A Benchmark for DDoS Attacks Detection in Microservice Architectures

M. Ficco[a], P. Fusco[a], A. Guerriero[b], R. Pietrantuono[b], M. Russo[a], F. Palmieri[a], and S. Russo[b]
[a]Università degli Studi di Salerno
[b]Università degli Studi di Napoli Federico II
{mficco, pfusco, fpalmieri}@unisa.it, m.russo269@studenti.unisa.it
{antonio.guerriero, roberto.pietrantuono, stefano.russo}@unina.it

***Abstract*—Microservices have become increasingly popular in modern software architectures due to their scalability and flexibility. However, this architectural paradigm introduces unique security challenges, particularly in the detection and mitigation of cyberattacks. This paper presents a collection of datasets designed to benchmark and evaluate attack detection strategies in microservices applications. The datasets include normal and malicious traffic patterns simulating real-world scenarios and attacks, such as classic DDoS, Slow DDoS, Syn Flood, GET Flood. Data was collected from experiments with a popular benchmark microservice system with diverse services interacting via standard protocols and API gateways. Each entry is labeled to distinguish between benign and malicious activities, providing a robust foundation for training and evaluating machine learning models aimed at intrusion detection. In addition to raw data, the dataset includes metadata detailing the configuration of microservices, the nature of simulated attacks, and the temporal sequence of events. This level of detail ensures that researchers and practitioners can reproduce experiments and gain deeper insights into the behavior of attacks in microservice contexts. By offering these datasets, we aim to facilitate the development of advanced detection algorithms and promote more effective security measures in microservice environments.**

***Index Terms*—Data collection, Machine Learning, Intrusion Detection, Denial of Service, Security benchmark.**

I. Introduction

Microservices architectures (MSA) are increasingly adopted due to their loosely coupled services, each running in its own process and interacting via lightweight mechanisms [1]. This architecture allows for independent development and deployment of services. However, the highly distributed nature of these systems makes them particularly vulnerable to security attacks. In particular, MSA are vulnerable to threats resulting from malicious load fluctuations and resource-exhausting attacks, whose goal is to overwhelm the target, rendering it unavailable to the users by exhausting its resources or bandwidth [2]. Anomaly detection techniques, including Machine Learning (ML)-based strategies, are commonly adopted in intrusion detection and prevention systems for such malicious behaviors [3]–[5]. While ML is indeed a valuable tool for anomaly detection, the most advanced strategies, *e.g.*, based on deep learning, require representative datasets to be properly trained and compared against each other. The generation of most current open datasets focused on collecting the sequences of API calls produced by microservices, as well as on the internal and external network traffic [6].

This paper contributes to the lack of datasets for network security research and practice for MSA, by presenting a collection of datasets that include a set of features useful for detecting resource-exhausting attacks. They are generated using a realistic microservices-based framework emulating train ticket reservations - `TrainTicket` [7], a well-known benchmark microservices system.

The `TrainTicket` framework has been extended with tools to generate and collect the datasets. In particular, the nominal load is generated using the *Locust* workload generator [8], while the attacks are conducted via the *MHDDoS* attack suite [9]. Data is collected using *CAdvisor*, a tool providing container-level resource usage metrics [10], and exported by *Prometheus*, a state-of-the-art monitoring tool [11].

The datasets include examples of normal data flow and of various types of DDoS attacks – *Classic DDoS*, *Slow DDoS*, *Syn Flood*, and *GET Flood* - recorded under different system conditions (idle and load scenarios). Overall, we provide 11 datasets: 2 representing normal conditions, 8 representing attack scenarios, and 1 unified dataset created by merging the previous 10. Each dataset is characterized by the following *feature sets*[1]: *Container Usage*, *CPU Usage*, *Memory Usage*, *Network I/O*, *Resource Limits*, *CPU Throttling*, *Pod Restarts*, and *Pod Status*. The unified dataset captures 494 features, representing the intersection of all features across these scenarios. Data collection for each scenario was conducted over a 30-minute period, with metrics sampled at 5-second intervals.

An example of use of the datasets is presented, in which several ML algorithms used to implement an Intrusion Detection System (IDS) are compared. To allow researchers and practitioners to work with these datasets and generate new ones, we provide data and code in an online repository[2].

