DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Biomedical Computing

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

Abdullah Gulraiz

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Biomedical Computing

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

Semantische Analyse zur Deduplizierung von Sicherheitsergebnissen in DevOps Security Tool Reports

Author:	Abdullah Gulraiz
Supervisor:	Prof. Dr. Florian Matthes
Advisor:	Phillip Schneider
Submission Date:	August 16, 2022

I confirm that this master's thesis in biomedical computing is my own work and I have documented all sources and material used.

Munich, August 16, 2022 Abdullah Gulraiz

Abstract

Adoption of DevOps practices has made modern-day software delivery rapid. To match this pace of delivery and to enable fixing of security vulnerabilities as early as they occur, automated cybersecurity practices are integrated at every stage of the DevOps process. One such cybersecurity practice is automated code analysis, whereby multiple security tools scan a software artifact before and after its deployment. These security tools produce semi-structured reports containing numerous security findings. Each security finding consists of verbose, machine-generated information regarding a security flaw identified in the software. Cybersecurity professionals need to read through all security findings to identify existing security flaws, before informing the relevant developers to fix them. Due to overlapping use-cases of security tools, duplicate findings identifying the same security flaw get generated. Reading through these duplicate findings at every deployment costs cybersecurity professionals valuable time and hinders the agility of DevOps. This thesis aims to explore the application of semantic similarity techniques from the field of Natural Language Processing (NLP) to identify and remove these duplicate security findings, in order to save time and efforts of cybersecurity professionals. We investigate possible ways of deduplicating security findings, and devise a framework for applying different categories of semantic similarity techniques to achieve the same. We also develop a tool for creating human-annotated datasets of security findings and for qualitatively evaluating our results. Finally, we use a method from each semantic similarity category to deduplicate security findings from two different datasets, and derive conclusions from the quantitative and qualitative evaluations of the results.

Contents

Abstract **iii**

1. Introduction **1**
1.1. Motivation . . . 1
1.2. Research Questions . . . 3
1.3. Thesis Outline . . . 4

2. Theoretical Foundations **5**
2.1. Natural Language Processing and Semantic Similarity . . . 5
2.1.1. Knowledge-based methods . . . 6
2.1.2. Corpus-based methods . . . 8
2.1.3. Deep Neural Network (DNN)-based methods . . . 9
2.1.4. NLP Concepts . . . 11
2.1.5. Metrics . . . 12
2.2. DevOps and Security Findings . . . 14
2.2.1. DevSecOps . . . 15
2.2.2. Security Tools . . . 17
2.2.3. Deduplication strategies . . . 19
2.2.4. Cybersecurity concepts . . . 19

3. Methodology **21**
3.1. Dataset . . . 25
3.1.1. Scanning . . . 25
3.1.2. Labeling . . . 25
3.2. Corpus formation . . . 26
3.2.1. Feature-based semantics aggregation . . . 27
3.3. NLP application . . . 28
3.3.1. Testbench . . . 28
3.3.2. Techniques . . . 29
3.3.3. Transitive clustering . . . 31
3.4. Evaluation . . . 33
3.4.1. Quantitative evaluation . . . 33
3.4.2. Qualitative evaluation . . . 34

4. Results **35**
4.1. Security Findings Labeler (SeFiLa) . . . 35
4.1.1. Label feature . . . 36

4.1.2. Evaluate feature . . . 40
4.2. Dataset statistics . . . 43
4.2.1. SAST dataset . . . 45
4.2.2. DAST dataset . . . 46
4.3. Quantitative evaluation . . . 49
4.3.1. Sentence-BERT (SBERT) Semantic Search-based deduplication . . . 49
4.3.2. Latent Semantic Indexing (LSI)-based deduplication . . . 54
4.3.3. Knowledge Graph (KG)-based deduplication . . . 57
4.3.4. Summary . . . 59
4.4. Qualitative evaluation . . . 64

5. Discussion **67**
5.1. RQ1: What are duplicate security tool findings in DevSecOps and how can they be deduplicated? . . . 67
5.2. RQ2: How can we use Natural Language Processing (NLP) to deduplicate security tool findings? . . . 68
5.3. RQ3: How do base NLP Semantic Similarity methods perform when applied to security tool reports corpus? . . . 69

6. Conclusion **73**
6.1. Future work . . . 75

A. General Addenda **77**
A.1. Definition of flowchart elements . . . 77

List of Figures **78**

List of Tables **80**

Bibliography **81**

1. Introduction

1.1. Motivation

Given the expanding ecosystem of high-available applications, maintained and advanced through continuous integrations and deployments, there is an ever-increasing pressure on the cybersecurity professionals of these applications to keep their end-users safe from cybersecurity exploits. These applications, before being handed to the user, are scanned through multiple *security tools* that scan for possible vulnerabilities or exploits in the code, dependencies, binaries, or even deployed software, and produce machine-generated, semi-structured human-readable security findings reports. The cybersecurity professionals then go through each of these reports and inform the right developers and teams in effort to get the vulnerabilities fixed and software deployed. Consequently, when there are more frequent updates to software to meet user and business demands, and an increasing number of available security tools covering the spectrum of modern-day cybersecurity needs, the effort to manage security findings multiplies exponentially.

Thanks to advancements in software engineering and DevOps (software Development (Dev) and IT Operations (Ops)), software that took months to update and cater to updated user demands now gets swiftly deployed and available as soon as the features get approved [1]. This is made possible by modern DevOps practices that allow for easy deployment, monitoring, and scaling of modern applications. An extension of DevOps is DevSecOps, which ensures safety and security in applications by treating cybersecurity as first citizen of the entire process [2]. In practice, this ranges from following secure coding practices (e.g., sanitizing user input, not storing passwords as plain text, etc.) to scanning code, dependencies, binaries and even a live deployed application with their specific security tools. Despite all these security practices, countless instances of cyberattacks still take place, like the recent DDoS attack on Microsoft Azure services [3]. This underlines the need for cybersecurity professionals to be comprehensive in their security checks, and recent advancements in the fields of artificial intelligence, helps us foresee that the quantity of security tools and the resulting security findings would only increase in the future.

Our study is motivated by a specific DevSecOps scenario in the industry where there exist comprehensive security guidelines for deploying and maintaining software. Typically, developers get assigned issues, which they fix and submit code for to their branch on the version control. The reviewers then examine the code for completeness and potential flaws and merge it in the *main* branch, which triggers a build of the application through pre-defined continuous integration / deployment (CI/CD) pipeline(s) [4]. In addition to containing stages necessary for building the deployable artifact(s), the pipeline includes stages which run security tools and proceed only if there were no vulnerabilities found or if there was

configuration to ignore some warnings. Otherwise, cybersecurity professionals read through the generated security tool reports to determine the underlying security issues and inform the related developer(s) of their solutions to get the problems fixed before restarting the pipeline. Typically, the security tool reports are *JSON* or *XML* files containing entries that represent security findings. This entire process of reporting back to the previous stage in pipeline to resolve security issues before proceeding to the next stage is known as a *security feedback loop*. Eventually, the goal is to make security feedback loops seamless and simple so security issues can be fixed effectively and efficiently.

The employed security tools, although covering different dimensions of cybersecurity, have overlapping use-cases to improve software coverage, cater for edge-cases with specialized tests and avoid black spots. This leads to production of duplicate security findings that highlight the same security issue, all of which need to be read by cybersecurity to identify existing security flaws. Knowing that there exist more than thirteen hundred security findings for the product we use in our study, and that there could be numerous additional security tools employed in the CI/CD pipelines, it gets inefficient for the security feedback loop if cybersecurity professionals read through similar security issues worded differently as separate security findings at every run of the pipeline. That is why we find it essential to investigate techniques and methods that support the automated analysis of security findings.

In this regard, the field of Natural Language Processing (NLP) shows to be of great relevance. Being a branch of artificial intelligence that works with natural language in texts and understands contextual nuances of the language used, modern-day NLP is heavily used in tasks like spam detection, machine translation (e.g., Google Translate), virtual agents (e.g., Apple's Siri), sentiment analysis and text summarization. Its immense success after the advent of deep learning-based transformer models like BERT [5], is evident from the fact that it is used in every single English language query processed by Google [6]. At its core, NLP basically reduces natural language texts into numerical representations, where each technique has a different way of forming these numerical representations and grasping aspects (e.g., semantics) of a given text.

For the studied DevSecOps scenario, the relevant task of NLP is *semantic similarity*, which quantifies the relationship between texts or documents based on the likeness of their meanings or semantic contents. In this research, we explore the usage and performance of various semantic similarity techniques from literature to identify duplicate security findings, since there exist no studies in literature achieving the same. For processing security findings, NLP is particularly attractive due to its ability to capture the semantic content of their texts and being impartial to the variation in wording of similar security findings from different tools. The usage of NLP semantic similarity for security findings deduplication contrasts with the presently used *rule-based deduplication*, whereby cybersecurity professionals attempt to match different aspects of security findings e.g., *file paths* to determine if they are duplicates. Our aim is to inform reader of the challenges and results of using NLP semantic similarity to deduplicate security findings, in order to reduce the time and effort of cybersecurity professionals when reading numerous findings and to increase the efficiency of security feedback loop.

To ensure that our results remain reproducible, all security tools used in our work are open-source and are used to scan the open-source software artifact *OWASP Juice Shop*[1]. This way we are able to present all data and results comprehensively without needing to conceal sensitive business information. Once cybersecurity professionals have scanned the software artifact with their choice of security tools, they pool all security findings together and label them as duplicates or non-duplicates of each other. As part of achieving this, we propose *SeFiLa*, a tool that seamlessly allows labelling of datasets and evaluation of results for this work by the cybersecurity professionals.

We then proceed to apply different semantic similarity techniques to score similarities between *corpus texts* of security findings, results of which allow us to identify duplicate security findings. These corpus texts are formed from features of the security findings data that explain the detected security vulnerability most descriptively. To avoid bias, we ensure that the chosen features are present in all security findings. We also provide the results of our semantic deduplication to cybersecurity professionals for reasoning of challenges, and report their qualitative evaluation along with the quantitative evaluation to the reader.

The end-goal from DevSecOps side of our work consists in the NLP techniques successfully identifying security findings as duplicates or non-duplicates from the larger pool of security findings, thereby recreating the ground-truth dataset labelled by cybersecurity professionals. This achievement will facilitate the bigger goal in DevSecOps of making the security feedback loop more efficient. From the NLP side of our work, our goal is to successfully contrast the efficacy of different semantic similarity techniques when applied to domain-specific security findings corpora. Consequently, our research is useful for both NLP researchers intending to explore the DevSecOps domain with regards to security tools and findings, as well as DevSecOps researchers aiming to gauge the usability of semantic similarity or NLP in security tools.

1.2. Research Questions

The main objective of our work is to determine the effectiveness of NLP semantic similarity techniques when applied to the domain-specific security findings corpora, in order to deduplicate the respective security findings. The first step to achieve this is to understand and establish what duplicate security findings mean. Furthermore, since security findings are semi-structured objects containing multiple features, it is necessary to understand which features are useful for deduplications, and the possible ways we can use them to deduplicate security findings:

RQ1: What are duplicate security tool findings in DevSecOps and how can they be deduplicated?

Having established the foundations of our goal, we now need to understand the crux of NLP along with the possibilities it offers to compare similarity between two texts. We also need to understand how we can apply these NLP semantic similarity methods, which work with natural language texts, to semi-structured security findings containing multiple features and domain-specific texts. Furthermore, we need to create a security findings dataset to use

[1]https://owasp.org/www-project-juice-shop/

with our NLP techniques. All these requirements help us constitute the second research question:

RQ2: How can we use Natural Language Processing (NLP) to deduplicate security tool findings?

The answer to above two research questions will allows us to correctly apply NLP techniques for deduplicating security findings. We now need to perform experiments to comprehensively evaluate the performance of base NLP semantic similarity methods for deduplicating security findings. This evaluation will allow us to conclusively inform cybersecurity professionals of the extent NLP can help in their deduplication process, and allow us to realize the challenges that occur in NLP-based deduplication of security findings. Hence, our final research question is phrased as:

RQ3: How do base NLP Semantic Similarity methods perform when applied to security tool reports corpus?

The answers to all above research questions will lead to successful completion of our research.

1.3. Thesis Outline

The remainder of this thesis is structured as follows:

In Chapter 2, we present the theoretical foundations describing theory and technical specificities of both Natural Language Processing and DevSecOps, which are required for a clear understanding of the concepts applied in this thesis. In Chapter 3, we describe our experimental designs for the creation of ground-truth datasets and their usage for deduplication, the implementation of our researched semantic similarity techniques, and the quantitative and qualitative evaluations of our results. In Chapter 4, we objectively state and visualize the quantitative results of our experiments. We also include a qualitative evaluation of the deduplicated findings by cybersecurity professionals. In Chapter 5, we make deductions from our results and use the information presented throughout the thesis answer our research questions. Finally, Chapter 6 is designated for the concluding remarks of our research and highlights the possible future work.

2. Theoretical Foundations

2.1. Natural Language Processing and Semantic Similarity

Natural Language Processing (NLP) [7] is an area of artificial intelligence that focuses on understanding how humans interpret and use natural language (theoretical linguistics), and aims to implement the same for computers (computational linguistics) for performing useful tasks. It does so by implementing computational techniques that allow automatic analysis and representation of human languages by computers. Most of these techniques are based on *syntactic representations* in texts, i.e., they rely on the co-occurrence frequencies of words instead of their semantic meanings. This limits them to only use the information present in the text being processed, unlike humans, who can relate a piece of text to related concepts, relevant episodes, or sensory experiences. However, this cognitive gap is being bridged by modern computational models that also consider semantic features not explicitly mentioned in text.

A domain of computational linguistics is sentence analysis, which can be divided into two main parts: *syntax analysis* and *semantic analysis* [7]. Syntax analysis deals with determining and regularizing syntactic structure of a sentence, which often results in omitting words not contributing to the meaning of the sentence e.g. "He talks faster than John ~~talks~~", and converting all passive voice sentences to active voice for better depicting operator-operand relations e.g. "Those grapes were crushed by me" to "I crushed those grapes". On the other hand, semantic analysis deals with determining meaning of a sentence by translating the input language to an intermediate language with simpler semantics e.g., translating a natural language instruction to formal logic or database command language. Therefore, syntax analysis helps ensure correct sentences and semantic analysis allows computers to perform some action in response to a natural language input, e.g., moving a robot arm when instructed in natural language.

An application of semantic analysis is determining similarity of texts or *textual similarity*. In early days, two texts were considered similar if they contained the same words/characters. However, as can be seen in the example "John and David studied Maths and Science" and "John studied Maths and David studied Science", two sentences could have very different meanings despite having the same words. This necessitates the concept of *semantic* (textual) similarity, which measures the equivalence between two blocks of text based on their semantic meaning. The equivalence or similarity is measured as a percentage or ranking instead of a binary decision of being similar or non-similar [8]. Semantic similarity forms an essential part in wide range of NLP applications including information retrieval [9], text summarization [10], text classification [11], essay evaluation [12], machine translation [13], and question answering [14, 15]. They can be broadly classified into knowledge-based methods,

corpus-based methods, deep neural network-based methods, and hybrid methods. In the following subsections, we explain these categories and the methods we adopt from them.

2.1.1. Knowledge-based methods

Knowledge-based methods determine semantic similarity between two terms based on information from an external knowledge source. These knowledge sources are well-structured representations of terms or concepts, which are interconnected through semantic relations and offer measures to compute semantic similarity of terms contained based on their actual meanings. [16]. These knowledge sources include ontologies, lexical databases, thesauri, and dictionaries [8]. For our work we use WordNet [17], a lexical database, as the knowledge source.

WordNet can be visualized as a graph-like structure, with nodes representing the meanings of words and concepts and edges representing the relationships between them [18]. The semantic similarity between two contained words depends on the path distance between their nodes [19]. WordNet currently contains more than 100,000 English concepts [16]. Its alternatives include Wiktionary and BabelNet [20].

Overall, knowledge-based methods are computationally simple and give exceptionally good results due to being backed by an external knowledge source. For the same reason, they also handle ambiguities like synonyms, idioms, and phrases efficiently. However, this dependency means that external knowledge source(s) used should be reliable and always updated, of which latter is a computationally expensive task. Furthermore, while strong ontologies like WordNet exist for the English language, similar resources do not exist for all languages and domains, which is a serious drawback for knowledge-based systems given there exist 6500 languages and numerous domains [8].

Below we discuss different knowledge-based methods used to calculate the semantic similarity score between two terms (concepts or words) while using an external knowledge source.

Edge-counting methods

Edge-counting methods deduce semantic similarity of two words based on the *distance* between their nodes, where a greater distance means lesser similarity. The measure *path* explicitly counts the edges between two terms t_1 and t_2, with similarity being inversely proportional to the shortest path length between them [21]:

$$sim_{path}(t_1, t_2) = \frac{1}{1 + min_len(t_1, t_2)}$$

Another measure, *wup*, gives importance to the fact that terms deeper down the knowledge source hierarchy are more specific, and hence more similar to each other than generic terms at lower depths. It, therefore, also includes the depth of *common ancestor term* of both terms t_1 and t_2 in calculation. This common ancestor term is referred to as the Least Common Subsumer (t_{lcs}), and is included in the similarity measure as [22]:

$$sim_{wup}(t_1, t_2) = \frac{2depth(t_{lcs})}{depth(t_1) + depth(t_2)}$$

A weakness of edge-counting methods is that they assume similar words to exist at the same depths in knowledge source hierarchy, which is not true for all cases.

Feature-based methods

Feature-based methods utilize the semantic features of words, like 'gloss' [16], which refers to meaning of a word as defined in the dictionary. They measure semantic similarity of two terms as the extent of overlap between their gloss [8]. A major limitation of feature-based methods is their requirement of rich semantic features to be incorporated by ontologies, which is not always met since most ontologies just include taxonomic relationships [16].

Information Content-based methods

Information content (IC)-based methods measure the semantic similarity of two words using the *IC value* associated with them [8]. The IC value of a word quantifies its specificity in a context, with a higher IC value indicating the word being specific and describing its concept clearly and a lower value indicating it having abstract meaning [18]. It can be measured by applying Inverse Document Frequency (IDF) [23] to a text corpus, by relating the specificity of a word inversely with the number of times it occurs in text. While being a corpus-based method semantic similarity method, it is also possible to measure the IC value from intrinsic structure of an ontology [24] or by using multiple ontologies [25]. IC-based semantic similarity measures include *res* [26], *lin* [27] and *jcn* [28].

Combined Knowledge-based methods

Combined knowledge-based methods allow us to yield the benefits of multiple knowledge-based methods. One of the proposed methods is *wpath* [18], which is a weighted path measure that adds IC value of the Least Common Subsumer (LCS) to the shortest path length of terms t_1 and t_2. It calculates semantic similarity as:

$$sim_{wpath}(t_1, t_2) = \frac{1}{1 + min_len(t_1, t_2) * k^{IC_{t_{lcs}}}}$$

Where k is a hyperparameter that needs to be tuned for different knowledge graphs (KG) based on their distributions of terms in each domain. This method was proposed to be used with various KGs including WordNet, DBPedia [29] and YAGO [30], and achieves great correlation in most of the gold-standard datasets [8]. It is, therefore, also our measure of choice when using knowledge-based methods in subsequent chapters.

2.1.2. Corpus-based methods

Corpus-based methods work by constructing a vector representation of a large corpus of text data, and for two sentences estimate the semantic similarity score by applying a distance measure to their respective vectors from the vector space. The vector representation is formed using *distributional hypothesis* [31], which is based on the idea that similar words occur together frequently. From the many distance measures proposed to estimate semantic similarity between distributional hypothesis-based vectors, *cosine similarity* is the most significant and widely used measure [32], and is also our choice of distance measure in subsequent chapters.

Since the formation of vector representation is based on statistical measures, explicit meanings (as specified in knowledge-bases) of words play no role in measuring the semantic similarity for corpus-based methods [8]. This makes corpus-based systems language- and domain-independent, unlike knowledge-based systems [33]. However, this also means that their performance is highly dependent on quality of corpuses the vector representations are formed from.

A manifestation of corpus-based methods is *word embeddings*, which are pre-trained models that convert sentences to numerical vectors. These vectors retain the contextual relationships between words, and can be directly used to compute semantic similarity between two sentences. Inside word embeddings, these vectors are computed through different approaches like neural networks [34], word co-occurrence matrix [35], or representations in terms of the context in which the word appears [36]. Some popular and widely used word embeddings include *word2vec* [37], *GloVe* [35], *fastText* [38], and *BERT* [5].

Below we describe workings of the corpus-based semantic similarity methods used in our work. Alternative corpus-based methods that can be explored for determining semantic similarity include Hyperspace Analogue to Language [39], Explicit Semantic Analysis [40], Word-alignment Models [41], Normalized Google Distance [42], Dependency-based Models [43], Kernel-based Models [44], and Word-attention Models [45].

Latent Semantic Analysis

Latent Semantic Analysis (LSA) [46] first forms a vector representation from the corpus of text data, by creating a *word co-occurrence matrix* X. The rows of matrix represent the unique words in text, and the columns represent individual documents (or sentences) we want to compute semantic similarity between. Each cell is then filled with the respective word counts. Since the matrix is formed from a large corpus of text data, dimensionality reduction must be achieved using the mathematical technique known as Singular Value Decomposition (SVD) [8]. SVD breaks the large matrix to three individual matrices, whose product results in the original matrix. Of these three matrices, two represent the rows and columns as vectors derived from their respective eigenvalues, and one is a diagonal matrix [47], as illustrated in Figure 2.1[1].

Dimensionality reduction is achieved by selecting k largest singular values $\sigma_1, ..., \sigma_l$ and their corresponding singular vectors from U and V. The row "word" vector $\hat{t}_i^T$ and column

[1]Source: https://wikimedia.org/api/rest_v1/media/math/render/svg/150ac35d701a80f0436220209c6b6f7ca938e854

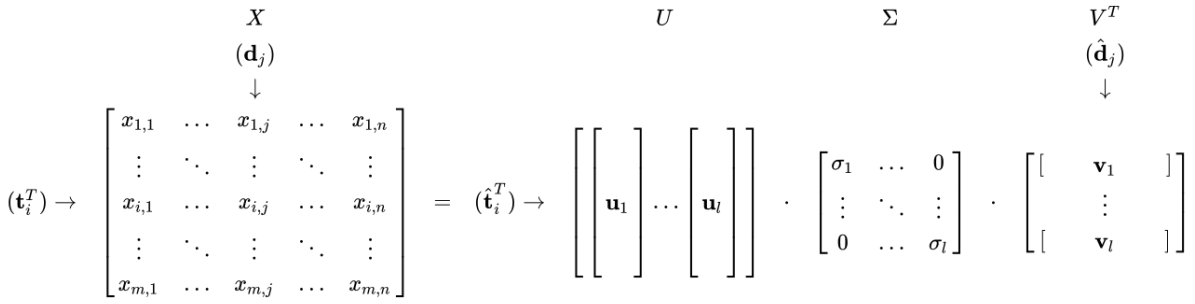


Figure 2.1.: SVD of the word co-occurence matrix

"document" vector $\hat{d_j}$ now have k entries, and they serve as the approximated lower dimension "semantic space". The dimensionality reduced approximation of the word co-occurrence matrix becomes:

$$X_k = U_k \Sigma_k V_k^T$$

This is also known as a rank k approximation to X with the smallest error. To compute the semantic similarity between two document entries d_j and d_q, they must first be translated to lower dimension space:

$$\hat{d_j} = \Sigma_k^{-1} U_k^T d_j$$
$$\hat{d_q} = \Sigma_k^{-1} U_k^T d_q$$

Where the inverse of Σ_k is found by inverting each nonzero value within the matrix. Cosine similarity is then used between vectors $\Sigma_k \cdot \hat{d_j}$ and $\Sigma_k \cdot \hat{d_q}$ to estimate the semantic similarity score.

Latent Dirichlet Allocation

Latent Dirichlet Allocation (LDA) [48] is used to calculate the semantic similarity between two complete documents of text. It achieves this by representing the general idea behind a document as a numerical vector, instead of considering every individual word in the document [8]. This gives it the advantage of reduced dimensionality, since modeled topics are less than total words in the document. Semantic similarity between documents is calculated by applying cosine similarity to their respective vectors [49].

2.1.3. Deep Neural Network (DNN)-based methods

Recent developments in neural networks have also helped develop better performing semantic similarity techniques. DNN-based semantic similarity methods have shown to outperform most traditional methods, with the success of transformer-based models serving as a breakthrough in semantic similarity research [8]. Other popular techniques include Convolutional

Neural Networks (CNN) [50], Long Short-Term Memory Networks (LSTM) [51], Bidirectional Long Short-Term Memory Networks (Bi-LSTM) [52], and Recursive Tree LSTM Networks [53].

Since the performance of DNN-based methods increases with increased data, computational power, and larger model sizes [54], their usage is restricted in scenarios where these resources are scarce. Furthermore, most deep-learning models are "black-boxes", where it is hard to debug and establish what features affect model performance or can cause failures. For this reason, many domains like finance, cybersecurity, etc., that deal with sensitive data are reluctant to employ DNN-based methods because of their lack of interpretability [8]. This give corpus-based methods, which have solid mathematical foundations, a clear advantage over DNN-based methods.

For our work, we explore the use of recent transformer-based models.

Transformer-based models

Transformer-based models [55] generate word embeddings by using *attention mechanisms* to capture semantic properties of words. The semantic similarity of two sentences can be simply calculated by taking cosine similarity of the generated word embedding vectors. Transformer networks basically consist of two parts: the *encoder* and the *decoder*. The encoder consists of layers of multi-head attention mechanisms followed by a fully connected feed-forward neural network, whereas the decoder is similar to the encoder with one additional layer of multi-head attention. Although originally proposed for machine translation task, the transformer-based models have proven very successful for generating word embeddings [5].

Initially, a transformer-based model is pre-trained on all domains of textual data, which helps learning of low-level features by the model. Later, when a new task is presented, the model is fine-tuned by adding an additional layer on top of the trained network [56]. This allows retention of all previous knowledge, while specializing the model for the task and domain of our choice. Many high-performing pre-trained models [54] are available to choose from and directly fine-tune for the downstream task of choice.

A popular pre-trained [16] transformer network is Bidirectional Encoder Representations from Transformers (BERT) [5], which gives state of the art results for various NLP tasks including question-answering [57], sentence classification [58] and sentence-pair regression. Its input consists of two sentences separated by a special [SEP] token, to which multi-head attention is applied and output is passed through a simple regression function for the final score.

For our work, we use Sentence-BERT (SBERT) [59], a modification of BERT network that uses siamese and triplet network [60] structures. The siamese network architecture enables fixed-size output vectors, which can be conveniently used to calculate semantic similarity using cosine similarity. The architecture of SBERT for computing similarity score between two sentences is shown in Figure 2.2. The major advantage of SBERT over BERT is that it reduces the time to find most similar sentence pairs in a dataset, while maintaining the same accuracy as BERT. For a 10,000 sentences dataset, BERT takes 65 hours to find similar sentence pairs whereas SBERT takes only 5 seconds.

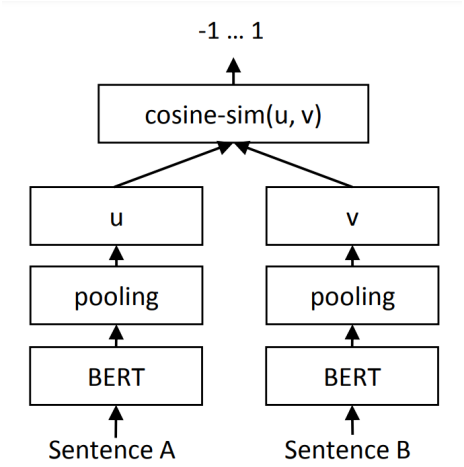


Figure 2.2.: SBERT architecture for inferring sentence similarity

2.1.4. NLP Concepts

Below we define some common NLP concepts used throughout this thesis, adapted from documentation of the popular NLP library, spaCy[2].

Tokenization

Segmenting text or sentence into individual words, punctuations marks etc. The results are called tokens.

Part-of-speech (POS) Tagging

Assigning word types to tokens, like verb or noun.

Dependency Parsing

Assigning syntactic dependency labels which describe the relations between individual tokens, like subject or object.

Lemmatization

Assigning the base forms to each token, e.g., the lemma of “was” is “be”, and the lemma of “rats” is “rat”.

Sentence Boundary Detection (SBD)

Finding and segmenting individual sentences.

[2]https://spacy.io/

Named Entity Recognition (NER)

Labelling each token that is a named "real-world" object, as its type e.g., "John" as *Person*, "Microsoft, Inc." as *Company*, etc.

Entity Linking (EL)

Relating textual entities to unique identifiers in a knowledge source, e.g., in sentence "Paris is the capital of France", "Paris" refers to the city of Paris and not to the celebrity Paris Hilton, and hence would be linked to the respective entity in knowledge source representing the city of Paris.

Similarity

Comparing how similar words, text spans or documents are to each other.

Text Classification

Assigning categories or labels to a whole document or parts of it.

Rule-based Matching

Finding sequences of tokens based on their texts and linguistic annotations, similar to regular expressions.

2.1.5. Metrics

As will be shown in subsequent chapters, our work results in formation of clusters that are evaluated against a gold standard dataset. Below we briefly explain the clustering metrics used to evaluate performance of the applied NLP techniques with. These metrics are generic and applicable to all such machine learning techniques.

Pair counting and Contingency Matrix

Pair counting allows comparison of entities with respect to magnitudes of their attributes or traits [61]. Members of a clustering are incrementally paired and are then compared to similarly paired members of the other clustering [62]. A contingency matrix, also known as the confusion matrix, shows instances of predicted classes as columns and instances of label classes as rows [63]. The cells show overlap between both class categories, which allows comparison of output and visualization of algorithm performance. Below, we outline the method of forming a contingency matrix from pair counting, as outlined in [62].

Given a set Q comprising of clustering present in gold-standard dataset, set P comprising of clustering that results from our algorithm, and set D comprising of all possible elements that can be present in Q and P, we form incremental pairing of each clustering as $Pairs_P$, $Pairs_Q$ and $Pairs_D$ respectively. An example of how this pairing is done is shown below:

$$P = \{1,2,3\},\{4,5\},\{6\}$$
$$Pairs_P = \{(1,2),(1,3),(2,3),(4,5)\}$$

Note above that we skip the elements that exist individually. Using the formed pairs, the values for four quadrants of the contingency matrix can be calculated as:

Member pairs that occur in both partitions,

$$a = |Pairs_P \cap Pairs_Q|$$

Member pairs that occur in $Pairs_Q$ and not $Pairs_P$,

$$b = |Pairs_Q \setminus Pairs_P|$$

Member pairs that occur in $Pairs_P$ and not $Pairs_Q$,

$$c = |Pairs_P \setminus Pairs_Q|$$

And non-member pairs that do not occur in any clusters in either partition,

$$d = |Pairs_D \setminus (Pairs_P \cap Pairs_Q)|$$

We also define

$$n = |Pairs_D| = a + b + c + d$$

The contingency matrix can then be written as shown in Table 2.1. We now describe the metrics that use values from the contingency matrix to perform different kinds of evaluations.

