Superviz25-SQL: High-Quality Dataset to Empower Unsupervised SQL Injection Detection Systems

Grégor Quetel[1], Eric Alata[2], Pierre-François Gimenez[3], Thomas Robert[1], and Laurent Pautet[1]

[1] LTCI, Télécom Paris, Institut Polytechnique de Paris, Paris, France
{firstname.lastname}@telecom-paris.fr
[2] LAAS CNRS, Toulouse, France ealata@laas.fr
[3] Univ. Rennes, Inria, IRISA, Rennes, France
pierre-francois.gimenez@inria.fr

Abstract. The digitalization of public and private services has led to more sophisticated and serious cybersecurity threats. Among them, SQL injection attacks leverage user inputs to remotely execute malicious actions on a database, such as data exfiltration and deletion, or privilege escalation. They are regularly classified as one of the most prominent threats to web services. Intrusion detection systems are widely used to detect such injection attacks and react to them, but it is difficult to assess their actual effectiveness and compare them because of a lack of high-quality datasets. Current SQL injection detection datasets lack diversity, are poorly documented, and the generated samples are not representative of real-world infrastructures. This article presents a new dataset Superviz25-SQL[4], whose design is structured around four quality dimensions: realism, diversity, benchmarking capabilities and the presence of good documentation. We examine the dataset diversity using lexical, syntactic and semantic metrics, and demonstrate that its size is sufficient to evaluate data-intensive detectors. Finally, we provide nine classical and state-of-the art SQL injection detection pipelines as baselines for future works.

Keywords: Dataset · SQL injection · Anomaly detection

1 Introduction

Stating that the Internet has taken a central place in professional and private life would be an understatement. The financial and geopolitical incentives of attacking such online systems have never been greater: French ANSSI measured an augmentation of 15% of cybersecurity incidents between 2023 and 2024 [3]. Intrusion Detection Systems (IDS) play an important role in the fight against malicious entities due to their ability to detect and react to threats.

[4] The dataset is available at: https://zenodo.org/records/17086037

One important threat in online systems is injection-based attacks: they are present in the Top 10 for web application security risk published by OWASP [21] in 2017 and 2021, and have been attributed more than 30 Common Weaknesses Enumeration (CWEs). The most well-known example of this class of attacks is the SQL injection attack (SQLia), with 17,000+ Common Vulnerabilities and Exposures (CVEs) associated with it at the time of writing. SQL is a query language used to query a database. Attackers exploit a common pattern in web applications: the reliance on templates of queries in which user inputs are integrated. Injection vulnerabilities are caused by faulty applications that do not properly sanitize user inputs before their integration into developers' query templates. As an example, consider a website that allows users to log in with the following template without any sanity check on user inputs:

```
SELECT * FROM users WHERE login=' __1 ' AND password =' __2 '
```

This template has two blanks (i.e., placeholders) that can be filled with user inputs. However, an attacker could input `admin` in the first blank and `' OR 1=1--` in the second blank, leading to the following query:

```
SELECT * FROM users WHERE login='admin' AND password ='' OR 1=1--'
```

Since `--` starts a comment in MySQL (an SQL dialect), the executed query will not verify the password due to the `OR 1=1` tautology. This example is technically very simple, but much more complex SQLias exist.

Due to these attacks' prevalence and impact, many works proposed methods to detect SQLia [5, 7, 10, 11, 14, 16, 30, 31]. However, comparing these works is hindered by the lack of high-quality reference datasets, resulting in the usage of private datasets that cannot be shared for privacy and confidentiality reasons. Several network-oriented datasets contain SQL injections, such as CICIDS-2017 [25] or ISCX2012 [26], but their SQL traffic is encrypted and cannot be used to evaluate methods that analyse SQL query content. To the best of our knowledge, there are only two datasets dedicated to SQLias: Kaggle SQL Injection [24, 29] and WAF-A-MoLE [4]. However, they exhibit quality issues such as errors in their artefacts, undocumented design choices or unrealistic attack proportions. Thus, they cannot be used to assess the detection performance on real industrial systems, drastically reducing their usefulness.

In this article, we introduce Superviz25-SQL, a novel dataset for the evaluation of SQLia detection system with a focus on quality. More precisely, Superviz25-SQL focuses on being realistic, diverse, useful for benchmarking and well documented. Our dataset is explicitly constructed for unsupervised intrusion detection to reflect real-world constraints, where labelled attacks are rare and realistic attack campaigns can hardly be performed on existing systems. Additionally, this focus allows the detection of undocumented attacks and payloads. We also include insider attacks [12] that are not injection-based. Such attacks are rarely included in datasets, and by including such attacks, we aim to raise awareness of them in the IDS community.

The rest of the article is structured as follows: Section 2 reviews traditional SQLia detection locations and techniques. Section 3 summarizes dataset quality

criteria and discusses the limits of existing SQL injection detection datasets, followed by Section 4 detailing the problem statement. Section 5 describes the construction of Superviz25-SQL, emphasizing realism, diversity, benchmarking and proper documentation. Section 6 presents and discusses diversity metrics and evaluates the sufficiency of the dataset's size. Finally, Section 7 presents the performances of classical novelty detectors on Superviz25-SQL, provided as baselines for future works.

2 Background

2.1 Probes location

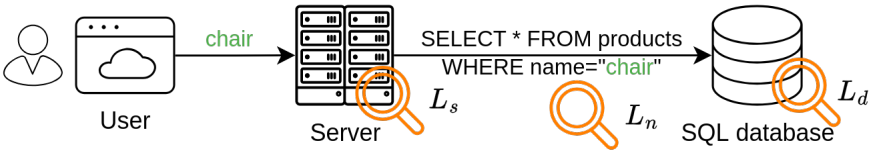


Fig. 1: The three different probe locations: L_s (server-side), L_n (network-side) and L_d (database-side).

Three traditional probe locations (L_s, L_n, and L_d) for SQL injection attack detection are presented in Fig. 1. In L_s, the probe is located on the Web server, where the user input and the query template can both be recorded. In L_n, the probe monitors network communications, which are typically encrypted. In L_d, the probe is located in the database, where the final query is visible, but the user input cannot be trivially distinguished from the template.

