Received 5 March 2025; revised 11 May 2025; accepted 2 June 2025. Date of publication 5 June 2025; date of current version 25 June 2025.
Digital Object Identifier 10.1109/OJCOMS.2025.3576749

Algorithms to Identify Copied and Manipulated Spectrum Occupancy Data in Cognitive Radio Networks

JOSEPH TOLLEY AND CARL B. DIETRICH (Senior Member, IEEE)

Bradley Department of Electrical and Computer Engineering, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061, USA

CORRESPONDING AUTHOR: J. TOLLEY (e-mail: jtolley@vt.edu)

This work was supported by the National Science Foundation under Award 2128584.

ABSTRACT Spectrum Access Systems (SASs) and similar systems coordinate access to shared radio frequency bands to efficiently allocate the use of spectrum between users in a locality. To fill the need for dense spectrum occupancy information, SASs will utilize crowdsourced data from nodes outside the SAS's control. This crowdsourcing of data, however, makes the SAS vulnerable to many types of attacks. The attacks covered in this paper include copying and manipulating existing data to create a Spectrum Sensing Data Falsification (SSDF) attack. We propose methods to identify two categories of easily implemented SSDF attacks and show the proposed methods to be both effective and efficient. Further, we recommend that the proposed techniques be used in conjunction with other SSDF thwarting methods that use statistics, probability, or machine learning, and can identify a wider range of SSDF attacks, albeit more slowly and less reliably than the proposed methods can identify the specific types of SSDF attacks for which they are effective. Our findings demonstrate the feasibility of discerning diverse forms of manipulated data while maintaining pace with the influx of incoming data. The ability to identify manipulated data rapidly without imposing undue strain on a centrally aggregated system helps reduce the number of ways to create a potentially successful SSDF attack and increases the accuracy of determining the radio transmission activity of a Primary User (PU). Several methods are explored and evaluated for identifying copied or manipulated spectrum data. We recommend utilizing an exact match identification algorithm with Elasticsearch to search for exact copies of spectrum data. Additionally, we recommend utilizing a cosine similarity function with Elasticsearch to search for manipulated spectrum data and exact copies when sufficient computational resources are available.

INDEX TERMS Dynamic spectrum access, cognitive radio, CBRS, malicious crowdsourced data detection, cosine similarity.

I. INTRODUCTION

To ADDRESS the increasing need for wireless data capacity, centrally coordinated Dynamic Spectrum Access (DSA) can be utilized to govern a spectrum-sharing strategy across multiple Radio Frequency (RF) bands.

Primary Users (PUs) are licensed incumbent users that have priority access to certain frequencies for wireless transmission and can use them without interference from other users. Secondary Users (SUs) are unlicensed users that do not have priority access and must participate in spectrum sharing with a frequency coordinator like a Spectrum Access System (SAS) to transmit legally.

Though potentially more effective, SASs, similar systems, and future multi-context-aware spectrum-sharing frameworks (e.g., as proposed for the 12.2-12.7 GHz band [1], [2]) leveraging crowdsourced spectrum occupancy and/or other observed operational-context data to enhance the detection of PU transmission activity and potential for interference, are susceptible to various types of attacks. This is an inherent risk of incorporating denser, crowdsourced spectrum occupancy observations, and potentially other operational-context information, from unverified devices. This paper focuses on the case of crowdsourced spectrum-occupancy data used by SASs in the United States (US) Citizens Broadband Radio

Service (CBRS) band due to its near-term relevance, and briefly address extension of the proposed approach to other crowdsourced operational-context observations as potential future work.

Maliciously generated, manipulated, or copied spectrum sensing data can be used to create Spectrum-Sensing Data Falsification (SSDF) attacks.

To thwart these attacks, we investigated ways of identifying specific types of easily-implemented, and therefore likely, SSDF attacks that maliciously manipulate and copy spectrum occupancy data; and propose specific methods for this purpose based on the results of our investigation. To (1) exploit the speed and effectiveness of the proposed methods and (2) maintain the potential to detect a wide variety of SSDF attacks, we recommend that the proposed methods be used in conjunction with other types of SSDF identification methods, including methods based on one or more of that are more broadly applicable, yet slower and less accurate. The ability to detect falsified spectrum data reliably has broad implications across various industries, including wireless communications, national security, next-generation networks, and regulatory compliance. In spectrum-sharing frameworks such as those managed by SASs, these detection methods help prevent unauthorized spectrum usage and ensure fair allocation of resources. Defense and security agencies can leverage these techniques to mitigate intentional interference and ensure the integrity of mission-critical communications. Additionally, 5G networks, Internet of Things (IoT) deployments, and smart city infrastructures benefit from improved spectrum monitoring, reducing the risk of data manipulation affecting intelligent transportation and energy grid systems. These applications highlight the necessity of effective SSDF detection mechanisms to secure dynamic spectrum access and maintain reliable communication environments.