Table 2.1.: Contingency Matrix

	Predicted (cluster)		Total
	Pairs *in P*	Pairs *not in P*	
Actual (gold std.)			
Pairs *in Q*	a (True Positive)	b (False Negative)	$a+b$
Pairs *not in Q*	c (False Positive)	d (True Negative)	$c+d$
Total	$a+c$	$b+d$	$a+b+c+d=n$

Accuracy

The accuracy metric measures the ratio of correct predictions over the total number of predictions made [64]. In clustering, it is also known as the Rand measure [65], and is useful in telling the ratio of predicted clusters that exactly match the clusters in gold-standard dataset. It can be measured as:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} = \frac{a+d}{n}$$

Precision

Precision is used to measure the positive patterns that are correctly predicted from the total predicted patterns in a positive class [64]. In clustering, this is useful in determining the ratio of correctly predicted clusters from all the clusters that were predicted.

$$Precision = \frac{TP}{TP + FP} = \frac{a}{a + c}$$

Recall

Recall is used to measure the fraction of positive patterns that are correctly classified [64]. This helps us identify the ratio of correctly predicted clusters to all the true labels, which consist of individually existing elements and clustered elements.

$$Recall = \frac{TP}{TP + FN} = \frac{a}{a + b}$$

F-Measure

The f-measure, also known as the Dice Measure [66], is calculated as the harmonic mean between precision and recall. Since increasing precision decreases recall and vice versa, the f-measure serves as an intermediary metric as it gives equal importance to both. It can be calculated as:

$$F - measure = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2TP}{2TP + FP + FN} = \frac{2a}{2a + b + c}$$

2.2. DevOps and Security Findings

DevOps is a collaborative and multidisciplinary effort to automate continuous delivery of new software versions, while guaranteeing their correctness and reliability [67]. It is widely adopted by enterprises [68, 69, 70] to improve their software delivery processes [71]. Evolving from the agile movement [72] that advocates small software release iterations with customer reviews, DevOps allows frequent software releases in a production-like environment. To do so, it proposes a complementary set of agile practices to effectively deliver software in short cycles iteratively [67].

DevOps is closely related to the concepts of *continuous delivery* and *continuous deployment* [73, 74, 75, 76]. Continuous delivery (CD) [77] is the deployment of software to production by the press of a button, after it has passed all stages of an *automated deployment pipeline*. These stages include compilation, automated tests, containerization, security tests, etc., and are run automatically once the software is committed to version control repository. Continuous deployment [78] is a variant of continuous delivery, which automatically sends to production every software that successfully passes through the pipeline. Both concepts differ from *continuous integration* (CI), which is the integration of code from multiple developers into

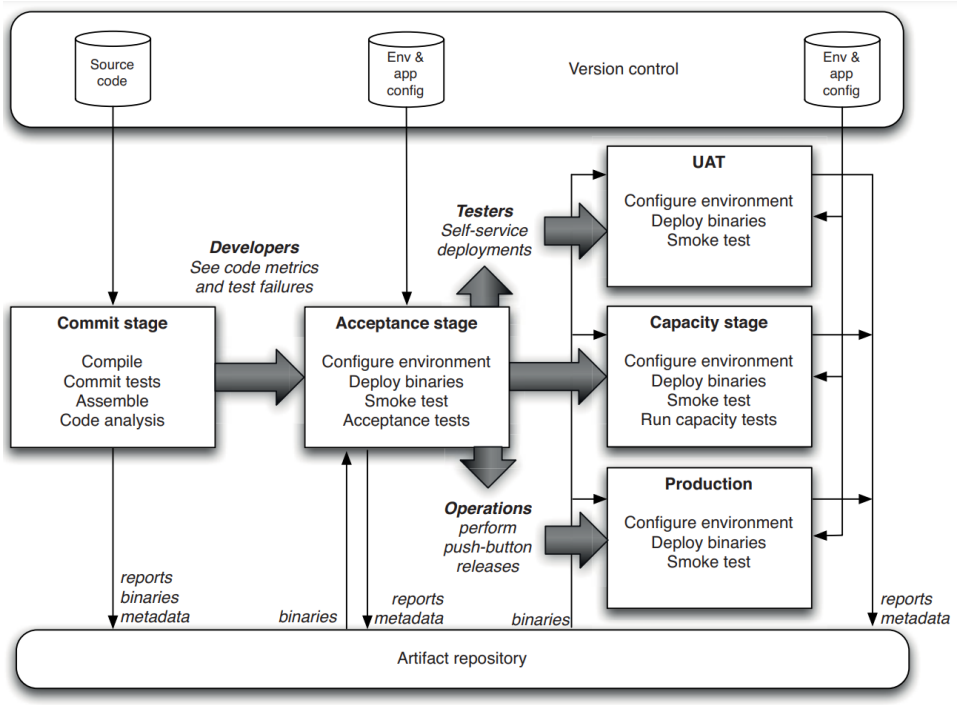


Figure 2.3.: Automated deployment pipeline

the same version of software after automated error checking [79]. A typical automated deployment pipeline, adapted from [68], is shown in Figure 2.3.

The process of DevOps heavily relies on automation tools. A combination of these tools helps achieve the goals of human collaboration, continuous delivery, and maintenance of software reliability. These tools broadly serve the purpose of knowledge-sharing (e.g., GitLab wiki), source code management (e.g., Git), build process (e.g., Maven), continuous integration (e.g., Jenkins), deployment automation (e.g., Docker) and monitoring & logging (e.g., Prometheus) [67]. They are vast in numbers [80, 81] and constantly evolve in their nature and function [73], which is a fact that strongly motivates our research.

2.2.1. DevSecOps

A necessary expansion to DevOps is DevSecOps, which integrates security controls and processes into the DevOps software development lifecycle [82]. It promotes collaboration between development teams, operations teams, and security teams [83], and makes sure that everyone agrees to the responsibility of security [84, 85]. It *shifts security to the left*, by including it in every part of the development process [86] instead of traditionally addressing it at the end of process [2]. Security experts are involved from initial planning to every iteration of the development cycle [87, 88], and help developers and operators on security considerations [89, 90, 84]. It also promotes use and development of metrics to track threats and vulnerabilities throughout the software development process [91], which allows an organization to verify

how good an application is on demand [92].

A *vulnerability* is defined as a property of system security requirements, design, implementation, or operation that could be accidentally triggered or intentionally exploited and result in a security failure [93]. It has been proven important experimentally [94, 95] and anecdotally [96] to use a variety of vulnerability prevention and discovery techniques throughout the software development lifecycle. Two such vulnerability discovery techniques are *static analysis* and *dynamic analysis*.

Static (code) analysis points to issues in source code such as non-compliance to a standard style, problems of maintainability, risk of bugs and even vulnerability breaches [67]. It provides a scalable way for security code review that can be used early in the lifecycle, does not require code to be executable, and can be used on subsections of the codebase [97]. Dynamic analysis, on the other hand, is the analysis of properties of a running program [98] that are investigated using information gathered at run-time. Both static and dynamic analysis are regarded as complementary approaches [99] and have been compared for their merits and demerits [100, 101]. Some major differences between them, as outlined in [99], are listed in Table 2.2.

Table 2.2.: A comparison of Static Analysis and Dynamic Analysis

Static Analysis	Dynamic Analysis
Does not require program to be executed	Requires program to be executed
Less precise	More precise
Holds for all executions	Holds for a particular execution
Lacks in handling run-time programming language features	Well suited to handle run-time programming language features like polymorphism, dynamic binding, threads, etc.
Incurs less overhead	Incurs large run-time overheads

Like DevOps uses automation tools to achieve rapid development, deployment [102, 92] and feedback from end-users [103], DevSecOps automates security controls that do not need manual interference [89]. Examples of such automation include the execution of static code analysis in automated deployment pipeline, and the automated dynamic analysis of deployed artifact once pipeline has finished. Moreover, these security controls run without hindering the agility of underlying DevOps processes [87, 104, 91].

As a result of automation of security procedures, multiple tools produce new *security findings*. A security finding is a single entry in the semi-structured output (report) of the security tool. Due to duplicate coverage of some features in the software to prevent blind spots, these security findings also get duplicated. A finding is duplicate if it is identified multiple times by the same security tool, or if it is identified by different security tools due to overlapping coverage of the software. In an experiment to assess the effectiveness of static code analyzers, it is observed that at least 41% of vulnerabilities get re-identified by all tools [97]. Given that it is important in DevSecOps to monitor every part of the inventory and

to log every resource [85, 105, 82], the total number of resulting findings become numerous and contain a significant percentage of duplicates.

In practice, the presence of security findings by automated security testing halts the automated deployment pipeline, prompting cybersecurity professionals to read through them and provide feedback to the relevant developers. Once resolved, the pipeline is re-run. However, reading through several duplicate security findings is inefficient and deters agility, further hindering the agility of underlying DevOps processes, which must be prevented [87, 104, 91]. Therefore, it becomes necessary to deduplicate the security findings and achieve a common truth of security information. A unique list of security findings allows better overview of the project security status, and enables developers to apply same solution approach to multiple locations by reducing the overall number of security findings dealt with.

Our work attempts to deduplicate security findings from multiple security tools, for both static and dynamic analysis, using NLP semantic similarity techniques. As will be shown in subsequent chapters, we create a dataset of security findings by scanning a common software artifact with seven static analysis and two dynamic analysis tools. Below we outline these tools and their purposes.

2.2.2. Security Tools

Security tools can be classified as Static Analysis Security Tools (SAST) or Dynamic Analysis Security Tools (DAST). The SAST can be further divided into Static Code Analysis Tools (SCAT), Third-Party Vulnerability Scanning Tools (TVST) and Secret Scanning Tools (SST) [106].

Anchore

Anchore[3] is a TVST that allows quick generation of a list of known vulnerabilities from a container image or project directory. It can scan operating system and programming-language specific packages, and shows scan results in terms of vulnerability sources, e.g., package name. It provides reports in JSON format, which typically include the vulnerability's package name/path, severity, package version that fixes it, and data from the National Vulnerability Database (NVD).

Dependency Check

The OWASP Dependency Check[4] tool attempts to detect publicly disclosed vulnerabilities in a project's dependencies by identifying a Common Platform Enumeration (CPE) [5] identifier for each dependency and linking it with its associated CVE-IDs. It updates automatically using the NVD Data Feeds hosted by National Institute of Standards and Technology, and generates a JSON format report.

[3]https://anchore.com/opensource/
[4]https://owasp.org/www-project-dependency-check/
[5]https://nvd.nist.gov/products/cpe

Trivy

Trivy[6] is a TVST and hardening check tool that comprises of multiple scanners. It scans for operating system packages, software dependencies, known vulnerabilities (CVEs), infrastructure-as-code misconfigurations, and sensitive information and secrets. It is able to scan container images, filesystem, remote Git repository, and Kubernetes cluster or resource. It produces the report in JSON file containing vulnerability information and vulnerable locations in the scanned artifact.

HorusSec

HorusSec[7] is SCAT and SST that identifies security flaws and key leaks in the files and Git history of a project. It is used during development, and supports analysis of C#, Java, Kotlin, Python, Ruby, Golang, Terraform, Javascript, Typescript, Kubernetes, PHP, C, HTML, JSON, Dart, Elixir, Shell, Nginx, Swift. It outputs a JSON format report that includes the vulnerable code instances and their files, as well as vulnerability details like identifier, details, and severity.

Semgrep

Semgrep[8] is SCAT and SST that scans the codebase locally or in the build environment using simple rules, e.g., a *no-print-python-prod* rule looks for any *print(...)* statements in production Python code and generates security findings if found. It contains two thousand community-built rules and allows users to define their own rules as well. Currently, it supports more than twenty programming languages. It outputs a JSON format report that includes locations and details of the rules found violated.

CodeQL

CodeQL[9] is a SCAT that discovers vulnerabilities across codebase using a semantic code-analysis engine. It supports writing queries to find all variants of a vulnerability, allowing the user to eradicate it from all instances. It outputs a JSON format report that contains locations of queried vulnerabilities with other context information.

Gitleaks

Gitleaks[10] is an SST that detects and helps prevent presence of hardcoded secrets like passwords, API keys, and tokens in git repos. It outputs a JSON format report that shows instances and locations of sensitive information in code.

[6] https://github.com/aquasecurity/trivy
[7] https://horusec.io/site/
[8] https://semgrep.dev/
[9] https://codeql.github.com/
[10] https://github.com/zricethezav/gitleaks

ZAP

OWASP Zed Attack Proxy (ZAP)[11] is a DAST used for penetration testing of web applications. It serves as a "man-in-the-middle proxy" by standing between the tester's browser and the web application, and intercepts and inspects messages exchanged between them. It produces JSON format report that includes discovered vulnerabilities, their instances of occurrence, as well as their possible fixes.

Arachni

Arachni[12] is a DAST for evaluating the security of web applications. It learns from a web application's behavior during the scan process, and performs meta-analysis to assess the trustworthiness of results and intelligently identify false positives. It detects changes caused while travelling through the paths of a web application's cyclomatic complexity, and detects the attack/input vectors that would otherwise be undetectable by non-humans. It can also inspect client-side code, and supports JavaScript, HTML5, DOM manipulation and AJAX.

2.2.3. Deduplication strategies

Since security tool reports are semi-structured and contain multiple pieces of information e.g., description, location, solution for each security finding or vulnerability, there exist multiple ways for a group of findings to be considered duplicates or unique. In Table 2.3, we present from the cybersecurity professionals' perspective of identifying security finding duplicates, possible strategies for deduplication of security findings.

2.2.4. Cybersecurity concepts

Below we brief some important cybersecurity concepts relevant to our work and used in different parts of this thesis.

National Vulnerability Database

National Vulnerability Database (NVD) is a data repository containing software vulnerabilities and configuration settings, that uses open standards to provide reliable and interoperable information about vulnerability impact metrics, technical assessment methods, and references to solutions and IT product identification data [107].

CVE

The CVE Program[13] identifies, defines, and catalogs publicly disclosed cybersecurity vulnerabilities as unique CVE Records, which are used by the IT and cybersecurity professionals

[11] https://www.zaproxy.org/
[12] https://github.com/Arachni/arachni
[13] https://cve.mitre.org/

Table 2.3.: Types of Deduplication Strategies

Type	Description	Mitigation Strategy	Advantage	Disadvantages
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

to ensure that they are discussing the same issue. As part of the CVE Program, partner organizations assign and publish CVE Records for newly discovered vulnerabilities to communicate consistent descriptions for them, in effort to prioritize and address the discovered vulnerabilities.

CWE

Common Weakness Enumeration (CWE)[14] is a list of common software and hardware weakness types that have security implications. Types of such weaknesses are flaws, faults, bugs, or other errors in software or hardware implementation, code, design, or architecture that if left unaddressed could result in systems, networks, or hardware being vulnerable to attack. The CWE List and associated classification taxonomy serve as a language that can be used to identify and describe these weaknesses.

Attack Vector

An attack vector is defined as the path or series of steps through which an attacker is able to gain access to a host [108]. This also includes vulnerabilities, since it may require several vulnerabilities to launch a successful attack. Examples of attack vectors include misconfiguration, kernel flaws, buffer overflow, insufficient input validation, symbolic links, race conditions, incorrect file or directory permissions and social engineering [109].

[14]https://cwe.mitre.org/

3. Methodology

In this section, we describe in detail our experimental design. Our research aims to explore the efficacy of NLP semantic similarity techniques to remove duplicate findings DevOps security tool reports. As described earlier, these security tool reports are machine-generated, semi-structured files containing entries referred to as findings. Each finding contains at least the information regarding detected security issue and its location, both of which vary in structure and syntax across different tools. Examples of location information include filenames, line numbers, instances of vulnerable code, and of vulnerability information include vulnerability title, details, unique identifier, reference URI.

Since we aim to process natural language (NL), the first step in our experimental design is to examine security tool reports individually and try to understand them with the perspective of a computer novice. In our case, this leads to us looking for the *title* or *description* of a finding, which are properties that describe the finding in NL. These properties appear with different names for different security tools. An example security finding from our SAST dataset is shown below:


```
{
  "VulnerabilityID": "NSWG-ECO-428",
  "PkgName": "base64url",
  "InstalledVersion": "0.0.6",
  "FixedVersion": ">=3.0.0",
  "Layer": {...},
  "SeveritySource": "nodejs-security-wg",
  "PrimaryURL": "https://hackerone.com/reports/321687",
  "Title": "Out-of-bounds Read",
  "Description": "`base64url` allocates uninitialized Buffers when number is
  ↪  passed in input on Node.js 4.x and below",
  "Severity": "HIGH",
  "References": [
    "https://github.com/brianloveswords/base64url/pull/25",
    "https://hackerone.com/reports/321687"
  ]
}
```


The security finding shown above is detected by Trivy and highlights the *problem* of *"trying to access buffers that do not exist"* in the listed version of package *base64url*. The finding also indicates the severity level and the location of vulnerability. Comparatively, a DAST security finding looks like:

```
{
    "name": "SQL Injection",
    "description": "Due to the requirement for dynamic content of today's web
     ↪  applications, many rely on a database backend to store data that will
     ↪  be called upon and processed by the web application (or other
     ↪  programs). Web applications retrieve data from the database by using
     ↪  Structured Query Language (SQL) queries. To meet demands of many
     ↪  developers, database servers (such as MSSQL, MySQL, Oracle etc.) have
     ↪  additional built-in functionality that can allow extensive control of
     ↪  the database and interaction with the host operating system itself. An
     ↪  SQL injection occurs when a value originating from the client's request
     ↪  is used within a SQL query without prior sanitisation. This could allow
     ↪  cyber-criminals to execute arbitrary SQL code and steal data or use the
     ↪  additional functionality of the database server to take control of more
     ↪  server components. The successful exploitation of a SQL injection can
     ↪  be devastating to an organisation and is one of the most commonly
     ↪  exploited web application vulnerabilities. This injection was detected
     ↪  as Arachni was able to cause the server to respond to the request with
     ↪  a database related error.",
    "references": {
      "UnixWiz": "http://unixwiz.net/techtips/sql-injection.html",
      "Wikipedia": "http://en.wikipedia.org/wiki/SQL_injection",
      "SecuriTeam":
       ↪  "http://www.securiteam.com/securityreviews/5DP0N1P76E.html",
      "OWASP": "https://www.owasp.org/index.php/SQL_Injection",
      "WASC": "http://projects.webappsec.org/w/page/13246963/SQL%20Injection",
      "W3 Schools": "http://www.w3schools.com/sql/sql_injection.asp"
    },
    "remedy_guidance": "The only proven method to prevent against SQL injection
     ↪  attacks while still maintaining full application functionality is to
     ↪  use parameterized queries (also known as prepared statements). When
     ↪  utilising this method of querying the database, any value supplied by
     ↪  the client will be handled as a string value rather than part of the
     ↪  SQL query. Additionally, when utilising parameterized queries, the
     ↪  database engine will automatically check to make sure the string being
     ↪  used matches that of the column. For example, the database engine will
     ↪  check that the user supplied input is an integer if the database column
     ↪  is configured to contain integers.",
    "severity": "high",
    "check": {
      "name": "SQL Injection",
```

```
      "description": "SQL injection check, uses known SQL DB errors to identify
       ↪  vulnerabilities.",
      "elements": ["link", "form", "cookie", "header", "link_template", "json",
       ↪  "xml"],
      "author": "Tasos 'Zapotek' Laskos <tasos.laskos@arachni-scanner.com>",
      "version": "0.2.3",
      "platforms": [...],
      "shortname": "sql_injection"
    },
    "vector": {...},
    "proof": "SQLITE_ERROR",
    "referring_page": {...},
    "platform_name": "sqlite",
    "platform_type": "db",
    "page": {...},
    "remarks": {},
    "trusted": true,
    "tags": ["sql", "injection", "regexp", "database", "error"],
    "cwe": 89,
    "signature": "SQLITE_ERROR",
    "cwe_url": "http://cwe.mitre.org/data/definitions/89.html",
    "digest": 1321544700,
    "response": {...},
    "request": {...}
  }
```


The above DAST finding is generated by the security tool Arachni and comprehensively describes the *problem* of *"arbitrary SQL code execution by a hacker in the running instance of application"*.

It can be seen that natural language explanations for SAST are very concise compared to DAST. This syntactic imbalance is also reflected in the semantics, whereby it is relatively easier for a cybersecurity novice to understand the descriptions in DAST due to their verbosity, while more domain expertise is required to understand a SAST finding upfront. As pointed out in Table 2.2, a reason for this is that SAST findings are simple and reproducible, and hence just a brief indication of a repetitive issue is enough for the cybersecurity professionals to help resolve the issue. Contrarily, DAST findings are instance specific and result from wholistic comprehensive checks, and are therefore verbose and include a *solution* to the vulnerability to help resolving it by the cybersecurity professionals. Owing to the overall difference in purposes of SAST and DAST, the imbalance in length of natural language texts of their findings, and that SAST findings contain just a *description* of the vulnerability as useful natural language feature whereas DAST findings also contain a *solution*, we deduplicate findings from SAST and DAST separately.

During this phase, we also explore extra features other than natural language ones that

can aid the deduplication process. We seek cybersecurity professionals' guidance regarding features that are completely representative of a finding. This discussion primarily leads to realizing the types of deduplication strategies as shown in Table 2.3. It also clarifies that *severity* of a finding has no role in identifying duplicate findings and is just used for prioritization. We also get to understand the importance of *CVE-ID*, an identifier that points to a unique vulnerability listed in the CVE catalog, over *CWE-ID*, an identifier that links to a security weakness leading to multiple vulnerabilities. Later in this section, we show how using *CVE-IDs* helps us increase deduplication accuracy for SAST by adding semantic content to the security finding descriptions.

Overall, our analysis of security tool reports leads us to perform *problem-based deduplication*, where we operate on NL descriptions that wholly describe the security flaw, contain the most semantic content, and are uniform in format across all security findings. This uniformity of format is contrary for *location-based deduplication*, where both an IP address and a file path represent locations, but are of different formats and contain no natural language content.

Our experimental design consists of creating individual, ground-truth datasets for SAST and DAST, which contain collections of similar findings from the perspective of cybersecurity professionals. We deduplicate findings from each dataset by extracting the respective findings without collections and pooling them together. We then apply different semantic similarity techniques to predict collections of similar findings, and quantitatively evaluate the results of a technique by comparing predicted collections with the ground-truth collections. Finally, we perform a qualitative evaluation by having a cybersecurity professional review the wrongly predicted clusters and select from list of possible challenges the most applicable challenges. Figure 3.1 summarizes the major elements in this process. In the following sections, we explain each of them in detail.

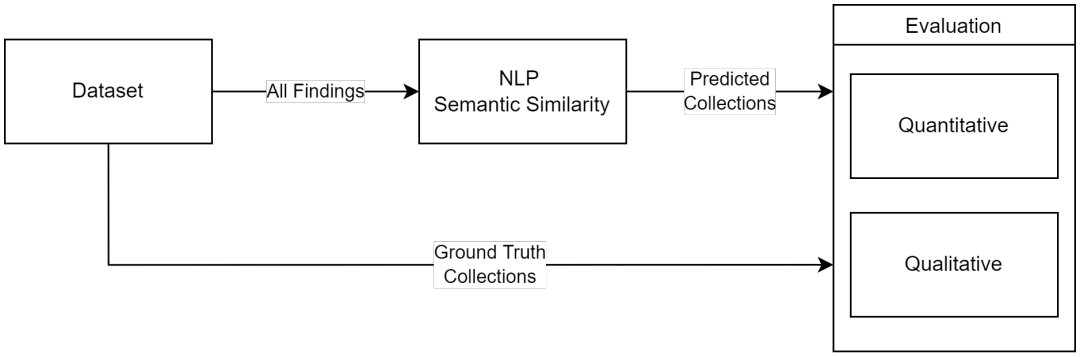


Figure 3.1.: Elements of deduplication process

3.1. Dataset

3.1.1. Scanning

To begin forming the dataset for our experiments, it is necessary to have a set of security tool reports from the SAST and DAST mentioned in subsection 2.2.2. To simulate the production environment in industry, the security tools should scan the same software's codebase and running instance, respectively. This will also result in semantically overlapping duplicates of security findings, which will help us effectively gauge the performance of NLP semantic similarity techniques. Our choice of such software is the OWASP JuiceShop[1], which is an insecure web application containing many vulnerabilities, including ones from the OWASP Top Ten list[2]. It is primarily used by security scanners as a "guinea pig" application to check how well they detect vulnerabilities. Its open-sourced nature also motivates its use since it will allow the reader to easily reproduce the results of our work. For our experiments, we will use SAST to analyze its source code, and DAST to analyze a running instance of its provided docker image. Our scanning pipeline for the dataset is shown in Figure 3.2.

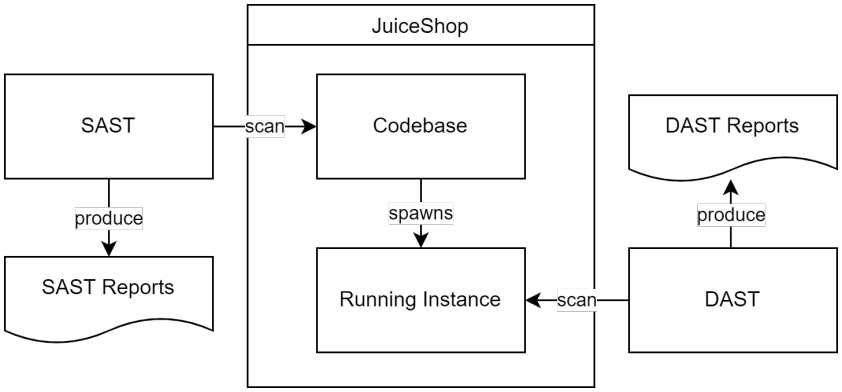


Figure 3.2.: Scanning pipeline

3.1.2. Labeling

Once we have collected all security tool reports, we need the expertise of cybersecurity professionals from industry to help us form collections of semantically similar findings from the numerous findings contained within each report. These collections, as shown in Figure 3.1, will serve as the ground truth for evaluating our predicted collections of similar findings.

However, since there are a total of 1351 SAST findings in 7 report files, each finding being a *JSON* object, it is cumbersome for the cybersecurity professionals to shift back and forth between files to look for similar findings. Furthermore, different tool reports have different names for the same property, e.g., security tool *Trivy* refers to *CVE ID* as *VulnerabilityID*, whereby security tool *Anchore* contains it as a sub-property of *nvd_data*. This challenges the

[1]https://owasp.org/www-project-juice-shop/
[2]https://owasp.org/www-project-top-ten/

cybersecurity professionals to memorize the names of relevant properties for each of seven tools while looking for duplicate findings, especially when they need to consider multiple properties. For a total 138 problems identified in 1351 security findings in our case, it is impractical for an expert with limited time to form a dataset with the given limitations. Therefore, we propose a tool, *SeFiLa*, as an additional research outcome of our work that helps overcome these essential challenges and allows cybersecurity professionals to label the ground truth dataset of security findings.

The design and functioning of the tool are described in the next chapter. Here we define the class diagram of the `Dataset`, which is essential to understanding how we apply the NLP techniques to it.

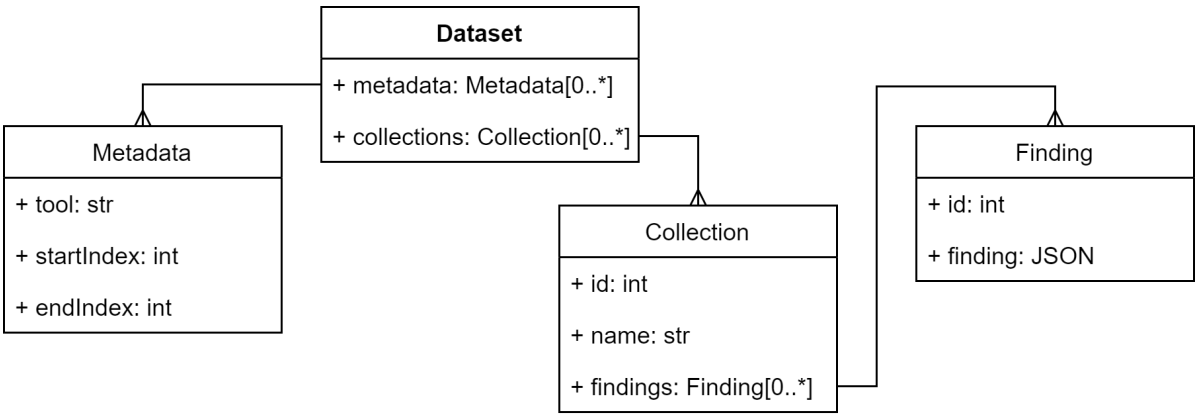


Figure 3.3.: Dataset class diagram

As can be seen in Figure 3.3, the `Dataset` downloaded from *SeFiLa* consists of *metadata* and *collections*. The collections contain groups of similar findings resulting from all generated security tool reports. Each collection and finding are assigned a unique identifier for easier manipulation during NLP application and evaluation. The metadata serves to relate findings with security tools, with `startIndex` and `endIndex` referring to finding identifiers.

3.2. Corpus formation

Having developed the ground-truth dataset, we now process it to extract information most meaningful for our NLP algorithms. The strategy involves deciding the feature names of security findings that wholly describe the problem being highlighted by the findings. Values of these features are concatenated to form a *corpus text* that is processed by the NLP technique. As pointed out in the start of this chapter, the most useful features are those that contain most semantic content, which are *description* in SAST reports, and *description* and *solution* in DAST reports. One challenge that needs to be overcome is that required features are not readily available in all findings e.g., security tool *Anchore* just contains a URI to the webpage describing the vulnerability. In such cases, we process the information contained in finding to get the required features e.g., for *Anchore* we scrape the *description* from webpage

of vulnerability. We also ensure that our selected features are present in all findings to avoid bias, particularly if we use more than one feature.

The next step is to extract our required corpus from the dataset. For this, we devise *Data Loaders* that process the dataset according to a certain *Corpus Format* and return us the *All Findings* corpus and *Ground Truth Collections* labels as illustrated in Figure 3.1. The *Corpus Format* allows us to specify the feature names to extract for each tool from the dataset, along with additional options like the *separator* to use between concatenated features and the option to stop corpus extraction if a specified feature is not present in the finding. Figure 3.4 shows the structure of `DataLoader` and `CorpusFormat` with their associations.

