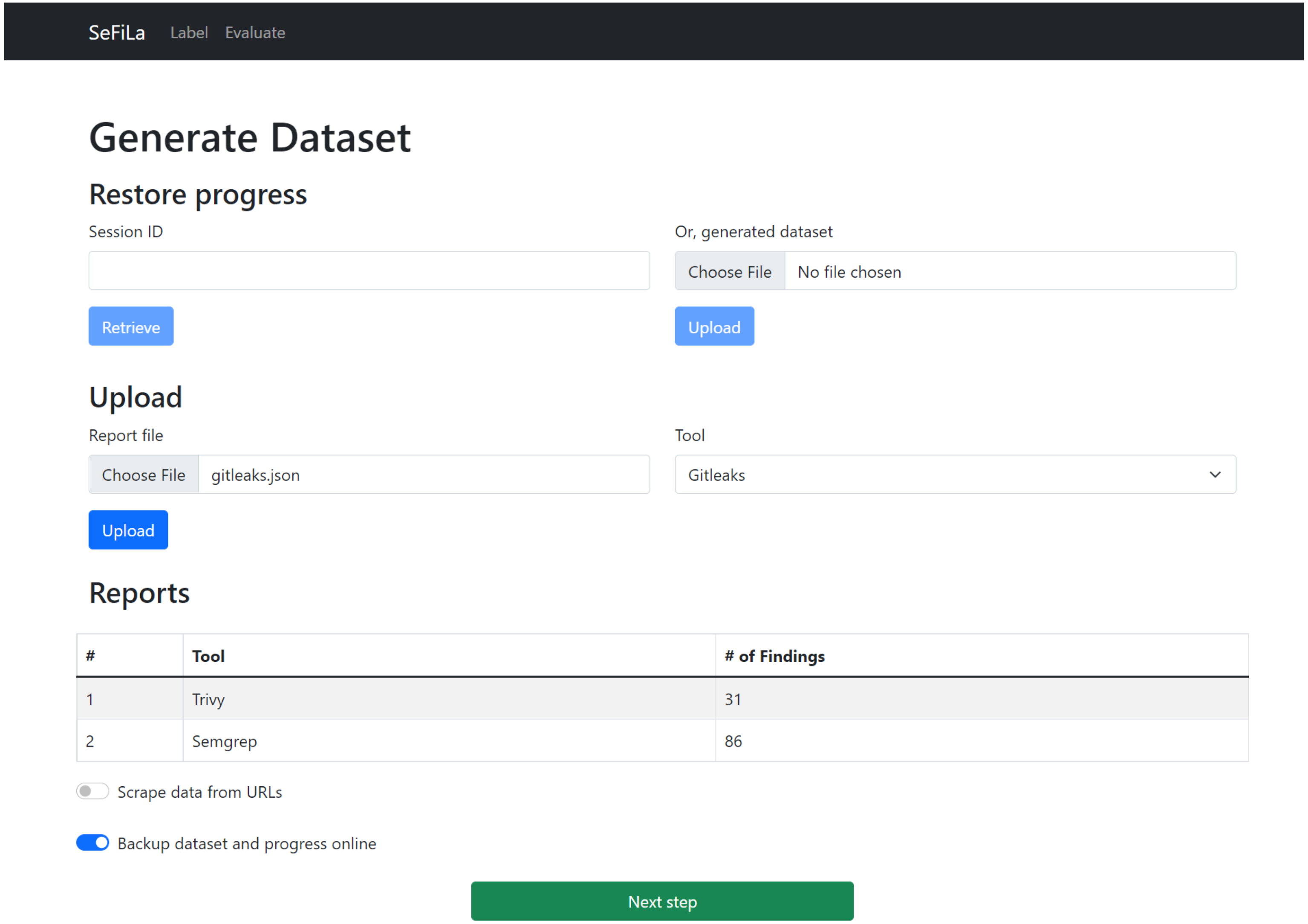
- **Develop a tool for the task**
  - Adapt it to experts' feedback
  - Necessity for repeating our work

How can we use Natural Language Processing (NLP) to deduplicate security tool findings?

Security Findings Labeler (SeFiLa)

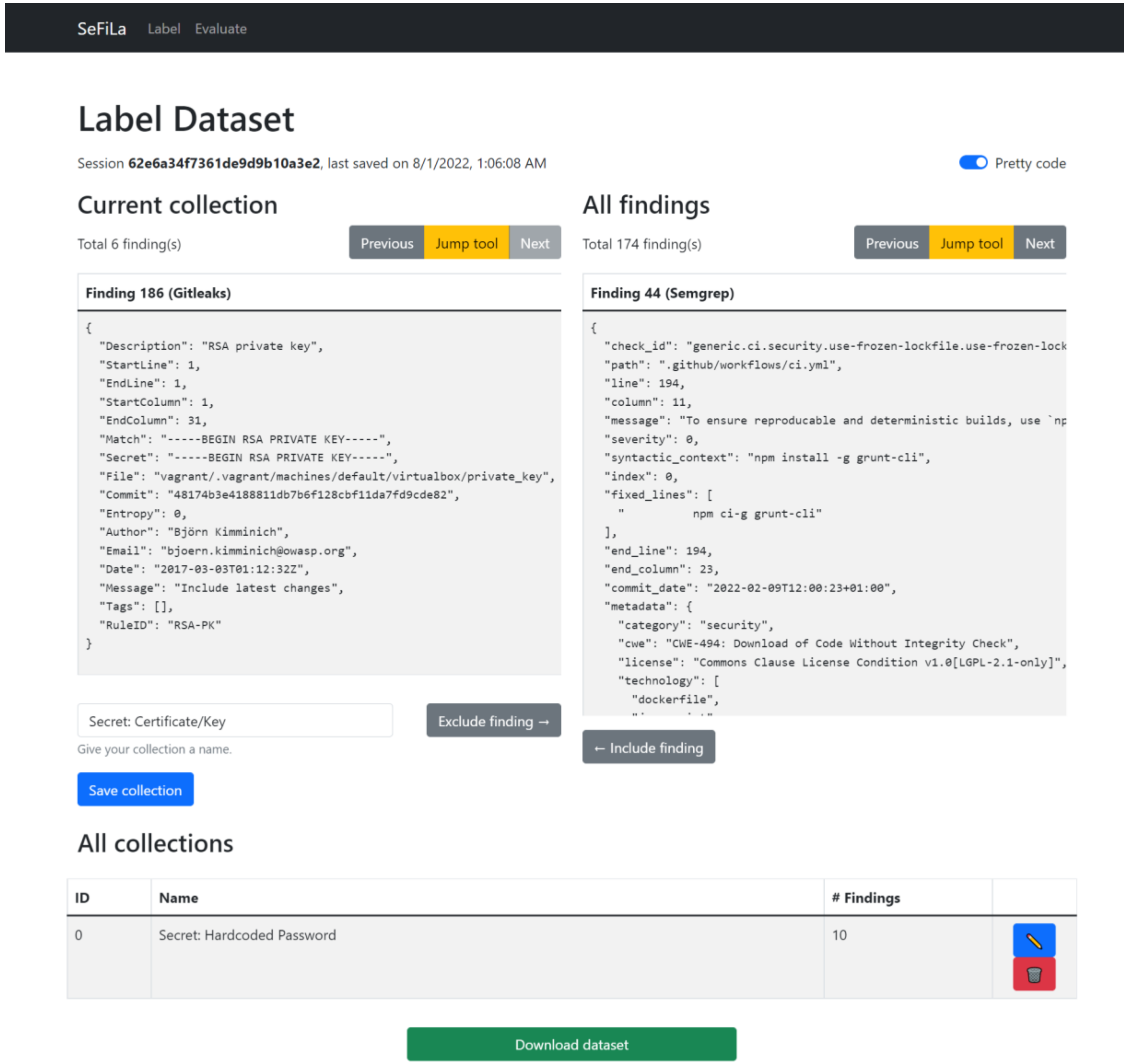
How can we use Natural Language Processing (NLP) to deduplicate security tool findings?

