





# TerraDS: A Dataset for Terraform HCL Programs

Christoph Bühler\*  University of St. Gallen  
St. Gallen, Switzerland christoph.buehler@unisg.ch

David Spielmann\*  University of St. Gallen  
St. Gallen, Switzerland david.spielmann@unisg.ch

Roland Meier  *armasuisse*  
Thun, Switzerland roland.meier@ar.admin.ch

Guido Salvaneschi  University of St. Gallen  
St. Gallen, Switzerland guido.salvaneschi@unisg.ch

**Abstract**—Infrastructure as Code (IaC) aims to automate infrastructure management by enabling the definition of infrastructure configurations in programs, rather than manually configuring hardware or cloud resources. Terraform is one of the most widely used IaC tools, gaining significant traction in recent years, as highlighted by its large and active user community and widespread adoption in both open-source and enterprise environments. Terraform’s code is written in the HashiCorp Configuration Language (HCL), which defines the infrastructure in a declarative manner. Despite the widespread adoption of Terraform, there is no large-scale dataset available for researchers to study IaC Terraform programs systematically. To address this gap, we present TerraDS, the first dataset of publicly available Terraform programs written in HCL. TerraDS contains the HCL code and the metadata of 67,360 open source repositories with permissive open-source licenses. The dataset includes 279,344 Terraform modules with 1,773,991 registered resources, all compiled into a reusable archive (~335 MB).

**Index Terms**—Cloud Computing, Configuration Management, Open Source Software, Static Analysis

## I. INTRODUCTION

Infrastructure as Code (IaC) automates the deployment and provisioning of infrastructure for cloud environments. In IaC, the infrastructure is defined in machine-readable configuration files and programs, rather than manual processes, reducing the likelihood of human error and helping developers manage and version their infrastructure as they do with application code [1]. There are several tools available for IaC, such as Ansible [2], Chef [3], and Puppet [4]. These solutions enable developers to define the steps to deploy infrastructure and the applications that run on it. Other tools, like AWS CloudFormation [5] and Terraform [6], allow developers to define a desired state and derive the required steps to reach that state from the current one. Terraform is widely used, with 371,981 public repositories on GitHub as of August 2024, and major companies like Uber and Slack adopting it for operations [7, 8]. Research on these programs is important to understand best practices, identify common vulnerabilities, and improve IaC tools and methodologies.

However, no comprehensive, publicly available dataset exists that provides a large-scale collection of Terraform HCL programs, which hinders systematic analysis, evaluation, and the development of new tools and techniques for IaC.

In this work, we present the TerraDS dataset of publicly available HCL programs from GitHub. TerraDS consists of

62,406 GitHub repositories with permissive open-source licenses, including 279,344 Terraform modules and 1,773,991 registered resources. To demonstrate the practical applications of the dataset, we perform example analyses, showcasing its potential for future research in the field.

## II. TERRAFORM IN A NUTSHELL

Terraform [6] allows declarative infrastructure management in the HCL configuration language [9]. A configuration consists of `.tf` files specifying the target state of the infrastructure [10].

Listing 1: A Terraform HCL program example.

```
1 provider "aws" {
2   region = "us-east-1"
3 }
4
5 data "aws_ami" "amazon_linux" {
6   most_recent = true
7   owners = ["amazon"]
8 }
9
10 resource "aws_instance" "instance" {
11   ami = data.aws_ami.amazon_linux.id
12   instance_type = "t2.micro"
13 }
```

Terraform programs are structured in *modules*, consisting of a directory in the file system with at least one `.tf` file, which can be referenced from other modules – fostering separate development and software reuse. Listing 1 shows one such module using the Amazon Web Services (AWS) provider. The module queries the `aws_ami` *data* resource to retrieve the latest Amazon Machine Image (AMI) ID (Section II) and defines a *managed* resource, `instance`, for creating an AWS EC2 instance with that AMI (Section II). The AMI serves as the base system image, which is the pre-configured operating system used to launch the instance.

*Managed* resources are objects in the target environment that are created and directly handled by Terraform. By contrast, *data* resources already exist in the infrastructure or environment, and are imported into the current IaC program as read-only, i.e., they can be used to retrieve information about existing resources. In Listing 1, the `aws_instance` retrieves the ID of an existing AMI.

## III. CONSTRUCTION OF THE DATASET

To construct TerraDS, we follow a multistep process to collect, filter, and analyze Terraform programs. We first search for repositories containing HCL files with a custom CLI

\*These authors contributed equally to this work.