The rest of the paper is structured as follows. Sec. II presents the current state of the art. The `TrainTicket` microservices-based framework is described in Sec. III. Sec. IV presents the implemented monitoring framework. The generated datasets are described in Sec. V. A use case is shown in Sec. VII. Sec. VIII is devoted to conclusion and future work.

[1]Each set comprises multiple features; some features are shared among sets.
[2]Available at:*https://github.com/iotresearchunisa/MicroservicesDDoS*.

979-8-3315-2096-0/25/$31.00 ©2025 IEEE

II. Related Work

Researchers are proposing various techniques for detecting DDoS attacks in microservices architectures. These techniques are based on anomaly detection methods [12] and machine learning models, employing both supervised [13] and unsupervised [14] approaches. This trend highlights the increasing need for extensive datasets to facilitate the development of novel DDoS detection strategies.

Castro *et al.* [4], [5] generated their own dataset using TeaStore [15], an application featuring six services and a database, in conjunction with JMeter[3]. They utilized the Hulk[4] and Torshammer[5] tools to perform attacks under four different attack profiles.

Similarly, Jacob *et al.* [6] employed a distributed tracing tool to monitor a microservices application and detect cyber attacks using a Diffusion Convolutional Recurrent Neural Network (DCRNN). The monitored microservices application was a Social Network consisting of 36 microservices, taken from DeathStarBench [16], an open-source benchmark suite comprising six microservice-based applications.

Gupta *et al.* [17] introduced HONEYKUBE, a microservices architecture designed to collect interaction data from real-world attacks. They provided a dataset of approximately 850 GB, including system trace files, network traces, and additional log files.

In line with current literature, we provide a new collection of datasets, which includes features useful to detect resource-exhausting attacks against MSA, generated using a realistic microservices-based application. These datasets are designed to assist researchers in experimenting with IDS for various types of DDoS attacks, providing the capability to replicate and potentially expand the attack types with new ones.

III. Microservice System for Datasets Generation

The `TrainTicket` booking system, developed by FudanSeLab, represents a microservices-based framework for train ticket reservations, designed to benchmark and test within the software engineering domain. This initiative addressed the scarcity of benchmark systems that accurately reflect the intricacies of industrial microservices ecosystems [7]. This system emerges as a quintessential model of microservices architecture, showcasing over 40 distinct microservices each dedicated to fulfilling a specific role to ensure seamless operation of the application. Its design facilitates performance testing and experimental research, particularly focusing on metrics such as latency and throughput.

The architecture of `TrainTicket` encompasses a diverse array of components, including databases, web applications, and Java applications, highlighting its complexity and versatility. `TrainTicket` distinguishes itself by addressing several critical gaps identified in prior benchmark systems, including limited interaction modes, inadequate scale and complexity, and the underutilization of core microservices design principles. Designed with a commitment to these principles, `TrainTicket` incorporates a variety of interaction modes and boasts a significant codebase with 61,136 lines of code, encompassing both Java and Node.js.

[3]https://jmeter.apache.org
[4]https://github.com/R3DHULK/HULK
[5]https://github.com/Karlheinzniebuhr/torshammer

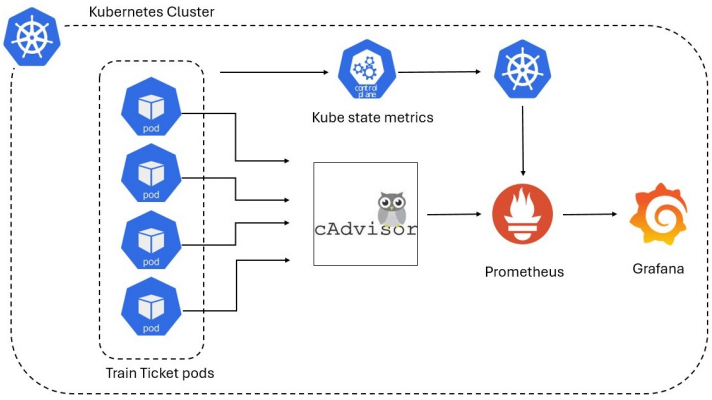

Fig. 1: The proposed monitoring architecture.