Security Findings Labeler (SeFiLa) – Label feature



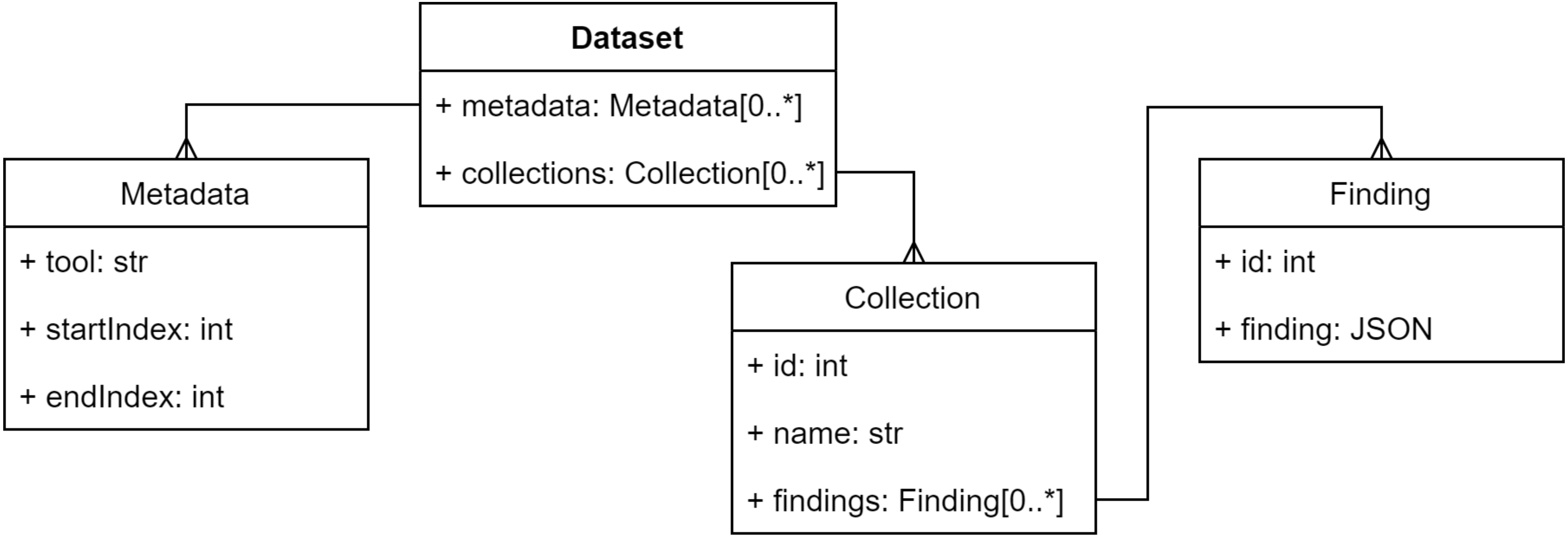
How can we use Natural Language Processing (NLP) to deduplicate security tool findings?

Security Findings Labeler (SeFiLa) – Label feature



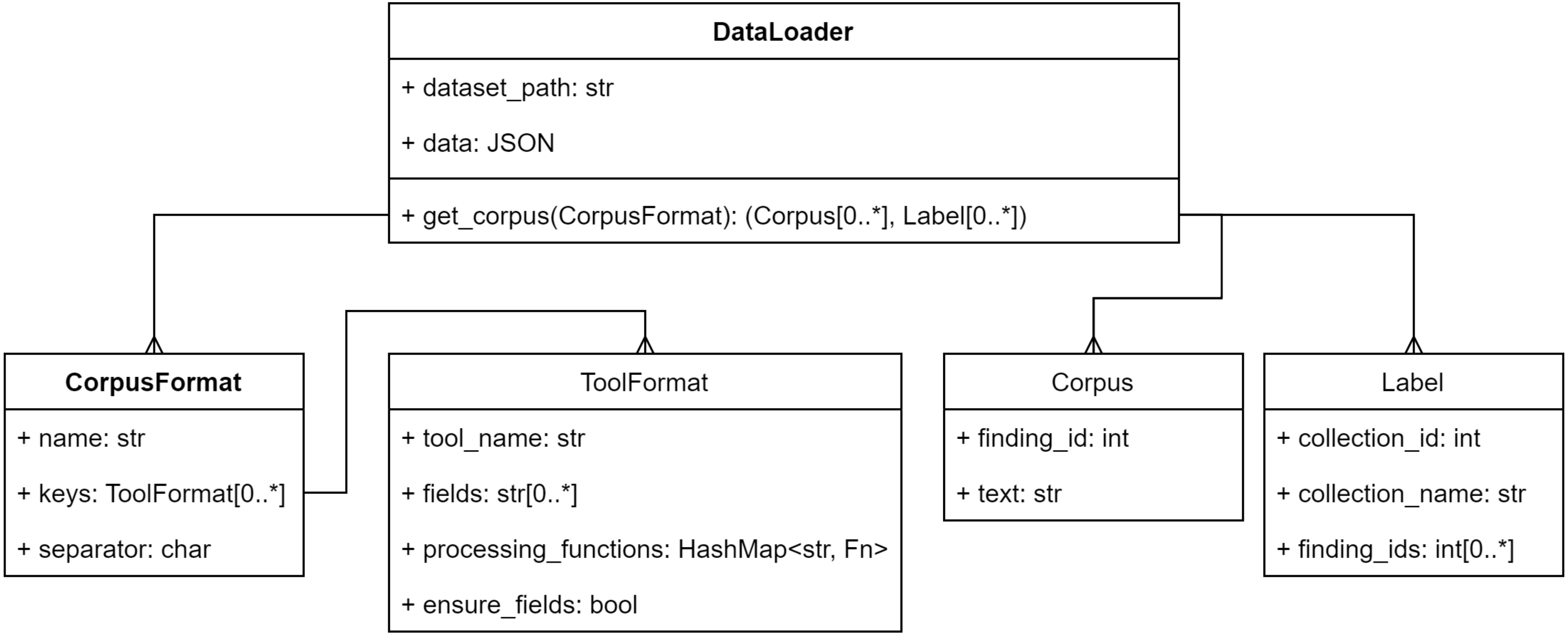
How can we use Natural Language Processing (NLP) to deduplicate security tool findings?