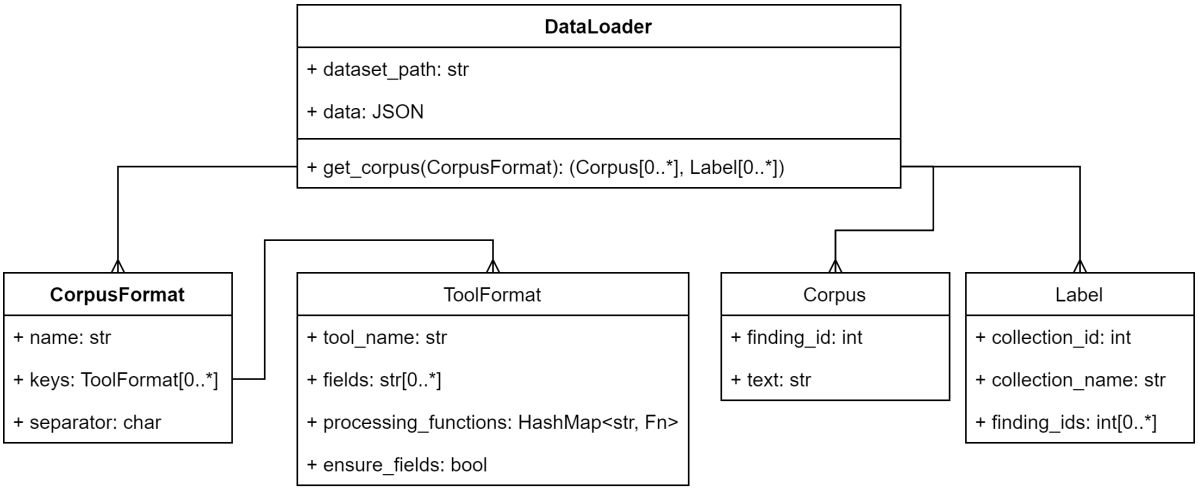


Figure 3.4.: Data Loader and Corpus Format class diagrams

The process starts with providing the dataset path to the `DataLoader`, which loads the file data into memory. The *Corpus* and *Labels* are then retrieved for our NLP technique by invoking the `get_corpus` method, which retrieves the values of specified features for each tool, passes the values through optional user-defined `processing_functions`, and concatenates the processed values with `separator` in between.

3.2.1. Feature-based semantics aggregation

The general approach to creating a corpus from security tool reports is to extract features from security findings and concatenate them together to form corpus texts. However, in SAST reports, semantic features like *description* are simple and of short lengths, thereby containing insufficient semantic content to completely describe the problem identified by security finding. Consequently, this results in their formed word embeddings being unable to fully capture the semantics of problem and affects the semantic similarity score.

To our advantage, duplicate findings resulting from different tools often provide differently worded descriptions for the same problem. Furthermore, features like *CVE-ID* uniquely identify vulnerabilities, and can be used to group findings that talk about the same vulnerabilities and problems, despite being generated by different security tools. We use these two

provisions to concatenate *descriptions* of security findings that have the same *CVE-ID* and assign the *concatenated descriptions* to each of the involved security findings. This aggregates the semantic content of corpus texts and allows NLP semantic similarity techniques to better grasp the context of the problem identified by security finding. Figure 3.5 summarizes the process of feature-based semantics aggregation.

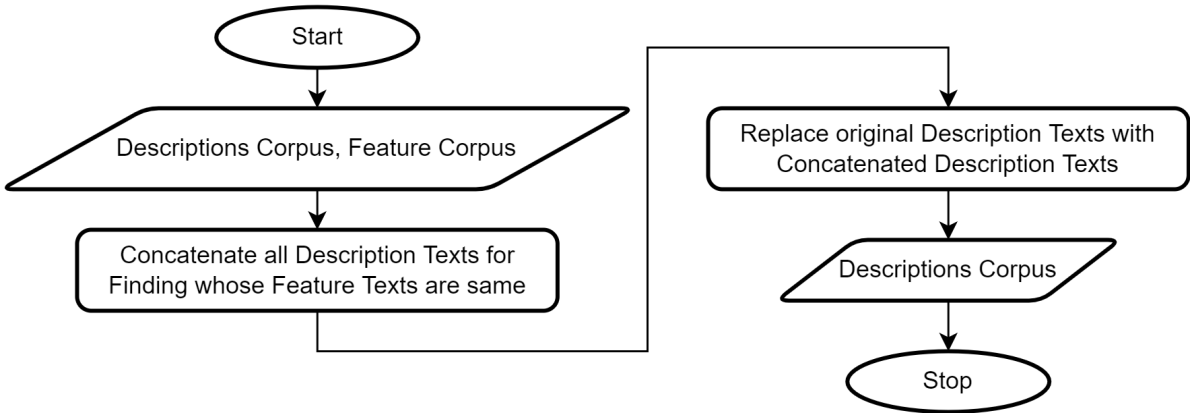


Figure 3.5.: Feature-based semantics aggregation

3.3. NLP application

3.3.1. Testbench

The corpus and labels retrieved from `DataLoader` are then used with different NLP semantic similarity techniques, which operate on the corpus to predict semantically similar findings and evaluates the results from predictions and labels. Different semantic similarity techniques e.g., knowledge-based, corpus-based, and transformer-based are implemented as separate classes, each derived from the `Technique` class. Our experimental setup allows running experiments with different parameters for the NLP techniques e.g., *similarity threshold* which configures the similarity score for considering two security findings as similar. A single run of predicting similar findings is referred to as a *Run Case*. Our experiments consist of executing different run cases that consist of a combination of techniques, corpus formats, data loaders and their configurations. The class diagram of our experimental setup is shown in Figure 3.6.

Each `RunCase` is executed as a separate process to improve execution speed and benefit from multiple compute cores. Once data has been loaded as described in section 3.2, the `apply` method of technique is invoked to get the predictions. Each prediction consists of a security finding and a sequence of semantically similar security findings from the corpus, implemented through their unique identifiers. These predictions are run through the `transitive_clustering` method, a process that sanitizes the predictions further as described in subsection 3.3.3. Finally, we evaluate the predictions through the `evaluate` method, which computes the values of each metric outlined in subsection 2.1.5 and returns the key-value pairing of metric name and its value.

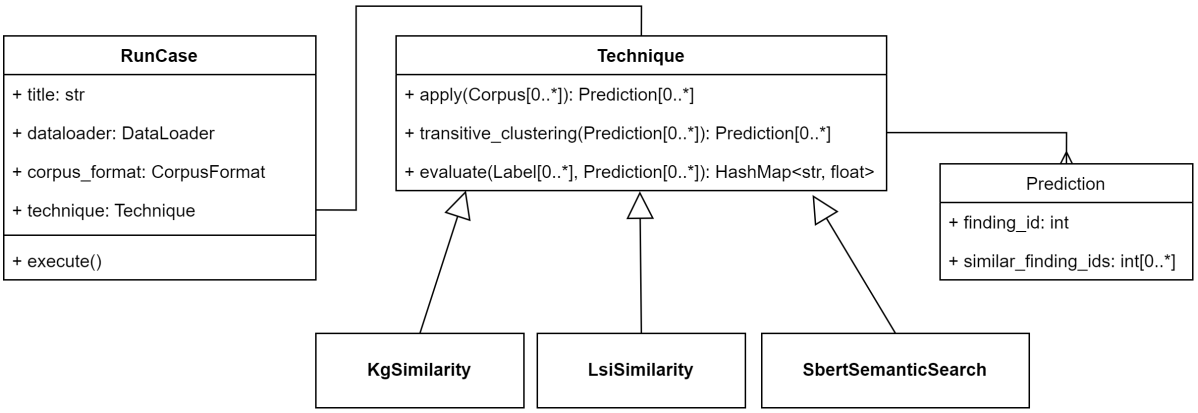


Figure 3.6.: Run Case and Technique class diagrams

In the following sections, we describe how each semantic similarity method that inherits from the `Technique` class works.

3.3.2. Techniques

Knowledge-graph (KG) similarity

As described in subsection 2.1.1, knowledge-based methods compute semantic similarity between two words based on an external knowledge source. Since our corpus contains texts, each of which consists of multiple words, we extend the usage of word similarity to sentence similarity using the *bag-of-words* (BOW) approach. The BOW approach aggregates similarity scores between each word in two sentences, where similarity scores of two words are computed through a knowledge-based method.

We use the popular NLTK[3] library to compute word similarities, from which we use *WordNet* as the knowledge source and *wup* as the knowledge-based method. For BOW approach, we write our own implementation. The computation of sentence similarity from corpus is illustrated in Figure 3.7, with the following abbreviations:

SS: Sentence Similarity, *WS*: Word Similarity, and *WC*: Word Count.

Latent Semantic Indexing (LSI) Similarity

LSI similarity, also known as Latent Semantic Analysis (LSA), is a corpus-based method that reduces dimensionality of the *word co-occurrence* vector space and allows computation of sentence similarity by application of *cosine similarity* to document vectors of the reduced latent space. The mathematical foundations have already been described in subsubsection 2.1.2. The LSI functionality for our experiments has been achieved using the Gensim[4] library.

[3]https://www.nltk.org/

[4]https://radimrehurek.com/gensim/index.html

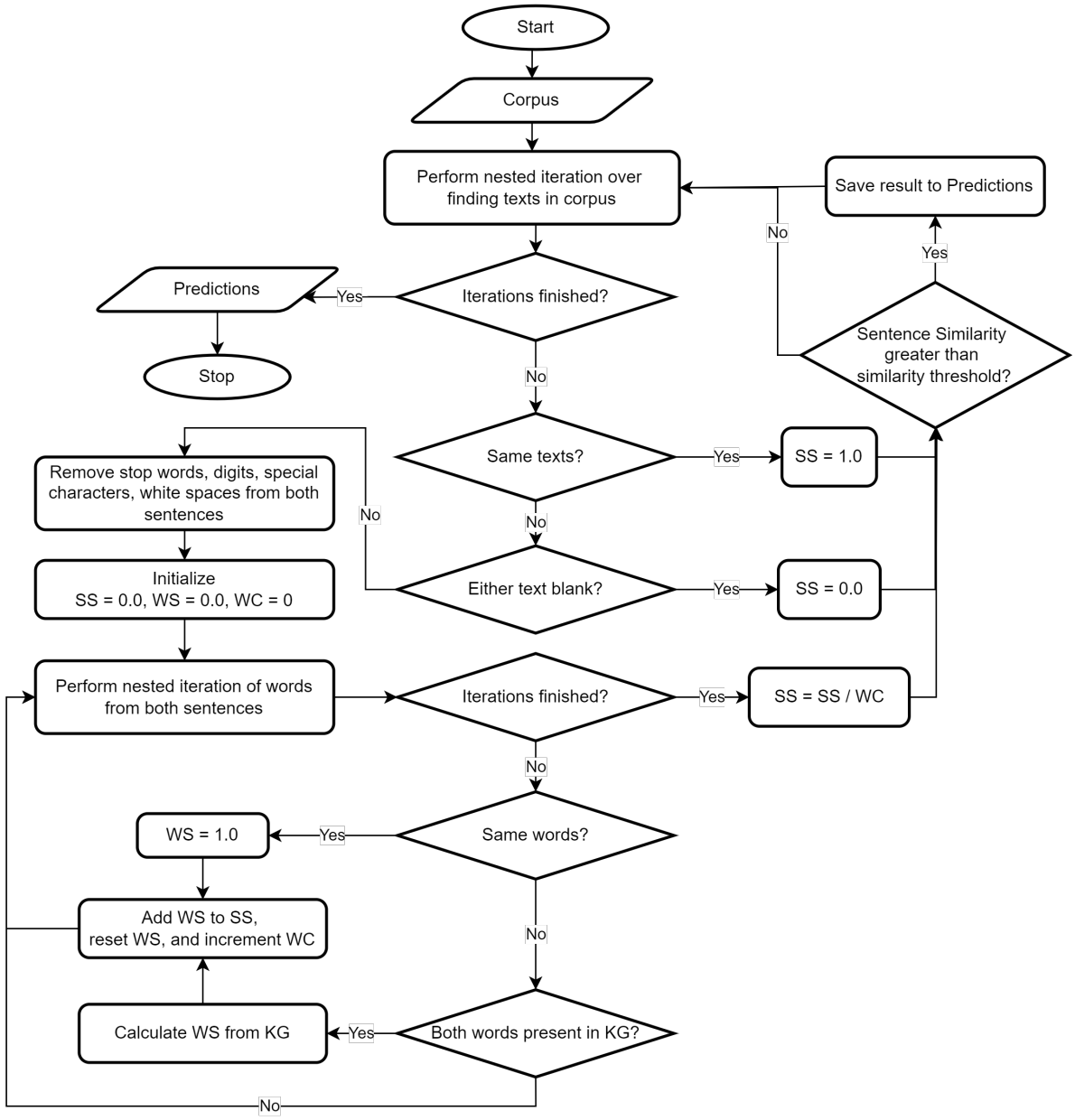


Figure 3.7.: Knowledge-graph Similarity

In Figure 3.8, we illustrate how the LSI technique is implemented in our experiments for computing semantic similarity of security findings corpus.

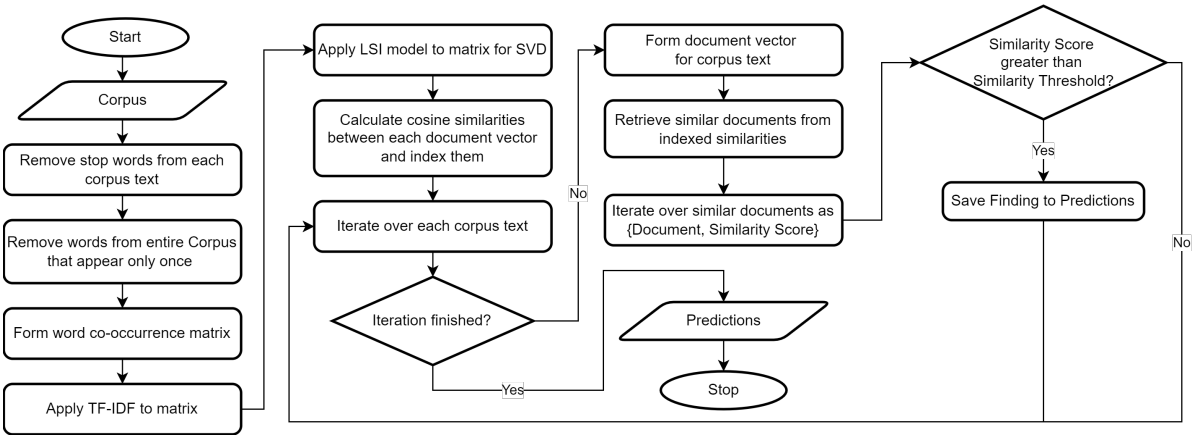


Figure 3.8.: LSI Similarity

Sentence BERT (SBERT) Semantic Search

As detailed in subsubsection 2.1.3, SBERT is a transformer-based method for computing sentence similarity, that is much faster than the original BERT model while maintaining its accuracy. It uses attention mechanisms to capture the contextual information in a sentence and generates word embeddings whose similarity can be conveniently computed with *cosine similarity*.

Semantic Search is a method of generating word embeddings by SBERT, which initially embeds all corpus texts into the vector space of one of its models. Each entry or *query* from the corpus texts is then embedded into the same vector space, and closest embeddings from the vector space are returned. These embeddings have a high semantic overlap with our query.

SBERT contains many pre-trained models[5] to choose from, based on whether the semantic search is *symmetric* or *asymmetric*. For symmetric semantic search, the query and the initially embedded corpus texts are of about the same length and have the same amount of content. Contrarily, asymmetric semantic search is when the query is a short text and the embedded corpus texts are paragraphs. In our case of SAST and DAST security findings, the semantic search we perform is symmetric using the *all-MiniLM-L6-v2* model. The application of semantic search to our corpus texts is illustrated in Figure 3.9.

3.3.3. Transitive clustering

As a result of applying semantic similarity techniques as explained in subsection 3.3.2, we get predictions that are a sequence of security finding identifiers semantically similar to a

[5]https://www.sbert.net/docs/pretrained_models.html#sentenceembeddingmodels

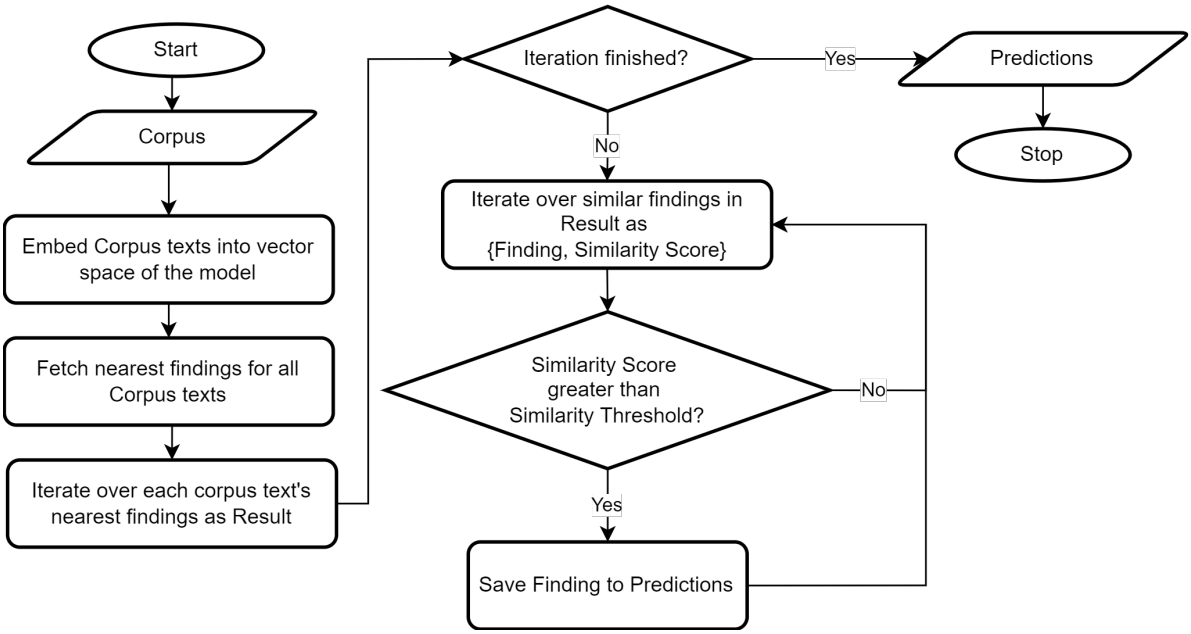


Figure 3.9.: SBERT Semantic Search

certain security finding. As also illustrated in flowcharts of different techniques, two findings are considered semantically similar if the calculated similarity score between them passes a certain similarity threshold. However, we encounter the following example predictions, with *Similar Findings* listed in descending order of semantic similarity score with the corresponding *Finding*:

$$Prediction\,1 = \{Finding : 1,\ Similar\ Findings : \{1,3,5,2\}\}$$
$$Prediction\,2 = \{Finding : 2,\ Similar\ Findings : \{2,4,5,3\}\}$$

We see that while *Finding* 1 is found similar to findings $\{1,3,5,2\}$, *Finding* 2 does not have *Finding* 1 in its set of similar findings. Likewise, if *Finding* 4 is similar to *Finding* 2, it should also be similar to *Finding* 1. One possible reason for this discrepancy is that semantic similarity score depends on vector space word embeddings from the perspective of *query* text. Therefore, even though there exists *Finding* 1 in the set of similar findings for *Finding* 2, it does not appear in the final set of similar findings due to having a semantic similarity score lower than the *similarity threshold*. Conversely, lowering the similarity threshold leads to a lot of false positive predictions. We, therefore, use transitive property to consider findings related even through *intermediate* findings, as semantically similar. As a result of applying this *transitive clustering* to predictions, both predictions mentioned above become:

$$Prediction\,1 = \{Finding : 1,\ Similar\ Findings : \{1,2,3,4,5\}\}$$
$$Prediction\,2 = \{Finding : 2,\ Similar\ Findings : \{1,2,3,4,5\}\}$$

In Figure 3.10, we illustrate how this transitive clustering is achieved for the predictions.

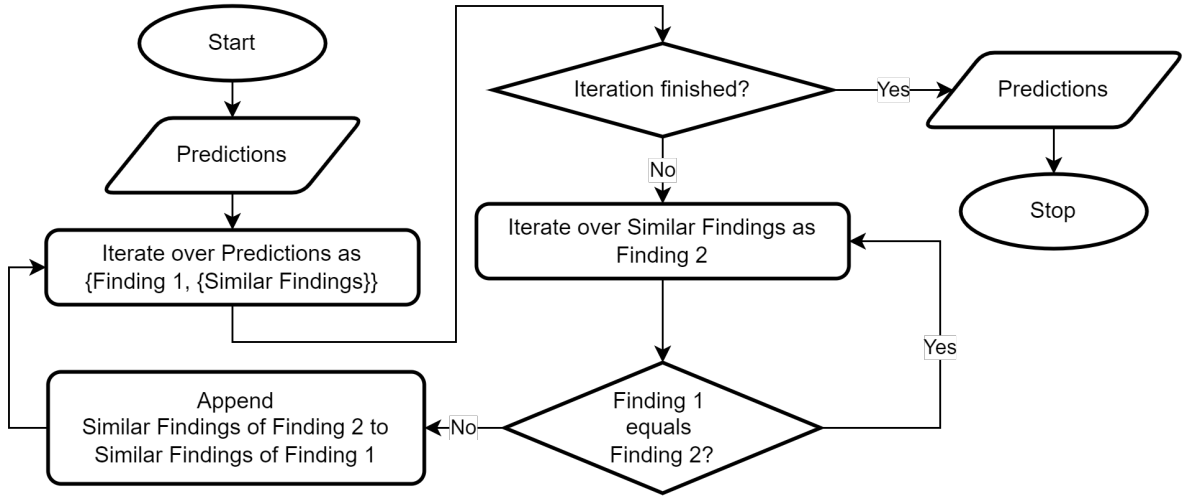


Figure 3.10.: Transitive clustering process

3.4. Evaluation

3.4.1. Quantitative evaluation

As part of invoking the `evaluate` method on a technique during execution of a `RunCase`, the predictions are processed and compared with the labels to compute the metrics listed in subsection 2.1.5. Figure 3.11 illustrates how quantitative evaluation is performed for our experiments. In addition to using the discussed pair-counting techniques, we also use the accuracy measures outlined below. The variables *Unique Predictions* and *Unique Labels* refer to sequence sets of finding identifiers from labels and predictions respectively.

We first compute the sequences of matched and unmatched predictions and labels:

$$Matched\ Predictions = Matched\ Labels = Unique\ Predictions \cap Unique\ Labels$$

$$Unmatched\ Predictions = Unique\ Predictions \ominus Matched\ Predictions$$

$$Unmatched\ Labels = Unique\ Labels \ominus Matched\ Labels$$

These sequences are used to compute three types of accuracies. The first is *Prediction Accuracy*, which gives us a measure of how many predictions are accurate from the total number of predictions. This allows us to know how well the NLP semantic similarity techniques work with security findings corpus. The second is *Label Accuracy*, which tells us the ratio of successfully identified labels. This helps realize the efficacy of deduplication when trying to achieve cybersecurity professionals' standards. Since we would like to achieve both aspects in our deduplication, we also compute the *Mean Accuracy*, which is an average of the former accuracy measures.

$$Prediction\ Accuracy = \frac{|Matched\ Predictions|}{|Unique\ Predictions|}$$

$$Label\ Accuracy = \frac{|Matched\ Labels|}{|Unique\ Labels|}$$

$$Total\ Accuracy = \frac{Prediction\ Accuracy + Label\ Accuracy}{2}$$

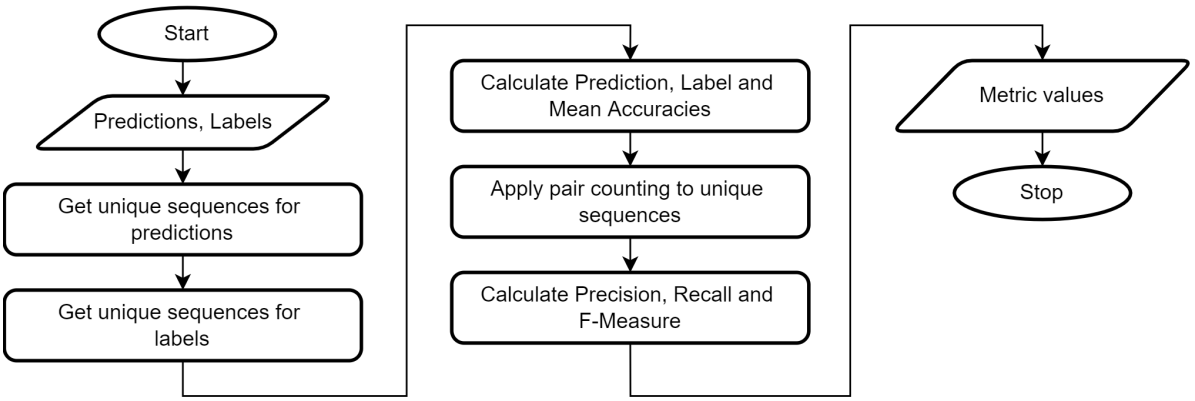


Figure 3.11.: Quantitative evaluation process

3.4.2. Qualitative evaluation

After quantitative evaluation, we let cybersecurity professionals observe which finding texts got incorrectly predicted. This is done by showing them the incorrect predictions, their corresponding ground truth labels, and original security finding structures of each. Consequently, they can define and select from the list of *challenges* and *reasons* ones that most explain the wrong prediction. The qualitative evaluation allows identification of challenges relating to deduplication using NLP. Figure 3.12 shows the process flow of qualitative evaluation. The process is performed through *SeFiLa*.

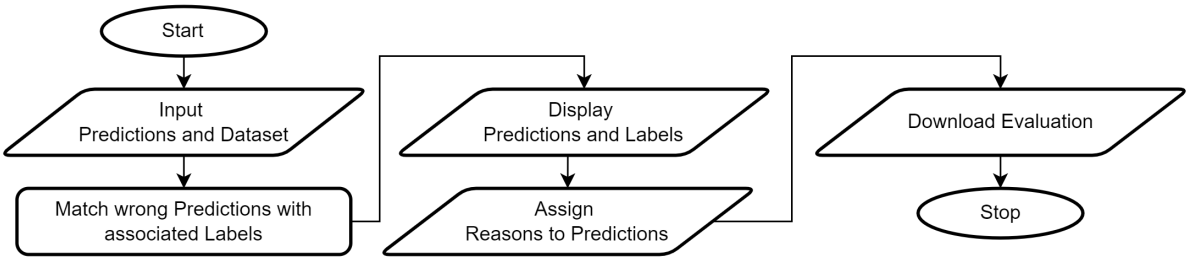


Figure 3.12.: Qualitative evaluation process

4. Results

In this chapter, we present the research outcomes and results of our experiments. These results arise from applying NLP semantic similarity techniques to our security findings datasets and indicate their efficacies in detecting semantically duplicate findings when applied to problem-describing features. Here, we just present an objective listing of results, and present our inferences from them in the next chapter.

One of the significant outcomes of our research is *SeFiLa*, a tool that solves the problem of labeling security finding datasets and enables qualitative evaluation of our semantic deduplication results. It has been designed for and in collaboration with cybersecurity professionals and serves to make future repetitions of our research seamless. In our experiments, it has also served as our interface with the cybersecurity professionals and has helped digitalize their interaction with our experiments.

The chapter proceeds with describing *SeFiLa*, with illustration and outline of its design and usage, respectively. This is followed by quantitative evaluation of semantic deduplication of both SAST and DAST datasets, where we present results from each applied semantic similarity technique as described in subsection 3.3.2 for different *similarity thresholds* and *corpora*. Finally, we present results of the qualitative evaluation, where cybersecurity professionals overview the incorrectly deduplicated findings and outline possible challenges and reasons for the respective occurrences.

4.1. Security Findings Labeler (SeFiLa)

As described in subsection 3.1.2, it is not feasible for a cybersecurity professional with limited time and memorization to shift between N security tool reports and keep in mind $N * M$ different names of M features scanned to detect similar findings. Furthermore, as listed in the previous chapter, SAST findings are brief but many, while DAST findings are verbose, which challenge cybersecurity professionals in terms of both quantity and quality when recognizing duplicate findings.

The complete answer to our second research question, *"How can we use Natural Language Processing (NLP) to deduplicate security tool findings?"*, involves the creation of datasets to begin with. To achieve it while overcoming the challenges mentioned above, we propose *SeFiLa*, a tool developed in collaboration with cybersecurity professionals that objectively caters to their needs when performing tasks related to our work. These tasks include *Labeling*, which is the formation of security findings dataset for our experiments, and *Evaluation*, which helps qualitatively analyze the results of NLP semantic similarity.

Another advantage of the tool is uniform interface of cybersecurity professionals with

our experiments. While quantitative evaluation is straightforward due to defined metrics, qualitative evaluation can include bias of the evaluator. *SeFiLa* solves this problem by providing a unified list of *challenges* and *reasons* formed by cybersecurity professionals while maintaining enough seclusion for the experts to provide their feedback on results. It also allows us to *quantify* the qualitative analyses, which would not have been possible given different feedback formats e.g., one-word, concise sentence, or descriptive paragraph feedbacks.

Overall, the structure and working of the tool prove to be necessary to achieve this and any future repetitions of our research. It is implemented as a single-page web application (SPA) developed using the React library. To design the tool exactly according to cybersecurity professionals' needs, it has been developed in an agile fashion while being hosted on the cloud. Its remote nature also allows backing up of data online and allows sharing of data between different cybersecurity professionals. Figure 4.1 shows the main page of *SeFiLa*.

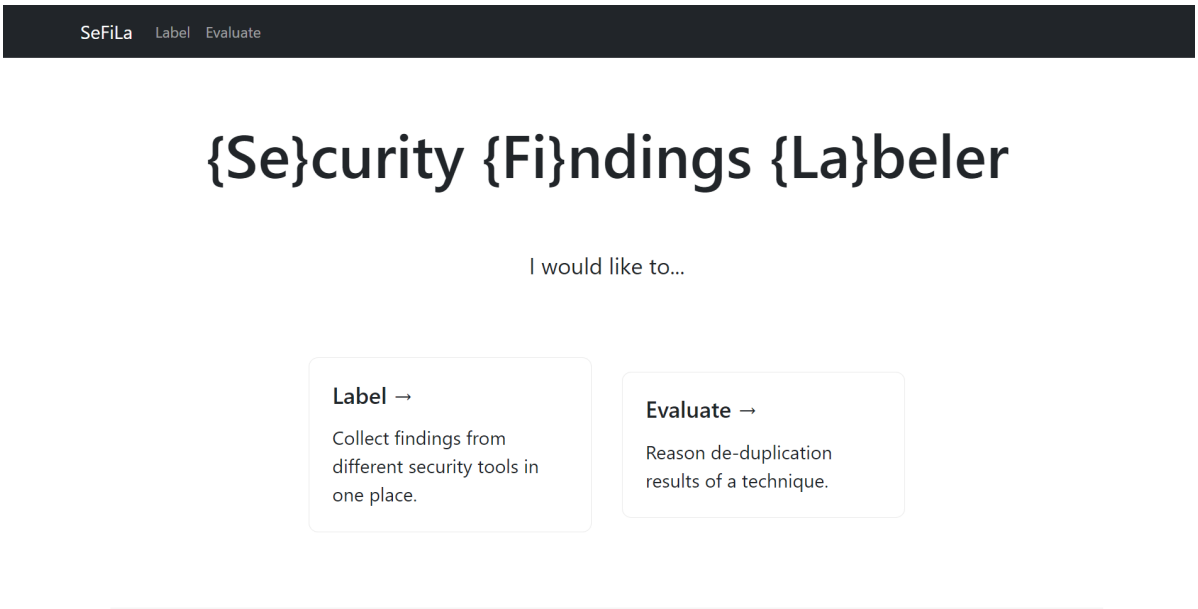


Figure 4.1.: Main page of *SeFiLa*

4.1.1. Label feature

To start labeling, cybersecurity professionals first consolidate all security tool reports in one place and get a quantitative overview of findings per tool for the dataset. This outcome is achieved through the *Generate Dataset* stage, that allows cybersecurity professionals to upload reports from each security tool and shows the number of findings from each report.