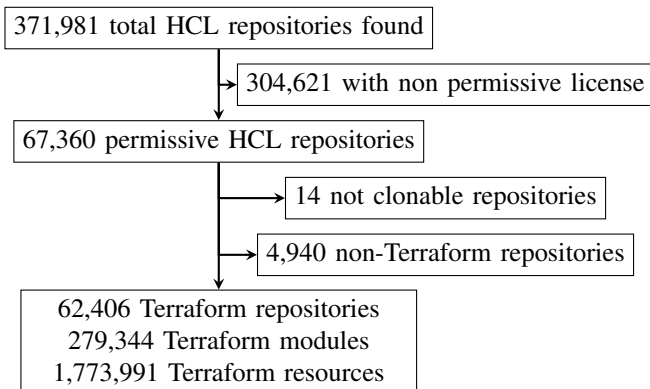


Fig. 1: Repository search and module analysis.

application in C#. Then, we filter the repositories based on their license and relevance. Finally, we analyze the Terraform modules with a Go program to extract metadata and defined resources inside the Terraform program.

#### A. Repository Search

Several open source platforms exist, such as GitHub<sup>1</sup>, GitLab<sup>2</sup>, and BitBucket<sup>3</sup>, just to name a few. Among those, GitHub is perceived as the most used platform [11, 12]. GitHub also provides an advanced code search API [13]. Therefore, we used GitHub as data source for TerraDS.

We identify the repositories by GitHub’s code search using the language search parameter as the query – since HCL is a programming language recognized by GitHub. Since GitHub limits search queries to 1,000 results, we sliced the queries by repository creation date.

All queries included the term `language:hcl` and do not include forks, i.e., TerraDS will not contain any forks. The `created:[date]` was iterated and paginated to comply with rate limits. For reference, the first query, using `created:<2016-01-01`, returned 549 repositories that were created before January 1, 2016. After that, we iterated day by day, adjusting the creation date. No query from January 1, 2016, to August 31, 2024, exceeded 1,000 repositories. This step results in 371,981 repositories.

#### B. Filtering

Figure 1 shows the filtering process of TerraDS. We identified 67,360 repositories that contain licenses which are *permissive*, i.e., they allow derivative work and redistribution. Repositories without a license fall back to normal copyright law, which does not allow reproduction, redistribution, or deriving further work [14], and thus, are not included in TerraDS. Next, we removed the non-clonable repositories (14), that errored during the cloning process. Finally, the non-Terraform repositories (4,940) were excluded: After parsing the `.tf` files to check for valid Terraform modules (Section III-C), we

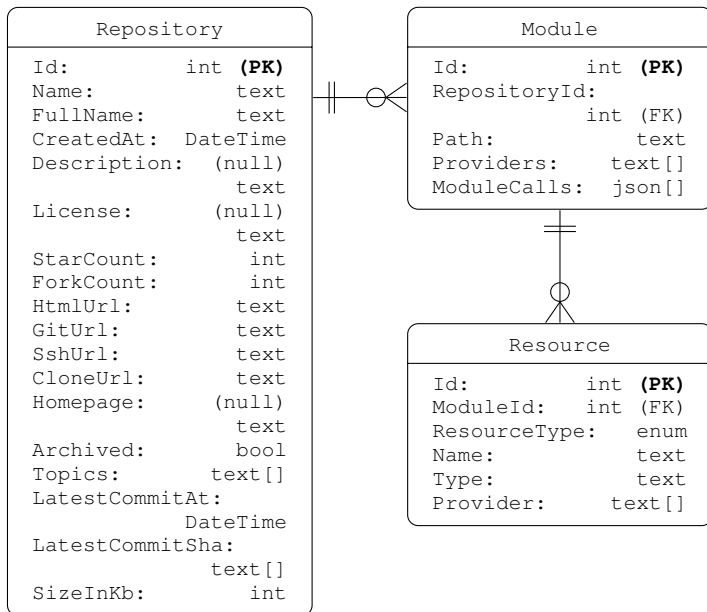


Fig. 2: Entity relation diagram (ERM) of the dataset

removed 4,940 repositories which contain `.tf` files but do not include any correct Terraform modules (and are incorrectly identified as HCL files by GitHub). These steps result in 62,406 relevant repositories.