As mentioned, these methods should be used in addition to other types of SSDF identification methods. We successfully applied both an exact match identification algorithm to identify copies of spectrum data and a cosine similarity algorithm to identify copied and manipulated spectrum data. Neither method imposes undue strain on a centrally aggregated system helping reduce the number of ways to create an SSDF attack. Applying either method or both increases the accuracy of determining the radio transmission activity of a PU.

II. BACKGROUND

SPECTRUM ACCESS SYSTEMS

Spectrum Access Systems (SASs) are used to manage the shared CBRS band in the US. Title 47 of the US Code of Federal Regulations, Chapter I, Subchapter D, Part 96 states that a SAS should ensure that when a PU appears on a channel or overlaps a range of frequencies allocated by the SAS, SUs participating in spectrum sharing are informed to stop transmitting within 300 seconds from the time the SAS is notified about the presence of a PU [3]. The SAS has no direct control over SUs and cannot prevent any SU from transmitting regardless of whether or not an SU is acting maliciously or not. As a compliance measure, the SAS notifies all SUs with active grants in the specified band to cease transmission. SASs often use spectrum occupancy data in the form of a received power measure report to determine the transmission activity of a PU in a given channel.

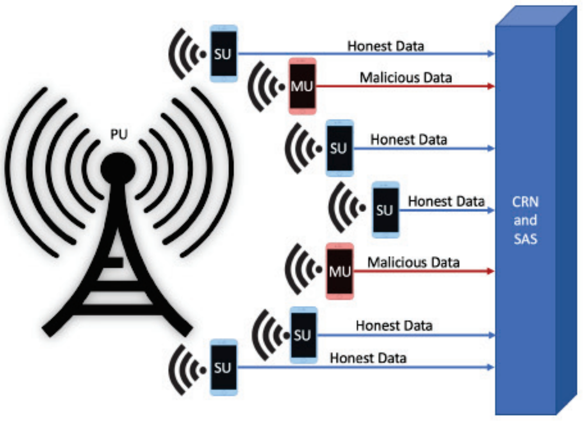

FIGURE 1. CRN during a Spectrum Sensing Data Falsification attack where malicious SUs, MUs, report malicious received power measure reports.

SPECTRUM SENSING DATA FALSIFICATION

An SSDF attack comes in the form of Malicious Users (MUs) transmitting false data to a fusion center attempting to change the decision-making of a Cognitive Radio Network (CRN) regarding the presence of a PU [4]. For a CRN to function properly, spectrum sensing data must be collected from receiving radios and reported to a fusion center to determine if a PU is transmitting within a specific frequency range. In an SSDF attack, MUs will report false information to corrupt the system. The data can be maliciously reported higher or lower than the actual sensed values. Figure 1 depicts a CRN during an SSDF attack. The MUs send false spectrum measure reports to the aggregate center. In non-centralized networks, MUs report false data to their respective endpoints.

An MU can benefit from both malicious measure report data that is higher than the actual value and lower than the actual value as they both affect the functionality of the systems.

If the reported data is higher than the actual measure report readings, a CRN might incorrectly decide that a PU is transmitting within the frequency range submitted and signal to all the other SUs in the system to stop transmitting on that channel. This allows the MU to access a channel or frequency range that would have been otherwise reserved for honest SUs. If everyone is told to stop transmitting within a specific frequency range, the MU will not have interference from other users. SUs and the SAS suffer from these attacks. SUs will potentially have an interrupted grant, degrading their quality of service (QoS) or they will miss out on a potential opportunity to transmit. The SAS, if benefiting from SUs monetarily, will lose money on the potential spectrum they could have licensed out in a grant.

When falsely reported values are too low SUs, PUs, the CRN, and the SAS all suffer. SUs and PUs transmissions will interfere with each other causing a degradation in the QoS. The CRN and SAS will lose the credibility and trust of their users and if the PU reports the SAS for allowing interference to occur, potentially the SAS operator could lose their license to operate. Unless the MU is choosing a PU as its intended target for interference purposes, an MU would benefit more directly by falsely reporting data that is higher than the actual sensed spectrum data.