Because of thousands of security findings that need to be labeled, cybersecurity professionals prefer to continue labeling at a later stage. Furthermore, they also want to prevent any losses in their progress due to unexpected computer restarts or browser crashes. The *Generate*

Dataset stage allows them to restore progress from the cloud using an identifier provided during each labeling stage, or by simply uploading a previously generated dataset file. It also allows cybersecurity professionals to add more findings to an existing dataset by allowing them to upload more files while restoring from a cloud backup or a dataset file.

As pointed out in earlier chapters, some SAST reports do not provide natural language descriptions in their findings. Instead, they contain reference URIs linking to webpages describing the problem. However, cybersecurity professionals require natural language descriptions to interpret the problems identified by security findings. *SeFiLa* provides the option to *"Scrape data from URLs"* before starting the labeling process, which scrapes data for and appends an extra feature containing natural language description to each security finding. Furthermore, there are times when cybersecurity professionals have limited connection to the internet, because of which backing up of labeling progress to cloud might be hindered. To address this, *SeFiLa* allows complete offline usage by providing the option to skip cloud backup for a labelling session.

Figure 4.2 shows an overview of the *Generate Dataset* stage, where two SAST reports have already been parsed and one is pending upload. Once the cybersecurity professionals are satisfied with the uploaded reports, they can proceed to the next step of *Label Dataset* shown in Figure 4.3. Before this stage gets visible, incrementing integers are assigned as finding identifiers to each security finding in the dataset, and progress is shown for backing up of all findings to the cloud as well as for scraping of URLs if applicable.

As shown in Figure 4.3, the *Label Dataset* stage contains three main sections, namely *Current collection*, *All findings*, and *All collections*. The intended outcome of labeling process is to form *collections* of similar findings, each collection representing a different problem. These collections are named after the identified problem and contain security findings that are *problem-based semantic duplicates*. The *All findings* section shows security findings from all reports which have not yet been assigned to any collection, present in the order of upload of their respective security tool reports. Conversely, the *Current collection* section contains findings regarding the problem being identified, present in the order they were added to the collection. Both sections preview a single finding at a time, allowing cybersecurity professionals to focus on the contents of each finding as well as preventing any browser crashes that may occur due to simultaneous rendering of large findings on limited memory machines. Each previewed finding is titled *Finding X (Y)*, where *X* refers to the numeric identifier assigned to the finding during previous stage and *Y* refers to security tool the finding belongs to. This helps cybersecurity professionals develop an understanding the structures and useful features of each tool in its report right from the labeling stage. Findings contained in each section can be navigated using the section's *Next* and *Previous* buttons. To help cybersecurity professionals preview findings from the next security tool in line when navigating through collections, the *Jump Tool* buttons are also provided. These save time particularly when labeling SAST reports, since numerous findings are generated by each tool and the next similar finding for currently labelled problem might only exist in findings from the next tool.

The labelling process is straightforward. cybersecurity professionals navigate through

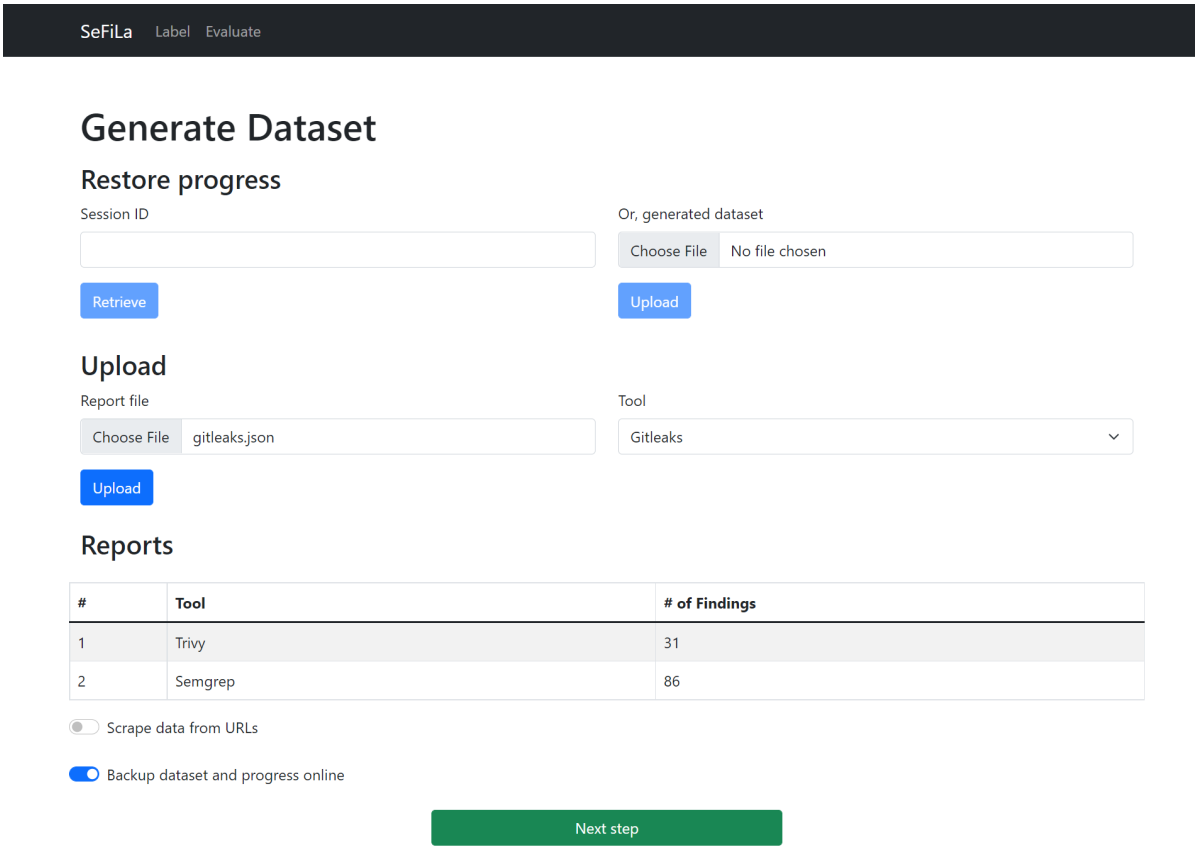


Figure 4.2.: *Generate Dataset* stage of *Label* feature

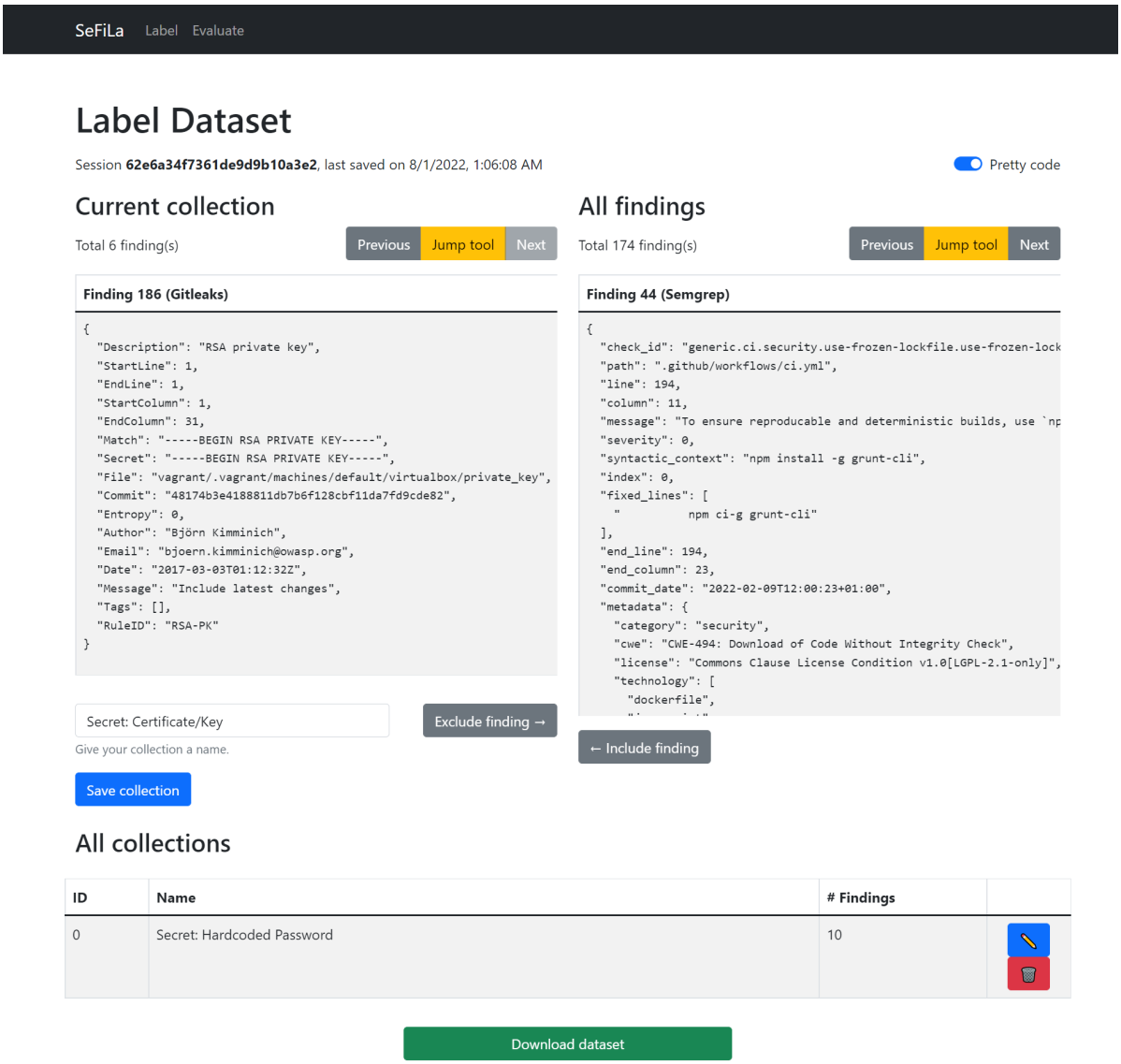


Figure 4.3.: Interface for *Label* feature

findings in the *All findings* section, and for any problem they want to start collecting findings for, they click the *Include finding* button to move the currently displayed finding from *All findings* section to the *Current collection* section. Conversely, a finding can be moved back to the *All findings* section by clicking the *Exclude finding* button. This way all findings identifying the same problem are moved to the same collection. Once the dataset has been fully traversed, or if the cybersecurity professionals prefer to start collecting findings for another problem, the findings in *Current collection* can be given a name and *Save collection* can be clicked to save the current collection to *All collections* section.

To change or preview findings in a collection, or to change a collection name, there is provision for editing a collection by clicking the pencil button in front of it. Editing a collection moves existing findings from the *Current collection* to *All findings* section, populates *Current collection* with findings from the edited collection, and removes the edited collection name from *All collections* section. The contents of the edited collection can then be modified and saved again. Likewise, if a collection needs to be re-labelled or is not needed altogether, there is possibility to delete it by clicking the trash can button located in front of it. This moves the findings of deleted collection to *All findings* section, and removes the name of deleted collection from *All collections* section.

Every time a finding is moved between *All findings* and *Current collection*, or when a collection is saved, deleted, or edited, the progress is backed up to the cloud. The timestamp of last saved progress, along with the session identifier that can be used to restore progress later, appear below the "Label Dataset" heading. To save the collections formed, cybersecurity professionals can click the *Download dataset* button to download a *JSON* file containing the formed collections and their contained findings. The dataset file can be readily used with our data loaders and experimental setup explained in the previous chapter.

4.1.2. Evaluate feature

Another important task we use *SeFiLa* for is the qualitative evaluation of our NLP semantic deduplication results. The goal of qualitative evaluation is to infer limitations of NLP when applied to security findings from the perspective of a cybersecurity professional. We achieve this with *SeFiLa* by enabling cybersecurity professionals to compare incorrectly predicted security finding collections with the expected collections from ground truth dataset, and requesting them to *reason* the challenges that lead to incorrect predictions if they try to predict the expected collection by looking at just the *corpus texts* of its findings. The underlying assumption is that if cybersecurity professionals can recognize from the corpus texts that are input to the NLP algorithms whether two findings are similar or not, the NLP algorithm should be able to as well. This qualitative evaluation is necessary because at time of labelling the data, cybersecurity professionals have a preview of all features of findings instead of just their corpus texts, which allows them to use their intuition and domain expertise while grouping semantically similar findings together. However, the NLP algorithms only work with the corpus texts of findings and do not have the contextual information or domain expertise to add weights to their predictions.

To start the qualitative evaluation, cybersecurity professionals are provided a file containing

the incorrect predictions and the corpus strings that were seen by the NLP techniques. They upload this along with the original dataset at the *Upload* stage of *Evaluation* in *SeFiLa*, as shown in Figure 4.4.

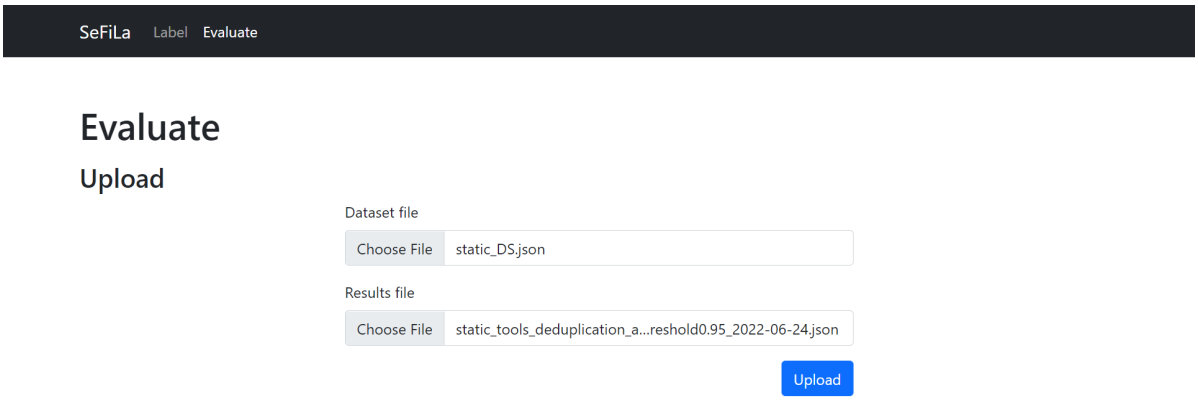


Figure 4.4.: *Upload* stage of *Evaluation* feature

Once uploaded, cybersecurity professionals see the *Evaluation* page as shown in **??**. The evaluation workflow is divided into two sections, namely *Predictions* and *Reasons*. The *Predictions* section shows an incorrectly predicted collection of finding identifiers, along with its related labels from the ground truth dataset. It also shows a listing of corpus text strings for each finding identifier mentioned in prediction and label(s). This format allows cybersecurity professionals to directly compare the text strings with each other. If they need more information regarding a finding, they are able to click the finding identifier of a specific corpus text to preview the original finding, as illustrated in Figure 4.6. The selected incorrect prediction is shown inside a *select* input that allows in-place searching of findings. This helps cybersecurity professionals conveniently shift between different incorrect predictions when evaluating collections for a certain problem.

The second part of evaluation workflow is assigning the reasons or challenges applicable to the selected incorrect prediction from the perspective of a cybersecurity professional reading just the listing of corpus texts. This can be done from the *Reasons* section by selecting as many options as needed. If an option needs to be added to the current list or an existing option needs to be edited, the yellow *Reasons* button can be clicked to open the *Reasons* dialog box as shown in Figure 4.7. The required changes get persisted to the cloud as they are made, and get visible on the *Evaluation* page as well. Once evaluated, cybersecurity professionals download their evaluation and send it to us.

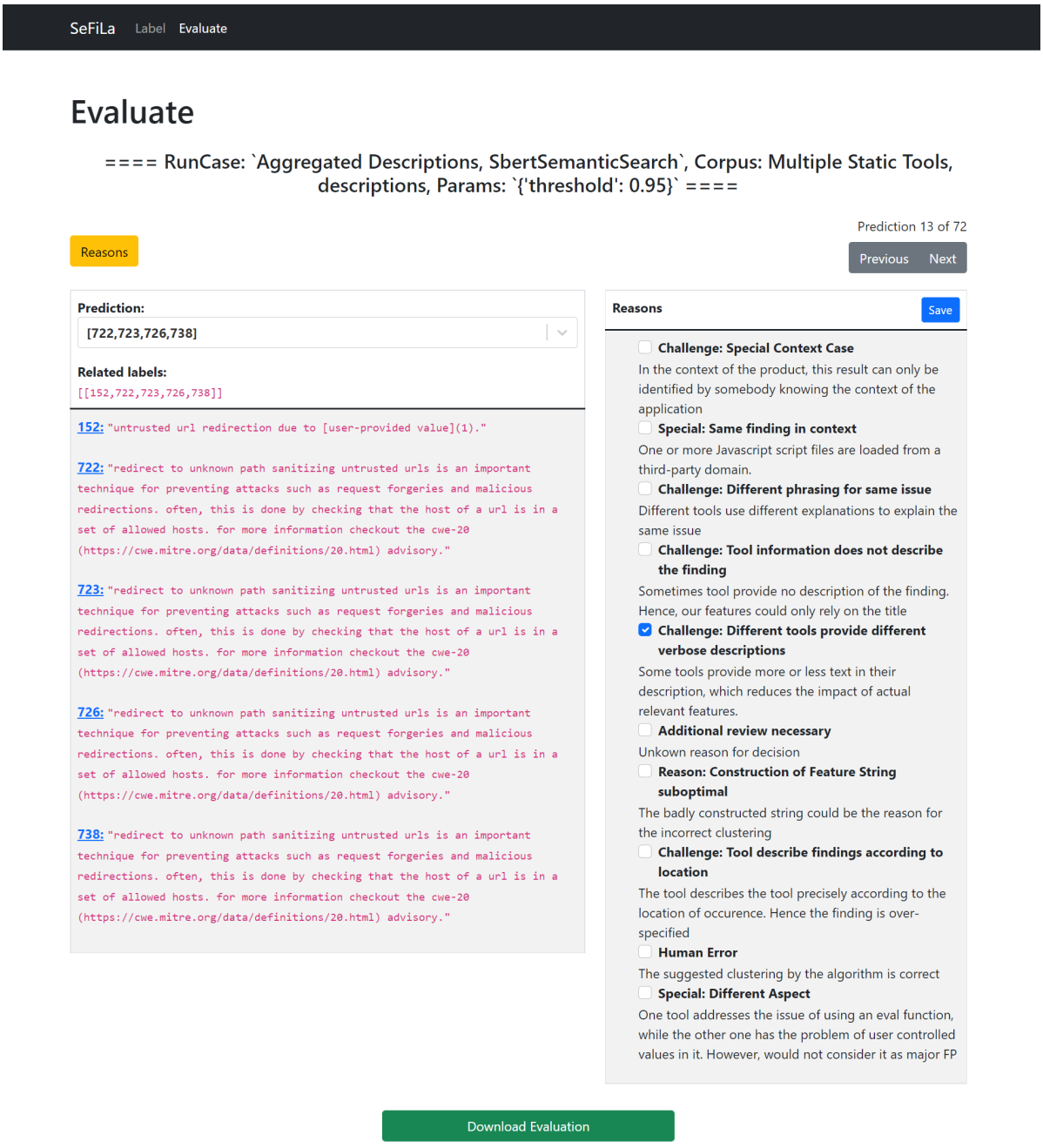


Figure 4.5.: Interface for *Evaluation* feature

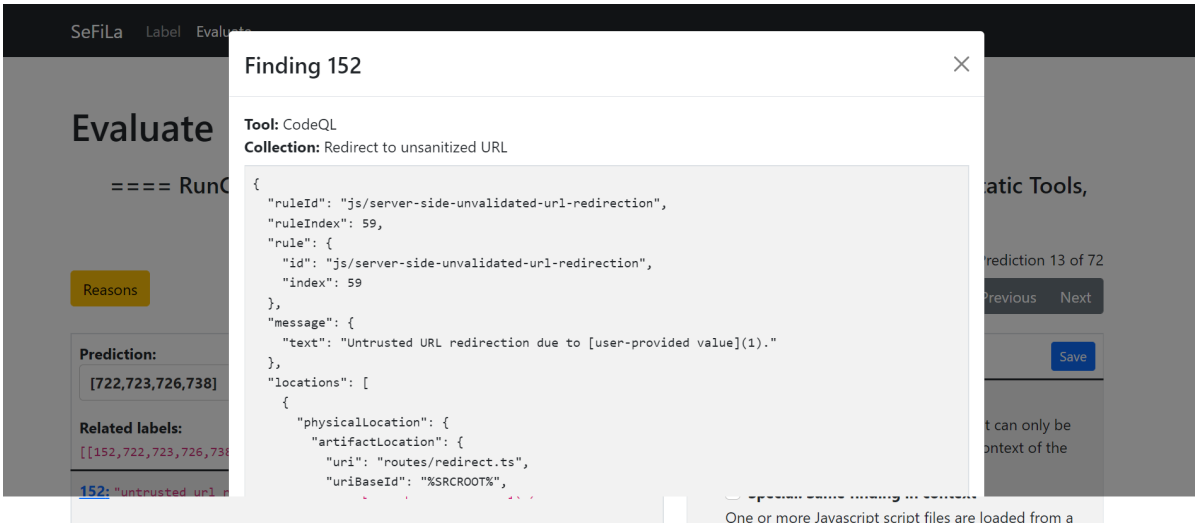


Figure 4.6.: Preview of the original finding

4.2. Dataset statistics

As a result of labeling with *SeFiLa*, cybersecurity professionals are able to provide us with a single ground-truth dataset for both SAST and DAST. The need for these datasets to be created individually has already been described in the previous chapter. Below, we compare aspects of both datasets that help understand the difference between them.

Table 4.1.: Comparison of SAST and DAST 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	*description*	*name*, *description*, *solution*
Average length of corpus text	302	471

As also highlighted in earlier chapters, SAST findings are greater in quantity, making up 97.4% of the total number of findings we encounter. While the number of security tools plays a part in this distribution, the *finding density* (average number of findings per security tool)

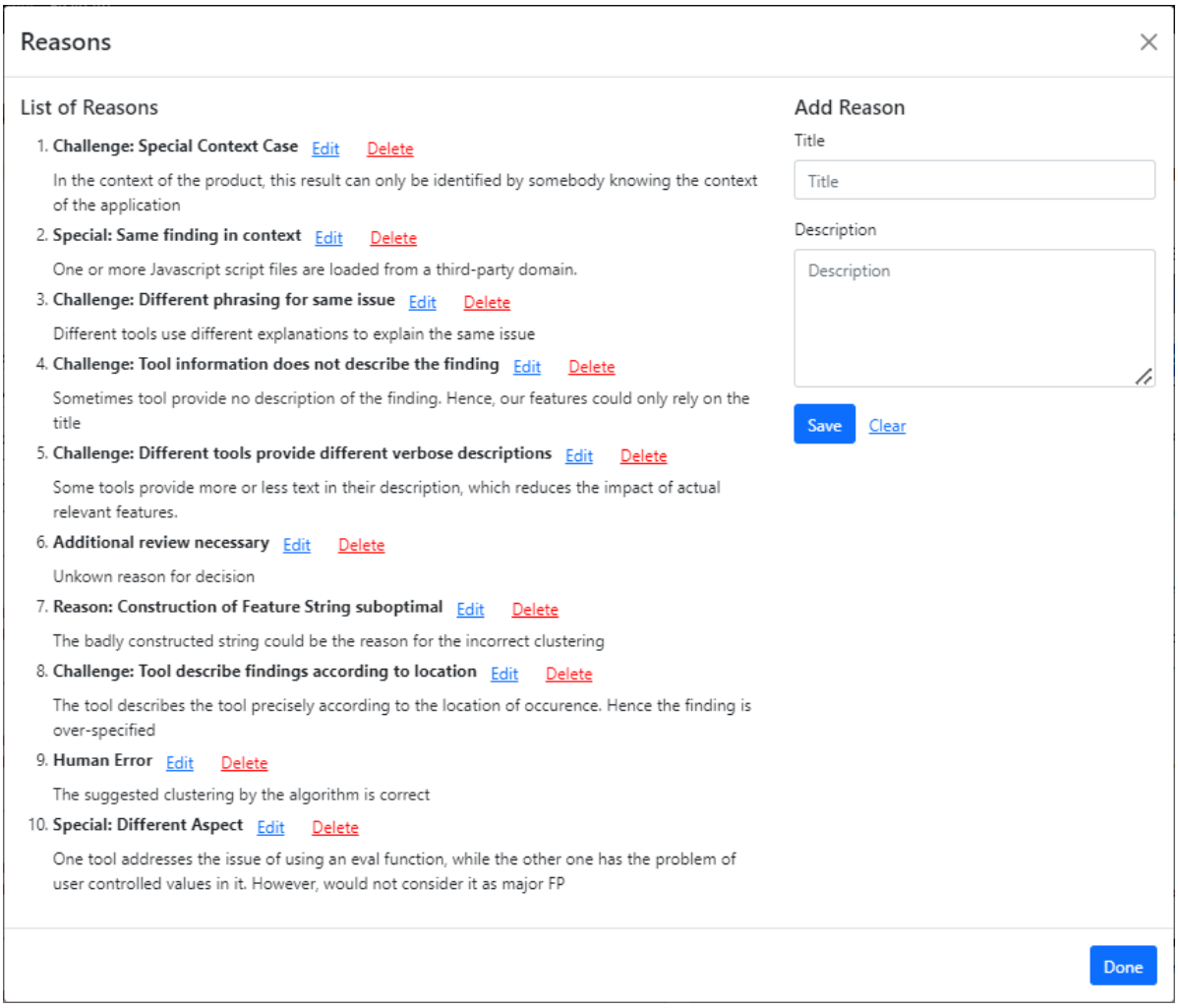


Figure 4.7.: Dialog box to add or edit *Reasons*

for SAST is still greater than 10× the finding density for DAST. There also exist 173 more collections for SAST than for DAST. However, the number of collections formed per total number of findings is still higher (67%) for DAST than it is for SAST (33%). Similarly, the number of findings per collection apparently shows to be higher for SAST than for DAST. However, when normalized with their respective total numbers of findings, the average number of findings per collection in DAST shows to be 16× to that of SAST. We can also see that the average lengths of corpus texts for DAST are greater than those of SAST by a factor greater than 1.5.

In the following subsections, we present some further statistics regarding each dataset.

4.2.1. SAST dataset

Our SAST dataset consists of reports from the seven security tools listed in subsection 2.2.2. As explained in previous chapters, different SAST tools perform different types of analysis. Figure 4.8 shows the number of findings produced by each tool.

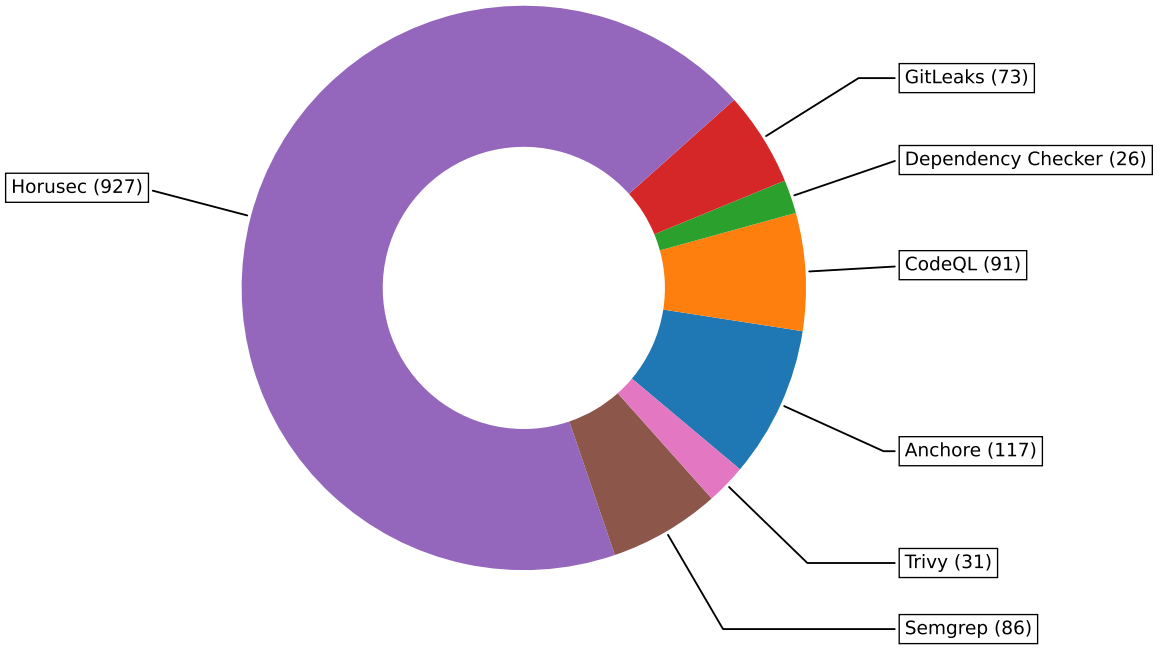


Figure 4.8.: Number of findings per SAST

The feature used for deduplication must be available in every tool report. We see that the *description* feature, which describes the overall problem being identified by the finding in natural language, fulfills this criterion and has semantic value for NLP algorithms. However, it occurs differently in findings from different tools. In Table 4.2, we show the property names and optional steps needed to extract the *description* feature from each tool.

The average lengths of corpus texts formed from this feature is shown in Figure 4.9. The *Plain* data represents feature used as-is from the security finding, whereas *Aggregated*

Table 4.2.: *Description* feature names for SAST

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*	-

data represents feature value whose semantic content has been aggregated as discussed in subsection 3.2.1. We can see that aggregating semantic content increases lengths of the corpus text strings on average by 2×.