Generated *Dataset*



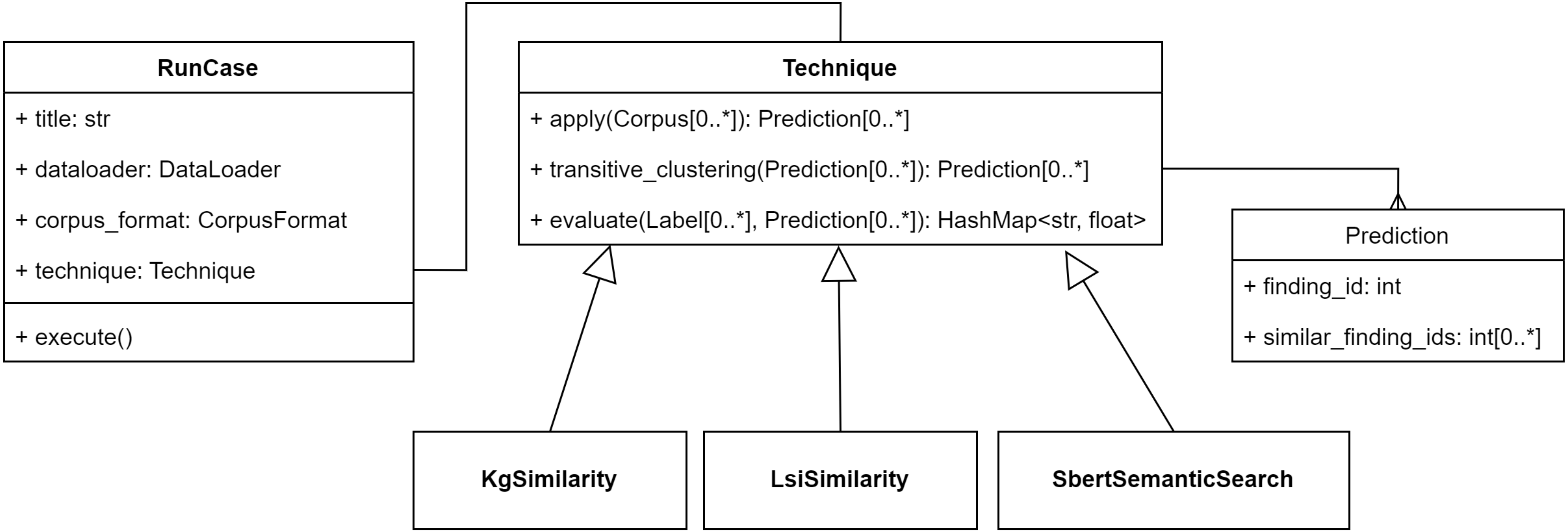
How can we use Natural Language Processing (NLP) to deduplicate security tool findings?

DataLoader to extract corpus from *Dataset*



How can we use Natural Language Processing (NLP) to deduplicate security tool findings?

Experimental design



Outline

Motivation

Research Questions

1. What are duplicate security tool findings in DevSecOps and how can they be deduplicated?
2. How can we use Natural Language Processing (NLP) to deduplicate security tool findings?
3. How do base NLP Semantic Similarity methods perform when applied to security tool reports corpus?

Conclusion

How do base NLP Semantic Similarity methods perform when applied to security tool reports corpus?

Overview of datasets

Aspect	**SAST**	**DAST**
Total number of findings	1351	36
Number of security tools	7	2
Collection type	Problem-based	Problem-based
Total number of collections	183	10
Maximum number of findings in a collection	408	25
Minimum number of findings in a collection	1	1
Average number of findings in a collection	7	3
Features concatenated to construct corpus text string	*description*	*description, solution, name*
Average length of corpus text string	302	471

How do base NLP Semantic Similarity methods perform when applied to security tool reports corpus?

Text corpora

- **DAST**
  - DAST-NDS
    - Features: *name description solution*
  - DAST-Description
    - Features: *description*

- **SAST**
  - SAST-Plain
    - Features: *description*
  - SAST-Aggregated
  - Features: *description* (aggregated via CVE-ID)

Quantitative metrics

- **Pair-wise counting**
  - Compares pairs of predictions and labels
  - **Metrics:**
    - Precision: ratio of correct predictions
    - Recall: ratio of predicted ground-truth pairs
    - F-Measure: harmonic mean of precision and recall
- **Example**

  predictions = {(1, 2, 3, 4, 5, 6)}

  ground-truth = {(1, 2, 3, 4), (5, 6)}

  Precision: 0.47, Recall: 1.0, F-Measure: 0.64

- **Direct cluster comparison**
  - Directly compares predictions with labels
  - **Metrics:**
    - Prediction accuracy: ratio of correct predictions
    - Label accuracy: ratio of predicted ground-truth pairs
    - Average accuracy: arithmetic mean of both
- **Example**

  predictions = {(1, 2, 3, 4, 5, 6)}

  ground-truth = {(1, 2, 3, 4), (5, 6)}

  Pred. Acc.: 0.0, Label Acc.: 0.0, Avg. Acc.: 0.0

How do base NLP Semantic Similarity methods perform when applied to security tool reports corpus?

Quantitative results summary

Technique	Corpus	Maximum Accuracy			Maximum Metric Value		
		Average	Prediction	Label	F-Measure	Precision	Recall
SBERT	*SAST-Plain*	0.723	0.621	0.825	0.901	0.820	1.000
	SAST-Aggregated	0.812	0.701	0.923	0.901	0.820	1.000
	DAST-NDS	0.859	0.818	0.900	0.998	0.997	1.000
	DAST-Description	0.859	0.818	0.900	0.998	0.997	1.000
LSI	*SAST-Plain*	0.750	0.658	0.842	0.901	0.820	0.999
	SAST-Aggregated	0.826	0.734	0.918	0.901	0.820	1.000
	DAST-NDS	0.859	0.818	0.900	0.998	0.998	0.998
	DAST-Description	0.859	0.818	0.900	0.997	0.997	0.997
KG	*SAST-Plain*	0.683	0.556	0.809	0.901	0.820	1.000
	SAST-Aggregated	0.794	0.676	0.913	0.901	0.820	1.000
	DAST-NDS	0.733	0.667	0.800	0.997	0.993	1.000
	DAST-Description	0.733	0.667	0.800	0.997	0.993	1.000