INTERMITTENT SPECTRUM SENSING DATA FALSIFICATION

The Intermittent Spectrum Sensing Data Falsification (ISSDF) attack is the same attack method as an SSDF but the attacker reports false data to the CRN with a lower rate of incidence to maintain the attacker's reputation with the targeted system. Various types of MU detection algorithms can identify if an MU is submitting false spectrum data, especially if they are doing so when their behavior is sustained or patterned. These include statistical anomaly detectors, similarity-based models, clustering techniques, and machine learning-based classifiers tailored for spatio-temporal signal patterns. Their spectrum data will almost always be an outlier if their attack is constant. Many algorithms that attempt to thwart an SSDF attack assign a score to the node, Citizens Broadband Radio Service Device (CBSD), or user that submits the data. In this case, if the system's decision on the presence of a PU constantly disagrees with the decision of the MU or its data, the score will go down.

The detection and response functionality is more difficult with an ISSDF attack. Oftentimes, MUs will submit false data only occasionally and honest data the rest of the time. If the MU does not know how the CRN assesses the trustworthiness of a user or node, the MU will often only attack when it benefits them. With intermittent attacks, the MU can assume their trust score will go back up, remain the same or a flag (identifying the MU) will be removed from their identity. If the user knows the algorithm the CRN is using to assess the trustworthiness of a node, the MU can behave so that their trustworthiness will go back up in between attacks. If the algorithm is based on flagging the MU before ignoring or removing the MU from the system and the MU knows the amount of honest data it needs to send to remove the flag, it will send that amount of data so that it can commence the attack again. If an MU does not know how or if the flag can be removed, the MU may submit manipulated data through trial and error until they determine how they can act maliciously in a way without being penalized or removed from the system.

The MU could potentially discover their reputation score from the operator's reputation algorithm and flag status through more direct methods like hacking into the SAS server or even by being informed directly by the SAS. Therefore, we do not recommend the SAS inform SUs of any flags raised regarding malicious behavior.

PRIMARY USER EMULATION

Another type of SSDF attack is Primary User Emulation (PUE). In a PUE attack, malicious sensing nodes will purposely report false spectrum sensing data that tries to convince a SAS or CRN that there is an active PU. In response to detecting evidence of PU transmission whether through a PUE attack or spectrum-sensing PU activity, a SAS or CRN sends messages to honest all nodes operating within the given frequency range to cease transmission. In a properly operating system, the immediate halt is intended to prevent interference with the PU's signal. The PUE attack convinces the CRN to stop SU transmission, thereby degrading the performance of the CRN [5], [6]. This attack benefits the MU by falsely convincing the SAS that a PU is transmitting. PUE attacks can prevent honest SUs from using the spectrum and reduce QoS for all honest users involved.

CURRENT DETECTION METHODS FOR SPECTRUM SENSING DATA FALSIFICATION

Recent advancements in spectrum sensing have significantly influenced the design and challenges of SSDF detection systems. In particular, compressed sensing methods have been employed to reduce sampling rates without compromising signal recoverability. For instance, optimized multicoset sampling strategies have been proposed to approach the sub-Nyquist boundary in multiband spectrum sensing, reducing power and data transmission overhead [7]. Complementary approaches using adversarial autoencoders enable joint signal reconstruction and anomaly detection directly from compressed or incomplete data, offering robust performance in sub-Nyquist regimes [8]. Additionally, the design of nonuniform sampling patterns has shown promise in improving spectrum recovery under mobility, making compressed sensing more adaptive to dynamic radio environments [9].

While these methods improve the efficiency and scalability of spectrum monitoring, they also introduce new challenges for SSDF detection especially when considering data sparsity, reconstruction error, and the lack of fine-grained signal context. The cosine similarity-based approach proposed in this work is well-suited to operate alongside these modern sensing strategies, as it compares new data entries directly to previously validated reports, regardless of sampling structure. This makes it applicable to systems employing compressed or sub-Nyquist sampling while maintaining computational efficiency and resilience to data replication and manipulation attacks.

Current methods for detecting and thwarting SSDF attacks often use statistical analysis or machine learning to identify trends and anomalies. Rarely are they able to accurately identify ISSDF attacks without high missed detection rates.