Datasets designed to evaluate network intrusion detection systems, such as CICIDS2017 [25], CICIDS2018 [26], or ISCX [18], locate their probe in L_n. They contain encrypted SQLias, so they can only be detected indirectly (for example, an exfiltration attack can significantly alter the network activity). In this work, we are interested in evaluating methods to detect SQLias from query contents, so we consider this probe location unsuited.

Locations L_s and L_d monitor different types of data. Observations from L_s can clearly distinguish user input from the query template: a detector can exploit the whole query as well as the user input for its prediction, simplifying the detection. Conversely, probes located on L_d capture all queries reaching the database, allowing the detection of attacks not originating from the server. However, the classifier cannot distinguish the user input from the query template written by the developer, making the detection more difficult.

2.2 SQL injection attack categories and sqlmap

SQLia alter the interaction between the server and the database. The outcome, such as data exfiltration or modification, depends on the application's level of

information disclosure and the structure of the vulnerable SQL template. These factors together determine which attack techniques can be used. We detail the most common techniques [23]. In applications collecting and disclosing all result sets of executed queries, **union-based attacks** use the `UNION ALL SELECT` keywords to exfiltrate information. **Stacked queries** rely on using a semicolon followed by an arbitrary malicious query. In applications leaking errors, **error-based attacks** expose subquery results via error messages. When applications do not disclose query results, but behave differently depending on whether a query succeeded or failed, **boolean-based attacks** and **time-based attacks** can infer the result of a query character by character. A last category of SQL attack is the **insider attack** [12] of a disgruntled employee, a malicious subcontractor, or an external attacker who acquired an initial access. Such an attacker has direct access to the database and does not need to exploit an injection vulnerability. They are rarely present in datasets but represent an important threat. Due to their nature, they can only be measured by probes located in L_n or L_d.

The state-of-the-art tool to identify and exploit SQLi vulnerabilities in web applications is `sqlmap` [28]. It performs an attack on a target URL with first, a recognition phase that consists of finding the correct prefix and suffix of payloads to craft syntactically valid queries. Then, the exploitation phase occurs: the tool retrieves the exploitation objective, such as the database banner, tables, schema, users, etc. To execute the attack, `sqlmap` employs the previously defined techniques along with an **inline attack**, which injects SQL queries directly into requests to identify vulnerable endpoints. Additionally, the tool implements a tamper-script feature that mutates the syntax of the attack payload with a semantically equivalent one to bypass potential detection systems. `sqlmap` can also generate insider attacks by providing the database credentials.

3 Related work

3.1 Dataset Quality

Several studies have examined the quality of datasets for network intrusion detection systems, notably [15], [6] and a more general article on datasets [8]. Based on these references, we distil dataset quality assessment into three dimensions: documentation, realism, and diversity, along with precise evaluation criteria, excluding those that are overly specific to network IDS datasets:

GenDoc dataset documentation should entail the entire generation process (and not only the attacks);
WLDoc the intended workload (i.e., benign queries) should be described;
Artefacts experimental artefacts should be transparently described;
Labels labelling should be error-free.
WLReal the intended workload should be realistic in terms of statement types, user-input values and query structure complexity;
AtkReal the attacker should use realistic attacks.
WLDiv the intended workload should be heterogeneous;

AtkDiv the attacker should use varied attacks.

Besides, with the surge of machine learning techniques used for anomaly detection, we consider the dataset benchmarking capabilities as another quality assessment dimension [6]. We derive the following criteria for this dimension:

Unbal The test dataset should not be balanced (as it could lead to over-optimistic precision and F1 scores);
FineLab the labels should be fine-grained: instead of *benign* or *malicious* labels, the attack types should be described as well as whether the attack was successful or not;
Size the dataset must be of enough size for novel machine-learning techniques that require considerable training data;

Several datasets containing SQL injection attacks have been proposed. Many network-oriented datasets, such as CICIDS-2017 [25], ISCX2012 [26], DAPT-2020 [18], CUPID [13] and Unraveled [19], contain SQLias among other attacks. However, the SQL payloads are encrypted, so they cannot be used to evaluate methods analysing user inputs or templates. In this work, we are interested in evaluating methods to detect SQL injection attacks from the queries, so we do not consider these datasets.

To the best of our knowledge, there exists no dataset to evaluate unsupervised SQLia detectors, and there are only two public datasets with clear-text queries for supervised detectors: Kaggle SQL Injection [24, 29] and WAF-A-MoLE [4].

3.2 Kaggle SQL Injection Dataset

The Kaggle SQL Injection dataset is frequently used to train detection mechanisms [7, 11, 16]. Two different versions of this dataset, [29] and [24], are commonly referenced. Both versions overlap significantly and contain approximately the same number of samples (30,873 and 30,905, respectively).

Despite being the most widely used, the latter version contains formatting issues caused by incorrect quoting of SQL queries, resulting in queries appearing in the label column and creating a third, mostly empty, column. It leads to the need for further preprocessing for the dataset users, which, when implemented differently, hinders the performance comparison of decision engines [6]. Moreover, the dataset provides no information about its origin or construction process, and fails to meet criteria **GenDoc**, **WLDoc**, and **Artefacts**. Since Kaggle SQL Injection provides no train-test split, users can partition the data freely. However, its roughly balanced default distribution may underestimate false positives, violating criterion **Unbal**.

The probe location of the Kaggle SQL Injection dataset is unclear: the dataset mixes both user input (such as `beecher@mailboxrussia.gu`) and complete SQL queries (such as `SELECT * FROM Orders WHERE OrderDate BETWEEN '1996-07-01' AND '1996-07-31'`). This inconsistency suggests an ambiguous data collection point comprised of observations made at L_s or L_d. This ambiguity highlights a lack of realism, preventing the correct evaluation of detection

techniques, and indicates a shortfall concerning criterion **WLReal**. Additionally, some (full) SQL queries labelled as "normal" data are syntactically invalid, which is difficult to justify and inconsistent with criterion **Labels**.

3.3 WAF-A-MoLE Dataset

The second dataset with both malicious and benign SQL statements, WAF-A-MoLE, was built by [4] to train IDS and then evaluate their mutation mechanism that seeks to bypass detection. The WAF-A-MoLE probe is located in the database (L_d), so the dataset only contains full queries. Samples were generated using `randgen`, `sqlmap` and `OWASP ZAP`. The generation process is documented, satisfying criteria **GenDoc**, **WLDoc**, and **Artefacts**, while attacks are generated using standard tools, fulfilling criterion **AtkReal**. Furthermore, the dataset is larger and supports training data-intensive models, thereby satisfying criterion **Size**.