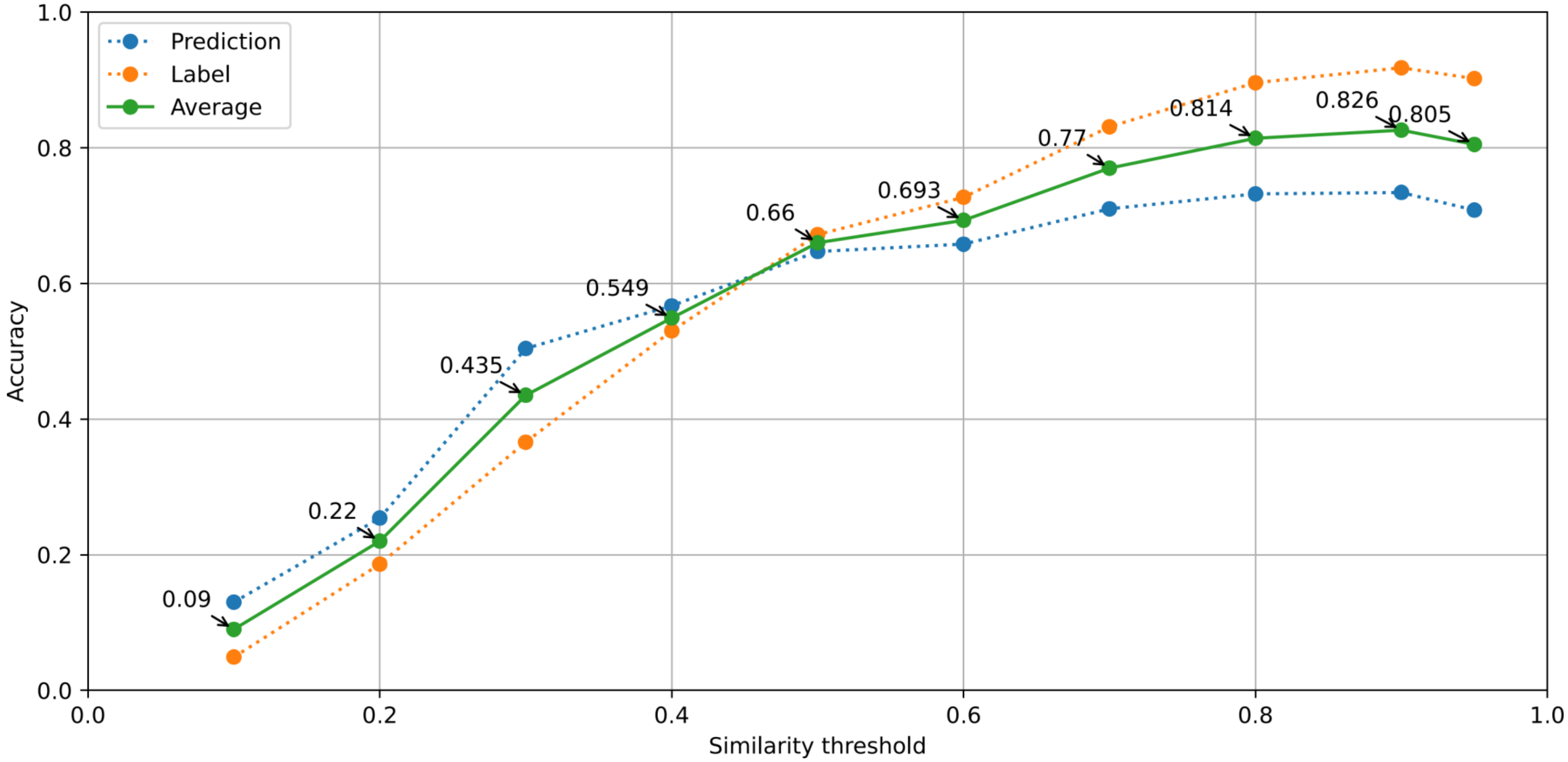
How do base NLP Semantic Similarity methods perform when applied to security tool reports corpus?

Quantitative results summary

Technique	Corpus	Maximum Accuracy			Maximum Metric Value		
		Average	Prediction	Label	F-Measure	Precision	Recall
SBERT	*SAST-Plain*	0.723	0.621	0.825	0.901	0.820	1.000
	SAST-Aggregated	0.812	0.701	0.923	0.901	0.820	1.000
	DAST-NDS	0.859	0.818	0.900	0.998	0.997	1.000
	DAST-Description	0.859	0.818	0.900	0.998	0.997	1.000
LSI	*SAST-Plain*	0.750	0.658	0.842	0.901	0.820	0.999
	SAST-Aggregated	0.826	0.734	0.918	0.901	0.820	1.000
	DAST-NDS	0.859	0.818	0.900	0.998	0.998	0.998
	DAST-Description	0.859	0.818	0.900	0.997	0.997	0.997
KG	*SAST-Plain*	0.683	0.556	0.809	0.901	0.820	1.000
	SAST-Aggregated	0.794	0.676	0.913	0.901	0.820	1.000
	DAST-NDS	0.733	0.667	0.800	0.997	0.993	1.000
	DAST-Description	0.733	0.667	0.800	0.997	0.993	1.000

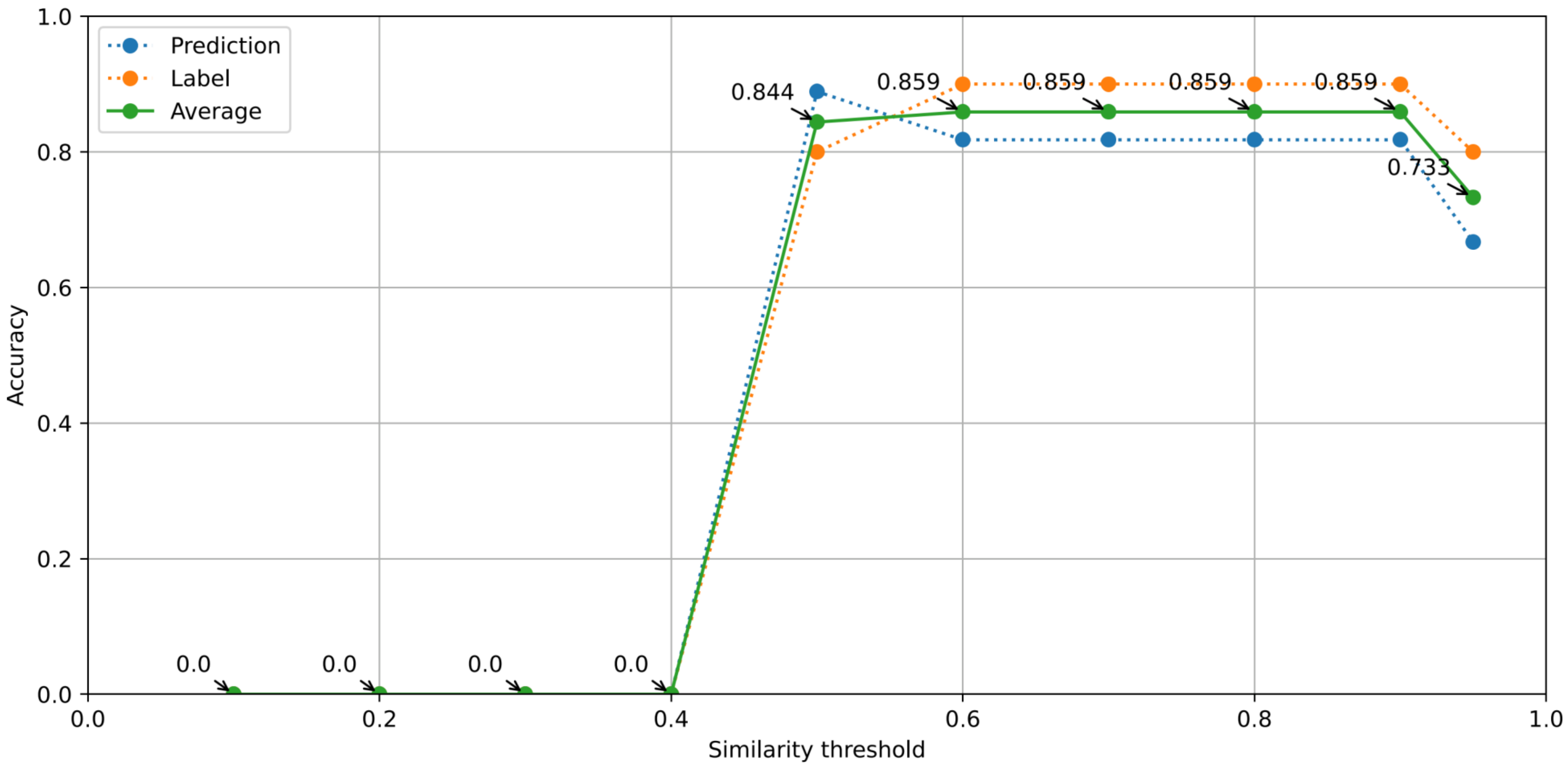
How do base NLP Semantic Similarity methods perform when applied to security tool reports corpus?