IV. Monitoring Framework

During the analysis phase aimed at establishing an efficient data collection environment for monitoring and metrics analysis, various technological solutions were carefully evaluated. The objective was to design a system capable of simulating a distributed environment, collecting and storing performance metrics, and providing insightful visual feedback on the monitoring system's efficacy. The selected environment comprises a synergistic integration of several key technologies, each chosen for its specific strengths in facilitating a comprehensive monitoring solution.

Figure 1 depicts the architecture of the framework developed for monitoring and data collection for the microservices-based system; it includes the following tools:

- *Minikube* [18]: Chosen for its ability to replicate a Kubernetes distributed environment locally, Minikube stands out as a pivotal tool. It enables the simulation of a distributed system on a single machine, offering a practical and cost-effective solution for development and testing purposes.
- *CAdvisor (Container Advisor)* [10]: Integral to the monitoring framework, CAdvisor is tasked with the collection of performance metrics from containers.
- *Prometheus* [11]: Serving as the cornerstone for metrics storage, Prometheus is deployed to aggregate and store the metrics harvested by CAdvisor.
- *Grafana* [19]: To complete the monitoring solution, Grafana is incorporated for the preprocessing and visual representation of the collected metrics. It transforms raw data into meaningful insights through its advanced data visualization capabilities by providing a real-time overview of the system's health and performance.

Finally, *Locust* is an open-source load testing tool utilized for simulating users (or "locusts") and for generating traffic needed to evaluate the performance of web applications and systems [8]. It empowers developers and testers to script test scenarios in Python, thus providing the flexibility to program complex behaviors and interactions that real users might exhibit with the system.

V. Data Generation

To accurately simulate and analyze the performance and security of the environment, particularly in the context of potential threats and operational demands, it became crucial that the framework for data generation encompasses both the attack surface and the system's standard behavioral patterns. This approach aimed to ensure comprehensive coverage of scenarios that the system might encounter, thereby enabling a thorough assessment of its resilience and performance under various conditions.

A. Defining Standard Behavior

Standard behavior within the environment was conceptualized through two distinct cases, which serve as benchmarks for assessing system performance and security posture:

1) *Idle State:* The system is in a state of minimal activity, not processing any external requests. This baseline scenario is critical for understanding the system's resource footprint in the absence of operational load.
2) *Under Load:* The system processes a continuous flow of requests, distinctively different from its idle state, reflecting a scenario where the system is actively utilized but not overwhelmed.

This bifurcation simulates two different states of a distributed system's operation, facilitating the differentiation of performance and security metrics under varied conditions.

B. Simulating System Load

We considered Locust to generate the load concerning the standard behavior. We exploited advanced algorithms designed to dynamically adjust the load by increasing and decreasing the number of requests. This approach simulates an approximation of the system's behavior under typical operational conditions, offering insights into performance bottlenecks and resilience under stress. However, it's important to note that real-world traffic patterns may be more erratic and complex than those simulated during testing [20].

C. Addressing the Attack Surface

In addressing the system's attack surface, a multi-faceted approach was designed to cover an extensive spectrum of potential threats that could compromise system integrity and security. The strategy pivoted towards an in-depth focus on identifying, simulating, and analyzing various forms of DDoS attacks. DDoS were prioritized due to their notable detectability through observable changes in system behavior and resource consumption patterns, which could be more readily monitored and analyzed compared to the subtler indicators of other forms of attacks. This strategic shift involved the development and implementation of advanced simulation models and analytical tools designed to accurately mimic the dynamics of DDoS under various scenarios and conditions.

Utilizing MHDDoS [9], a comprehensive suite of DDoS attack simulations was conducted on the microservices systems. The simulations encompassed a range of attack methodologies, each chosen for their prevalence in real-world attack scenarios and their relevance to the security landscape faced by contemporary microservices architectures. The attack types included, but were not limited to, the following:

- *Classic DDoS*: This form of attack floods the target system with an overwhelming volume of traffic, aiming to exhaust network bandwidth or system resources, thereby rendering the service unavailable to legitimate users.
- *Slow DDoS*: Unlike the classic DDoS, Slow DDoS attacks are characterized by sending traffic at a slow rate. This method is insidious as it aims to open and maintain as many connections as possible over a long period, gradually depleting server resources without triggering rate-based defense mechanisms.
- *Syn Flood*: A specific type of DDoS attack that exploits the TCP handshake process. Attackers send a flood of TCP/SYN packets, often with a spoofed source IP address, to overwhelm the target's ability to respond to new legitimate connections.
- *Get Flood*: This attack targets web servers and applications by flooding them with a massive number of HTTP GET requests. The goal is to exhaust the server's resources and bandwidth, leading to denial of service.