#### C. Module Collection and Analysis

Our Go program to collect and analyze the modules (1) downloaded each repository, (2) searched directories for at least one `.tf` file, (3) analyzed the Terraform module to implement the filtering described above and to obtain the metadata, and (4) stored the repository as a redistributive `*.tar.gz` file.

In the analysis, we parse the IaC program and store the relative path, the used/referenced providers, and the external module calls. We also store additional information for all modules about the resources, such as the resource type (managed or data), the used provider, the provider type (e.g. `aws_vpc`), and the resource name.

### IV. STRUCTURE OF THE DATASET

TerraDS contains the data of the 62,406 relevant repositories, including 279,344 modules that contain 1,773,991 resources – 1,484,185 managed and 289,806 data resources. TerraDS consists of two packages: a SQLite database with the metadata (Figure 2) and an archive with the source code.

In the SQLite database, the *repository* table contains the repositories’ metadata, such as name, clone URL, and the license. The *module* table references a repository by its ID and contains the relative path within the repository’s file structure. Module entries also contain a string array with all used providers and a list of JSON objects (with a name and source property) with the external module calls. The *resource* table stores, for each resource in a module, name, type, required provider and whether it is a managed or data resource.

<sup>1</sup><https://github.com/>

<sup>2</sup><https://gitlab.com/>

<sup>3</sup><https://bitbucket.org/>

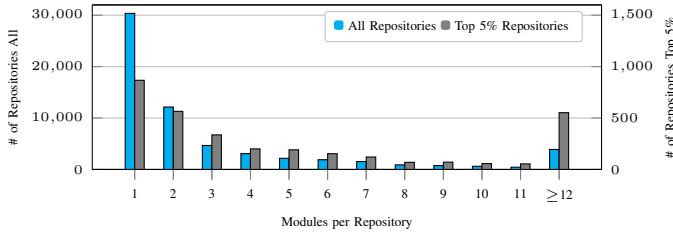


Fig. 3: Distribution modules per repository.

The source code archive contains ~335 MB of compressed source code. The archive only contains the relevant HCL code and the version history of the repositories is not preserved.

TerraDS is long term archived on Zenodo<sup>4</sup>. The tools that were used to gather and analyze the data are also provided to allow reproducibility. In addition to the Zenodo entry, the scripts are also publicly available on GitHub in an open source repository<sup>5</sup> under the CC-BY-4.0 license.

## V. DATA ANALYSIS

To demonstrate the use of TerraDS, in this section, we present an example analysis on the metadata and one on the source code using the Checkov [15] static analyzer.

### A. Metadata Analysis

Figure 3 shows the distribution of modules per repository through two histograms: one for all repositories and one for the top 5% (based on their star ratings). The histograms are based on the metadata and can be simply retrieved with a SQL query (see Figure 2). The y-axes have different absolute values, but are scaled proportionally.

Figure 3 shows that about half of the repository contain one module. The last bin ( $\geq 12$ ) aggregates all repositories with 12 or more modules. On average, each repository includes 4.5 modules and the repository with the most modules has 1,189 modules.<sup>6</sup> Moreover, the histogram of the top 5% by stars is skewed to the right, indicating that repositories with more stars tend to have more modules.

Figure 4 shows the distribution of lines of code (LOC), again for all repositories and for the top 5%. Note that the x-axis is in logarithmic scale. The average LOC per repository is 912, while for the top 5% by stars, the average increases to 2,657. The maximum LOC for a repository is 550,188<sup>7</sup>.

Figure 5 displays the number of repositories created each year (left), as well as the distribution of providers used in the repositories (right). First, the number of repositories that are created is increasing, indicating the growing popularity of Terraform and IaC in general. Second, AWS is the most popular provider, followed by other cloud providers such as Azure and Google Cloud. Notably, the Random provider is also popular, offering randomness specifically designed for

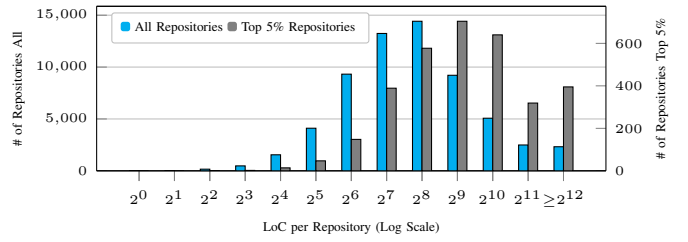


Fig. 4: Distribution of lines of code per repository.