Quantitative results – LSI *SAST-Aggregated*



How do base NLP Semantic Similarity methods perform when applied to security tool reports corpus?

Quantitative results – SBERT *DAST-NDS*



How do base NLP Semantic Similarity methods perform when applied to security tool reports corpus?

Qualitative evaluation

Cybersecurity professionals

- review the incorrect predictions and related labels
- read corresponding findings' sentences
- highlight the challenges or reasons that lead to incorrect predictions from their perspective

- Use best results achieved

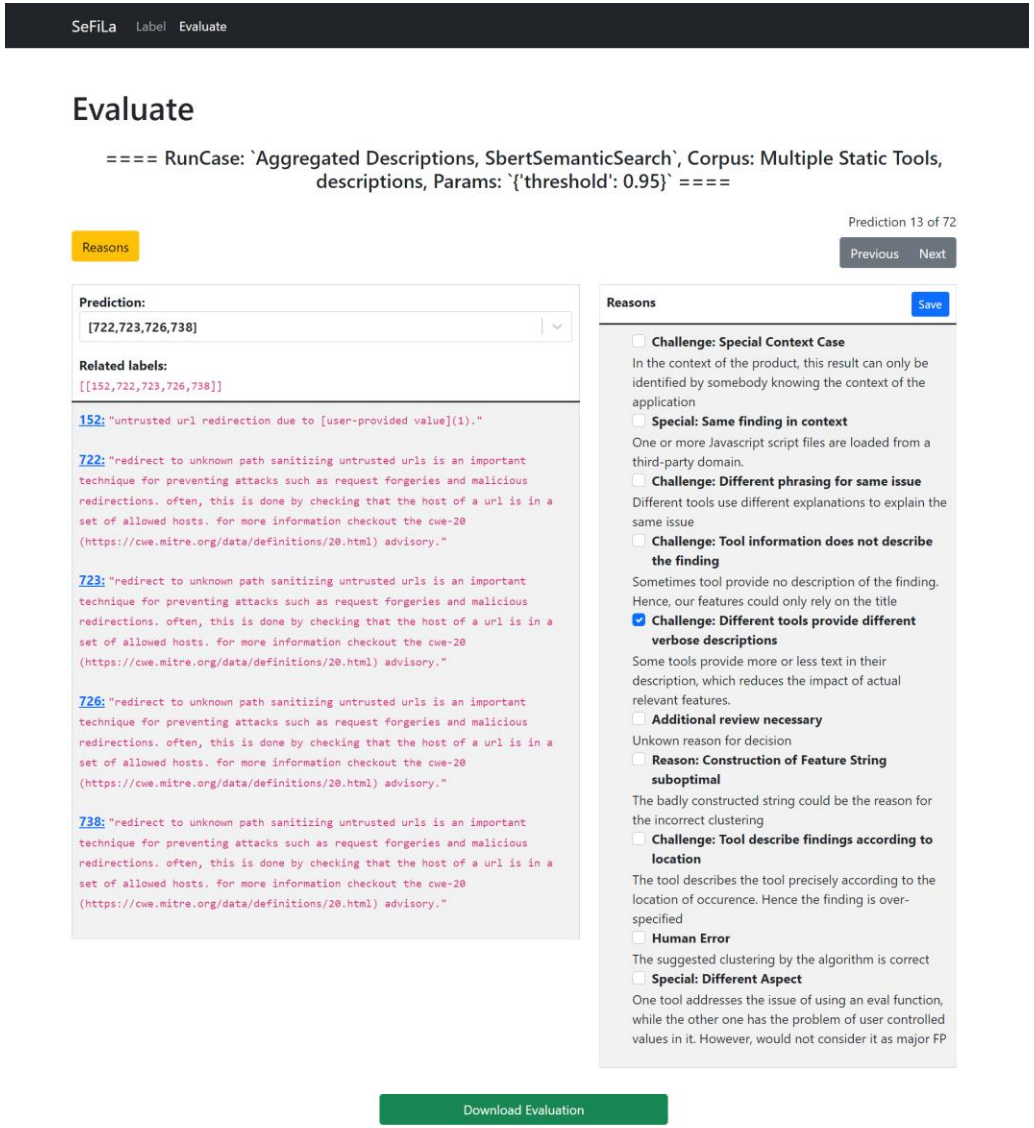
- 72 wrong predictions for SAST
  - 114 reason assignments

- 2 wrong predictions for DAST
  - 4 reason assignments

- Multiple reasons can be assigned to an incorrect prediction

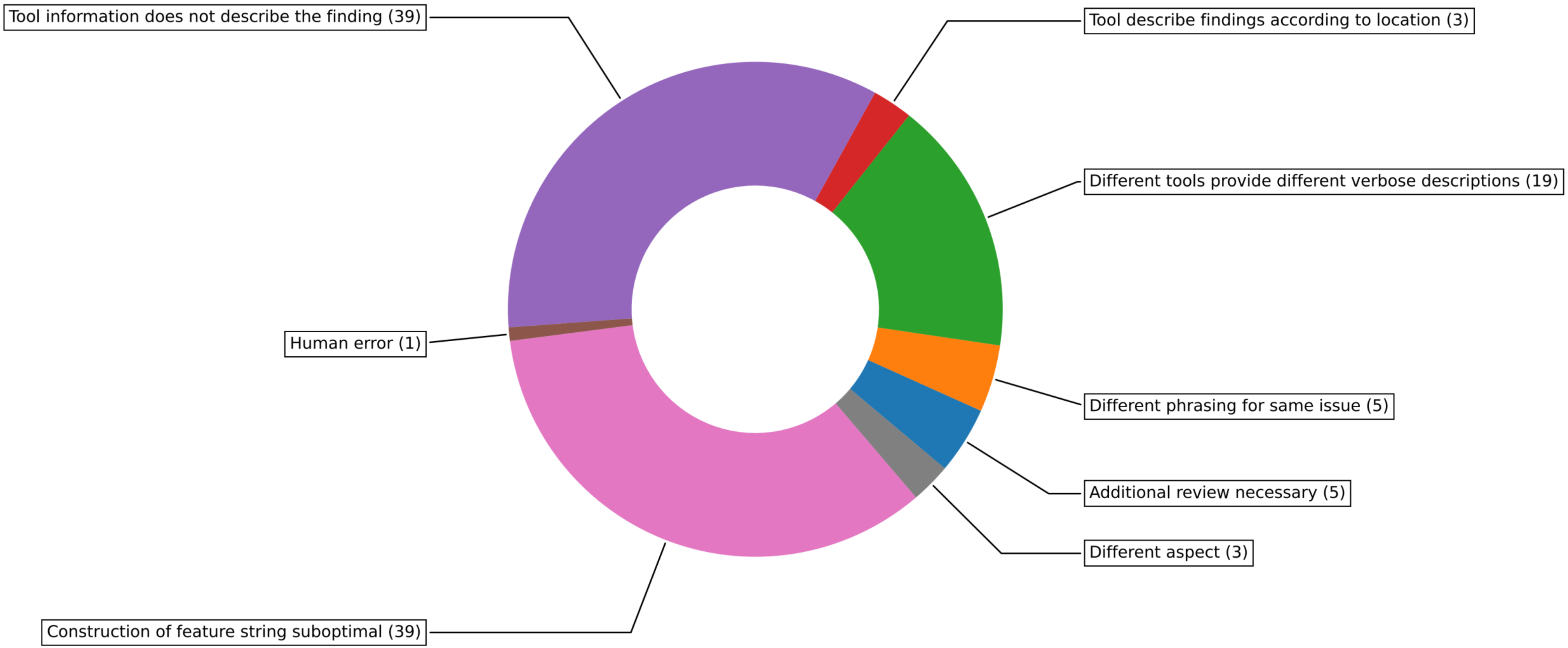
How do base NLP Semantic Similarity methods perform when applied to security tool reports corpus?