These attack types were chosen based on their representativeness of common threats faced by microservices systems, with the premise that simulating these attacks would account for a significant portion of real-world scenarios.

D. Data Collection and Metrics

For each simulated attack scenario, datasets were generated under both idle and loaded system states using Locust. The study encompassed a total of 10 scenarios (5 under load and 5 in idle conditions) with data collection spanning 30 minutes for each scenario and metrics sampled every 5 seconds. The collected metrics for each container within the `TrainTicket` application are gathered in the following *feature sets*:

- *Container Usage* (128 features): Tracks the number of containers in use, providing insights into the overall utilization and potential scaling requirements.
- *CPU Usage* (64 features): It measures the percentage of the allocated CPU resources being utilized by the container. High values suggest need for optimization or scaling, whereas low usage may refer to over-provisioning.
- *Memory Usage* (151 features): It monitors the amount of memory consumed by the container. Similar to CPU usage, this metric helps in identifying memory bottlenecks and optimizing resource allocation.

- *Network I/O* (296 features): It quantifies the amount of data being sent to and received from the network. This metric is useful for detecting bottlenecks and understanding the performance of the application as for data I/O over the network.
- *Resource Limits* (116 features): It defines the maximum amount of CPU and memory resources that a container can consume. Monitoring these limits against actual usage can prevent resource contention and ensure fair resource distribution among containers.
- *CPU Throttling* (68 features): It indicates the extent to which CPU has been throttled for a container, revealing if the application is hitting its CPU usage limits. Frequent throttling can degrade performance and may necessitate adjustment of resource allocations.
- *Pod Restarts* (68 features): The number of times a pod has restarted can indicate instability or issues within the application. High restart counts necessitate a deeper investigation into logs and system events.
- *Pod Status* (168 features): It shows whether a pod is ready to serve requests. Monitoring pod readiness helps ensure that the application remains available to end-users and operates efficiently.

The reported sets overlap because some features may appear multiple times. This comprehensive data collection strategy captured a detailed performance profile of the system in various operational and attack scenarios, providing valuable insights on the resilience and efficiency of the distributed environment. Figure 2 shows Network I/O during a DDoS attack, highlighting spikes at the start and end of the attack.

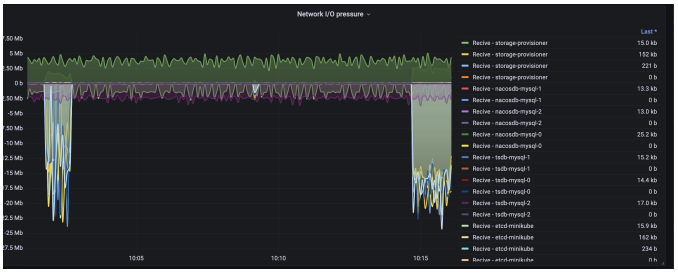

Fig. 2: Network I/O snapshot showing DDoS attack spikes at start and end.

VI. Data Collection and Preprocessing

To effectively visualize and analyze the extensive metrics collected from CAdvisor and exported by Prometheus, a comprehensive set of dashboards was created in Grafana. These dashboards played a crucial role in organizing the data, enabling a clear and intuitive representation of the system's performance under various conditions.

A. Dashboard Configuration in Grafana

Grafana's versatile dashboard creation tools facilitated the development of custom views for collected metrics. Figure 3 displays two examples views of the dashboard, concerning

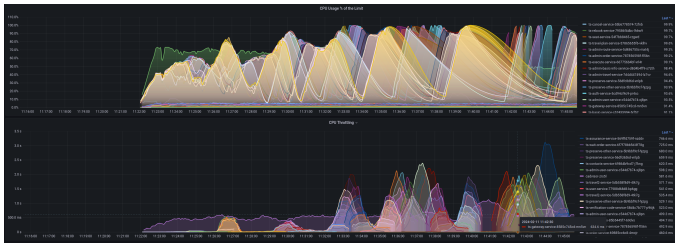

Fig. 3: CPU Usage and CPU Throttling Dashboards.