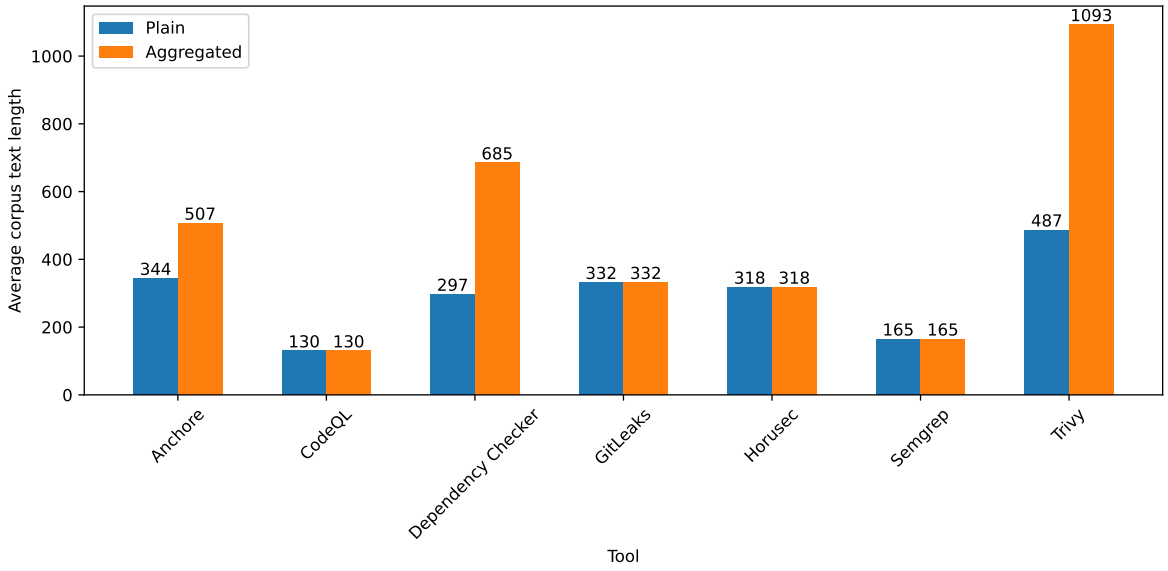


Figure 4.9.: Average lengths of SAST corpus texts

It is also helpful to know the quantity of findings present in ground-truth collections for interpreting the predicted collections. For SAST, the dataset contains 183 collections containing 1351 findings, whose quantitative distribution is shown in Figure 4.10.

4.2.2. DAST dataset

The DAST dataset is generated from two tools, *ZAP* and *Arachni*, as mentioned in subsection 2.2.2. They produce a total of 36 verbose findings, with their breakdown shown in Figure 4.11.

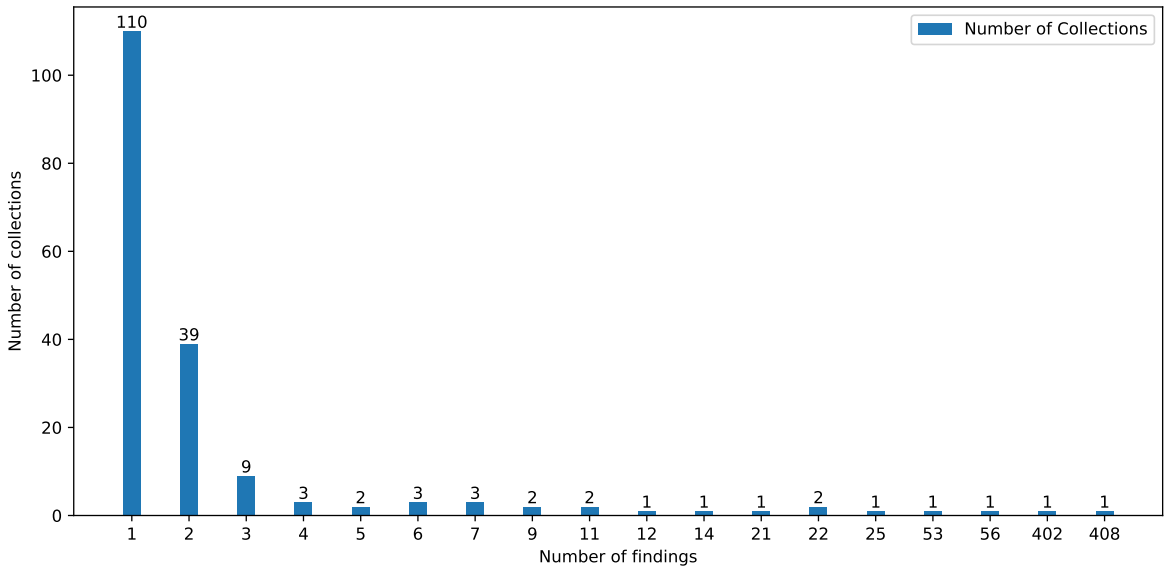


Figure 4.10.: Collections finding frequency for SAST

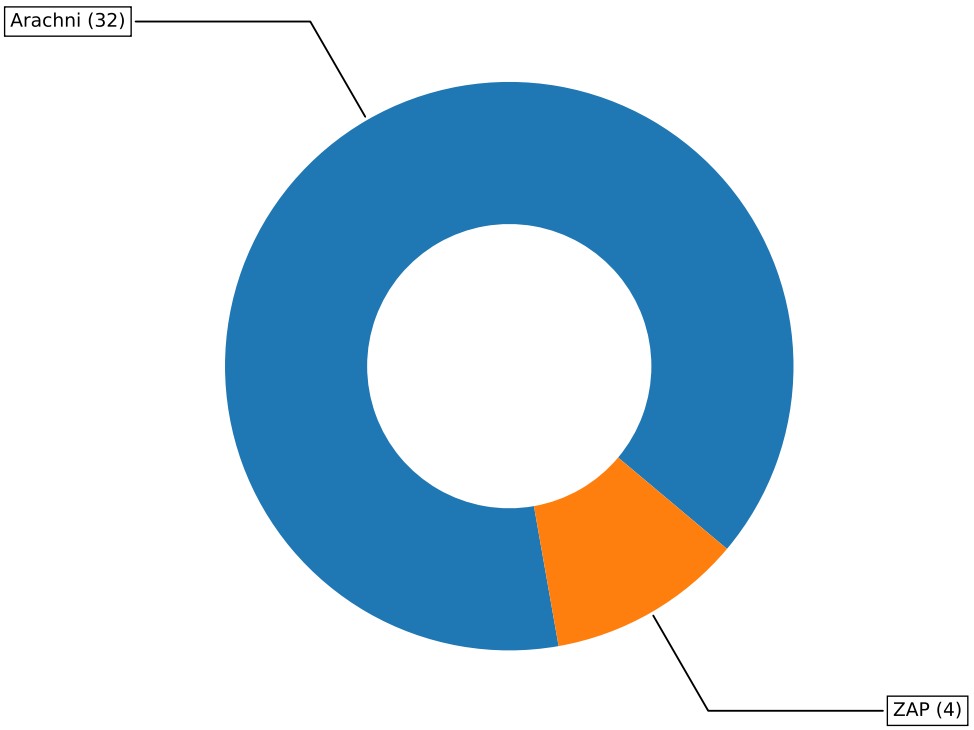


Figure 4.11.: Number of findings per DAST

Compared to SAST findings, we are able to extract more semantic features from DAST findings. The features *name*, *description*, and *solution* are available in reports from both DAST tools and can be extracted using properties listed in Table 4.3.

Table 4.3.: *Description* feature names for DAST

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

Values of these features are concatenated to form the corpus texts, which are used as inputs to NLP semantic similarity techniques. The average lengths of the formed corpus text strings for each DAST is shown in Figure 4.12. Furthermore, the DAST dataset contains a total of 10 collections, whose quantitative relation with their contained number of findings is shown in Figure 4.13.

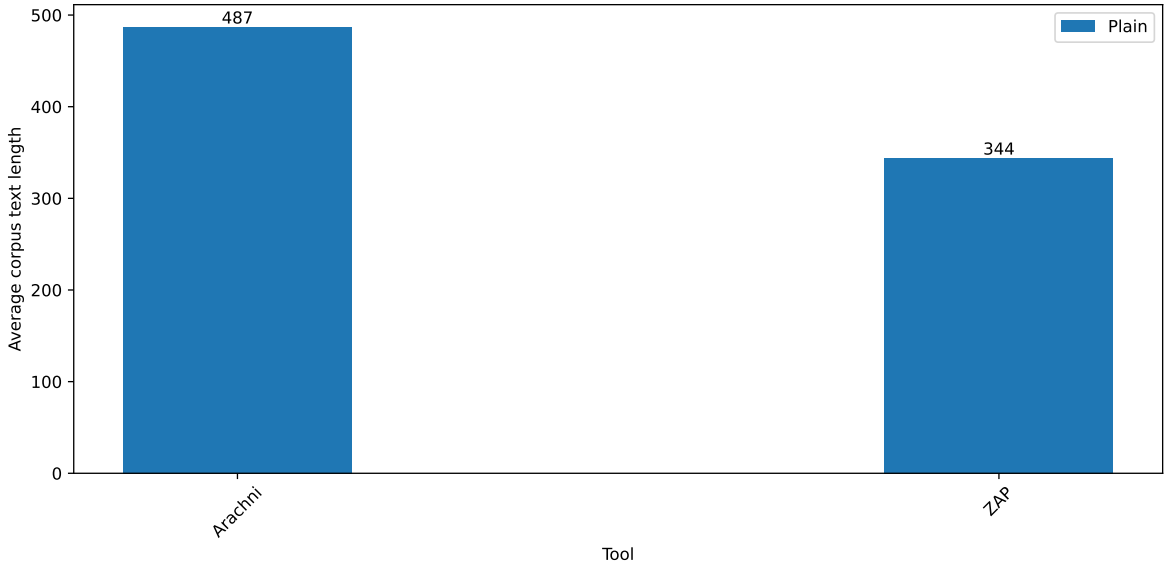


Figure 4.12.: Average lengths of DAST corpus texts

Having understood the specifications of both datasets, we now present the quantitative results of applying our chosen NLP semantic similarity techniques to deduplicate their findings.

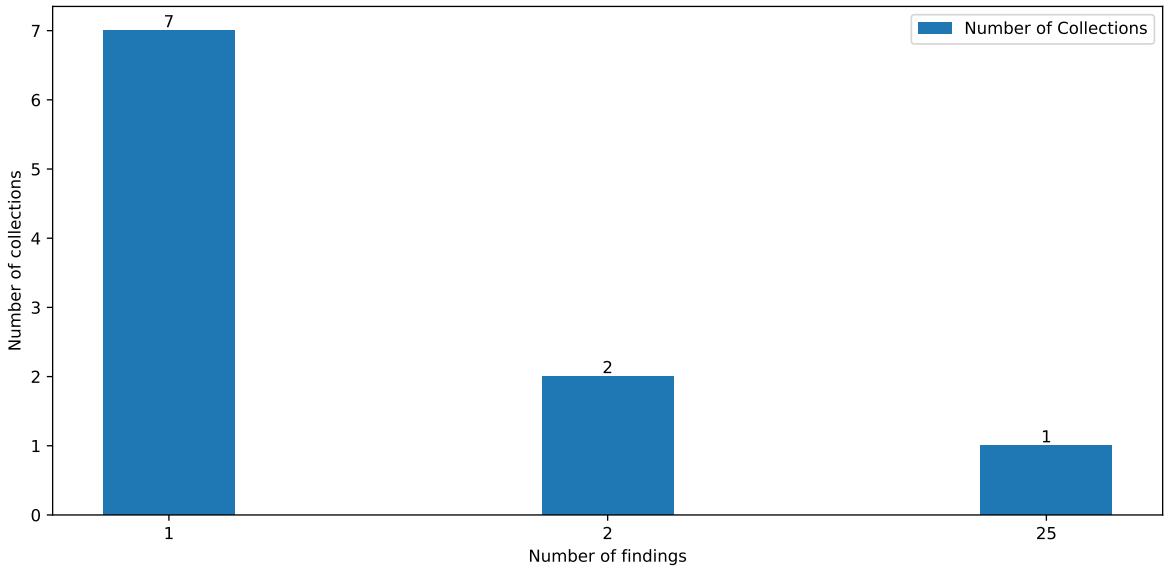


Figure 4.13.: Collections finding frequency for DAST

4.3. Quantitative evaluation

In the following subsections, we present the results of deduplicating the SAST and DAST datasets, by applying semantic similarity techniques to two corpora from each. The SAST corpora are named *SAST-Plain* and *SAST-Aggregated*, based on whether they contain just the *description* feature or the semantically aggregated version of *description*. The two DAST corpora are *Name-Description-Solution* (NDS) and *Description*, based on the features used when forming their corpus text strings.

4.3.1. Sentence-BERT (SBERT) Semantic Search-based deduplication

SAST

As shown in Figure 4.14, applying SBERT Semantic Search to *SAST-Plain* achieves a maximum 72.3% average accuracy at the similarity threshold of 0.9, which decreases if we increase the similarity threshold. However, applying the same to *SAST-Aggregated* leads to an increased average accuracy of almost 79% at a 0.9 similarity threshold, which increases further to 81.2% when the similarity threshold is increased to 0.95, as shown in Figure 4.15. It can also be seen from both plots that the increase in accuracies for both corpora follows the same trend with the increase in similarity thresholds. Another significant observation is that in both plots the prediction accuracy is higher than label accuracy for lower similarity thresholds, and is overtaken by label accuracy at higher similarity thresholds.

In contrary to the above observation, the f-measure, precision and recall measures for both corpora exhibit same trends and values as shown in Figure 4.16 and Figure 4.17. This shows no significant gain when applying SBERT Semantic Search to the *SAST-Aggregated*

corpus over *SAST-Plain* corpus. Furthermore, the gain in f-measure occurs steeply during a specific interval across the similarity threshold axis, whereas the gain in accuracy happens continuously as we increase the similarity thresholds.

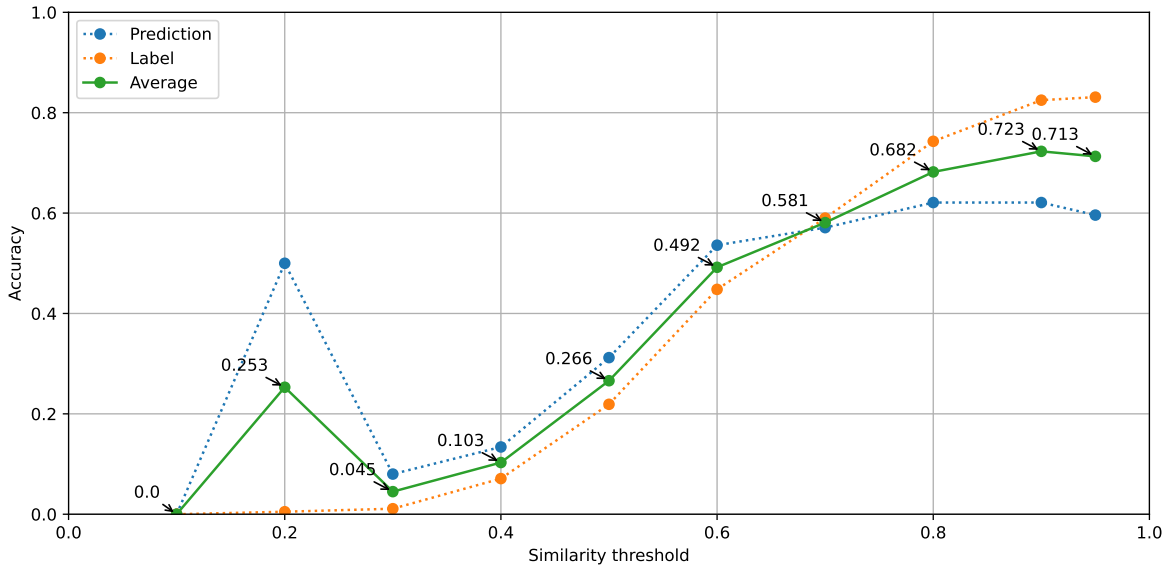


Figure 4.14.: SBERT Semantic Search accuracies for *SAST-Plain* corpus

DAST

The accuracies of results when applying SBERT Semantic Search to DAST copora are shown in Figure 4.18 and Figure 4.19 respectively. It can be seen for both corpora that a similarity threshold of at least 0.4 is needed to get a non-zero accuracy. At a similarity threshold of 0.5, both corpora yield their first non-zero accuracies, with results from the *DAST-NDS* corpus being 84.4% accurate (average) as compared to the 63.3% accurate results from *DAST-Description* corpus. For similarity thresholds greater than 0.5, both corpora exhibit the maximum achieved accuracies of 85.9%. However, at similarity threshold of 0.95, the *DAST-NDS* corpus yields an outlier lower average accuracy value of 73.3%. It can be observed that like the trend in accuracies from SAST corpora, the DAST corpora also yield a higher prediction accuracy at lower similarity thresholds, succeeded by label accuracy at higher similarity thresholds.

The results of f-measure, precision and recall metrics for both DAST corpora are shown in Figure 4.20 and Figure 4.21. It can be seen in both plots that until a similarity threshold of 0.5, the f-measure value is decided by the value of recall, while the precision maintains its highest value of 1.0. From the similarity threshold of 0.5 and onwards, values of all three metrics stay the same for both corpora. Like their counterparts from SAST corpora, these values indicate no significant difference when varying DAST corpora for deduplication through SBERT Semantic Search.

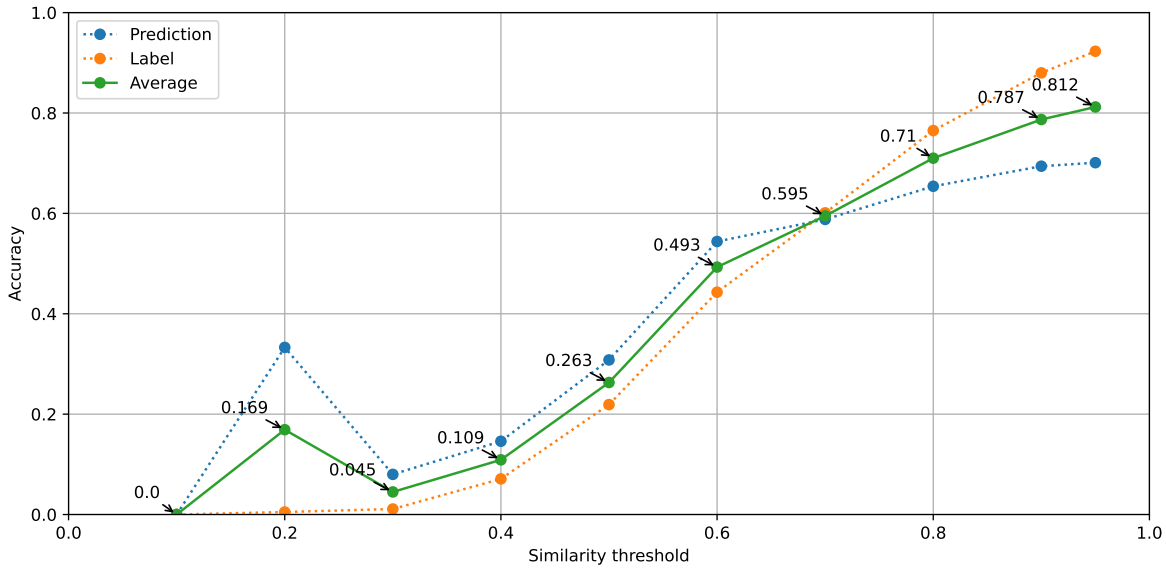


Figure 4.15.: SBERT Semantic Search accuracies for *SAST-Aggregated* corpus

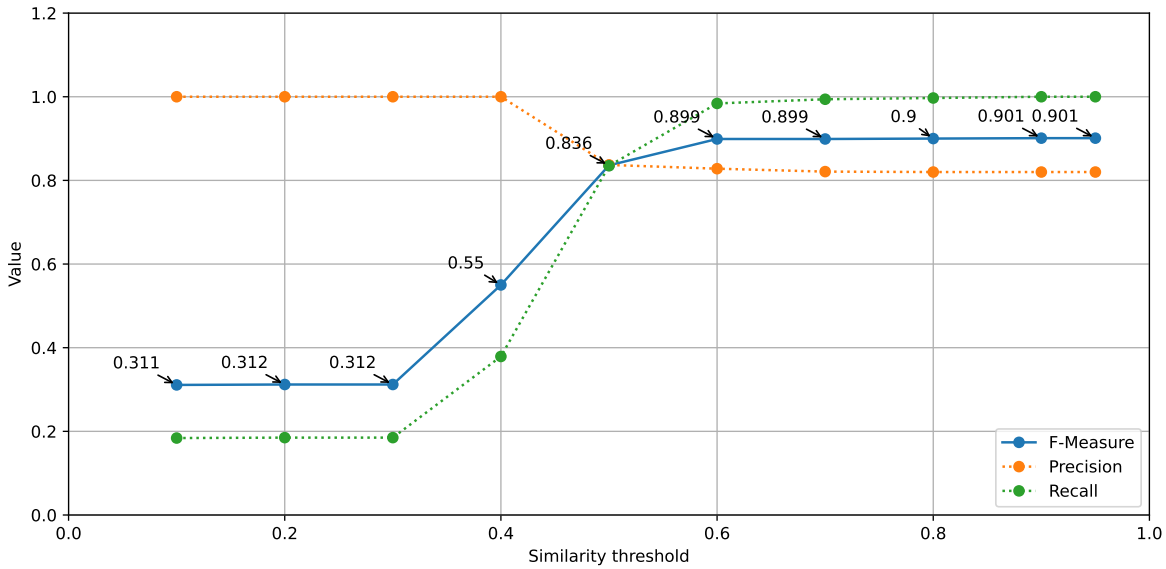


Figure 4.16.: SBERT Semantic Search metrics for *SAST-Plain* corpus

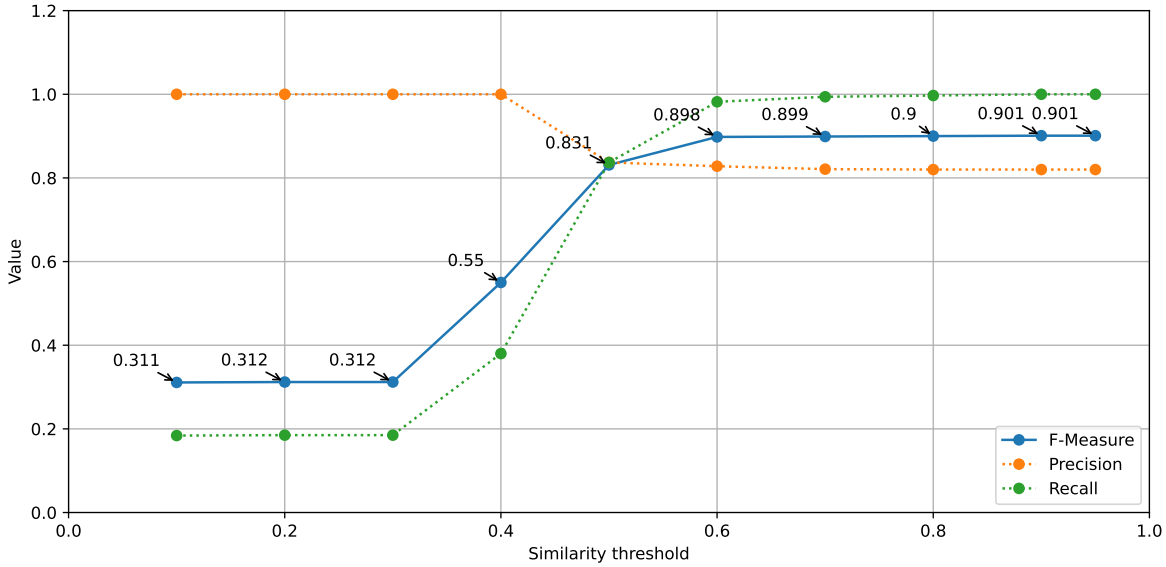


Figure 4.17.: SBERT Semantic Search metrics for *SAST-Aggregated* corpus

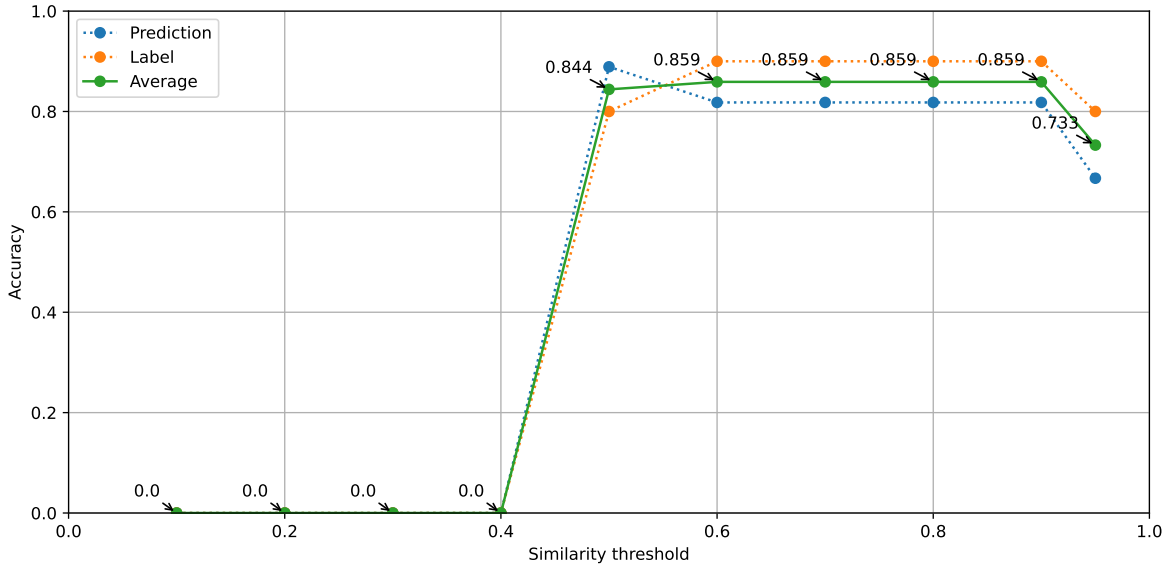


Figure 4.18.: SBERT Semantic Search accuracies for *DAST-NDS* corpus

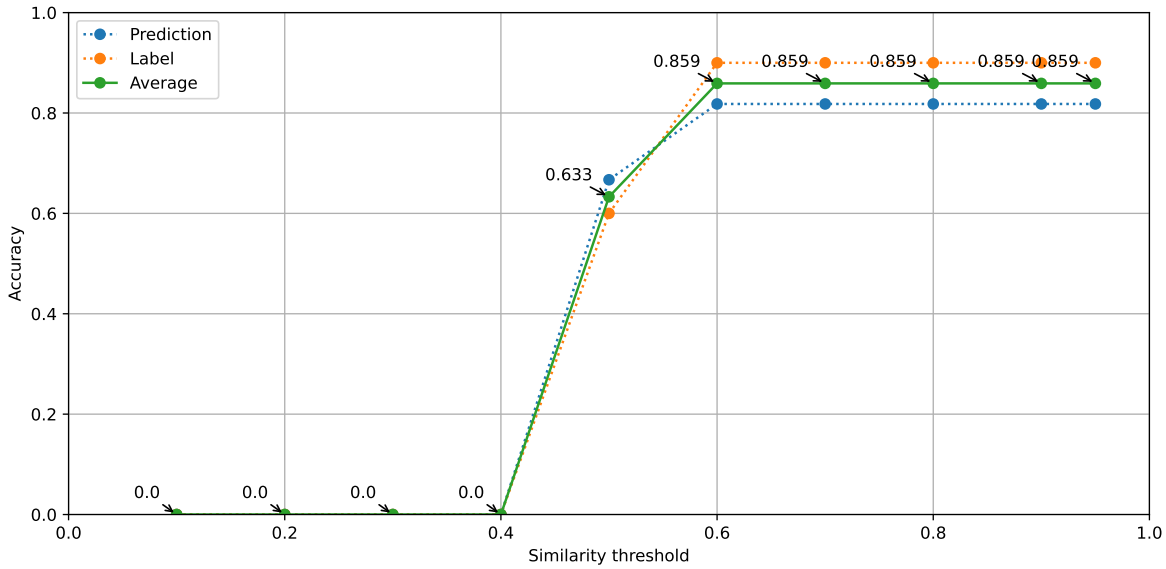


Figure 4.19.: SBERT Semantic Search accuracies for *DAST-Description* corpus

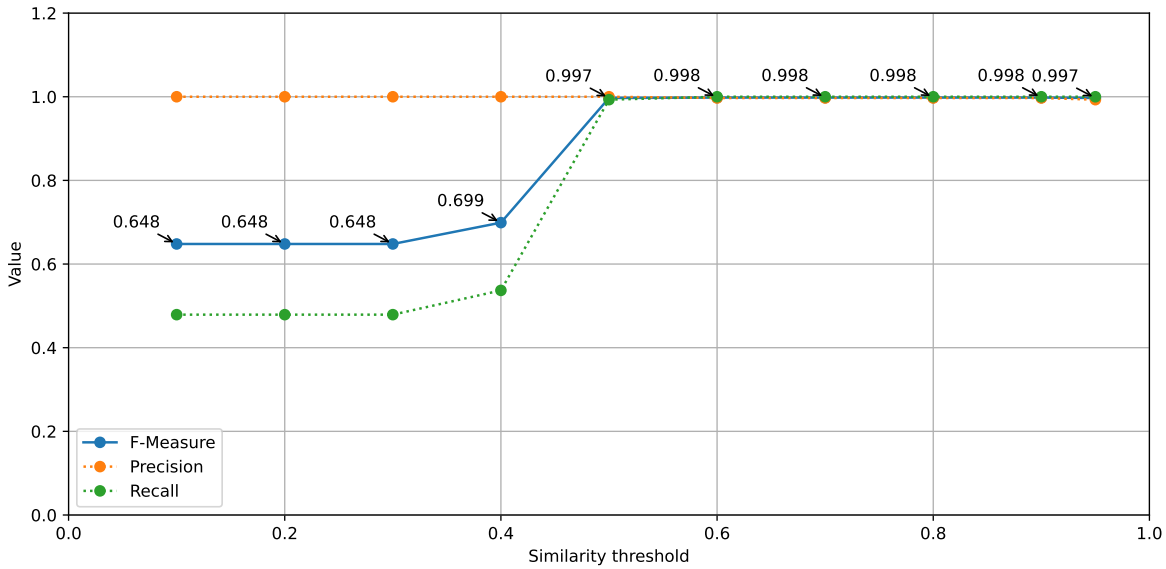


Figure 4.20.: SBERT Semantic Search metrics for *DAST-NDS* corpus

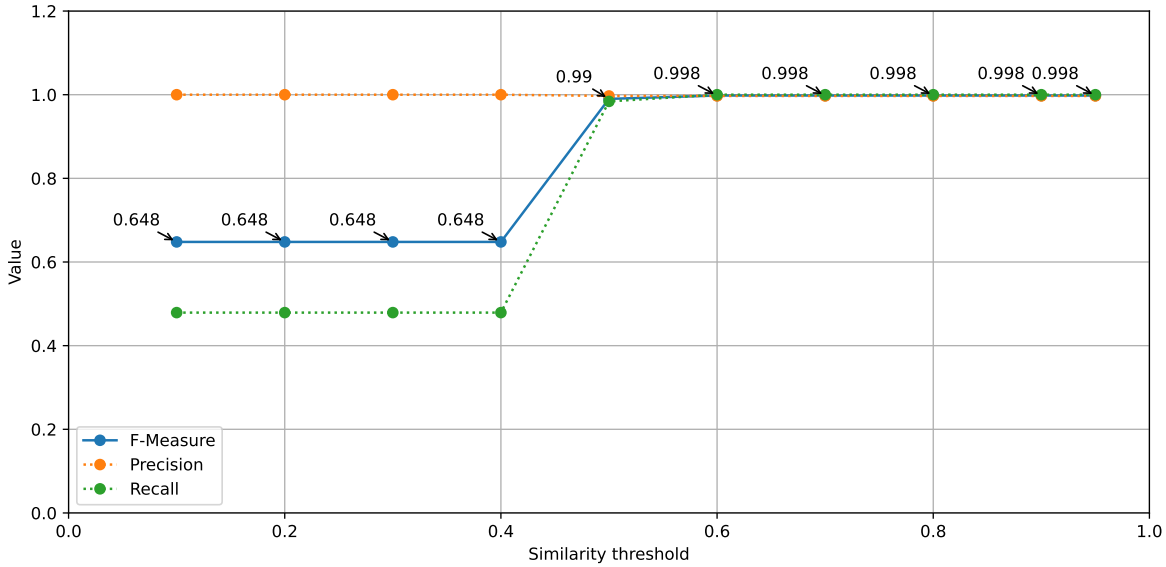


Figure 4.21.: SBERT Semantic Search metrics for *DAST-Description* corpus

4.3.2. Latent Semantic Indexing (LSI)-based deduplication

SAST

Applying LSI similarity to the *SAST-Plain* corpus yields a maximum (average) accuracy of 75% at similarity threshold of 0.8, whereas applying the same to *SAST-Aggregated* corpus yields 81.4% accuracy. This positive offset in accuracy for *SAST-Aggregated* corpus is also observed for all other similarity thresholds, as visible in Figure 4.22 and Figure 4.23. We also observe a proportional growth in accuracy for both corpora with increasing similarity thresholds. However, the difference in results occurs after similarity threshold 0.8, whereby the accuracy for *SAST-Plain* corpus falls by 14%, before recovering to 69.7% . This is contrary to the accuracy from *SAST-Aggregated* corpus, which first increases to 82.6% and then maintains above 80%. Similar to their SBERT Semantic Search counterparts, we observe that the prediction accuracy has a higher value for similarity thresholds under 0.5, after which label accuracy value becomes higher.