The dataset's queries correspond to the most common SQL statements [2] but with a limited variety of structure limiting realism and diversity of both normal and attack samples, which is inconsistent with criteria **WLReal** and **WLDiv**. Furthermore, the dataset contains string escaping issues, which lead to the generation of queries that deviate from the described structure.

Additionally, we note that among the 345,199 benign and 393,345 attack statements parsed by `sqlparse` there is an intersection of 3,761 statements, which alters classification complexity [6] and suggests the presence of labelling inconsistencies, raising concerns with respect to the **Labels** quality criterion.

4 Problem Statement

The evaluation of SQLi detection systems heavily relies on dataset quality, but the existing datasets fall short across several quality criteria (cf. Table 1). Besides, both datasets are designed for supervised learning, which requires labelled data. This assumption is unrealistic in practice: labelled attacks are expensive to produce, and real-world systems must detect undocumented threats.

To address these limitations, we introduce Superviz25-SQL, a dataset specifically tailored for the evaluation of unsupervised SQLi detection systems. Its design is guided by various dataset quality dimensions, including the full documentation of the generation process, realism, and diversity of samples.

A central goal of Superviz25-SQL is to ensure high diversity in the generated samples. This diversity is quantitatively assessed across multiple aspects (lexical, syntactic, and semantic) against existing datasets in Section 6. Additionally, Section 6 assess whether the size of Superviz25-SQL is enough.

Finally, to foster progress in unsupervised SQLi detection research, we evaluate the performance of classical detection methods on Superviz25-SQL. Providing these baselines will facilitate the performance assessment of future work.

Criteria	Kaggle SQL Injection	WAF-A-MoLE	Superviz25-SQL
GenDoc	No	Yes	Yes
WLDoc	No	Yes	Yes
Artefacts	No, and errors	Yes, but errors	Yes
Labels	No	No	Minimized
WLReal	No	No	Yes
AtkReal	Unknown	Yes	Yes
WLDiv	Yes	Yes	Yes
AtkDiv	Unknown	Yes	Yes
Unbal	No	No	Yes
FineLab	No	No	Yes
Size	Not evaluated	Not evaluated	Yes

Table 1: Comparison of Kaggle SQL Injection, WAF-A-MoLE and Superviz25-SQL using dataset quality criteria.

5 Dataset Generation

Following the recommendation **GenDoc**, this section offers a comprehensive overview of the Superviz25-SQL generation process and the corresponding design decisions undertaken to produce a high-quality dataset with respect to realism, diversity, and benchmarking capabilities. Design choices relevant to the documentation criterion are also discussed throughout the section. We conclude with an analysis of the generated dataset.

5.1 Generation Overview

Superviz25-SQL's generation process is summarized in Fig. 2. The produced dataset is centred on a single information system corresponding to a specific deployment of MySQL. As such, it reflects a scenario in which a security administrator can access benign traffic associated with this particular deployment. While incorporating multiple SQL dialects could increase diversity, Superviz25-SQL is intended to reflect the traffic of a single application deployment, where such variation would be unrealistic.

We base the dataset on the database schema provided by the OurAirports project[5]. The project reports airport infrastructure around the world and was selected for its real-world usage, well-documented database schema, and the availability of publicly accessible data dumps. A generator relying on defined templates has been designed to create both normal and attack samples. The generator creates benign queries by filling placeholders (such as `{countries_code}` in Fig. 3) with values obtained from the OurAirports project dumps. For example, the ISO country code `FI` is a value the generator can select to produce a legitimate query.

[5] https://ourairports.com

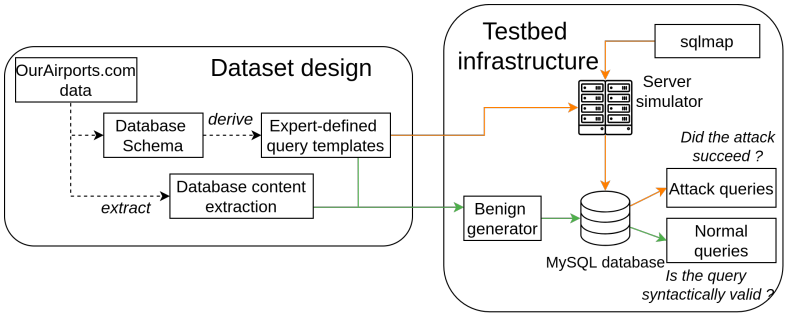


Fig. 2: Dataset generation process: design phase with the extraction of OurAirports values and the design of query templates. Then, the testbed leverages query templates, extracted values, and `sqlmap` to generate samples.

```
SELECT r.name AS region_name, COUNT(a.id) AS airport_count
    FROM airport a JOIN regions r ON a.iso_region = r.code
    WHERE a.iso_country = {countries_code} GROUP BY r.name
    ORDER BY r.name ASC;
```

Fig. 3: `SELECT` statement template used in Superviz25-SQL generation. The placeholder `{countries_code}` is dynamically replaced with either a real value extracted from the OurAirports project or with payloads generated by `sqlmap`.

The first step of the generation consists of analysing the project database schema and extracting the values populating the database. The database schema[6] comprises six tables: `airports`, `airport-frequencies`, `runways`, `countries`, `navaids`, and `regions`. We derived from the schema query templates in the MySQL dialect, simulating the behaviour of typical web applications. In total, we defined 62 SQL templates covering a variety of statement types: 23 `SELECT`, 10 `UPDATE`, 10 `INSERT`, 8 `DELETE`, and 11 administrative queries involving user and privilege management. Figure 3 shows one of the derived query templates that accepts an ISO country code as user input and retrieves regional airport statistics through a multi-table join operation.

The widely recognized tool for automated SQL Injection exploitation `sqlmap` is used to create attack samples. We intentionally avoided using human pentesters to ensure the dataset could be automatically reproduced. We simulate an HTTP endpoint for each query template and conduct automated attack campaigns leveraging all `sqlmap` techniques. Each attack campaign consists of targeting an endpoint with a given technique and an exploitation objective. Specifically, the reconnaissance phase consists of multiple `sqlmap` invocations: one per query parameter. If reconnaissance reveals a vulnerable parameter, the exploitation phase is run. The exploitation objective is set to extract all data from the

[6] https://ourairports.com/help/data-dictionary.html

database for all techniques except time-based and stacked queries, to which we set smaller objectives due to their slower execution. Insider attack queries were generated using `sqlmap` in direct connection mode, with four campaigns conducted using different exfiltration objectives.