Security Findings Labeler (SeFiLa) – Evaluate feature



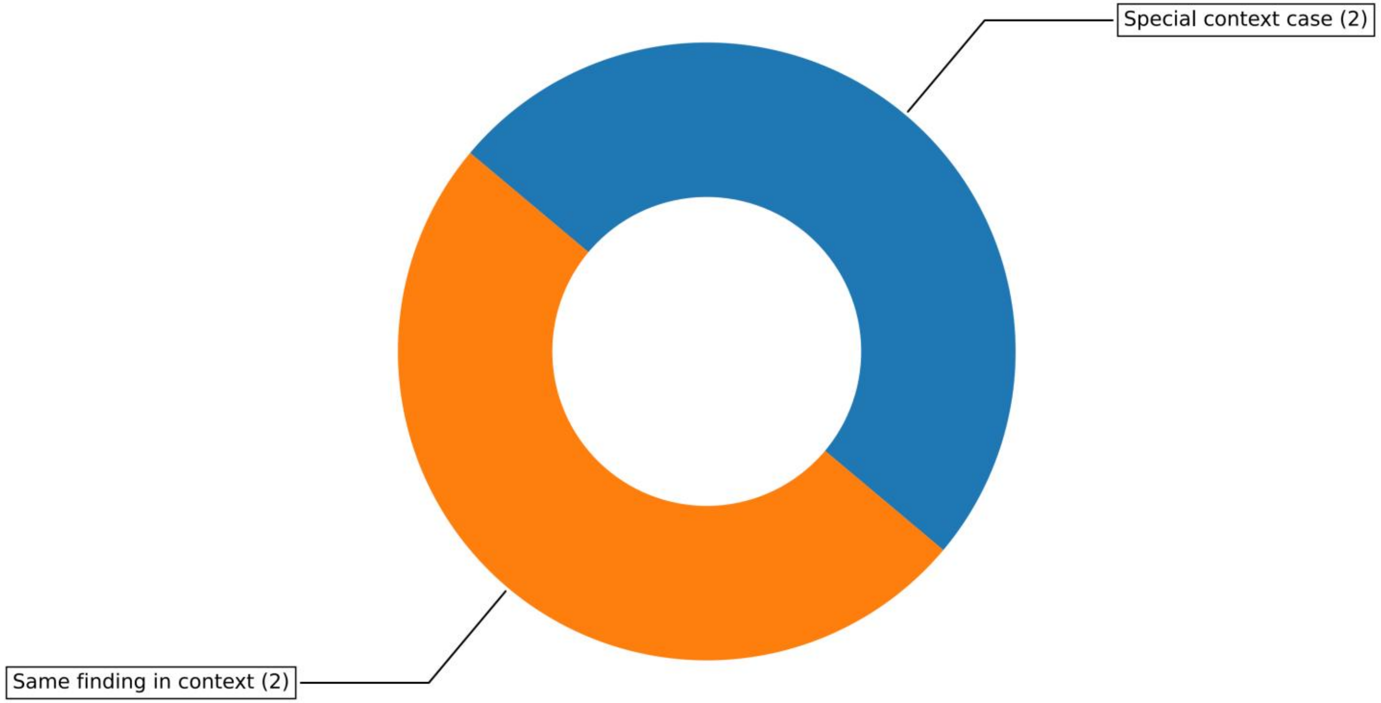
How do base NLP Semantic Similarity methods perform when applied to security tool reports corpus?

Qualitative results for SAST findings deduplication



How do base NLP Semantic Similarity methods perform when applied to security tool reports corpus?

Qualitative results for DAST findings deduplication



Outline

Motivation

Research Questions

1. What are duplicate security tool findings in DevSecOps and how can they be deduplicated?
2. How can we use Natural Language Processing (NLP) to deduplicate security tool findings?
3. How do base NLP Semantic Similarity methods perform when applied to security tool reports corpus?

Conclusion

Conclusion

- For semantic deduplication of security findings:
  - Base NLP techniques perform adequately
  - Challenge lies with formed corpus texts
- Deduce which base technique works best for required deduplication
  - **SAST:** LSI with *SAST-Aggregated*
  - **DAST:** SBERT with *DAST-NDS*
- When successfully engineering a solution for semantic deduplication:
  - Cater to limitations from qualitative analysis
  - Understand which category of semantic similarity techniques is feasible
- Presented methods and results to *Security Lifecycle* team at Siemens
- Intend to publish code and results online

Usage

- Improves current *security findings management*
- Results from this thesis implemented post existing CVE-ID-based deduplication of security findings

Thank you

Technische Universität München
Faculty of Informatics
Chair of Software Engineering for Business Information Systems

Boltzmannstraße 3
85748 Garching bei München

Tel +49.89.289. 17132
Fax +49.89.289.17136

matthes@in.tum.de
wwwmatthes.in.tum.de

Appendix

Transitive clustering

Before:

- *Prediction* 1 = {*Finding* : 1, *Similar Findings* : {1, 3, 5, 2}}
- *Prediction* 2 = {*Finding* : 2, *Similar Findings* : {2, 4, 5, 3}}

After applying transitive clustering:

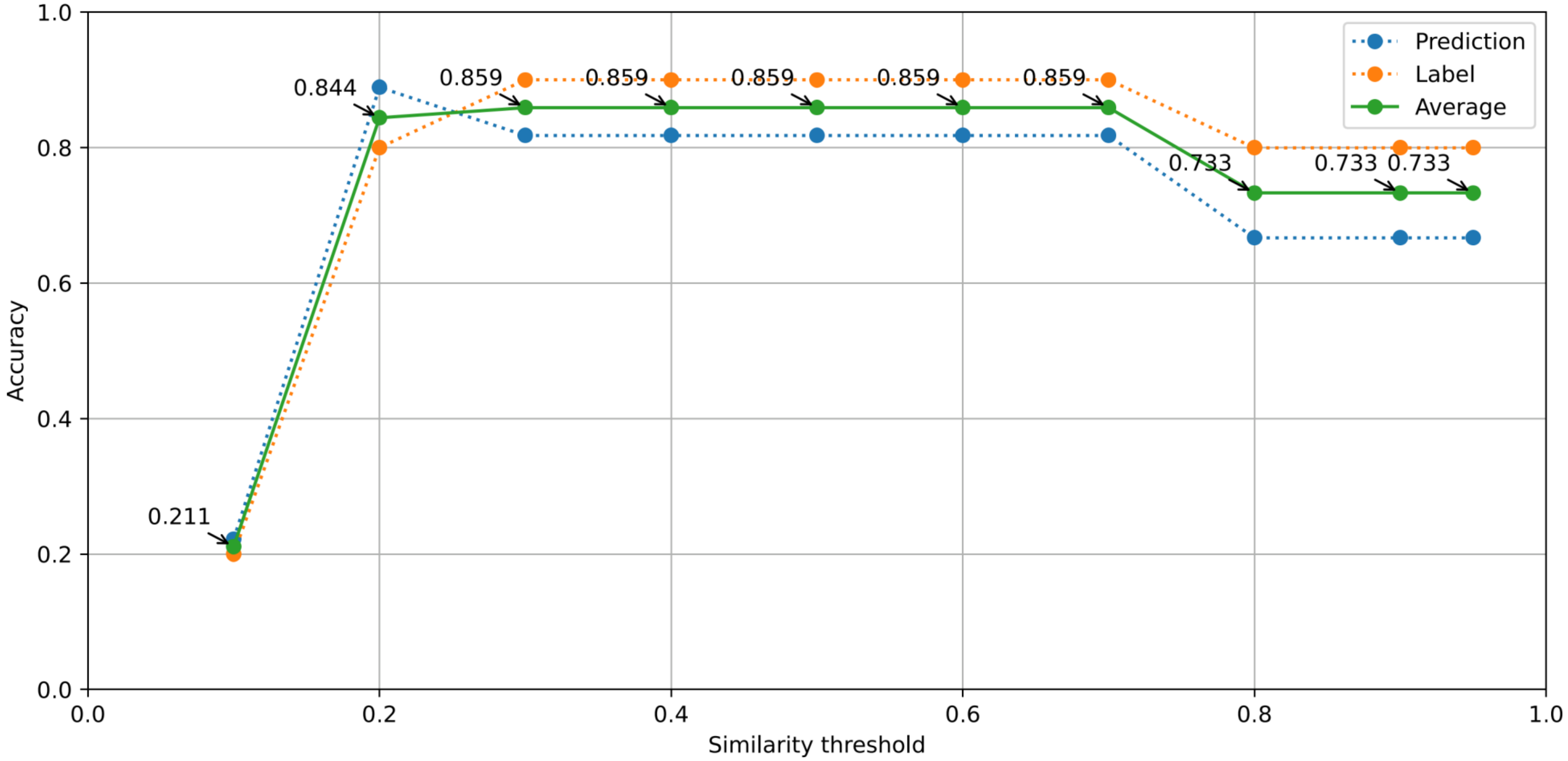
- *Prediction* 1 = {*Finding* : 1, *Similar Findings* : {1, 2, 3, 4, 5}}
- *Prediction* 2 = {*Finding* : 2, *Similar Findings* : {1, 2, 3, 4, 5}}

Reason:

- Semantic similarity score depends on vector space word embeddings from perspective of *query* text
- Similarity score of *Finding* 1 could be less than similarity threshold when looked from perspective of *Finding* 2
- Very low similarity scores causes a lot of false positives