Interestingly, the results of f-measure, precision and recall are similar for both corpora, as can be seen in Figure 4.24 and Figure 4.25. The highest value for f-measure, 90%, is achieved right after similarity threshold 0.2, which stays constant for all remaining similarity thresholds. This is varies from the SBERT results, where the highest and constant value for f-measure is achieved for similarity thresholds greater than 0.5. From an objective point of view, the results of f-measure, precision and recall illustrate no gain in performance by variation of corpora, for different similarity thresholds.

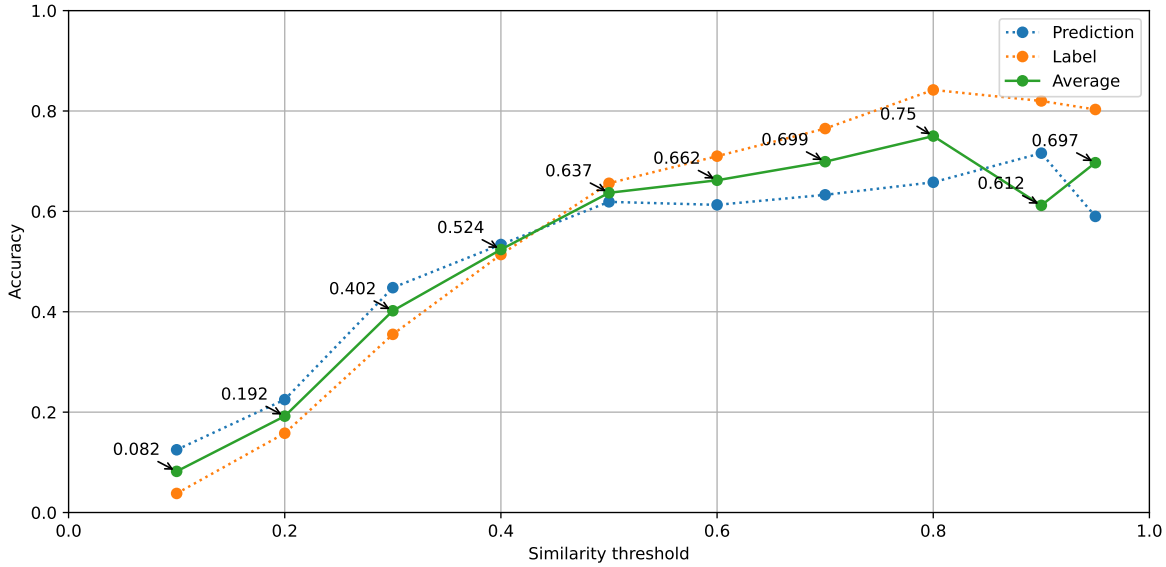


Figure 4.22.: LSI accuracies for *SAST-Plain* corpus

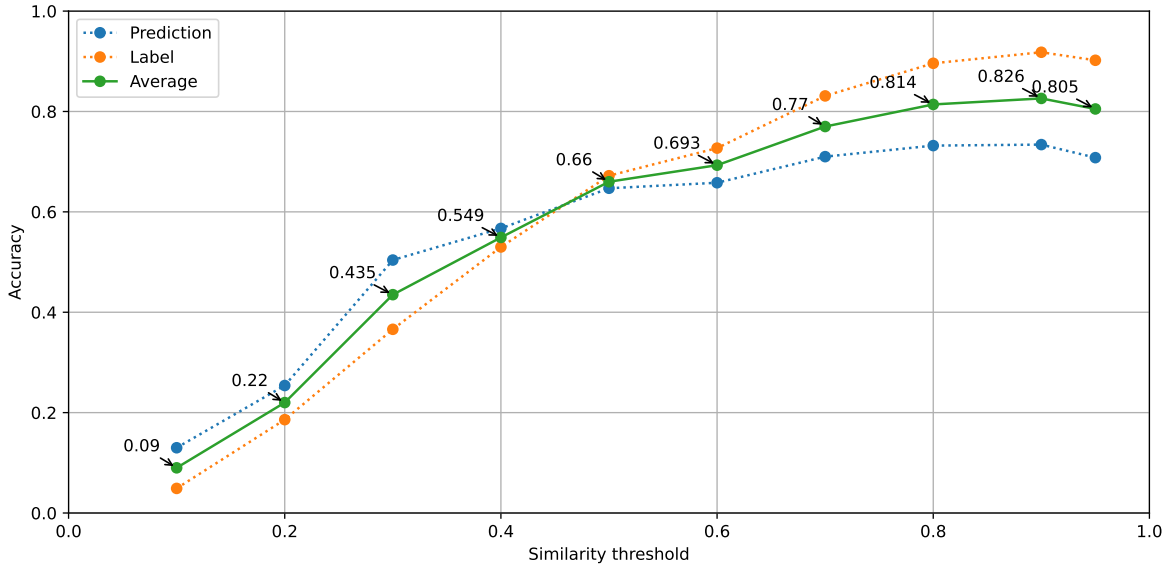


Figure 4.23.: LSI accuracies for *SAST-Aggregated* corpus

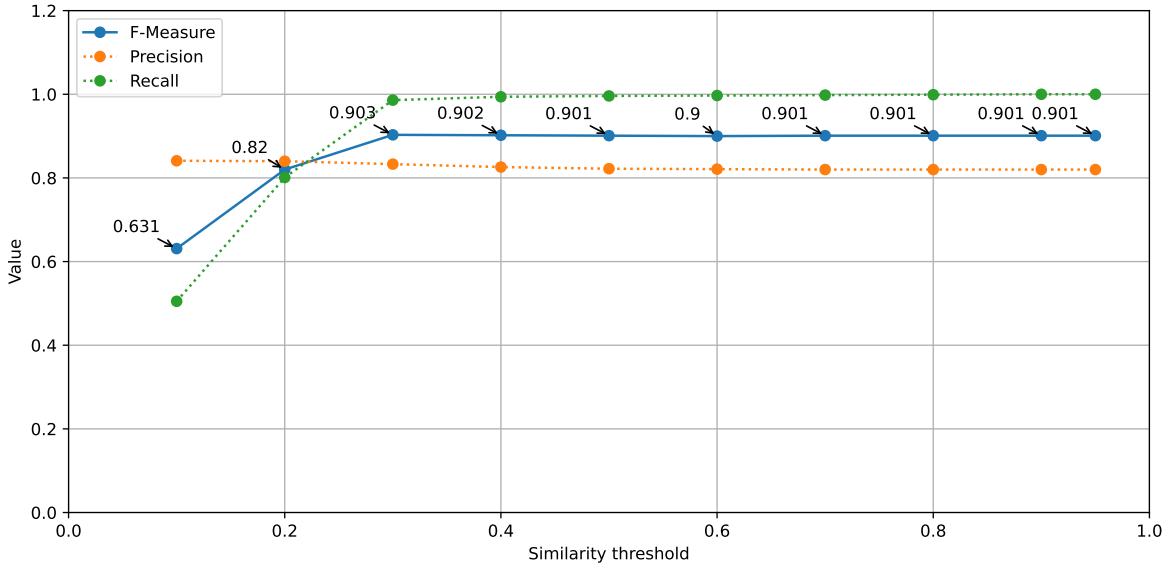


Figure 4.24.: LSI metrics for *SAST-Plain* corpus

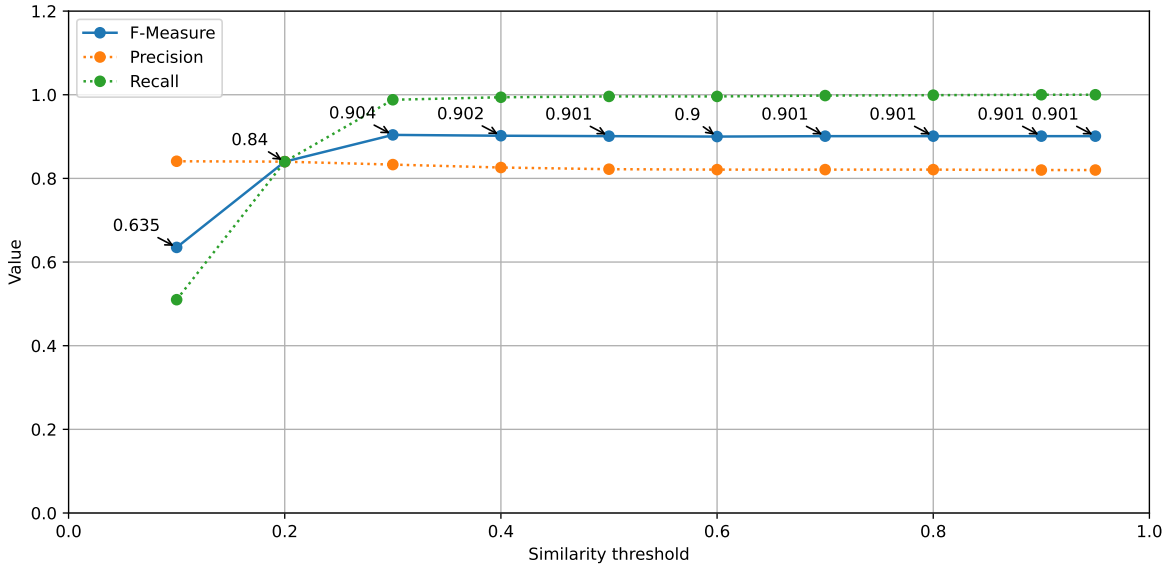


Figure 4.25.: LSI metrics for *SAST-Aggregated* corpus

DAST

The accuracies resulting from applying LSI to *DAST-NDS* and *DAST-Description* corpora at different similarity thresholds are shown in Figure 4.26 and Figure 4.27 respectively. It can be seen that the average accuracy resulting from *DAST-NDS* corpus is higher for four similarity threshold values compared to a single value for *DAST-Description* corpus. However, for similarity thresholds useful for our case i.e., > 0.5, both corpora yield the same accuracy of 85.9%. We also observe that the accuracy gained from both corpora drops to 73.3% for similarity thresholds ≥ 0.8.

The values of f-measure, precision and recall resulting from *DAST-NDS* and *DAST-Description* are shown in Figure 4.28 and Figure 4.29. It can be seen that values of all three metrics are the highest possible and similar for similarity thresholds ≥ 0.2. This is different from the SBERT Semantic Search results for both corpora, where the highest metric values are achieved for similarity thresholds ≥ 0.5. Consequently, it is visible from results of LSI on DAST corpora that varying corpora does not result in better deduplication.

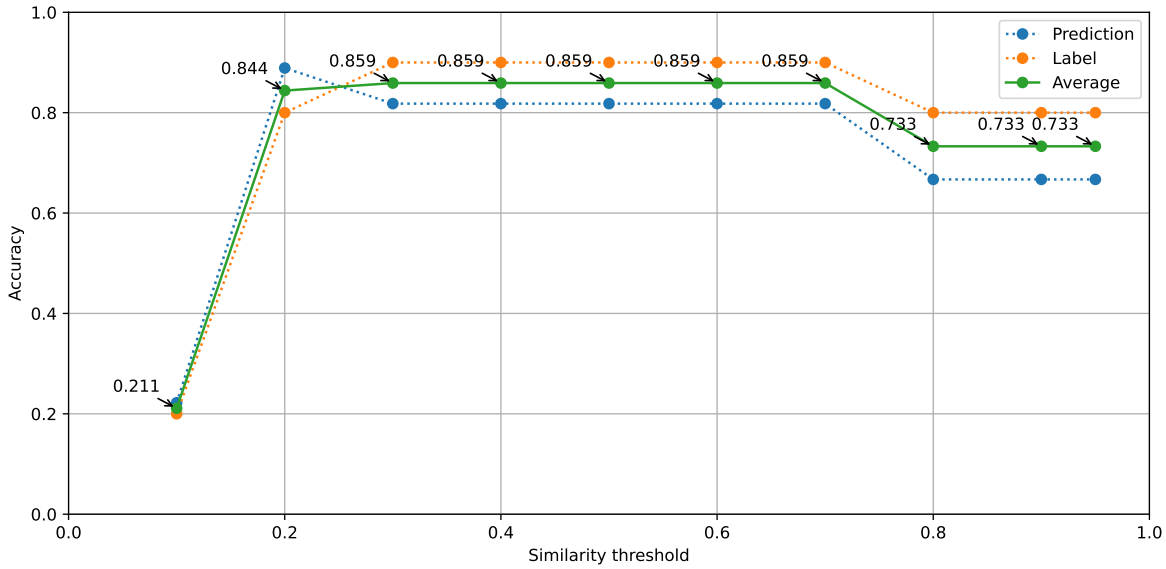


Figure 4.26.: LSI accuracies for *DAST-NDS* corpus

4.3.3. Knowledge Graph (KG)-based deduplication

SAST

Applying KG *bag-of-words* approach to *SAST-Plain* and *SAST-Aggregated* corpora yields the accuracies shown in Figure 4.30 and Figure 4.31, respectively. We observe that for both the highest average accuracies are achieved at similarity threshold of 0.3, after which the accuracies plateau. However, all values for the *SAST-Aggregated* corpus are considerably higher than those of the *SAST-Plain* corpus.

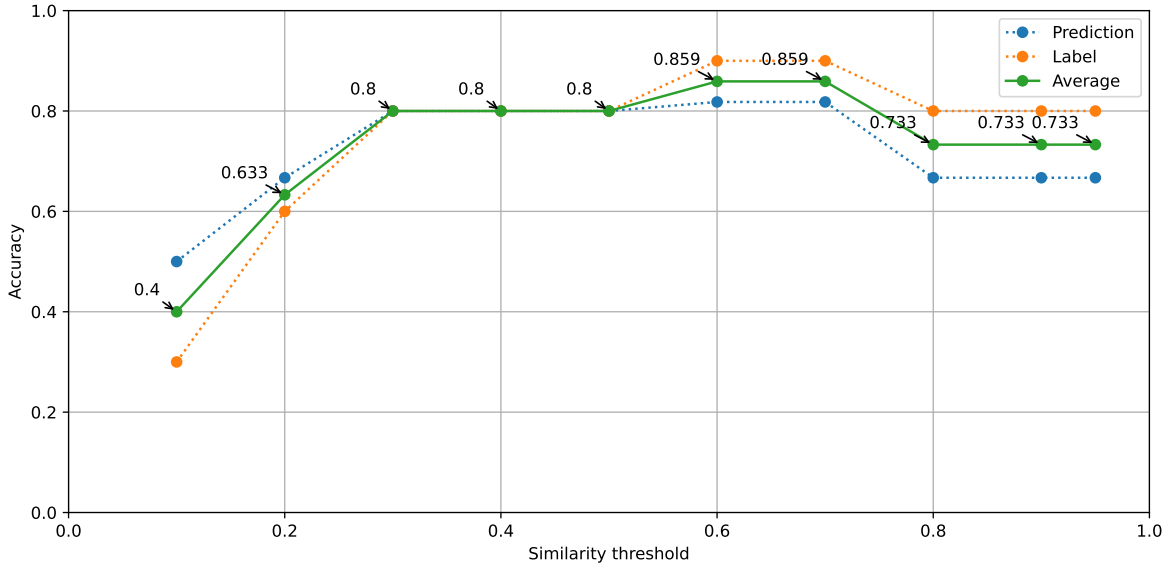


Figure 4.27.: LSI accuracies for *DAST-Description* corpus

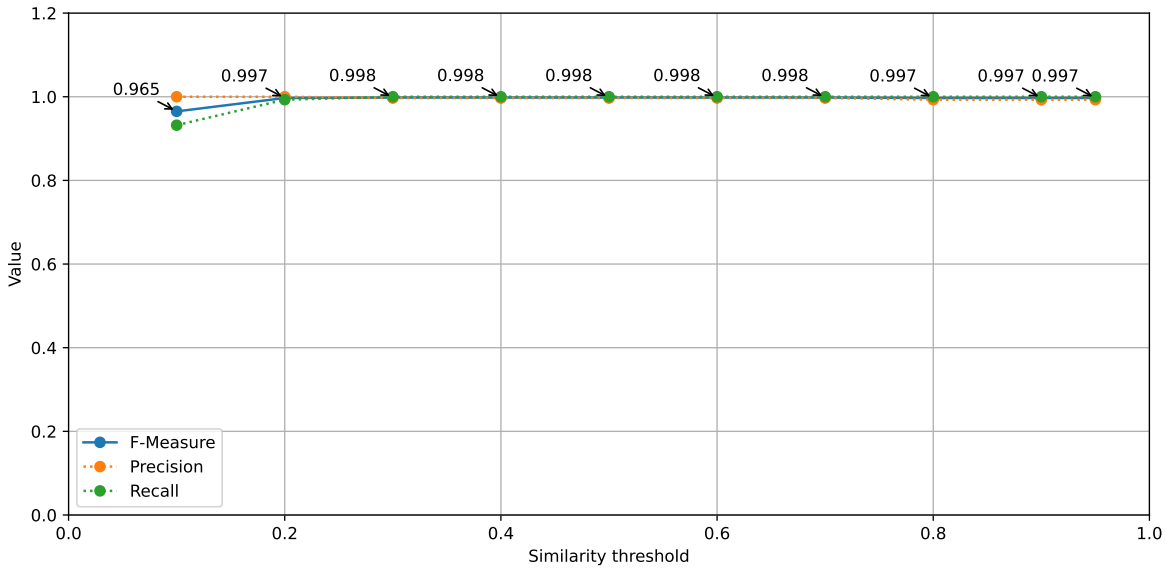


Figure 4.28.: LSI metrics for *DAST-NDS* corpus

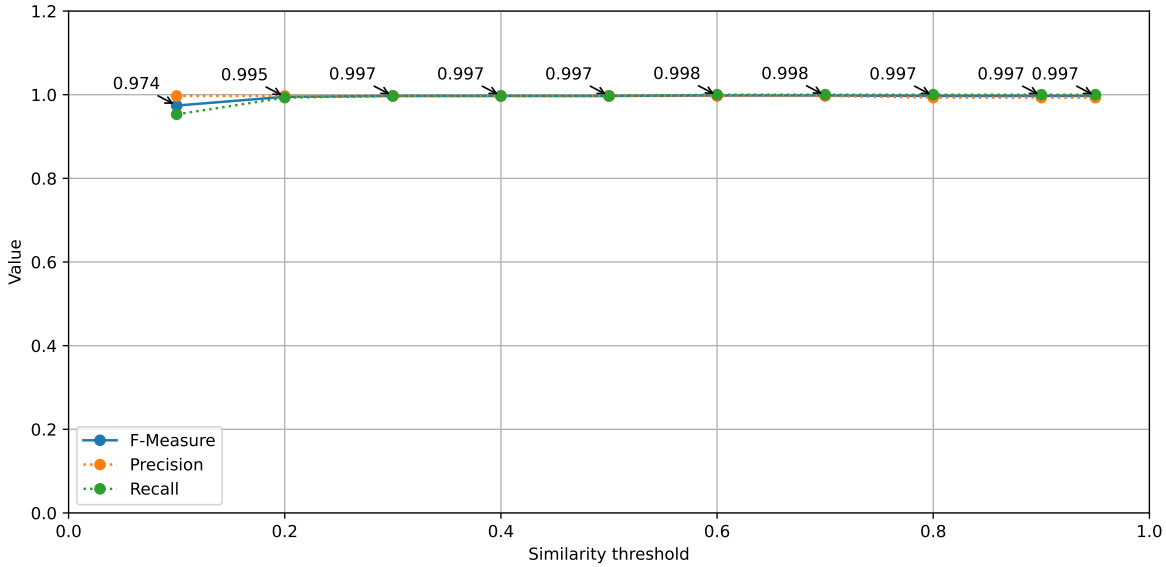


Figure 4.29.: LSI metrics for *DAST-Description* corpus

Likewise, the values of f-measure, precision and recall for KG-based deduplication of *SAST-Plain* and *SAST-Aggregated* corpora are illustrated in Figure 4.32 and Figure 4.33. We see that all values are exactly the same over all similarity thresholds. We also observe that the highest f-measure value is achieved for a large ratio of similarity thresholds.

DAST

The KG-based deduplication of *DAST-NDS* and *DAST-Description* corpora results in the accuracies illustrated in Figure 4.34 and Figure 4.35, respectively. Unlike SAST, both corpora exhibit the same accuracies over all similarity thresholds, which shows that varying corpora has no effect on the performance. However, similar to the trend for SAST, the highest accuracies for both corpora are achieved for similarity thresholds ≥ 0.3.

Likewise, the f-measure, precision and recall values for *DAST-NDS* and *DAST-Description* corpora are shown in Figure 4.36 and Figure 4.37, respectively. All values for both corpora turn out to be same, similar to their counterparts from SAST results. We also observe that values converge and plateau at a high of 0.997 for similarity thresholds ≥ 0.3.

4.3.4. Summary

Table 4.4 shows a summary of achieved accuracies and metrics when deduplicating the four corpora using techniques listed in this section. The values have been selected based on the *highest average accuracy* of each experiment, and allow us to have an overview of which technique is best for deduplicating a given corpus.