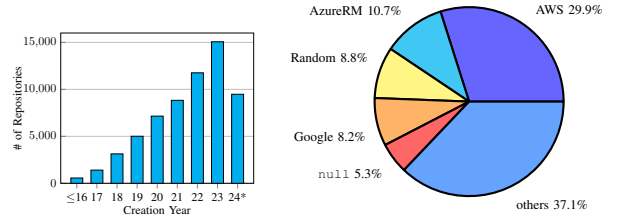


Fig. 5: Repositories created per year (left) and the distribution of providers used in repositories (right).

IaC. Unlike typical providers, it does not interact with external services. Instead, it generates random values during resource creation and holds them steady throughout the resource’s lifecycle, preserving idempotency and reproducibility.

Table I lists the top 10 resource types by total occurrences and their coverage across the 62,406 repositories. The most frequently used is AWS Security Group, which appears 47,468 times and is in more than a quarter (26.21%) of the repositories. Notably, the Null Resource is the only non-AWS resource in the top 10. It acts as a placeholder without creating actual infrastructure and is commonly used to run arbitrary commands on the local machine.

TABLE I: Top 10 resource types by total occurrences (percentage of 1,773,991 total resource occurrences) and repository coverage (percentage of 62,406 total repositories).

Resource Type	Occurrences	Repo Coverage
AWS Security Group	47,468 (2.68%)	16,359 (26.21%)
AWS Subnet	43,529 (2.45%)	11,106 (17.80%)
AWS IAM Role Policy Attachment	39,586 (2.23%)	8,628 (13.83%)
AWS IAM Role	39,145 (2.21%)	13,060 (20.93%)
AWS IAM Policy Document	38,967 (2.20%)	9,217 (14.77%)
AWS Security Group Rule	33,283 (1.88%)	4,615 (7.40%)
AWS Route Table Association	31,547 (1.78%)	8,758 (14.03%)
AWS Instance	31,502 (1.78%)	12,749 (20.43%)
Null Resource	30,618 (1.73%)	8,920 (14.29%)
AWS IAM Policy	24,377 (1.37%)	7,467 (11.97%)

### B. Analysis with Checkov

In this section, we analyze TerraDS to determine which security checks are most frequently violated by users. To this end, we run Checkov [15], a static analysis tool for IaC, and show the most frequently triggered checks in Table II. Checkov classifies checks into four severity levels: *low*, *medium*, *high*,

<sup>4</sup><https://doi.org/10.5281/zenodo.14217385>

<sup>5</sup><https://github.com/prg-grp/hcl-dataset-tools>

<sup>6</sup><https://github.com/epam/ecc-aws-rulepack>

<sup>7</sup><https://github.com/tamsalem/terragoat-gh-1>

and *critical*. This is because some checks are simply guideline violations, while others point out serious security issues. The top 10 checks in Table II are of *low* or *medium* severity, except for the sixth and eighth check, which are of *high* severity. For example, the sixth check indicates that data stored in the AWS Launch Configuration or Elastic Block Store is not encrypted and can potentially be accessed by unauthorized parties.

TABLE II: Top 10 most frequently triggered checks by Checkov in 62,406 repositories.

Description of Violation	Violations
Terraform module source does not use a commit hash	109,655
AWS security group rule does not have a description	62,321
AWS Instance Metadata Service has version 1 enabled	41,732
Detailed monitoring is disabled for AWS EC2 instances	35,534
AWS EC2 instance is not EBS optimized	35,353
Data stored in AWS Launch Configuration or EBS is unencrypted	32,603
No IAM role is attached to AWS EC2 instance	29,635
Terraform module source does not use a commit tag	28,622
AWS S3 Bucket does not have cross-region replication enabled	20,460
AWS S3 Bucket does not have event notifications enabled	20,414

We summarize the top 10 most frequently triggered checks classified as *high* severity in Table III. For example, the fourth violation highlights the creation of an Amazon EC2 instance with a public IP address, which poses a security risk because it directly exposes the resource to the public internet. The only *critical* issue we found is the creation of an AWS IAM policy that grants full administrative privileges and provides unlimited access to all resources in an AWS account [16]. This critical check is violated 392 times across 267 repositories.

TABLE III: Top 10 most frequently triggered checks by Checkov classified as *high*.