The generation process led to the creation of a training set containing 335,306 benign queries and a test set composed of 336,281 (i.e., 10%) malicious queries, along with 3,017,390 benign queries (i.e., 90%). Table 6, provided in Appendix A enumerates the fields provided for each entry, their description, and examples of benign and attack samples in the dataset.

5.2 Realism criteria

Realism directly impacts the applicability of research to real-world systems. Unrealistic data, either benign or malicious, can lead to misleading evaluations and reduced generalizability [6, 15].

In our dataset, the realism of the benign traffic stems from the use of query templates derived from a real-world application's schema and from placeholder values populated using data extracted from the OurAirports project. Anderson et al. [2] analyzed SQL query patterns in PHP web applications, reporting 62.5% SELECT, 9.4% UPDATE, 8.7% INSERT, 7.7% DELETE, and 12.3% other queries. Acting in accordance with criteria **WLDoc** and **WLReal**, we propose sampling benign queries similar to these figures. Our final workload is composed of 70% SELECT, 10% UPDATE, 10% INSERT, 9% DELETE, and 1% administrative queries. The latter reflects typical MySQL console operations. Finally, a MySQL server verifies all benign queries for syntactic correctness.

The generation of attacks relies on the widely used SQL injection exploitation tool, `sqlmap`, meeting criterion **AtkReal**. To better emulate real-world conditions, where only some endpoints are typically vulnerable or made public, we limit `sqlmap` attack campaigns to a randomly selected subset of 45 query templates. For each simulated vulnerable endpoint, six distinct attack campaigns are run, each using a different `sqlmap` technique, totalling 270 campaigns. Table 7 provided in Appendix B, summarizes the success rates of the different SQLias techniques used and the number of resulting samples. To minimize the labelling errors that could harm the correct evaluation of detection mechanisms and adhere to criterion **Labels**, we exclude initial `sqlmap` queries used solely for connectivity testing that usually do not contain malicious payloads. Furthermore, in line with the transparency principle emphasized in criterion **Artefacts**, we acknowledge that not all queries generated by `sqlmap` can be confirmed to contain malicious payloads. Accordingly, we found and removed from the dataset 8233 samples with contradictory labels (i.e., identical queries found in both malicious and benign sets).

5.3 Diversity criteria

A diverse dataset is essential for evaluating the generalization capabilities of detection models. Limited diversity can lead to overfitting and reduced robust-

ness [6]. In line with criteria **WLDiv** and **AtkDiv**, we foster syntactic and semantic diversity of samples by designing query templates such that each table in the OurAirports schema is represented by at least one template per major SQL statement type. Both aspects are closely linked as the impact of the query is dictated by how the input will be parsed. Syntactic variety is further enhanced by incorporating templates performing joins across multiple tables. Lexical diversity in queries arises from different placeholder types such as floats (coordinates), integers (identifiers), strings (airport codes, names), keyword lists, or URLs.

Benign samples are generated by randomly selecting authentic database values, according to their distribution in the OurAirports project. In contrast, variability is introduced for malicious samples through the different `sqlmap` techniques: both keywords and query structure will vary. Lexical diversity is further increased by applying a subset of tamper scripts (such as `sleep2getlock`, `equaltolike` or `randomcase`). We filtered out those not compatible with the MySQL dialect and those breaking payloads. Additionally, we randomized the value of non-attacked HTTP parameters using OurAirports values, provided to `sqlmap` through the `--eval` option.

The diversity is empirically evaluated in Section 6.

5.4 Benchmarking capabilities criteria

Superviz25-SQL is designed to support the evaluation and comparison of SQL injection detection approaches. As outlined in Section 2, detection can occur at multiple observation points. Accordingly, we provide both L_s (user input) and L_d (SQL query) representations for each sample, enabling comparison across various detection architectures. Following the recommendation of [6], we report the success or failure of each attack campaign based on whether exfiltration was achieved. For fine-grained performance analysis, attack samples are annotated with the associated technique and tamper script, addressing criterion **FineLab**. Furthermore, consistent with criterion **Unbal**, our test set reflects a realistic class imbalance, containing 90% benign and 10% attack samples.

To foster reproducibility in detector evaluation, we provide a ready-to-use train/test split. We give usage instructions and a summary of the dataset's contents in the datasheet [8] included with the artefact. To ensure the reproducibility of dataset generation, we also release the complete generation pipeline[7], along with a Docker-based environment, in compliance with the **GenDoc** criterion.

Finally, the **Size** criterion is evaluated in the following section.

6 Dataset Evaluation

The realism of the database schema and templates stems from their basis in the OurAirports project, while the realism of the attacks arises from the use of a genuine attack tool. However, the diversity of the dataset is not guaranteed by design, nor that its size is sufficient. This section evaluates both.

[7] https://github.com/gquetel/sqlia-dataset-generator

6.1 Diversity Evaluation

Inspired by recent work in natural language processing [20], we propose to characterize the diversity of SQL queries by *the variability of the data across specific dimensions (e.g., semantic, syntactic, lexical, SQL dialect)*. Dialect diversity was purposely not addressed in generating this application-specific dataset, resulting in a single covered dialect: MySQL. However, to evaluate our alignment with criteria **WLDiv** and **AtkDiv**, we propose the following metrics:

- *Vocabulary size*: The number of tokens in the vocabulary constructed from all samples in each dataset is used to evaluate their lexical diversity. In practice, we use the vocabulary constructed by CountVectorizer provided by Scikit-learn [22].
- *Number of Unique Parse Trees*: Syntactic diversity is examined through the computation of the number of unique parse trees in each dataset. We use `sqlglot` [17] to compute the parse trees. The tool does not allow to retrieve the incomplete parse trees that failed to parse. Some attack parse trees are therefore not studied.
- *Semantic Space Dispersion*: Inspired by [9], we assess the semantic diversity of generated queries by measuring the average pairwise cosine distance between their embeddings in the semantic space of a Sentence-BERT model. We use Secure-BERT [1], a pre-trained model fine-tuned on cybersecurity-specific corpora, to obtain these embeddings. This metric is computed using 10,000 randomly sampled queries per class for each dataset.