CPU usage and CPU Throttling. The key features of the dashboard setup include:

- *Time Queries:* Grafana allows us to formulate time-based queries we used to isolate data corresponding to the `TrainTicket` application's namespace pods. This feature allowed for precisely selecting the time window associated with each simulated scenario, ensuring accurate visualization of the relevant metrics.
- *Data Export:* the "Export as CSV" functionality offered by Graphana was utilized to export the data. This step enabled the extraction of metric values into a structured format, suitable for further analysis.

B. Data Processing

The exported CSV data files were aggregated into a single Pandas DataFrame for analysis. This consolidation step was crucial for creating a unified dataset that encompasses all scenarios, facilitating comprehensive analysis. The subsequent data processing steps involved:

- *Data Cleaning:* The DataFrame was scrutinized for "dirty" samples, characterized by missing or null metric values. These samples were removed to ensure the integrity and reliability of the dataset.
- *Normalization:* The metrics in the DataFrame were normalized using the MinMaxScaler from Scikit-learn. This process adjusted data to a common scale, enhancing the comparability and effectiveness of subsequent analyses.

VII. Example of Use

We discuss the use of the unified dataset, created by merging data from 10 scenarios, to compare the different machine learning algorithms used to implement an IDS.

A. Dataset preparation

An important aspect was ensuring a balanced data set to prevent the IDS model from being biased. We identified the set of characteristics with the fewest samples, which was 17,494. This minimum number was used as a reference for each feature set, ensuring that both idle and loaded states had an equal number of samples. In particular, Each set comprises a different number of features, and the final dataset includes the intersection of all these feature sets. Considering four types of attacks and normal data flow, a total of 87,470 samples were used. Each block of 17,494 samples, derived from the same set of features, was labeled with a unique identifier. The

final dataset includes a column titled "label", containing label numbers ranging from 0 to 4 (0 means normal behavior, 1 Classic DDoS, 2 Slow DDoD, 3 Syn Flood, and 4 GET Flood).

B. ML models for IDS experiments

As an example of usage of the provided merged datasets, we implemented an IDS with three ML models: Support Vector Machine (SVM), Random Forest (RF), and KNeighborsClassifier (KNN). All models are available through the Scikit-learn library [21]. The SVM classifier is configured using a linear kernel in its definition. The RF classifier is configured with the number of trees in the forest equal to 100. The KNN classifier is configured with the number of neighbors equal to 7.

C. Results

Each of the 87,470 samples in the dataset has 494 features. The shuffled data set was divided into two parts: a training set with 69,976 samples and a testing set with 17,494 samples.

Table I shows the accuracy of the three ML algorithms experimented on the proposed dataset. It is worth noting that all algorithms provide comparable results in both the training and testing steps. This similarity suggests investigating more sophisticated algorithms, like neural networks, ensemble models, or transformers, to determine if the data reveals deeper correlations among features. The use of cross-validation and hyperparameter tuning is crucial for optimizing model performance. The experimented models accurately predict the label of the input sample, whether it is a normal data sample or an attack sample. The accuracy achieved witnesses the good balance of the dataset.

TABLE I: Machine Learning algorithms accuracy.

	SVM	RF	KNN
Training	0.8018	0.8082	0.8062
Test	0.7929	0.7670	0.7705

VIII. Conclusions and Future Work

Datasets are valuable resources for experimental research. We have presented the design of a dataset for experimenting with attacks detection for microservice applications. We have made it available on a publicly repository.[2]

In the future, we plan to expand the dataset by incorporating a broader range of attack types and additional detection features. Moreover, other microservice-based applications will be considered to make the framework more generic. Finally, we aim at implementing anomaly detection solutions based on advanced AI models, like transformers and ensemble models, to achieve superior results in detecting and mitigating microservices security threats.

Acknowledgment

This work is part of the research activities realized within the projects SERICS (CUP PE00000014, under the NRRP MUR program funded by the EU - NGEUm) and FLEGREA (CUP E53D23007950001, Bando PRIN 2022).