Description of Violation	Violations
Data stored in AWS Launch Configuration or EBS is unencrypted	32,603
Terraform module source does not use a commit tag	28,622
Project-wide SSH keys are used for GCP VM instances	8,734
AWS EC2 instance has a public IP	8,217
GCP boot disks do not use customer supplied encryption keys	7,210
Azure Storage is not encrypted with a customer key	6,622
Azure Storage Blobs do not restrict public access	6,099
AWS Load Balancer is not using TLS 1.2 or newer	5,829
Azure Key Vault secrets do not have expiration date	4,294
Not all data stored in AWS Aurora is securely encrypted at rest	4,131

## VI. LIMITATIONS

In this section, we discuss the threats to validity related to the creation and the adoption of TerraDS.

*a) Internal threats to validity:* The internal validity of studies based on TerraDS may be impacted by the quality of the dataset. The dataset is constructed based on the GitHub API, on whose reliability we depend upon. However, the API has been used successfully for other datasets [17, 18]. With regard to the dataset’s quality, we applied several techniques to ensure that TerraDS only contains repositories with valid Terraform code. However, the dataset may still contain errors, inaccuracies or irrelevant repositories due to the limitations of the methods applied.

*b) External threats to validity:* Since IaC is an evolving field, the dataset may not be up-to-date with the latest Terraform practices. We only analyzed created repositories up to August 31, 2024. Therefore, the dataset does not contain any repositories created after this date. Additionally, only publicly available repositories on GitHub are considered. There exist more repositories on other platforms like GitLab or BitBucket as well as private repositories (e.g. for proprietary software of companies) that are not included in the dataset.

## VII. RELATED WORK AND DATASETS

To the best of our knowledge, there is no publicly available dataset that focuses on Terraform (HCL) programs. Sokolowski et al. constructed the PIPr dataset, which focuses on Programming Languages Infrastructure as Code (PL-IaC) programs [19] and which is the closest to our work. Markovtsev and Long introduced the Public Git Archive, containing 182,014 repositories, of various languages, that they ordered by the number of stars [17].

In the context of IaC and Terraform, Begoug et al. introduced TerraMetrics, a tool that extracts code metrics from Terraform programs to analyze the quality of IaC programs [20]. Before TerraMetrics, Begoug et al. conducted an empirical study of Stack Overflow posts to identify the current problems and solutions in IaC for the practitioners [21].

Other work focuses more on the quality of IaC programs. Bessghaier et al. analyzed the occurrence and the impact of code smells in Ansible programs [22]. Also, Opdebeek et al. analyzed security IaC smells in Ansible [23].

## VIII. CONCLUSION

We presented TerraDS, the first publicly available dataset of Terraform HCL programs. TerraDS is built by collecting and indexing metadata from 371,981 GitHub repositories, including 62,406 repositories with valid Terraform code and licenses permitting redistribution and further analysis. The metadata includes repository details such as star counts and licenses, along with information on Terraform modules and their contained resources. Along with the metadata, the HCL source code of all 62,406 repositories is provided in an archive. We further demonstrated the dataset’s utility by analyzing violated checks identified by Checkov. We believe that TerraDS further fosters the advancement of research in the area of Infrastructure as Code. As an example, researchers can use the dataset to improve the current state of the art in static analysis or testing and verification tools for HCL programs.

## IX. ACKNOWLEDGMENTS

This work has been co-funded by the Swiss National Science Foundation (SNSF, Grant No. 200429), by armasuisse Science and Technology, and by European Union’s Horizon research and innovation programme (CAPE Project, Grant No. 101189899).