Dataset	Sample Type	Number of Samples	Vocabulary Size	Unique Parse Trees	Semantic Space Dispersion
Superviz25-SQL	Normal	3,352,696	**459,215**	**58,891**	**0.01197**
WAF-A-MoLE		345,199	332,827	15,131	0.00623
Kaggle SQL Injection		19,537	14,471	515	0.01156
Superviz25-SQL	Attack	336,281	**119,721**	5,133	**0.01255**
WAF-A-MoLE		393,345	12,669	**7,336**	0.00991
Kaggle SQL Injection		11,382	10,805	242	0.01099

Table 2: Diversity Metrics Across Superviz25-SQL, WAF-A-MoLE, and Kaggle SQL Injection datasets.

In Table 2, we display the lexical, syntactic and semantic diversity metrics of malicious and benign samples for Superviz25-SQL, WAF-A-MoLE, and Kaggle SQL Injection. Regarding the lexical diversity of normal queries, Superviz25-SQL has the largest vocabulary (459,215 tokens). While WAF-A-MoLE's vocabulary size is similar (332,827), much of it comprises meaningless strings. Its attack queries have far lower lexical diversity than ours (12,669 vs. 119,721). Kaggle

SQL Injection, due to its smaller dataset size, exhibits lower vocabulary sizes: 14,471 for normal queries and 10,805 for attacks.

Syntactically, Superviz25-SQL shows the most diverse normal queries with 58,891 unique parse trees, largely due to one template simulating form-based searches with multiple logical conditions. This template was not used for attack queries, leading to a lower diversity (5,133 trees). In WAF-A-MoLE, syntactic variety originates from the usage of logical conditions, resulting in 15,131 normal and 7,336 attack parse trees. Kaggle SQL Injection exhibits the lowest diversity overall (515 for normal and 242 for attacks), attributable to the heterogeneous nature of the dataset.

In terms of semantic diversity, Superviz25-SQL yields the highest scores for both attack and benign queries (0.01197 and 0.01255). Kaggle SQL Injection exhibits similar scores (0.01099 and 0.01156), again explained by the dual nature of its samples. WAF-A-MoLE shows the lowest diversity (0.00623 and 0.00991), likely due to their usage of meaningless strings and similar query patterns.

Our finding indicates that Superviz25-SQL demonstrates high lexical, syntactic and semantic diversity amongst the compared datasets, particularly for normal queries. Notably, it achieves this despite using realistic query templates, genuine database values and a real-world attack generation tool, all of which typically constrain diversity.

6.2 Dataset Size Evaluation

We perform further study of Superviz25-SQL to examine the compliance of criterion **Size**, i.e, whether its size is sufficient to evaluate SQLia novelty detectors. This question is addressed by evaluating the performance of the most performant detection baseline (later detailed in Section 7) on a bigger train set. If a significant performance improvement is observed, this would suggest that the original Superviz25-SQL training set does not fully capture the range of normal behaviour. Conversely, if the performance remains unchanged, it would indicate that the detectors have reached a performance plateau, implying that the training dataset is of sufficient size.

The detection baseline uses Secure-BERT [1] paired with an Autoencoder for novelty detection (presented in the next section). We study the performance of the pipeline trained on the standard Superviz25-SQL training set versus a larger variant, Big-Superviz25-SQL, which is twice as big. Big-Superviz25-SQL is built upon the original Superviz25-SQL samples, augmented with an equivalent number of new benign samples. In both experiments, the test set is the same.

Table 3 reports the performance of the baseline on both Superviz25-SQL and Big-Superviz25-SQL. The lack of major improvements in AUPRC (-0.0316) and AUROC (-0.0003) when training on Big-Superviz25-SQL suggest that the model captured the normal behaviour with the original Superviz25-SQL. These results indicate diminishing returns from adding more data. Therefore, Superviz25-SQL appears sufficiently large for describing the system.

Dataset	Precision	Recall	F1	FPR	AUPRC	AUROC
Big-Superviz25-SQL	98.08%	88.20%	92.88%	0.192%	0.9260	0.9881
Superviz25-SQL	99.07%	84.44%	91.17%	0.09%	0.9576	0.9884
Performance Difference	-0.99%	+3.76%	+1.71%	+0.10%	-0.0316	-0.0003

Table 3: Secure-BERT + Autoencoder Performance on Superviz25-SQL vs. Big-Superviz25-SQL

7 Reference Experimental Protocol and Baselines

This section provides the reference experimental protocol for using Superviz25-SQL, as well as the performance of nine classical and state-of-the-art detection pipelines as baselines for future work.

Each pipeline consists of a feature extraction mechanism combined with a novelty detection model. We study the detection performance using the full queries (i.e., assuming a probe in L_d). The process is similar for a probe in L_s, but both administrative queries and insider attacks should be dismissed because they cannot be observed from L_s. Hyperparameters follow default configurations, and all implementations are publicly available in our repository[8]. We first detail the preprocessing pipelines, novelty detectors and used metrics, then discuss the overall model performances and per-attack recall.

Feature extraction techniques

- *CountVectorizer*: This feature extraction method provided by Scikit-learn [22] is representative of word-count-based approaches employed for SQL injection detection [27, 31]. It builds a vocabulary from training tokens and represents each sample as a vector with the occurrence count for each vocabulary token. Vocabulary size is unrestricted except when used with the Autoencoder detector, whose vocabulary is limited to 20,000 to avoid excessive network size.
- *Manually Selected Features (Li)*: Based on the work of Li et al. [14], this approach extracts handcrafted lexical and syntactic features tailored for SQL queries. These include pattern matching or token count. The features are normalized using Scikit-learn's StandardScaler [22] to ensure zero mean and unit variance before being passed to the detectors.
- *Secure-BERT*: This method represents a state-of-the-art feature extraction approach, leveraging Sentence-BERT-based architectures. Specifically, it employs Secure-BERT [1], a model fine-tuned on cybersecurity-related content, to produce fixed-size embeddings of SQL queries.