The work by R. Pietrantuono, A. Guerriero, and S. Russo received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 871342 "uDEVOPS".

References

[1] "Microservices," Jun. 2023. [Online]. Available: https://martinfowler.com/articles/microservices.html

[2] J. Mirkovic and P. Reiher, "A taxonomy of ddos attack and ddos defense mechanisms," *SIGCOMM Computing Communication Reviews*, vol. 34, no. 2, p. 39–53, 2004.

[3] F. B. Saghezchi, G. Mantas, M. A. Violas, A. M. de Oliveira Duarte, and J. Rodriguez, "Machine Learning for DDoS Attack Detection in Industry 4.0 CPPSs," *Electronics*, vol. 11, no. 4, 2022.

[4] J. Castro, N. Laranjeiro, and M. Vieira, "Detecting DoS Attacks in Microservice Applications: Approach and Case Study," in *Proceedings of the 11th Latin-American Symposium on Dependable Computing (LADC)*. ACM, 2023, p. 73–78.

[5] ——, "Exploring Logic Scoring of Preference for DoS Attack Detection in Microservice Applications," in *IEEE International Conference on Web Services (ICWS)*, 2023, pp. 573–584.

[6] S. Jacob, Y. Qiao, Y. Ye, and B. Lee, "Anomalous distributed traffic: Detecting cyber security attacks amongst microservices using graph convolutional networks," *Computers & Security*, vol. 118, 2022.

[7] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, W. Li, and D. Ding, "Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study," *IEEE Transactions on Software Engineering*, vol. 47, no. 2, pp. 243–260, 2021.

[8] "Locust documentation," https://docs.locust.io/en/stable/index.html, Accessed 11/07/2024.

[9] "MHDDos," https://github.com/MatrixTM/MHDDoS, Accessed 11/07/2024.

[10] Google, "cadvisor: Analyzes resource usage and performance characteristics of running containers," https://github.com/google/cadvisor, Accessed 11/07/2024.

[11] "Introduction to prometheus," https://prometheus.io/docs/introduction/overview/, Accessed 11/07/2024.

[12] Q. Du, T. Xie, and Y. He, "Anomaly detection and diagnosis for container-based microservices with performance monitoring," in *Algorithms and Architectures for Parallel Processing*, J. Vaidya and J. Li, Eds. Cham: Springer International Publishing, 2018, pp. 560–572.

[13] J. Grohmann, P. K. Nicholson, J. O. Iglesias, S. Kounev, and D. Lugones, "Monitorless: Predicting performance degradation in cloud applications with machine learning," in *Proceedings of the 20th International Middleware Conference*. ACM, 2019, p. 149–162.

[14] A. F. Baarzi, G. Kesidis, D. Fleck, and A. Stavrou, "Microservices made attack-resilient using unsupervised service fissioning," in *Proceedings of the 13th European Workshop on Systems Security (EuroSec)*. ACM, 2020, p. 31–36.

[15] J. von Kistowski, S. Eismann, N. Schmitt, A. Bauer, J. Grohmann, and S. Kounev, "Teastore: A micro-service reference application for benchmarking, modeling and resource management research," in *IEEE 26th Int. Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2018, pp. 223–236.

[16] Y. Gan and et al., "An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems," in *Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2019, p. 3–18.

[17] C. Gupta, T. Van Ede, and A. Continella, "Honeykube: Designing and deploying a microservices-based web honeypot," in *2023 IEEE Security and Privacy Workshops (SPW)*, 2023, pp. 1–11.

[18] "Minikube documentation," https://minikube.sigs.k8s.io/docs/, Accessed 11/07/2024.

[19] "Grafana documentation," https://grafana.com/docs/, Accessed 11/07/2024.

[20] A. Avritzer, D. Menasché, V. Rufino, B. Russo, A. Janes, V. Ferme, A. van Hoorn, and H. Schulz, "PPTAM: Production and Performance Testing Based Application Monitoring," in *Companion of the 2019 ACM/SPEC International Conference on Performance Engineering (ICPE)*. ACM, 2019, p. 39–40.

[21] F. Pedregosa and et al., "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.