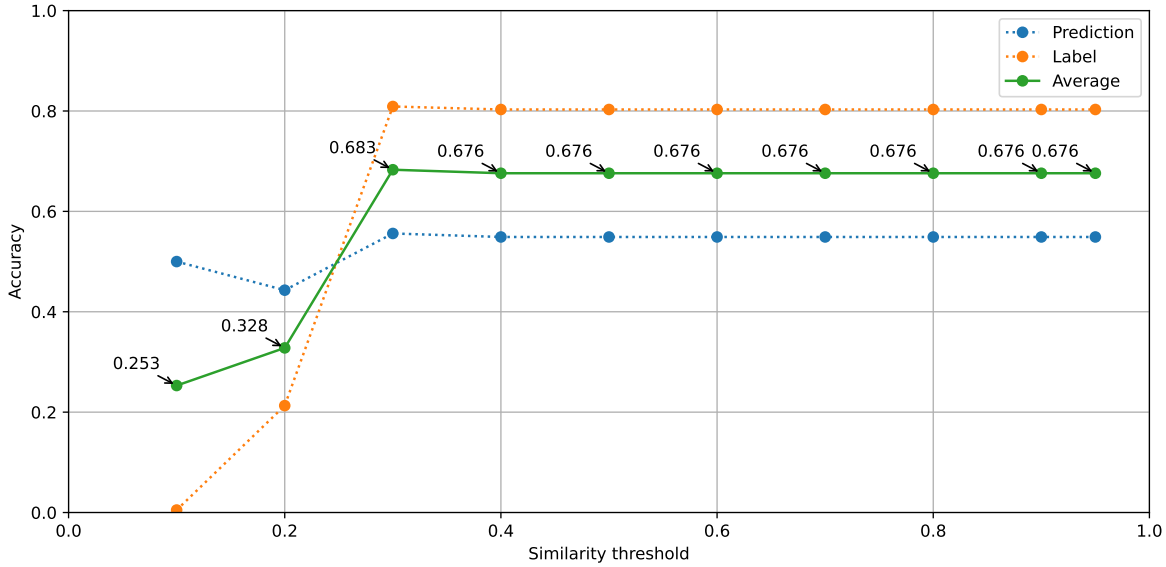


Figure 4.30.: KG accuracies for *SAST-Plain* corpus

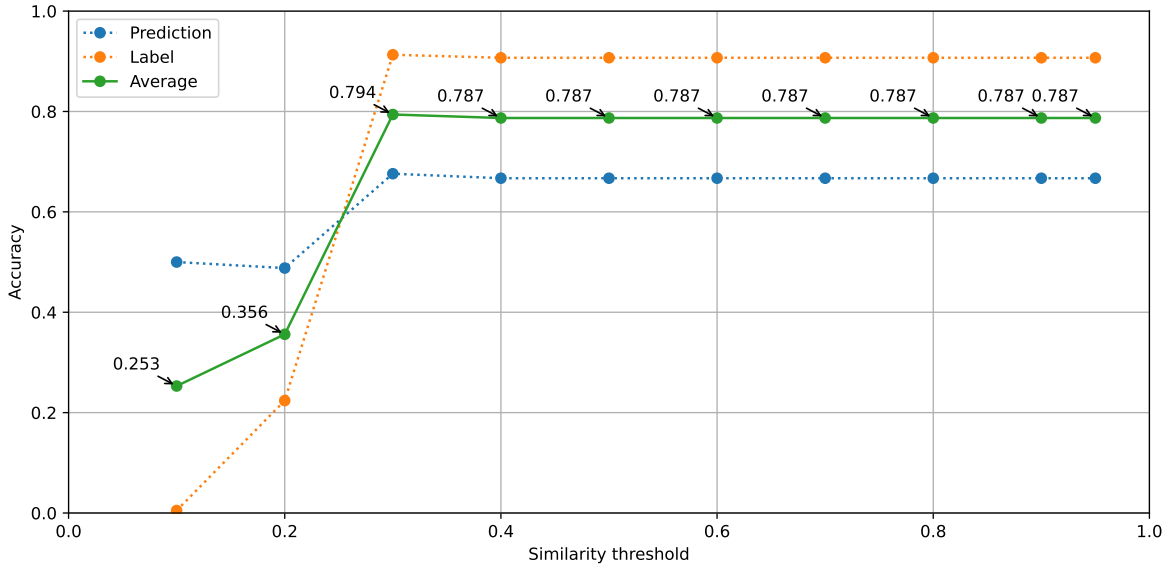


Figure 4.31.: KG accuracies for *SAST-Aggregated* corpus

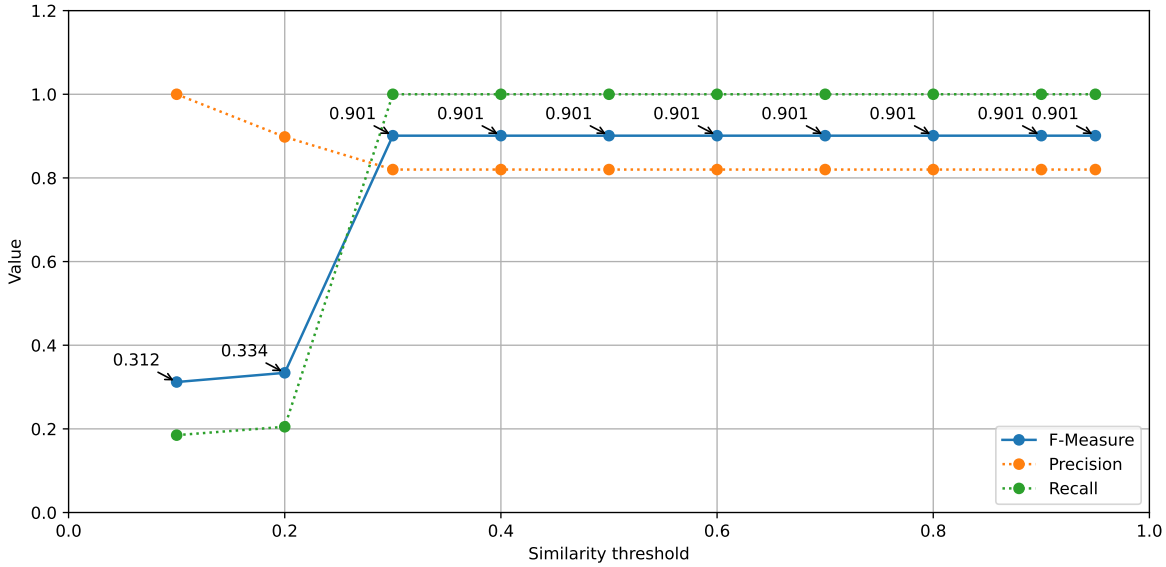


Figure 4.32.: KG metrics for *SAST-Plain* corpus

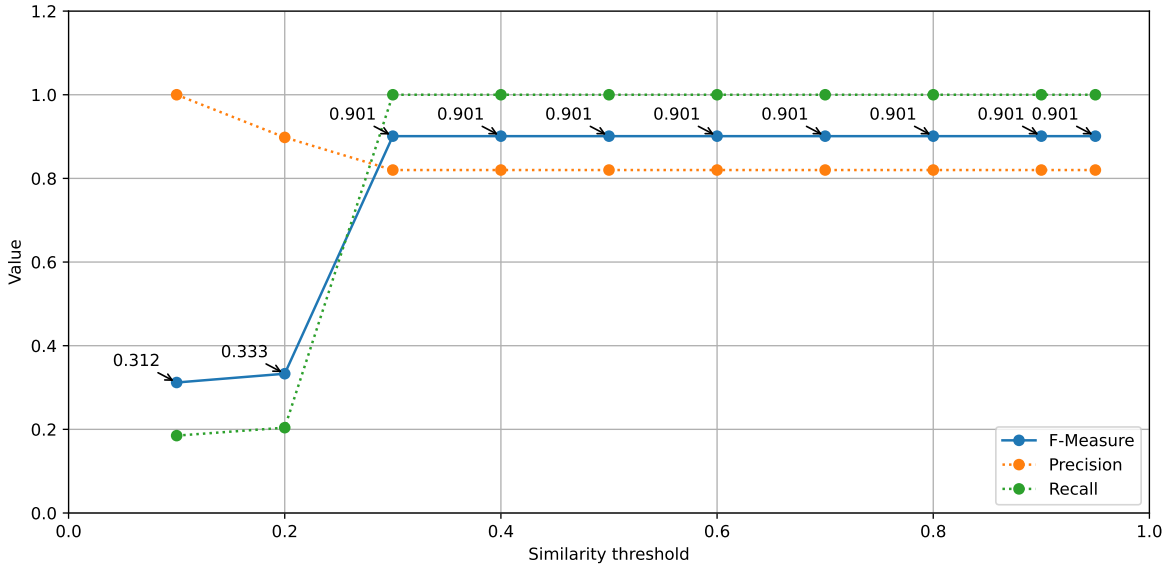


Figure 4.33.: KG metrics for *SAST-Aggregated* corpus

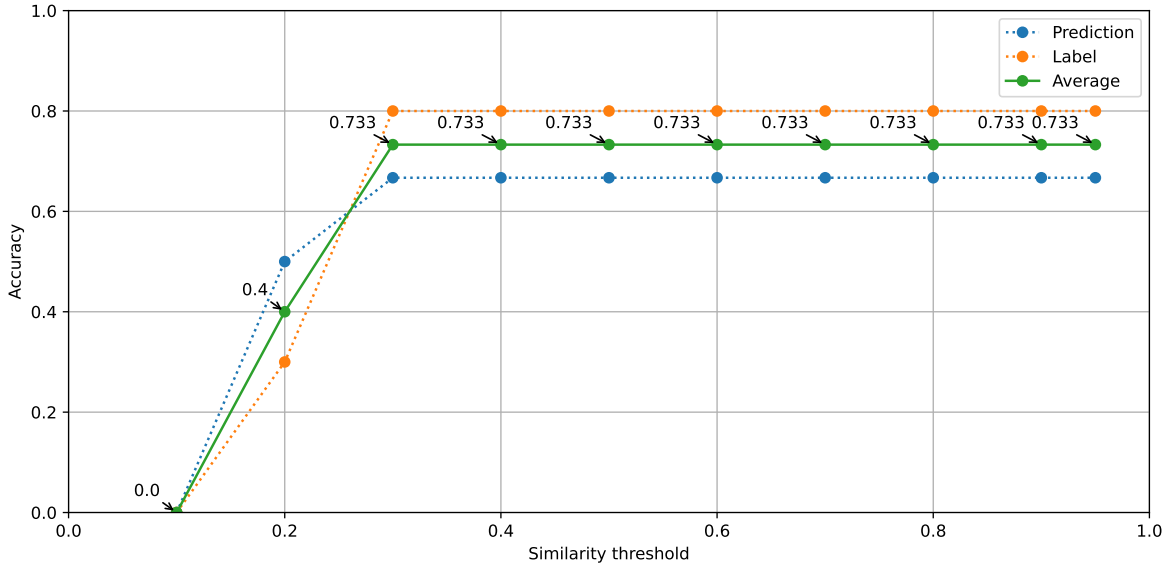


Figure 4.34.: KG accuracies for *DAST-NDS* corpus

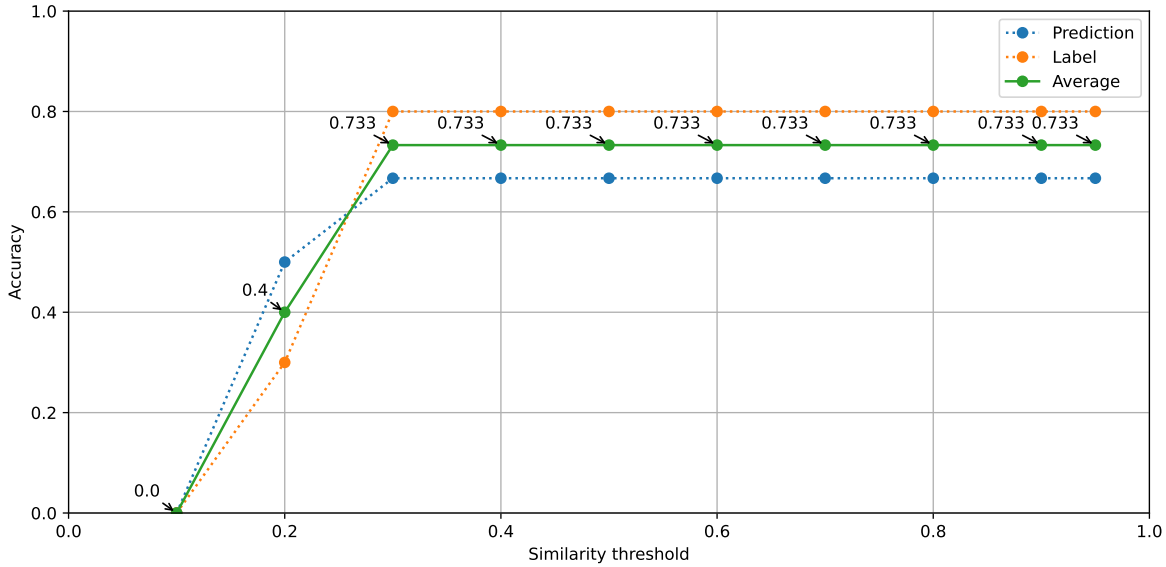


Figure 4.35.: KG accuracies for *DAST-Description* corpus

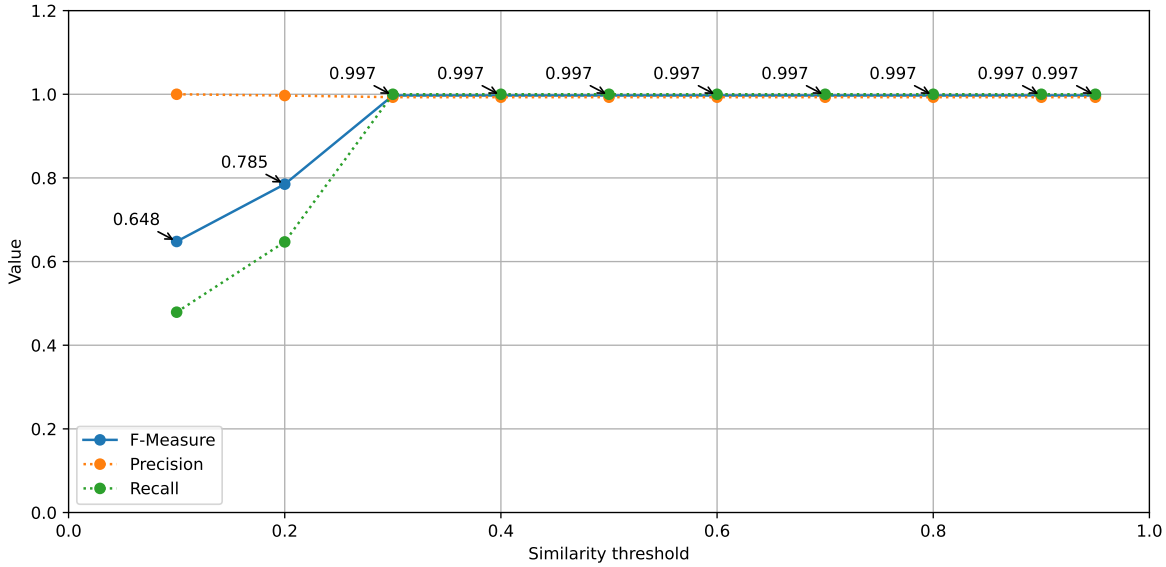


Figure 4.36.: KG metrics for *DAST-NDS* corpus

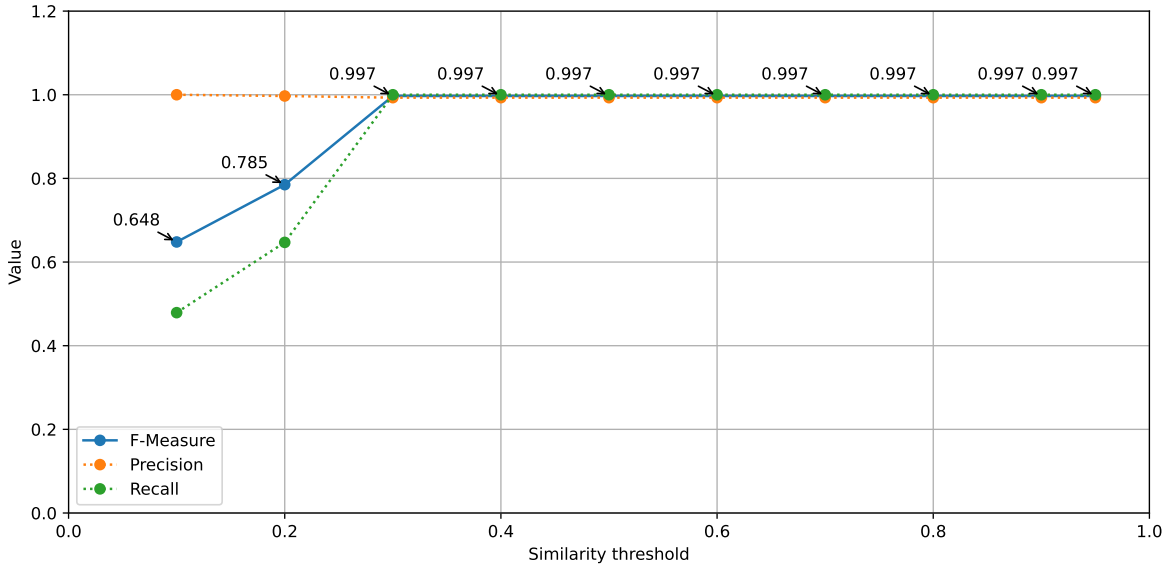


Figure 4.37.: KG metrics for *DAST-Description* corpus

Table 4.4.: Summary of accuracies and metric values

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

4.4. Qualitative evaluation

In this section, we present the results of qualitative evaluation of our experiments performed on SAST and DAST datasets. The cybersecurity professionals use *SeFiLa* to associate reasons and challenges with incorrect predictions of an experiment, and provide us with the files containing evaluation results. To save cybersecurity professionals' time, we only qualitatively evaluate the experiment results that have the *highest average accuracies*. As indicated in Table 4.4, for SAST dataset we use results from LSI applied to the *SAST-Aggregated* corpus, and for DAST dataset we use results from SBERT applied to the *DAST-NDS* corpus. The unified list of reasons and challenges used in both evaluations is shown in Table 4.5.

In our experiments, we had 72 incorrect predictions from the SAST experiment, and 2 incorrect predictions from the DAST experiment. This led to assignment of 114 and 4 reasons/challenges to wrong predictions from both experiments, respectively. Figure 4.38 and Figure 4.39 show the quantitative distribution of these assigned reasons and challenges.

Table 4.5.: List of reasons and challenges for qualitative evaluation

Type	Body
Challenge	**Special context case:** In the context of the product, this result can only be identified by somebody knowing the context of the application.
Special	**Same finding in context:** One or more Javascript script files are loaded from a third-party domain.
Challenge	**Different phrasing for same issue:** Different tools use different explanations to explain the same issue.
Challenge	**Tool information does not describe the finding:** Sometimes tool provide no description of the finding. Hence, our features could only rely on the title.
Challenge	**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.
Reason	**Construction of feature string sub-optimal:** The badly constructed string could be the reason for the incorrect clustering.
Challenge	**Tool describe findings according to location:** The tool describes the finding precisely according to the location of occurrence. Hence the finding is over-specified.
Human Error	The suggested clustering by the algorithm is correct.
Special	**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.

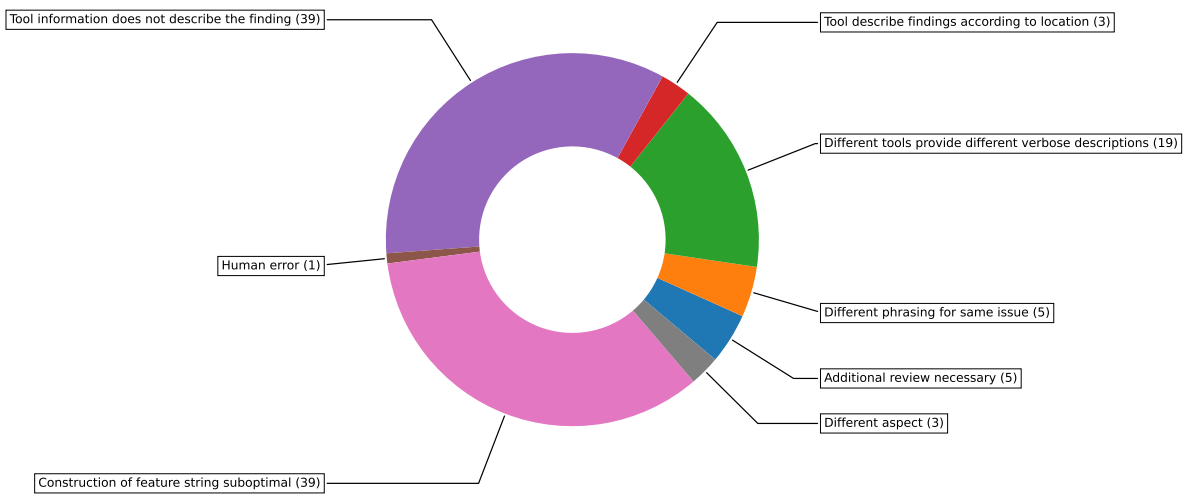


Figure 4.38.: Qualitative reasoning distribution for SAST experiment

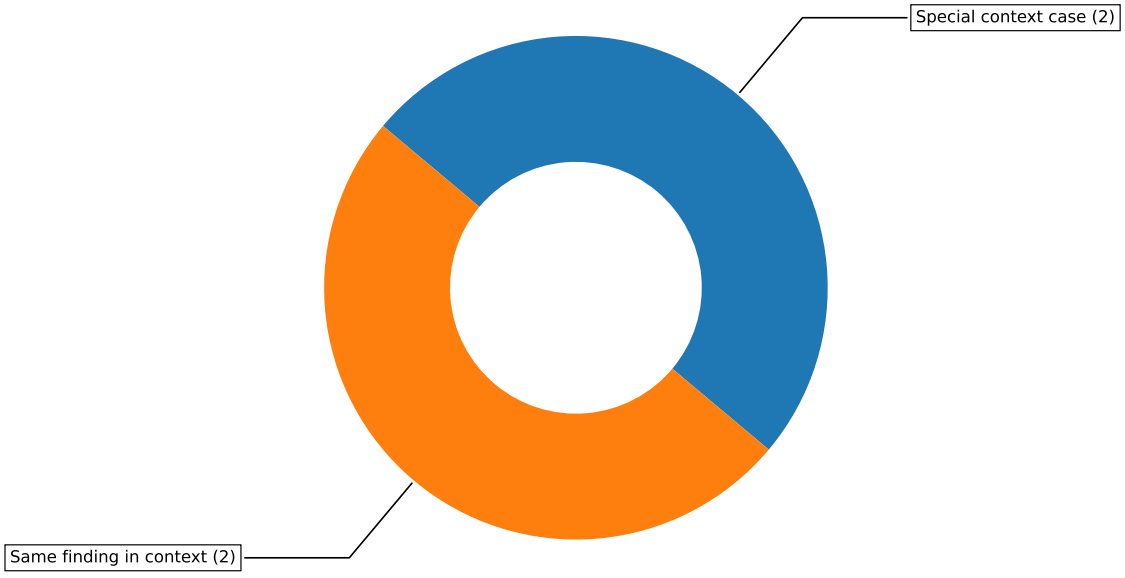


Figure 4.39.: Qualitative reasoning distribution for DAST experiment

5. Discussion

The goal of our work is to understand how base NLP semantic similarity techniques perform when trying to remove duplicates from DevOps security tool reports. Using theoretical knowledge from both domains, and input from the cybersecurity professionals, we implement the experimental setup to deduplicate security findings from SAST and DAST datasets. In this chapter, we make deductions from our results and use the information presented throughout this thesis answer our research questions.

5.1. RQ1: What are duplicate security tool findings in DevSecOps and how can they be deduplicated?

In section 2.2, we discuss the need for security tools which scan a software artifact before and after deployment, and the difference between use-cases of SAST and DAST. Upon scanning completion, these tools generate semi-structured reports containing security findings, where each finding indicates an instance of a vulnerability in the scanned artifact. Due to presence of a vulnerability in multiple locations in software and due to overlapping use-cases of security tools, findings identifying the same vulnerability appear multiple times in the generated reports. We confirm this by generating our own security tool reports for an open-source software artifact, *JuiceShop*, and understand these reports with the guidance of cybersecurity professionals. We then analyze findings that might be duplicate based on different features e.g., their *descriptions* or their *locations*, and through communication with cybersecurity professionals establish domain-specific ways of determining duplicate security findings. The results of this coordination state that duplicate findings exist as 1). being exact replica of each other (findings-based), 2). talking about the same identified issue (problem-based), 3). having a common solution (solution-based), or 4). talking about vulnerabilities at the same location (location-based). The security findings from reports can be deduplicated using any of these approaches, each having its advantages and disadvantages. A detailed overview of these deduplication strategies has been presented in Table 2.3.

As part of answering this research question, we are able to understand the types of vulnerabilities scanned by security tools, the structures of reports they produce, and the contents of their findings. We are also able to realize the importance of deduplication in an agile DevSecOps environment, whereby scanning takes place frequently and cybersecurity professionals need to review numerous findings and report back to developers. Just for *JuiceShop*, there exist 1351 SAST findings for our selection of tools that represent a cumulative of 138 issues. It would be so much more efficient for a cybersecurity professionals to go through 10x lesser content and report back to the developers for corrections before the pipeline

can be re-run. For larger projects and more security tools, the amount of security findings is even larger and hinders the agility much more. Thereby, we are able to understand a very useful industrial challenge that needs to be solved, in addition to the theoretical knowledge regarding how to solve it from the viewpoint of cybersecurity professionals.

5.2. RQ2: How can we use Natural Language Processing (NLP) to deduplicate security tool findings?

To answer this research question, we need to have an overview of what natural language processing offers, both on the theoretical and application level. In chapter 2, we first understand the difference between syntax and semantics, and realize that the NLP task relevant for our work is *semantic textual similarity*. We then understand how the three basic categories of modern semantic similarity methods, namely *knowledge-based*, *corpus-based*, *transformer-based*, differ. For our ease, we choose the simplest method from each category for ease of debugging and more importantly to better understand why each technique performs the way it does in our results.

Having determined and understood the NLP semantic similarity techniques we want to use, we now want to have the security tool reports corpora to which we can apply each chosen technique to, for determining duplicate findings. Since there's no way in literature on how to form a corpora from security tool reports, we need to research different ways of doing so. The first challenge we solve is how to bring findings from different reports together to establish a ground-truth dataset. This is because our findings yield from multiple security tool reports, where each report is a semi-structured JSON file of objects that even has different property names referring to the same feature. Improvising on this challenge with cybersecurity professionals leads us to the creation of *SeFiLa*, a tool where cybersecurity professionals are able to upload reports from different security tools, forming named collections from duplicate security findings, and download the labelled dataset. We integrate functionalities like jumping between different tools and the saving progress to the cloud as part of feedback from cybersecurity professionals, to enable them to seamlessly label thousands of findings from different security tool reports. After hours of labeling, our cybersecurity professionals are able to provide us with separate SAST and DAST datasets.

We then determine ways of using the findings contained in our dataset with NLP semantic similarity methods. This is because each finding has multiple features e.g., *description*, *URL*, *location*, etc., whereas the semantic similarity features work with texts or sentences. This is where we use our answer from RQ1, and analyze what forms of deduplication work best with *semantic* similarity methods and our attained security findings dataset. We conclude that the most suitable type of deduplication is *problem-based*, which results if we deduplicate based on textual *descriptions* of findings. We justify this from the perspective of both domains involved. For the DevSecOps domain, we intend to group duplicates in a way that helps cybersecurity professionals get an overview of the problems present from results of a certain scan. From there, using their domain expertise, they can prioritize between issues based on how important a certain issue is in their context, and get it resolved faster. On the NLP end,

we plan to work with features that have the most semantic value, so that we can explore fully the potential of semantic similarity methods when applied to security findings corpus. We also analyze the SAST and DAST datasets separately, and avoid bias by ensuring that our selected features are present in reports from all security tools. For some SAST, we do not readily find the *description* feature contained in the generated reports. For them, we scrape the descriptions from the provided reference URLs and use them as a feature. Furthermore, we discover that the *description* feature for many SAST reports has brief textual content. We improvise and discuss with cybersecurity professionals as to what secondary feature they consider as a unique descriptor of the *problem* being represented by a finding. This leads us to using the *CVE-ID*, which is a unique vulnerability identifier, to aggregate semantic content of the *description* features for SAST, as also explained in subsection 3.2.1. We conclude that to apply NLP, two corpora can be formed from SAST reports using the *description* feature of findings, by 1). using the feature directly, and 2). using the feature's semantically aggregated version. Similarly for DAST, we form one corpora using the *description* feature and second by concatenating the *name*, *description* and *solution* features of findings.

To implement semantic similarity techniques for deduplicating security findings, we develop an experimental setup as elaborated in subsection 3.3.1. This allows us to run experiments by creating our desired corpora right from the labelled datasets, identifying and grouping together duplicate findings, and quantitatively evaluating the results against the ground-truth collections from the dataset.

As part of answering this research question, we are able to understand the kinds of textual information present in the *description* feature of security findings from different tools. Furthermore, we are able to realize the challenge of balancing semantic content amongst security findings from different tools, whereby some tools provide a verbose textual descriptions while others provide no description and require their descriptions to be scraped from reference sources. Finally, we have discovered unique identifiers like *CVE-ID* which can be used as an anchor when trying to determine if two findings are similar, thus complementing information from the textual descriptions.

5.3. RQ3: How do base NLP Semantic Similarity methods perform when applied to security tool reports corpus?

Having understood how security findings can be deduplicated from RQ1, and how the task of semantic similarity from NLP can be applied to deduplicate security findings as part of RQ2, we now answer the final research questions of how effective NLP semantic similarity techniques are for deduplicating security findings. In doing so, we make deductions from the quantitative and qualitative results mentioned in chapter 4.

Before we apply the semantic similarity methods, we decide the metrics for quantitatively evaluating our results. Since our case involves grouping duplicate findings together into *clusters* and comparing them with ground-truth data, we use *pair-counting methods* from the literature to compute the widely used *f-measure*, *precision* and *recall* metrics. These metrics compare the clusters based on the elements contained in them. While this is useful for

deducing how well our techniques work, the cybersecurity professionals highly emphasize the need for absence of *any* false positives in the results, since it is considered disastrous to the purpose of scanning software with security tools. To prevent such occurrences, we form a stricter set of *accuracy* metrics, that directly compare the clusters while disregarding their element coverage. These accuracy metrics involve *label accuracy*, *prediction accuracy*, and *average accuracy*, as described in subsection 3.4.1. The presence of former two types of accuracies allow us to evaluate our results from the vantage point of both DevSecOps and NLP, respectively.

We then apply the three methods, *(SBERT) Semantic Search*, *Latent Semantic Indexing (LSI)* and *Knowledge Graphs (KG)* to deduplicate security findings through four corpora from SAST and DAST datasets. As a result of deduplication, we get predicted clusters of findings which are deemed as semantically similar by a technique, after passing a *similarity threshold*. The similarity threshold signifies the similarity score below which we consider two findings as dissimilar. We evaluate the formed clusters using our defined metrics, and plot the results for different similarity thresholds. This enables us to empirically determine the right semantic thresholds for different techniques, and helps us understand the evolution of metric values with changing similarity thresholds.

When deduplicating SAST findings, it can be seen from Table 4.4 that the highest deduplication accuracy in our experiments is achieved by using *SAST-Aggregated* corpus with LSI. We observe the average accuracy to be 82.6%, where our predicted collections match 91.8% of the ground-truth collections and 73.4% of our predicted collections turn out to be useful. This is better than using *SAST-Aggregated* with SBERT, where although our predicted collections match 92.3% of the labels, only 70% of the predictions are useful, meaning we get more false positives which are undesirable for the cybersecurity professionals. Another definitive conclusion from our experiments is that using the *SAST-Aggregated* corpus always yields better results than using the *SAST-Plain* corpus when deduplicating SAST findings.

For DAST findings, we observe that all corpora yield the same, high average accuracy of 85.9% when used with both SBERT and LSI. However, we can see from Figure 4.26 that at higher similarity thresholds, the accuracy of LSI falls to 73.3%. This is problematic in production usage, where cybersecurity professionals would naturally keep a high similarity threshold to prevent false positives. If doing so leads to a lower accuracy with LSI, it is better to use SBERT to deduplicate DAST findings, since the latter retains a high accuracy at high similarity thresholds. Furthermore, we see for SBERT that at a similarity threshold of 0.95, the average accuracy of the *DAST-NDS* corpus also falls to 73.3%, whereas a high value is retained with the *DAST-Description* corpus. Our interpretation of this result is that since *DAST-NDS* corpus contains two finding features of *description* and *solution*, at higher thresholds the semantic similarity measure gets very strict when comparing two corpus texts. Thereby, even if the *description* features of two findings are semantically similar, the *solution* features might differ, leading to a lower semantic similarity score between their formed corpus texts, and consequently leading them to being dissimilar findings. This also leads us to the conclusion that it is better to use the *DAST-NDS* corpus with a similarity threshold of 0.8-0.9. However, for strict deduplication at higher thresholds like 0.95, it is better to use

the *DAST-Descriptions* corpus, whose corpus texts consist just the *description* feature, thereby preventing any misinterpretation of the semantic context due to presence of another feature.

From the results of KG-based deduplication of all SAST and DAST corpora, we observe a lower performance compared to its LSI and SBERT-based counterparts. Further, we observe that the accuracies and metric values for f-measure, precision and recall plateau for similarity thresholds ≥ 0.3 (or any arbitrary value). The lower performance of KG-based deduplication can be justified due to the approach taken for deduplication. We use a *bag-of-words* approach, which compares the similarities of individual words in corpus texts and disregards the context in which those words occur. Furthermore, the underlying knowledge-source, *WordNet*, does not contain many domain-specific terms e.g., *jwt*, *sql*, *json*. Since a combination of such domain-specific terms allow a cybersecurity professional to discern findings and their impacts, they are vital when semantically comparing the corpus texts of different findings. In our implementation, we skip adding similarities of words that do not exist in *WordNet*, unless the words are exactly alike. This means that we ignore adding the similarity score of *jwt* and *sql* to our final similarity score, however, we add the similarity score of *jwt* and *jwt* as 1.0. Likewise, the plateau in accuracy and metric values also occurs due to our naive similarity calculation approach. Apart from domain-specific words, words like *vulnerability*, *attacker*, *compromises*, etc., occur frequently in the compared corpus texts and are available in *WordNet*. The similarity scores for word pairs *vulnerability-attacker*, *vulnerability-compromises* and *attacker-compromises* are 0.143, 0.182 and 0.167, respectively. If we add any or all of these to our final similarity score, and then normalize the final similarity score with number of word pairs, we get a very low overall similarity score. Thereby, beyond a certain similarity threshold, we observe the results of KG-based deduplication to plateau. However, given that KG-based deduplication still achieves 79.4% and 73.3% average accuracies for SAST and DAST findings respectively, we believe that it holds promise if the limitations of *bag-of-words* approach are mitigated.

For all techniques and corpora, we see that the trends of f-measure, precision and recall *metrics* are high-valued and constant compared to their corresponding accuracy values. For instance, when applying SBERT to the *SAST-Plain* corpus, we see a continuous positive improvement in all accuracies with the increase of similarity threshold. However, its metrics plot shows an initial high value for precision and low value for recall, both of which intersect at similarity threshold 0.5, before becoming constantly high above 80%. These consistent values of metrics result from pair-counting measures as described in subsection 2.1.5, which forms pairs of findings identifiers from predictions and ground-truth labels to compute the values of associated metrics. Since our experiments do not generate new findings, and instead just re-group the existing ones based on their semantic similarity score and similarity threshold, it so happens across the range of similarity thresholds that cluster sizes of predictions change from larger to smaller. This is because at lower similarity thresholds, more findings get considered similar to each other, whereas at higher similarity thresholds, less findings are able to cross the similarity threshold and hence appear separately. The large prediction clusters lead to formation of pairs that cover all findings. When these pairs are compared with all pairs of ground-truth labels, there exists high correspondence between them, leading to high

precision values initially. However, as the similarity threshold gets high, the grouping of findings transforms to include just findings with high similarity scores between corpus texts. Some of these groupings are also incorrect due to reasons mentioned in section 4.4, resulting in pairs that cannot be formed from the ground-truth groupings. This increases the false positive pairs, thereby leading to a decrease in precision. However, since the overall accuracies also increase with an increase in similarity threshold, we see that the value of precision falls by only a small amount. Since precision and recall are converse measures, initially the recall value is very low. However, as predictions are regrouped to include findings with higher similarity scores, a greater ratio of formed prediction pairs start matching the labels pairs, eventually bettering the recall.

The main difference of pair-counting measures with accuracy measures is that the former rewards clusters for including any percentage of correct findings, whereas the latter considers a prediction even with 98% correct findings as incorrect. Thereby, our motivator for including pair-counting measures is to analyze how our algorithms behave at *findings level*, in addition to evaluating clusters of findings that exactly match those in ground-truth labels. Our conclusion from f-measure, precision and recall for all techniques and corpora is that the performance of algorithms at findings level stays decent and consistent, and we can focus on improving our accuracies.

In addition to reporting the quantitative results of deduplication, we also perform qualitative evaluation by having cybersecurity professionals review the incorrect predictions and related labels, and from reading the corresponding corpus texts of each, highlight the challenges or reasons that lead to incorrect predictions in their perspective. We discover that for SAST, a major challenge is some tools' *description* features consisting just of title of the problem instead of description that provides details about the problem. This leads to insufficient semantic content to be derived from the finding corpus texts, thereby leading to incorrect deduplication. Another highly occurring reason for incorrect predictions in SAST is sub-optimally constructed corpus texts, which even challenge the cybersecurity professionals when they try to discern duplicate findings from reading them. The third most highlighted challenge for SAST is the imbalance in verbosity of *description* features of findings, which either contains contextually rich problem descriptions or under-specified ones, leading to different extents of semantic content being captured and thereby incorrect comparisons being made. Conversely, for DAST we only have 2 incorrect predictions, which are evaluated to be caused due to the challenge of being very application- and domain-specific.

Having answered the final research question, we are able to successfully realize the entire process of 1). understanding duplicate findings in security tool reports, 2). exploring how NLP can be used to identify these duplicates, 3). implementing different NLP techniques to deduplicate findings, and 4). evaluating the results of our experiments to give wholistic insights to cybersecurity professionals regarding the role of NLP in helping them deduplicate security findings from SAST and DAST reports.