Novelty detectors

- *Autoencoder (AE)*: A neural network trained to reconstruct its input from a compressed latent representation. Samples with high reconstruction error are

[8] https://github.com/gquetel/sqlia-dataset-generator

considered anomalous. Our PyTorch implementation uses two encoding layers, reducing the dimension to one-third of the input's one and two decoding layers. This implementation is available in our repository.
- *Local Outlier Factor (LOF)*: A density-based method that quantifies the local deviation of a given sample with respect to its neighbours. The Scikit-learn [22] implementation with default parameters is used.
- *One-Class Support Vector Machines (OCSVM)*: An algorithm that learns a decision boundary around the training data in high-dimensional space. Samples outside this boundary are considered anomalies. The Scikit-learn [22] implementation with default parameters is used.

Metrics

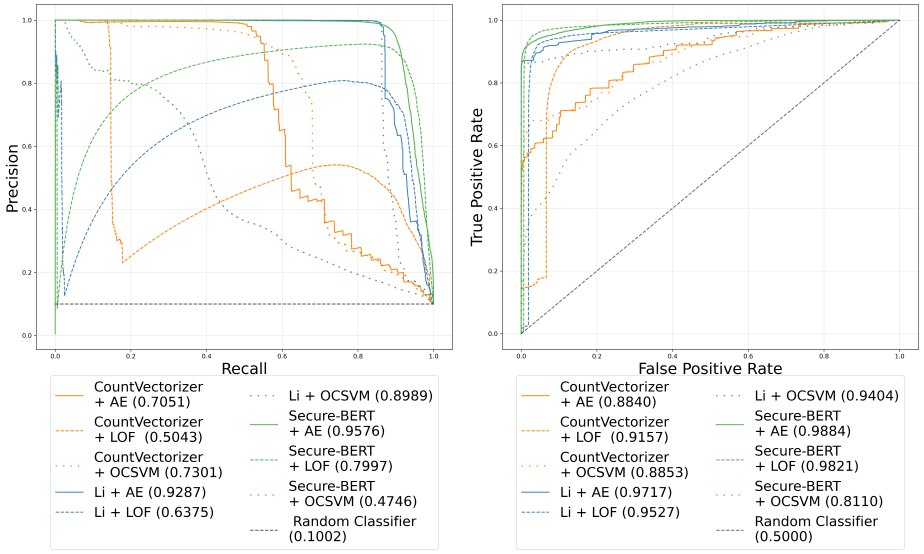
The detection performances of the different pipelines are evaluated with the Receiver Operating Characteristic curves and the Precision-Recall curves. The Area under the ROC curve (AUROC) metric evaluates the trade-off between true positive and false positive rates across different thresholds. The Area Under the Precision-Recall Curve (AUPRC) focuses on the precision-recall trade-off and emphasizes the model's ability to identify the minority class correctly.

We also compute standard intrusion detection metrics: precision (the proportion of alerts that correspond to an attack), recall (the proportion of attacks that are detected), F1 score (harmonic mean of precision and recall), and false positive rate (the proportion of benign queries that trigger an alert). Metrics are evaluated on the full test set using thresholds tuned to a maximum tolerated FPR of 0.1% on a validation set comprising 10% of the training data.

7.1 Results

AUPRC and AUROC analysis Figure 4b presents the ROC curves for all nine detection pipelines, along with their AUROC scores. Across all pipelines, AUCROC scores remain relatively high; the lowest AUCROC (0.8110) is observed with the Secure-BERT + OCSVM pipeline, suggesting that models are generally effective at separating benign from malicious queries. In particular, for low FPR values (which are often required in real-world applications), Secure-BERT + AE, Li + AE and Li + OCSVM can obtain about 85% recall. However, the other methods cannot reach a high recall without a large FPR. This analysis of the ROC curve is consistent with the PR curve, where only these three methods can achieve high recall with a very high precision. Regarding AUPRC scores shown in Figure 4a, the best performance was achieved by the Secure-BERT + AE pipeline with an AUPRC of 0.9576, followed closely by pipelines using manually engineered features from Li et al. [14]. However, other Secure-BERT pipelines show reduced effectiveness and those based on CountVectorizer yielded the weakest results (AUPRC of 0.4746 and 0.7988).

F1 score analysis Table 4 shows the F1 score, precision, recall, FPR metrics for each pipeline given a maximum FPR on the validation dataset. We observe



(a) PR curves. (b) ROC curves.

Fig. 4: Performance curves of the pipelines on Superviz25-SQL. The AUC scores are shown in parentheses for each pipeline.

Model	Precision	Recall	F1	FPR	AUPRC	AUROC
CountVectorizer + AE	98.59%	50.65%	66.92%	0.08%	0.7051	0.8840
CountVectorizer + LOF	91.82%	14.35%	24.82%	0.15%	0.5043	0.9157
CountVectorizer + OCSVM	97.67%	33.67%	50.08%	0.09%	0.7301	0.8853
Li + AE	98.56%	86.00%	91.85%	0.14%	0.9287	0.9717
Li + LOF	79.28%	1.58%	3.09%	0.06%	0.6375	0.9527
Li + OCSVM	98.97%	85.88%	91.96%	0.10%	0.8989	0.9404
Secure-BERT + AE	99.07%	84.44%	91.17%	0.09%	0.9576	0.9884
Secure-BERT + LOF	33.66%	0.52%	1.02%	0.11%	0.7997	0.9821
Secure-BERT + OCSVM	90.99%	6.91%	12.85%	0.08%	0.4746	0.8110

Table 4: Performance metrics for studied novelty detection pipelines.

high F1 scores for pipelines using Autoencoder (91.85% with Li, 91.17% with Secure-BERT). In contrast, performance sharply drops for other configurations, primarily due to lower precision or recall caused by our low FPR threshold setting.

Recall per attack analysis The attack technique column included with each attack sample allows computing per-technique detection rates, enabling deeper analysis of pipeline performance. Table 5 presents recall scores per technique for the pipelines with the best F1 score (from which we can deduce trends). Boolean-based and union-based attacks are the most reliably detected across

Model	Union	Boolean	Stacked	Error	Time	Inline	Insider
CountVectorizer + AE	37.20%	53.37%	6.57%	25.61%	14.52%	1.87%	0.00%
Li + AE	93.37%	91.33%	65.97%	79.80%	59.61%	74.30%	79.61%
Li + OCSVM	93.77%	91.49%	67.12%	77.04%	61.98%	66.74%	73.74%
Secure-BERT + AE	83.14%	92.30%	63.47%	78.14%	62.11%	75.95%	99.27%

Table 5: Recall score per technique for the best performing detection pipelines.

all pipelines. Pipelines using Secure-BERT or Li feature extraction show similar detection rates for most techniques, with time-based attacks being the least detected (recall between 59.61% and 62.11%). Dictionary-based pipelines show weaker performance, with recall decreasing down to 1.87% with the Autoencoder and no identified insider attack query. Conversely, insider attacks are correctly identified by the other pipelines with a recall score reaching 99.27% with Secure-BERT + AE. In summary, pipelines based on Autoencoders get good detection performances, specifically when combined with the state-of-the-art feature extraction using Secure-BERT [1] or domain-specific features [14]. Nevertheless, no pipeline achieves perfect detection. Some SQLia techniques remain difficult to identify, and false positive rates are still high, limiting these methods' applicability in the industry. These results advocate for developing new, more robust feature extraction techniques and novelty detectors.