## REFERENCES

- [1] K. Morris, *Infrastructure as Code*. O'Reilly Media, 2020.
- [2] Red Hat, "Ansible," 2024. [Online]. Available: <https://www.ansible.com/>
- [3] Progress Software Corporation, "Chef," 2024. [Online]. Available: <https://www.chef.io/>
- [4] Perforce Software, "Puppet," 2024. [Online]. Available: <https://www.puppet.com/>
- [5] Amazon Web Services AWS, "AWS CloudFormation," 2024. [Online]. Available: <https://aws.amazon.com/cloudformation/>
- [6] HashiCorp, "Terraform," 2024. [Online]. Available: <https://www.terraform.io/>
- [7] Uber, "Our Tech Stack: Part One — The Foundation," 2024, accessed: 2024-10-15. [Online]. Available: <https://www.uber.com/en-CH/blog/tech-stack-part-one-foundation/>
- [8] Slack Engineering, "How We Use Terraform at Slack," 2024, accessed: 2024-10-15. [Online]. Available: <https://slack.engineering/how-we-use-terraform-at-slack/>
- [9] HashiCorp, "HashiCorp Configuration Language," <https://github.com/hashicorp/hcl>, 2024, accessed: 2024-08-16.
- [10] M. Howard, "Terraform — Automating Infrastructure as a Service," *CoRR*, vol. abs/2205.10676, 2022. [Online]. Available: <https://doi.org/10.48550/arXiv.2205.10676>
- [11] N. McDonald and S. Goggins, "Performance and Participation in Open Source Software on GitHub," in *CHI '13 Extended Abstracts on Human Factors in Computing Systems*. New York, NY, USA: Association for Computing Machinery, Apr. 2013, pp. 139–144. [Online]. Available: <https://dl.acm.org/doi/10.1145/2468356.2468382>
- [12] V. Cosentino, J. L. Cánovas Izquierdo, and J. Cabot, "A Systematic Mapping Study of Software Development With GitHub," *IEEE Access*, vol. 5, pp. 7173–7192, 2017. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/7887704>
- [13] GitHub, "Searching for Repositories," 2024, accessed: 2024-08-16. [Online]. Available: <https://docs.github.com/en/search-github/searching-on-github/searching-for-repositories>
- [14] —, "Licensing a repository," 2024, accessed: 2024-08-19. [Online]. Available: <https://docs.github.com/en/repositories/managing-your-repositorys-settings-and-features/customizing-your-repository/licensing-a-repository>
- [15] Checkov by Prisma Cloud, "Checkov," 2024. [Online]. Available: <https://www.checkov.io/>
- [16] Prisma Cloud, "AWS IAM Policies — BC AWS IAM 47," 2024, accessed: 2024-10-15. [Online]. Available: <https://docs.prismacloud.io/en/enterprise-edition/policy-reference/aws-policies/aws-iam-policies/bc-aws-iam-47>
- [17] V. Markovtsev and W. Long, "Public Git Archive: a Big Code dataset for all," in *Proceedings of the 15th International Conference on Mining Software Repositories*. New York, NY, USA: Association for Computing Machinery, May 2018, pp. 34–37. [Online]. Available: <https://dl.acm.org/doi/10.1145/3196398.3196464>
- [18] O. Dabic, E. Aghajani, and G. Bavota, "Sampling Projects in GitHub for MSR Studies," in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, May 2021, pp. 560–564. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/9463094>
- [19] D. Sokolowski, D. Spielmann, and G. Salvaneschi, "The PIPr Dataset of Public Infrastructure as Code Programs," in *Proceedings of the 21st International Conference on Mining Software Repositories*. New York, NY, USA: Association for Computing Machinery, 2024, pp. 498–503. [Online]. Available: <https://doi.org/10.1145/3643991.3644888>
- [20] M. Begoug, M. Chouchen, and A. Ouni, "TerraMetrics: An Open Source Tool for Infrastructure-as-Code (IaC) Quality Metrics in Terraform," in *Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension*. New York, NY, USA: Association for Computing Machinery, Jun. 2024, pp. 450–454. [Online]. Available: <https://dl.acm.org/doi/10.1145/3643916.3644439>
- [21] M. Begoug, N. Bessghaier, A. Ouni, E. A. AlOmar, and M. W. Mkaouer, "What Do Infrastructure-as-Code Practitioners Discuss: An Empirical Study on Stack Overflow," in *2023 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. New Orleans, LA, USA: IEEE, Oct. 2023, pp. 1–12. [Online]. Available: <https://ieeexplore.ieee.org/document/10304847/>
- [22] N. Bessghaier, M. Begoug, C. Mebarki, A. Ouni, M. Sayagh, and M. W. Mkaouer, "On the Prevalence, Co-occurrence, and Impact of Infrastructure-as-Code Smells," in *2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. Rovaniemi, Finland: IEEE, Mar. 2024, pp. 23–34. [Online]. Available: <https://ieeexplore.ieee.org/document/10589858/>
- [23] R. Opdebeeck, A. Zerouali, and C. De Roover, "Control and Data Flow in Security Smell Detection for Infrastructure as Code: Is It Worth the Effort?" in *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*. Melbourne, Australia: IEEE, May 2023, pp. 534–545. [Online]. Available: <https://ieeexplore.ieee.org/document/10174011/>