6. Conclusion

In this chapter, we present concluding arguments to our thesis, and discuss possible future work. As mentioned in the beginning of this thesis, our work is a cross-domain research useful for both NLP researchers intending to explore the DevSecOps domain with regards to security tools and their findings, as well as DevSecOps researchers aiming to gauge the usability of semantic similarity or NLP in security tools.

In this thesis, we explore the deduplication of security findings contained in DevOps security tool reports using Natural Language Processing (NLP). To initiate such research, it is necessary to grasp important concepts from both domains of knowledge. Therefore, we first explore what DevSecOps is and how it is related to DevOps. We find that principles of agility and automation best describe the aim of both, whereby the former aims to implement security as first member of the latter. We then explore the types of security analysis and understand the difference between static analysis and dynamic analysis, performed using SAST and DAST, respectively. To practically understand the difference between them, our cybersecurity professionals scan an open-source insecure web application known as *JuiceShop* with 7 SAST and 2 DAST that are publicly available, and provide us with the generated security tool reports. A major motivation of using open-source tools and scan target is for the reader to be easily able to reproduce the results of our work. We then go over different security tool reports, and try to establish from a novice's perspective the factors that lead to findings being called duplicates of each other. In doing so, we identify features like *description* and *solution* that are repetitive in different findings, and actively collaborate with cybersecurity professionals to understand their approach for identifying duplicate findings. The entire exploration helps us answer our first research question and leads to the deduplication strategies mentioned in Table 2.3.

To answer our second research question, it is necessary to understand NLP and the possibilities it offers. We discover that the task of *semantic similarity* in NLP will help us identify texts that are semantically similar in meaning. We also explore different types of semantic similarity techniques and conclude that there are four main categories of semantic similarity techniques, namely *knowledge-based*, *corpus-based*, *deep neural network-based*, and hybrid of these. Since the core of our research intends to explore how NLP can be applied to security findings, we only use the base methods from each category. This helps us better understand how each method deduces a similarity score for the finding texts passed to it, and allows us to debug at a deeper level. We also realize that to fully explore the potential of *semantic* similarity techniques, we need to provide input texts that have the most semantic content. In other words, we need to provide textual features that are descriptive of the entire finding. From the deduplication techniques figured out in our answer to RQ1, we realize that this will lead us to performing problem-based deduplication, where the deduplication results

would contain groups of similar findings representing the same issue or vulnerability. We also find problem-based deduplication advantageous from the perspective of both domains involved. For the DevSecOps domain, we believe it helps cybersecurity professionals get an overview of all existing vulnerabilities in the scanned artifact, from which they can prioritize based on the importance for their application or environment, and get the findings resolved accordingly. Likewise, for the NLP domain, problem-based deduplication would help us conclude the extent to which base NLP techniques understand the semantic context of security findings. This is because the semantic similarity techniques would be encountering just texts to determine the similarity of findings, and only texts describing the identified problem of their respective findings are conclusive enough to fully represent the purpose of a finding. This embarks us to determine the finding features which fully describe the problem and are present in findings from all tools. For SAST findings, we discover it to be the *description* feature, and for DAST findings, the *description* and *solution* features. We also observe that the *description* feature for SAST findings is brief and thereby not completely descriptive of the identified problem in most cases. We discuss this limitation with cybersecurity professionals and ask them about the features they next consider to be completely representative of the problem identified by a security finding. Using their domain expertise that suggests using the *CVE-ID* feature, we increase the semantic content of SAST *description* feature by concatenating texts from findings having the same *CVE-ID*, and using the concatenated descriptions as inputs to semantic similarity techniques instead of the plain description feature. Having selected the features for deduplication, we realize the need of a ground-truth dataset to compare the results of our semantic deduplication with. This leads to another research outcome of developing *SeFiLa*, a tool which allows cybersecurity professionals to gather reports from different security tools and conveniently group their findings into different *collections*, each collection representing a single problem. Using *SeFiLa*, the cybersecurity professionals provide us with the ground-truth datasets for SAST and DAST. We also develop an experimental setup to use with the labelled datasets that allows us to conduct different experiments by effortlessly varying the techniques and corpora (texts formed from finding features). These milestones allow us cumulatively to the answer our second research question and enable us to finally run our deduplication experiments.

Finally, we conduct our experiments and evaluate their results quantitatively and qualitatively. In quantitative evaluation, we determine how each semantic similarity technique performs with respect to reproducing the collections from ground-truth dataset. We evaluate this by using two types of metrics; the general clustering metrics based on pair-counting, and our own accuracy metrics based on collection comparison. These metrics allow us to determine the efficacy of NLP semantic similarity techniques for deduplication on both *findings level* and *collections level*, respectively. We conduct our experiments for four corpora, two from each SAST and DAST, using a base technique from each category of semantic similarity techniques. Our results indicate that corpus-based methods are best suited when applied to SAST findings, and regardless of the method applied, the semantic content of *description* features should always be increased for SAST findings to achieve good deduplication. We also discover that deep learning-based methods perform better when deduplicating DAST

findings, with medium-high similarity thresholds working good for *DAST-NDS* corpus and very high similarity thresholds working well for the *DAST-Description* corpus.

For the qualitative part of our evaluation, we provide cybersecurity professionals with the texts of findings that get grouped incorrectly. Since semantic similarity techniques see just these finding texts when trying to group findings, the cybersecurity professionals select from a list of reasons and challenges those that best describe why these findings are predicted incorrectly given their finding texts. For SAST, we see that the most prominent reasons are findings containing brief titles instead of descriptions, and finding texts being formed sub-optimally. This highlights that the challenge lies with the provision of required information from security tool reports, instead of the failure of NLP semantic similarity techniques in understanding the semantic content of a finding text. For DAST, the most prominent reason of incorrect predictions is the *special context case*, which means that the result can only be identified by a cybersecurity professional knowing enough context of the application. The process of qualitative evaluation is also achieved through *SeFiLa*, which allows us to gather the responses of cybersecurity professionals in a uniform format that can be quantified. The successful evaluation of our techniques allows us to answer to the final research question of how base NLP methods perform when deduplicating security findings from DevOps security tool reports.

The source code for *SeFiLa* and our experimental setup are publicly available on the webpage of this thesis on the TUM sebis chair's website.

6.1. Future work

Having answered our research questions, we now outline the future work that can be pursued. A very important insight for our topic would be to see how location-based, solution-based, or their hybrid with problem-based deduplication performs. This would help us explore the possibility of achieving a greater overall accuracy when using aspects from multiple deduplication strategies. For instance, both problem-based and location-based deduplications can be performed separately, and a greater weight can be assigned to the results of location-based deduplication if the *description* feature does not contain enough semantic content. The final result can be evaluated against the ground-truth dataset.

A second possibility in future work is generating more textual data for the findings from different sources, and then performing problem-based deduplication as carried out in this thesis. The goal should be for textual data of each finding to consist of multiple paragraphs and contain enough semantic content for the semantic similarity techniques to grasp as much contextual information as possible. It would be interesting to see if increased textual data for findings helps to grasp better the semantics of the problem as well, leading to improved deduplication results. Such texts could be scraped from the multiple reference sources listed in a finding, or retrieved from a vulnerability database based on an identifier. It should be taken care that the final corpus texts have the same level of verbosity to avoid bias. Overall, this approach could help overcome the challenges highlighted for SAST, where insufficient textual information is encountered for semantic similarity techniques leading to a challenge

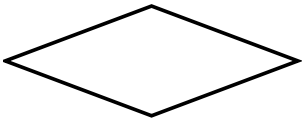
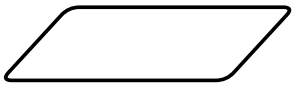
in deduplication. This work can also be extended by using fine-tuned NLP models for deduplication.

With modern advancements in neural networks, it would also be an interesting comparison to the NLP approach presented in this thesis to see how plain neural networks perform when trained with semi-structured findings as-is. Since neural networks automatically prioritize important features using layers like max-pooling, the manual effort undertaken by us to determine best features and deduplicate based on those might be eliminated altogether. However, training a neural network requires lot more data, so it would be necessary to devise a way to multiply the findings from existing datasets, or come up with a way to *generate* findings for the problems from our existing datasets, so that the manual labeling part can be avoided.

A. General Addenda

A.1. Definition of flowchart elements

The following table summarizes the shapes, names and purposes of flowcharts elements used in this thesis.

Element	Name	Purpose
	Terminator	Represents the start point or the end point of a program.
	Decision	Represents a conditional operation determining which one of the two paths a program will take.
	Process	Represents a process or set of operations that change value, form, or location of data.
	Input/Output	Represents input of data to the upcoming element or output of data from the preceding element.

List of Figures

2.1. SVD of the word co-occurence matrix . . . 9
2.2. SBERT architecture for inferring sentence similarity . . . 11
2.3. Automated deployment pipeline . . . 15

3.1. Elements of deduplication process . . . 24
3.2. Scanning pipeline . . . 25
3.3. Dataset class diagram . . . 26
3.4. Data Loader and Corpus Format class diagrams . . . 27
3.5. Feature-based semantics aggregation . . . 28
3.6. Run Case and Technique class diagrams . . . 29
3.7. Knowledge-graph Similarity . . . 30
3.8. LSI Similarity . . . 31
3.9. SBERT Semantic Search . . . 32
3.10. Transitive clustering process . . . 33
3.11. Quantitative evaluation process . . . 34
3.12. Qualitative evaluation process . . . 34

4.1. *SeFiLa*: Main page . . . 36
4.2. *SeFiLa*: *Generate Dataset* stage of *Label* feature . . . 38
4.3. *SeFiLa*: Interface for *Label* feature . . . 39
4.4. *SeFiLa*: *Upload* stage of *Evaluation* feature . . . 41
4.5. *SeFiLa*: Interface for *Evaluation* feature . . . 42
4.6. *SeFiLa*: Preview of the original finding . . . 43
4.7. *SeFiLa*: Dialog box to add or edit *Reasons* . . . 44
4.8. Number of findings per SAST . . . 45
4.9. Average lengths of SAST corpus texts . . . 46
4.10. Collections finding frequency for SAST . . . 47
4.11. Number of findings per DAST . . . 47
4.12. Average lengths of DAST corpus texts . . . 48
4.13. Collections finding frequency for DAST . . . 49
4.14. SBERT Semantic Search accuracies for *SAST-Plain* corpus . . . 50
4.15. SBERT Semantic Search accuracies for *SAST-Aggregated* corpus . . . 51
4.16. SBERT Semantic Search metrics for *SAST-Plain* corpus . . . 51
4.17. SBERT Semantic Search metrics for *SAST-Aggregated* corpus . . . 52
4.18. SBERT Semantic Search accuracies for *DAST-NDS* corpus . . . 52
4.19. SBERT Semantic Search accuracies for *DAST-Description* corpus . . . 53

4.20. SBERT Semantic Search metrics for *DAST-NDS* corpus 53
4.21. SBERT Semantic Search metrics for *DAST-Description* corpus 54
4.22. LSI accuracies for *SAST-Plain* corpus 55
4.23. LSI accuracies for *SAST-Aggregated* corpus 55
4.24. LSI metrics for *SAST-Plain* corpus 56
4.25. LSI metrics for *SAST-Aggregated* corpus 56
4.26. LSI accuracies for *DAST-NDS* corpus 57
4.27. LSI accuracies for *DAST-Description* corpus 58
4.28. LSI metrics for *DAST-NDS* corpus 58
4.29. LSI metrics for *DAST-Description* corpus 59
4.30. KG accuracies for *SAST-Plain* corpus 60
4.31. KG accuracies for *SAST-Aggregated* corpus 60
4.32. KG metrics for *SAST-Plain* corpus 61
4.33. KG metrics for *SAST-Aggregated* corpus 61
4.34. KG accuracies for *DAST-NDS* corpus 62
4.35. KG accuracies for *DAST-Description* corpus 62
4.36. KG metrics for *DAST-NDS* corpus 63
4.37. KG metrics for *DAST-Description* corpus 63
4.38. Qualitative reasoning distribution for SAST experiment 66
4.39. Qualitative reasoning distribution for DAST experiment 66

List of Tables

2.1. Contingency Matrix 13
2.2. Comparison of Static and Dynamic Analyses 16
2.3. Deduplication Strategies 20

4.1. Comparison of SAST and DAST datasets 43
4.2. *Description* feature names for SAST 46
4.3. *Description* feature names for DAST 48
4.4. Summary of accuracies and metric values 64
4.5. List of reasons and challenges for qualitative evaluation 65

Bibliography

[1] L. Riungu-Kalliosaari, S. Mäkinen, L. E. Lwakatare, J. Tiihonen, and T. Männistö. "DevOps adoption benefits and challenges in practice: A case study". In: *International conference on product-focused software process improvement*. Springer. 2016, pp. 590–597.

[2] H. Myrbakken and R. Colomo-Palacios. "DevSecOps: a multivocal literature review". In: *International Conference on Software Process Improvement and Capability Determination*. Springer. 2017, pp. 17–29.

[3] D. Schirrmacher. *Microsoft kontert Rekord-DDoS-Attacke mit 3,47 Terabit auf Cloud-Plattform Azure — heise.de*. `https://www.heise.de/news/Microsoft-kontert-Rekord-DDoS-Attacke-mit-3-47-Terabit-auf-Cloud-Plattform-Azure-6341800.html`. [Accessed 12-Aug-2022].

[4] M. Fowler and M. Foemmel. *Continuous integration*. 2006.

[5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[6] *Understanding searches better than ever before — blog.google*. `https://blog.google/products/search/search-language-understanding-bert/`. [Accessed 12-Aug-2022].

[7] K. Chowdhary. *Fundamentals of artificial intelligence*. Springer, 2020.

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

[9] S. Kim, N. Fiorini, W. J. Wilbur, and Z. Lu. "Bridging the gap: Incorporating a semantic similarity measure for effectively mapping PubMed queries to documents". In: *Journal of biomedical informatics* 75 (2017), pp. 122–127.

[10] M. Mohamed and M. Oussalah. "SRL-ESA-TextSum: A text summarization approach based on semantic role labeling and explicit semantic analysis". In: *Information Processing & Management* 56.4 (2019), pp. 1356–1372.

[11] Y. Kim. "Convolutional Neural Networks for Sentence Classification". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1746–1751. DOI: `10.3115/v1/D14-1181`. URL: `https://aclanthology.org/D14-1181`.

[12] H. K. Janda, A. Pawar, S. Du, and V. Mago. "Syntactic, semantic and sentiment analysis: The joint effect on automated essay evaluation". In: *IEEE Access* 7 (2019), pp. 108486–108503.

[13] W. Y. Zou, R. Socher, D. Cer, and C. D. Manning. "Bilingual word embeddings for phrase-based machine translation". In: *Proceedings of the 2013 conference on empirical methods in natural language processing*. 2013, pp. 1393–1398.

[14] A. Bordes, S. Chopra, and J. Weston. "Question answering with subgraph embeddings". In: *arXiv preprint arXiv:1406.3676* (2014).

[15] I. Lopez-Gazpio, M. Maritxalar, A. Gonzalez-Agirre, G. Rigau, L. Uria, and E. Agirre. "Interpretable semantic textual similarity: Finding and explaining differences between sentences". In: *Knowledge-Based Systems* 119 (2017), pp. 186–199.

[16] D. Sánchez, M. Batet, D. Isern, and A. Valls. "Ontology-based semantic similarity: A new feature-based approach". In: *Expert Systems with Applications* 39 (Mar. 2012), pp. 7718–7728. DOI: 10.1016/j.eswa.2012.01.082.

[17] G. A. Miller. "WordNet: a lexical database for English". In: *Communications of the ACM* 38.11 (1995), pp. 39–41.

[18] G. Zhu and C. A. Iglesias. "Computing semantic similarity of concepts in knowledge graphs". In: *IEEE Transactions on Knowledge and Data Engineering* 29.1 (2016), pp. 72–85.

[19] A. Pawar and V. Mago. "Challenging the boundaries of unsupervised learning for semantic similarity". In: *IEEE Access* 7 (2019), pp. 16291–16308.

[20] R. Navigli and S. P. Ponzetto. "BabelNet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network". In: *Artificial intelligence* 193 (2012), pp. 217–250.

[21] R. Rada, H. Mili, E. Bicknell, and M. Blettner. "Development and application of a metric on semantic nets". In: *IEEE transactions on systems, man, and cybernetics* 19.1 (1989), pp. 17–30.

[22] Z. Wu. "Palmer, M.: Verbs Semantics and Lexical Selection". In: *Proceedings of the 32nd Annual Meeting of Association for Computational Linguistics, Las Cruces, New Mexico*. 1994.

[23] S. Robertson. "Understanding inverse document frequency: on theoretical arguments for IDF". In: *Journal of documentation* (2004).

[24] D. Sánchez, M. Batet, and D. Isern. "Ontology-based information content computation". In: *Knowledge-based systems* 24.2 (2011), pp. 297–303.

[25] M. A. Rodriguez and M. J. Egenhofer. "Determining semantic similarity among entity classes from different ontologies". In: *IEEE transactions on knowledge and data engineering* 15.2 (2003), pp. 442–456.

[26] P. Resnik. "Using information content to evaluate semantic similarity in a taxonomy". In: *arXiv preprint cmp-lg/9511007* (1995).

[27] D. Lin et al. "An information-theoretic definition of similarity." In: *Icml*. Vol. 98. 1998. 1998, pp. 296–304.

[28] J. J. Jiang and D. W. Conrath. "Semantic similarity based on corpus statistics and lexical taxonomy". In: *arXiv preprint cmp-lg/9709008* (1997).

[29] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. *DBpedia—a crystallization point for the web of data. J Web Semant 7 (3): 154–165*. 2009.

[30] J. Hoffart, F. M. Suchanek, K. Berberich, and G. Weikum. "YAGO2: A spatially and temporally enhanced knowledge base from Wikipedia". In: *Artificial intelligence* 194 (2013), pp. 28–61.

[31] J. Gorman and J. R. Curran. "Scaling distributional similarity to large corpora". In: *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*. 2006, pp. 361–368.

[32] S. M. Mohammad and G. Hirst. "Distributional measures of semantic distance: A survey". In: *arXiv preprint arXiv:1203.1858* (2012).

[33] B. Altınel and M. C. Ganiz. "Semantic text classification: A survey of past and recent advances". In: *Information Processing & Management* 54.6 (2018), pp. 1129–1153.

[34] Q. Le and T. Mikolov. "Distributed representations of sentences and documents". In: *International conference on machine learning*. PMLR. 2014, pp. 1188–1196.

[35] J. Pennington, R. Socher, and C. D. Manning. "Glove: Global vectors for word representation". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543.

[36] O. Levy and Y. Goldberg. "Dependency-based word embeddings". In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. 2014, pp. 302–308.

[37] T. Mikolov, K. Chen, G. Corrado, and J. Dean. "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013).

[38] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. "Enriching word vectors with subword information". In: *Transactions of the association for computational linguistics* 5 (2017), pp. 135–146.

[39] K. Lund and C. Burgess. "Producing high-dimensional semantic spaces from lexical co-occurrence". In: *Behavior research methods, instruments, & computers* 28.2 (1996), pp. 203–208.

[40] E. Gabrilovich, S. Markovitch, et al. "Computing semantic relatedness using Wikipedia-based explicit semantic analysis." In: *IJcAI*. Vol. 7. 2007, pp. 1606–1611.

[41] M. A. Sultan, S. Bethard, and T. Sumner. "Dls@ cu: Sentence similarity from word alignment and semantic vector composition". In: *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*. 2015, pp. 148–153.

[42] R. L. Cilibrasi and P. M. Vitanyi. "The google similarity distance". In: *IEEE Transactions on knowledge and data engineering* 19.3 (2007), pp. 370–383.

[43] E. Agirre, E. Alfonseca, K. Hall, J. Kravalova, M. Pasca, and A. Soroa. "A study on similarity and relatedness using distributional and wordnet-based approaches". In: (2009).

[44] J. Shawe-Taylor, N. Cristianini, et al. *Kernel methods for pattern analysis*. Cambridge university press, 2004.

[45] Y. Le, Z.-J. Wang, Z. Quan, J. He, and B. Yao. "ACV-tree: A New Method for Sentence Similarity Modeling". In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization, July 2018, pp. 4137–4143. DOI: 10.24963/ijcai.2018/575. URL: https://doi.org/10.24963/ijcai.2018/575.

[46] T. K. Landauer and S. T. Dumais. "A solution to Plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge." In: *Psychological review* 104.2 (1997), p. 211.

[47] T. K. Landauer, P. W. Foltz, and D. Laham. "An introduction to latent semantic analysis". In: *Discourse processes* 25.2-3 (1998), pp. 259–284.

[48] R. A. Sinoara, J. Camacho-Collados, R. G. Rossi, R. Navigli, and S. O. Rezende. "Knowledge-enhanced document embeddings for text classification". In: *Knowledge-Based Systems* 163 (2019), pp. 955–971.

[49] F. Benedetti, D. Beneventano, S. Bergamaschi, and G. Simonini. "Computing inter-document similarity with Context Semantic Analysis". In: *Information Systems* 80 (2019), pp. 136–147. ISSN: 0306-4379. DOI: https://doi.org/10.1016/j.is.2018.02.009. URL: https://www.sciencedirect.com/science/article/pii/S0306437917301503.

[50] A. A. M. Al-Saffar, H. Tao, and M. A. Talab. "Review of deep convolution neural network in image classification". In: *2017 International conference on radar, antenna, microwave, electronics, and telecommunications (ICRAMET)*. IEEE. 2017, pp. 26–31.

[51] S. Hochreiter and J. Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[52] P. Zhou, W. Shi, J. Tian, Z. Qi, B. Li, H. Hao, and B. Xu. "Attention-based bidirectional long short-term memory networks for relation classification". In: *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 2: Short papers)*. 2016, pp. 207–212.

[53] X. Zhu, P. Sobihani, and H. Guo. "Long short-term memory over recursive structures". In: *International Conference on Machine Learning*. PMLR. 2015, pp. 1604–1612.

[54] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, P. J. Liu, et al. "Exploring the limits of transfer learning with a unified text-to-text transformer." In: *J. Mach. Learn. Res.* 21.140 (2020), pp. 1–67.

[55] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[56] Y. Sun, S. Wang, Y. Li, S. Feng, H. Tian, H. Wu, and H. Wang. "Ernie 2.0: A continual pre-training framework for language understanding". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 05. 2020, pp. 8968–8975.

[57] A. M. N. Allam and M. H. Haggag. "The question answering systems: A survey". In: *International Journal of Research and Reviews in Information Sciences (IJRRIS)* 2.3 (2012).

[58] A. Hassan and A. Mahmood. "Convolutional recurrent deep learning model for sentence classification". In: *Ieee Access* 6 (2018), pp. 13949–13957.

[59] N. Reimers and I. Gurevych. "Sentence-bert: Sentence embeddings using siamese bert-networks". In: *arXiv preprint arXiv:1908.10084* (2019).

[60] F. Schroff, D. Kalenichenko, and J. Philbin. "Facenet: A unified embedding for face recognition and clustering". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 815–823.

[61] L. L. Thurstone. "A law of comparative judgment." In: *Psychological Review* 34.4 (1927), pp. 273–286. DOI: 10.1037/h0070288. URL: https://doi.org/10.1037/h0070288.

[62] D. Pfitzner, R. Leibbrandt, and D. Powers. "Characterization and evaluation of similarity measures for pairs of clusterings". In: *Knowledge and Information Systems* 19.3 (2009), pp. 361–394.

[63] D. M. Powers. "Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation". In: *arXiv preprint arXiv:2010.16061* (2020).

[64] M. Hossin and M. N. Sulaiman. "A review on evaluation metrics for data classification evaluations". In: *International journal of data mining & knowledge management process* 5.2 (2015), p. 1.

[65] W. M. Rand. "Objective criteria for the evaluation of clustering methods". In: *Journal of the American Statistical association* 66.336 (1971), pp. 846–850.

[66] L. R. Dice. "Measures of the amount of ecologic association between species". In: *Ecology* 26.3 (1945), pp. 297–302.

[67] L. Leite, C. Rocha, F. Kon, D. Milojicic, and P. Meirelles. "A survey of DevOps concepts and challenges". In: *ACM Computing Surveys (CSUR)* 52.6 (2019), pp. 1–35.

[68] L. Chen. "Continuous delivery: Huge benefits, but challenges too". In: *IEEE software* 32.2 (2015), pp. 50–54.

[69] D. G. Feitelson, E. Frachtenberg, and K. L. Beck. "Development and deployment at facebook". In: *IEEE Internet Computing* 17.4 (2013), pp. 8–17.

[70] C. O'Hanlon. "A Conversation with Werner Vogels: Learning from the Amazon technology platform: Many think of Amazon as' that hugely successful online bookstore.'You would expect Amazon CTO Werner Vogels to embrace this distinction, but in fact it causes him some concern." In: *Queue* 4.4 (2006), pp. 14–22.

[71] N. F. Velasquez, G. Kim, N. Kersten, and J. Humble. "State of DevOps report". In: *Retrieved September* 19 (2014), p. 2021.

[72] H. B. Christensen. "Teaching DevOps and cloud computing using a cognitive apprenticeship and story-telling approach". In: *Proceedings of the 2016 ACM conference on innovation and technology in computer science education*. 2016, pp. 174–179.

[73] M. Shahin, M. A. Babar, and L. Zhu. "The intersection of continuous deployment and architecting process: practitioners' perspectives". In: *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. 2016, pp. 1–10.

[74] J. Wettinger, V. Andrikopoulos, and F. Leymann. "Automated capturing and systematic usage of devops knowledge for cloud applications". In: *2015 IEEE International Conference on Cloud Engineering*. IEEE. 2015, pp. 60–65.

[75] J. Wettinger, V. Andrikopoulos, and F. Leymann. "Enabling DevOps collaboration and continuous delivery using diverse application environments". In: *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer. 2015, pp. 348–358.

[76] H. Yasar and K. Kontostathis. "Where to integrate security practices on DevOps platform". In: *International Journal of Secure Software Engineering (IJSSE)* 7.4 (2016), pp. 39–50.

[77] J. Humble and D. Farley. *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education, 2010.

[78] J. Humble. "Continuous delivery vs continuous deployment". In: *Obtenido de https://continuousdelivery.com/2010/08/continuous-delivery-vscontinuousdeployment* (2010).

[79] G. G. Claps, R. B. Svensson, and A. Aurum. "On the journey to continuous deployment: Technical and social challenges along the way". In: *Information and Software technology* 57 (2015), pp. 21–31.

[80] J. Hernantes, C. Ebert, G. Gallardo, and N. Serrano. "Devops". In: *IEEE Softw* 33.3 (2016), pp. 94–100.

[81] M. Kersten. "A cambrian explosion of DevOps tools". In: *IEEE Software* 35.02 (2018), pp. 14–17.

[82] D. Shackleford. "A devsecops playbook". In: *SANS Institute* (2016).

[83] V. Mohan and L. B. Othmane. "Secdevops: Is it a marketing buzzword?-mapping research on security in devops". In: *2016 11th international conference on availability, reliability and security (ARES)*. IEEE. 2016, pp. 542–547.

[84] G. Bledsoe. *Getting to devsecops: 5 best practices for integrating security into your devops*. 2016. URL: `https://goo.gl/ZPzgxa`.

[85] F. Lim. *Devsecops is the krav maga of security*. 2016. URL: `https://goo.gl/BH4MS2`.

[86] S. Lietz. *Shifting security to the left*. 2016. URL: `https://goo.gl/sbheKS`.

[87] D. Shackleford. "The devsecops approach to securing your code and your cloud". In: *SANS Institute InfoSec Reading Room A DevSecOps Playbook* (2017).

[88] C. Caum. *Getting started with policy-driven development and devsecops*. 2016. URL: `https://goo.gl/AevVcX`.

[89] A. User. *Security breaks devops – here's how to fix it*. 2015. URL: `https://goo.gl/Yr1jk3`.

[90] W. Security. *Devops invites security to "join the party"*. 2016. URL: https://goo.gl/spj0wK.

[91] N. MacDonald and I. Head. "Devsecops: How to seamlessly integrate security into devops". In: *Gartner, Tech. Rep.* (2016).

[92] S. Vonnegut. *4 keys to integrating security into devops*. 2016.

[93] P. E. Black, P. E. Black, M. Kass, M. Koo, and E. Fong. *Source code security analysis tool functional specification version 1.0*. US Department of Commerce, National Institute of Standards and Technology, 2007.

[94] J. Zheng, L. Williams, N. Nagappan, W. Snipes, J. P. Hudepohl, and M. A. Vouk. "On the value of static analysis for fault detection in software". In: *IEEE transactions on software engineering* 32.4 (2006), pp. 240–253.

[95] A. Austin, C. Holmgreen, and L. Williams. "A comparison of the efficiency and effectiveness of vulnerability discovery techniques". In: *Information and Software Technology* 55.7 (2013), pp. 1279–1288.

[96] G. McGraw. "Software security". In: *Building security in* (2006).

[97] K. Goseva-Popstojanova and A. Perhinschi. "On the capability of static code analysis to detect security vulnerabilities". In: *Information and Software Technology* 68 (2015), pp. 18–33.

[98] T. Ball. "The concept of dynamic analysis". In: *Software Engineering—ESEC/FSE'99*. Springer. 1999, pp. 216–234.

[99] A. Gosain and G. Sharma. "A survey of dynamic program analysis techniques and tools". In: *Proceedings of the 3rd International Conference on Frontiers of Intelligent Computing: Theory and Applications (FICTA) 2014*. Springer. 2015, pp. 113–122.

[100] M. D. Ernst. *Static and dynamic analysis: Synergy and duality*. 2003.

[101] V. Gupta and J. K. Chhabra. "Measurement of dynamic metrics using dynamic analysis of programs". In: *Proceedings of the Applied Computing Conference*. Citeseer. 2008, pp. 81–86.

[102] J. Humble and J. Molesky. "Why enterprises must adopt devops to enable continuous delivery". In: *Cutter IT Journal* 24.8 (2011), p. 6.

[103] B. Fitzgerald and K.-J. Stol. "Continuous software engineering: A roadmap and agenda". In: *Journal of Systems and Software* 123 (2017), pp. 176–189.

[104] T. Greene. *What security teams need to know about devops*. 2016. URL: https://goo.gl/c8VOn4.

[105] K. Lindros. *How to craft an effective devsecops process with your team*. 2016. URL: https://goo.gl/ppWtjx.

[106] F. Angermeir, M. Voggenreiter, F. Moyón, and D. Mendez. "Enterprise-driven open source software: a case study on security automation". In: *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE. 2021, pp. 278–287.

[107] H. Booth, D. Rike, G. A. Witte, et al. "The national vulnerability database (nvd): Overview". In: (2013).

[108] A. Vector. URL: `http://searchsecurity.techtarget.com/dictionary/definition/1005812/attack%20-vector.html`.

[109] C. Simmons, C. Ellis, S. Shiva, D. Dasgupta, and Q. Wu. "AVOIDIT: A cyber attack taxonomy". In: *University of Memphis, Technical Report CS-09-003* (2009).