8 Conclusion

SQL Injections are an emblematic threat in web applications. A plethora of papers propose new detection architectures, but the usage of private datasets or faulty ones hinders their evaluation. We propose Superviz25-SQL, a SQL Injection Detection dataset for unsupervised detection. The design choices of the dataset are guided by four key quality dimensions: sample realism, diversity, benchmarking capabilities, and comprehensive documentation. The generation code, a reproducible environment and further detailed documentation are made available in our repository[9]. We assess dataset diversity using lexical, syntactic, and semantic metrics, and show that its size supports the evaluation of data-intensive detectors. The performances of common feature extraction techniques coupled with novelty detectors are provided as baselines. Future work will explore the transferability of detection models trained on generic datasets that include different database schemas and dialects.

Acknowledgements This work has been partially supported by the French National Research Agency under the France 2030 label (Superviz ANR-22-PECY-0008). The views reflected herein do not necessarily reflect the opinion of the French government.

[9] https://github.com/gquetel/sqlia-dataset-generator

9 Appendix

A Dataset Columns Details

Table 6 lists the columns characterizing each sample Superviz25-SQL. For each column, a description is provided along with its possible values, accompanied by examples of both an attack and a benign sample.

Column	Benign Sample	Attack Sample	Description
full query	DELETE FROM runways WHERE airport_ident = "KTUL";	SELECT * FROM airport WHERE icao_code = 'UKKK' AND 5487=6284 OR 'jmGH'='ZnfX'	The full statement, as observed in L_d
label	0	1	Benign (0) or attack (1)
user inputs	KTUL	UKKK' AND 5487=6284 OR 'jmGH'='ZnfX	The user input without the template, as observed in L_s
attack stage		recon	Corresponds to `sqlmap`'s attack stage, either 'recon' or 'exploit'.
tamper method		sleep2getlock	The sqlmap tamper-script used for this sample
attack status		success	The sqlmap attack campaign status, either 'success' or 'failure'
statement type	delete	delete	Statement type, either 'select','delete', 'execute', 'update', 'admin' or 'insider'.
query template id	airport-D5	airport-S1	The expert-defined template ID from which the sample was generated
attack id		boolean-156	Attack campaign identifier.
attack technique		boolean	The technique used by `sqlmap`, either 'boolean', 'error', 'inline', 'stacked', 'time' 'union' or 'insider'.
split	test	test	Proposed split for the dataset, either 'train' or 'test'

Table 6: Overview of the Superviz25-SQL columns that characterise each sample. Each column is described, along with example values for both a benign and an attack sample.

B Attack Campaigns Details

Table 7 shows the number of successful attack campaigns per SQL Injection technique (i.e. whether the `sqlmap` exfiltration step was achieved). For each

attack technique, we indicate the number of attack samples generated. We also provide this information for the four insider attack campaigns generated via direct access.

Attack Type	Successful Campaigns	Success Rate	Total Samples
Error-based	45	100%	40,995
Time-based	39	86.7%	24,713
Boolean-based	35	77.8%	203,845
Stacked queries	30	66.7%	24,659
Union-based	15	33.3%	39,591
Inline queries	5	11.1%	1,389
Insider attacks	4	100%	1,089

Table 7: Detail of success rate for the different `sqlmap` attack campaigns in Superviz25-SQL and the resulting number of samples.

References

1. Aghaei, E., Niu, X., Shadid, W., Al-Shaer, E.: SecureBERT: A Domain-Specific Language Model for Cybersecurity. In: Li, F., Liang, K., Lin, Z., Katsikas, S.K. (eds.) Security and Privacy in Communication Networks. pp. 39–56. Springer Nature Switzerland, Cham (2023). https://doi.org/10.1007/978-3-031-25538-0_3
2. Anderson, D.: Modeling And Analysis Of SQL Queries in PHP Systems. Master's thesis, East Carolina University (2018)
3. ANSSI: Panorama de la cybermenace 2024. https://www.cert.ssi.gouv.fr/cti/CERTFR-2025-CTI-003/ (2025), Accessed 7 August 2025
4. Demetrio, L., Valenza, A., Costa, G., Lagorio, G.: WAF-A-MoLE: Evading web application firewalls through adversarial machine learning. In: Proceedings of the 35th Annual ACM Symposium on Applied Computing. pp. 1745–1752. ACM, Brno Czech Republic (Mar 2020). https://doi.org/10.1145/3341105.3373962
5. Deva Priyaa, B., Devi, M.I.: Fragmented query parse tree based SQL injection detection system for web applications. In: 2016 International Conference on Computing Technologies and Intelligent Data Engineering (ICCTIDE'16). pp. 1–5 (2016). https://doi.org/10.1109/ICCTIDE.2016.7725367
6. Flood, R., Engelen, G., Aspinall, D., Desmet, L.: Bad Design Smells in Benchmark NIDS Datasets. In: 2024 IEEE 9th European Symposium on Security and Privacy (EuroS&P). pp. 658–675 (Jul 2024). https://doi.org/10.1109/EuroSP60621.2024.00042
7. Floris, G., Scano, C., Montaruli, B., Demetrio, L., Valenza, A., Compagna, L., Ariu, D., Piras, L., Balzarotti, D., Biggio, B.: ModSec-AdvLearn: Countering Adversarial SQL Injections with Robust Machine Learning. IEEE Trans.Inform.Forensic Secur. **20**, 6693–6705 (2025). https://doi.org/10.1109/TIFS.2025.3583234
8. Gebru, T., Morgenstern, J., Vecchione, B., Vaughan, J.W., Wallach, H., III, H.D., Crawford, K.: Datasheets for Datasets (Dec 2021). https://doi.org/10.48550/arXiv.1803.09010