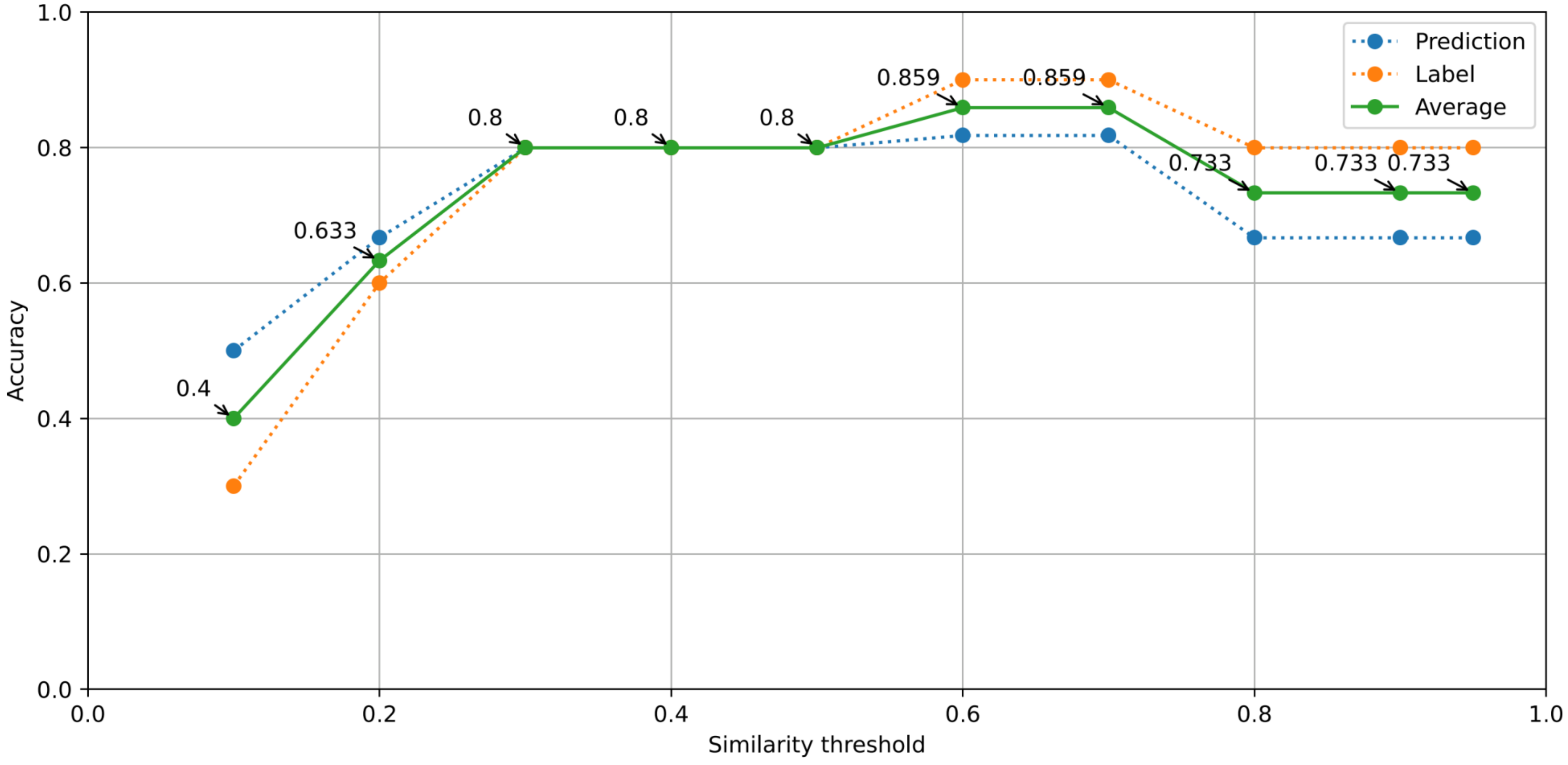
Appendix

Quantitative results – LSI *DAST-NDS*



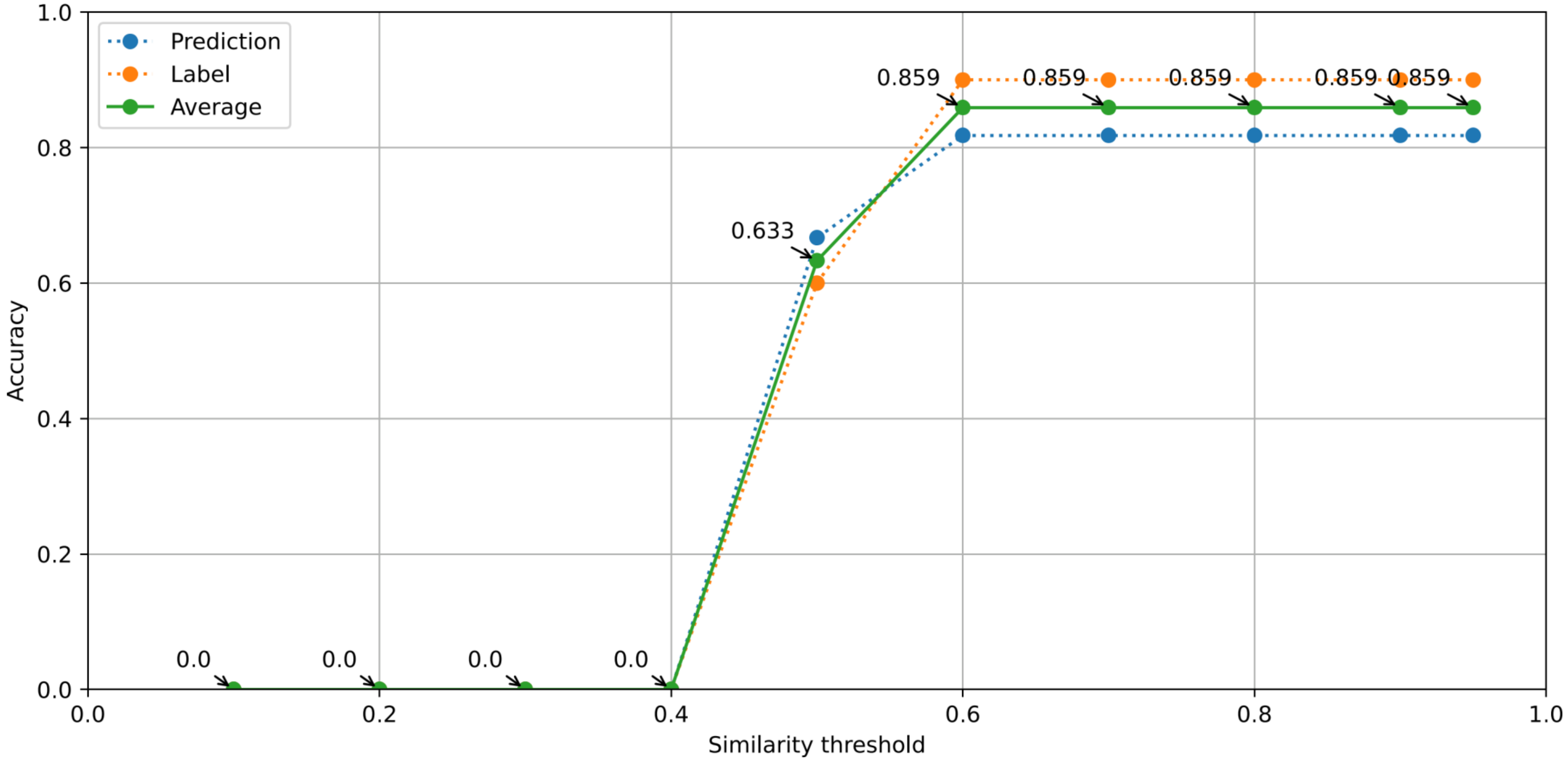
Appendix

Quantitative results – LSI *DAST-Description*



Appendix

Quantitative results – SBERT *DAST-Description*



Appendix

Quantitative results – corpus texts

1. *“no use .get method using data from request of user input allows user input data to be used as parameters for the 'request.get' method. without proper handling, it could cause a server side request forgery vulnerability. which is a type of exploitation in which an attacker abuses the functionality of a server, causing it to access or manipulate information in that server's domain. for more information checkout the cwe-918 (https://cwe.mitre.org/data/definitions/918.html) advisory.”*
2. *“no use request method using data from request of user input allows user input data to be used as parameters for the 'request' method. without proper handling, it could cause a server side request forgery vulnerability. which is a type of exploitation in which an attacker abuses the functionality of a server, causing it to access or manipulate information in that server's domain. for more information checkout the cwe-918 (https://cwe.mitre.org/data/definitions/918.html) advisory.”*

Prediction: {(1, 2)}

Labels: [{1}, {2}]

Appendix

Quantitative results – High f-measure, precision and recall values

Predictions = {(1, 2, 3, 4, 5, 6)}, Labels = {(1, 2, 3, 4), (5, 6)}

- Prediction pairs
  - P = {(2, 4), (1, 2), (3, 4), (1, 5), (4, 6), (1, 4), (2, 3), (4, 5), (2, 6), (5, 6), (3, 6), (1, 6), (2, 5), (1, 3), (3, 5)}
- Label pairs
  - Q = {(2, 4), (1, 2), (3, 4), (1, 4), (2, 3), (5, 6), (1, 3)}

- TP = |P ∩ Q| = | {(2, 4), (1, 2), (3, 4), (1, 4), (2, 3), (5, 6), (1, 3)} | = 7
  - pairs that occur in both P and Q
- FN = |Q – P| = | {} | = 0
  - pairs that occur in Q but not in P
- FP = |P – Q| = | {(1, 5), (4, 6), (4, 5), (2, 6), (3, 6), (1, 6), (2, 5), (3, 5)} | = 8
  - pairs that occur in P but not in Q

Precision = TP / (TP + FP) => 7 / (7 + 8) => 0.47

Recall = TP / (TP + FN) => 7 / (7 + 0) => 1.0

Appendix

List of reasons for poor deduplication

- **Special context case:** In the context of the product, this result can only be identified by somebody knowing the context of the application.
- **Same finding in context:** One or more Javascript script files are loaded from a third-party domain.
- **Different phrasing for same issue:** Different tools use different explanations to explain the same issue.
- **Tool information does not describe the finding:** Sometimes tool provide no description of the finding. Hence, our features could only rely on the title.
- **Different tools provide different verbose descriptions:** Some tools provide more or less text in their description, which reduces the impact of actual relevant features.
- **Additional review necessary:** Unknown reason for decision.
- **Construction of feature string sub-optimal:** The badly constructed string could be the reason for the incorrect clustering.
- **Tool describe findings according to location:** The tool describes the tool precisely according to the location of occurrence. Hence the finding is over-specified.
- **Human Error:** The suggested clustering by the algorithm is correct.
- **Different aspect:** One tool addresses the issue of using an eval function, while the other one has the problem of user controlled values in it. However, would not consider it as major false positive.