9. Guo, Y., Shang, G., Vazirgiannis, M., Clavel, C.: The Curious Decline of Linguistic Diversity: Training Language Models on Synthetic Text. In: Findings of the Association for Computational Linguistics: NAACL 2024. pp. 3589–3604. Association for Computational Linguistics, Mexico City, Mexico (2024). https://doi.org/10.18653/v1/2024.findings-naacl.228
10. Hasan, M., Balbahaith, Z., Tarique, M.: Detection of SQL Injection Attacks: A Machine Learning Approach. In: 2019 International Conference on Electrical and Computing Technologies and Applications (ICECTA). pp. 1–6. IEEE, Ras Al Khaimah, United Arab Emirates (Nov 2019). https://doi.org/10.1109/ICECTA48151.2019.8959617
11. Kakisim, A.G.: A deep learning approach based on multi-view consensus for SQL injection detection. Int. J. Inf. Secur. **23**(2), 1541–1556 (Apr 2024). https://doi.org/10.1007/s10207-023-00791-y
12. Khan, M.I., Foley, S.N., O'Sullivan, B.: Database Intrusion Detection Systems (DIDs): Insider Threat Detection via Behaviour-Based Anomaly Detection Systems - A Brief Survey of Concepts and Approaches. In: Meng, W., Katsikas, S.K. (eds.) Emerging Information Security and Applications. pp. 178–197. Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-030-93956-4_11
13. Lawrence, H., Ezeobi, U., Tauil, O., Nosal, J., Redwood, O., Zhuang, Y., Bloom, G.: CUPID: A labeled dataset with Pentesting for evaluation of network intrusion detection. Journal of Systems Architecture **129**, 102621 (Aug 2022). https://doi.org/10.1016/j.sysarc.2022.102621
14. Li, Q., Li, W., Wang, J., Cheng, M.: A SQL Injection Detection Method Based on Adaptive Deep Forest. IEEE Access **7**, 145385–145394 (2019). https://doi.org/10.1109/ACCESS.2019.2944951
15. Liu, J., Inam, M.A., Goyal, A., Riddle, A., Westfall, K., Bates, A.: What We Talk About When We Talk About Logs: Understanding the Effects of Dataset Quality on Endpoint Threat Detection Research. In: 2025 IEEE Symposium on Security and Privacy (SP). pp. 112–129. IEEE Computer Society (Apr 2025). https://doi.org/10.1109/SP61157.2025.00112
16. M, G., H B, P.: Semantic Query-Featured Ensemble Learning Model for SQL-Injection Attack Detection in IoT-Ecosystems. IEEE Trans. Rel. **71**(2), 1057–1074 (Jun 2022). https://doi.org/10.1109/TR.2021.3124331
17. Mao, T.: Sqlglot. https://github.com/tobymao/sqlglot (2025)
18. Myneni, S., Chowdhary, A., Sabur, A., Sengupta, S., Agrawal, G., Huang, D., Kang, M.: DAPT 2020 - Constructing a Benchmark Dataset for Advanced Persistent Threats. In: Wang, G., Ciptadi, A., Ahmadzadeh, A. (eds.) Deployable Machine Learning for Security Defense, vol. 1271, pp. 138–163. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-59621-7_8
19. Myneni, S., Jha, K., Sabur, A., Agrawal, G., Deng, Y., Chowdhary, A., Huang, D.: Unraveled — A semi-synthetic dataset for Advanced Persistent Threats. Computer Networks **227**, 109688 (May 2023). https://doi.org/10.1016/j.comnet.2023.109688
20. Nguyen, D., Ploeger, E.: We Need to Measure Data Diversity in NLP – Better and Broader (May 2025). https://doi.org/10.48550/arXiv.2505.20264
21. OWASP Foundation: OWASP Top Ten. https://owasp.org/www-project-top-ten/ (2021), Accessed 7 August 2025
22. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, É.: Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research **12**(85), 2825–2830 (2011). https://doi.org/10.48550/arXiv.1201.0490

23. Polop, C.: HackTricks. https://book.hacktricks.wiki/en/pentesting-web/sql-injection/index.html (2025), Accessed 7 August 2025
24. sajid576: SQL Injection Dataset. https://www.kaggle.com/datasets/sajid576/sql-injection-dataset (2022), Accessed 7 August 2025
25. Sharafaldin, I., Habibi Lashkari, A., Ghorbani, A.A.: Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization:. In: Proceedings of the 4th International Conference on Information Systems Security and Privacy. pp. 108–116. SCITEPRESS - Science and Technology Publications, Funchal, Madeira, Portugal (2018). https://doi.org/10.5220/0006639801080116
26. Shiravi, A., Shiravi, H., Tavallaee, M., Ghorbani, A.A.: Toward developing a systematic approach to generate benchmark datasets for intrusion detection. Computers & Security **31**(3), 357–374 (May 2012). https://doi.org/10.1016/j.cose.2011.12.012
27. Singh, K., Kokardekar, S., Khonde, G., Dekate, P., Badkas, N., Lachure, S.: Cloud Engineering-based on Machine Learning Model for SQL Injection Attack. In: 2023 International Conference on Communication, Circuits, and Systems (IC3S). pp. 1–6 (May 2023). https://doi.org/10.1109/IC3S57698.2023.10169533
28. sqlmapproject: Sqlmap. https://github.com/sqlmapproject/sqlmap (Apr 2024), Accessed 7 August 2025
29. Syed Saqlain Hussain Shah: Sql injection dataset. https://www.kaggle.com/datasets/syedsaqlainhussain/sql-injection-dataset (2021), Accessed 7 August 2025
30. Zhuo, Z., Cai, T., Zhang, X., Lv, F.: Long short-term memory on abstract syntax tree for SQL injection detection. IET Software **15**(2), 188–197 (2021). https://doi.org/10.1049/sfw2.12018
31. Zulu, J., Han, B., Alsmadi, I., Liang, G.: Enhancing Machine Learning Based SQL Injection Detection Using Contextualized Word Embedding. In: Proceedings of the 2024 ACM Southeast Conference on ZZZ. pp. 211–216. ACM, Marietta GA USA (Apr 2024). https://doi.org/10.1145/3603287.3651187