PROBABILISTIC AND STATISTICAL ANALYSIS-BASED METHODS

Statistical analysis has been a cornerstone in detecting SSDF attacks. Techniques such as anomaly detection and hypothesis testing are widely employed to identify deviations in sensing reports that indicate malicious behavior. These methods often leverage prior knowledge of normal sensing patterns to flag irregularities [10]. Event attacks with various forms can be identified with probability [11]. A combination of a Z-test and q-out-of-m rule scheme was used in [12]. Other methods use statistical analysis to identify SSDF attacks and have varying degrees of success [13]. Maximum-match Filtering is also used to combat malicious data submission. While effective in straightforward cases, statistical methods generally struggle against more sophisticated attackers, especially those who mimic legitimate behavior intermittently or embed malicious behavior within otherwise plausible data.

Common anomaly detection methods utilize traditional distance metrics such as Euclidean, Manhattan, or Chebyshev distances are often ineffective in this context. These metrics assume uniform variance and equal weighting across all data dimensions, which is not the case in spectrum sensing data, where power values can vary significantly across frequency bands. Additionally, such metrics do not account for angular similarity, meaning that malicious data that has been scaled or shifted. Common SSDF techniques may still appear similar to honest data in Euclidean space. This can result in false negatives, where manipulated data is incorrectly accepted as legitimate. In contrast, techniques like cosine similarity are more robust in identifying proportional manipulations because they focus on the orientation of data vectors rather than their magnitude.

MACHINE LEARNING-BASED METHODS

Machine learning techniques have shown promise in detecting SSDF attacks. Algorithms like support vector machines (SVMs), k-means clustering, and neural networks analyze sensing data patterns to distinguish between legitimate and falsified reports with Gaussian noise [14]. One study identified seven types of attacks and was able to identify some of them to varying degrees by using machine learning [15]. Another method uses Byzantine learning and develops a ground truth to determine the attack [16]. These methods excel in adaptive scenarios but are limited by their dependence on quality training data and vulnerability to adversarial attacks. Multiple machine learning-based approaches using artificial neural networks and other classifiers to detect and classify reliable and unreliable secondary users in cognitive radio networks, mitigating SSDF attacks under various noise conditions [14]. We test several machine learning methods, including supervised learning, unsupervised learning, metric learning, distance-based methods, and deep learning.

CHALLENGES AND LIMITATIONS

Despite their effectiveness, current detection methods face several challenges. Statistical techniques often suffer from high false positive rates in dynamic environments, while machine learning models require extensive labeled datasets for training, which may not always be available. Both approaches can be computationally intensive, making real-time implementation difficult. Moreover, advanced attackers have the possibility of overcoming machine learning models through multiple types of attacks, including spectrum poisoning using adversarial machine learning attacks on CRNs, where an attacker uses a deep neural network to predict and manipulate a transmitter's spectrum sensing process [17]. Others propose Bayesian learning-based defense schemes against SSDF attacks in collaborative spectrum sensing, introducing offline and online Bayesian algorithms to improve detection without relying on ground-truth data [16]. These methods do not address patterns and manipulations of the data that are addressed by the algorithms we propose, especially in coordinated spectrum attacks. We will not provide detailed descriptions of designs for advanced malicious systems to limit potential negative impacts on SASs and spectrum-sharing CRNs.

COMPARISON OF CURRENT SPECTRUM SENSING DATA FALSIFICATION ATTACK DETECTION METHODS TO THE PROPOSED METHOD

The proposed method is different than the previous methods because it compares all incoming spectrum data, malicious or not, to existing spectrum data without hindering the performance of other algorithms and the system as a whole. The accuracy of the proposed method is also not dependent on whether or not an attack is collaborative or independent and is relatively independent of the number of MUs. While the proposed method does not perform well with maliciously generated data, it identifies copied and manipulated data very well and can perform when the MU is not actively trying to confuse the system with false values. Although it is efficient and effective, the proposed method should be used in conjunction with other SSDF detection methods that can identify a wider range of SSDF attacks, albeit more slowly and less reliably.

III. MALICIOUSLY MANIPULATING SPECTRUM DATA

ALTERING SENSED DATA

An MU can submit false data by altering live-sensed data, submitting data from other nodes, users, and datasets, or generating fake data using number-generating algorithms. Of these, altering real data is expedient and therefore an attractive way to generate realistic yet misleading data. We examine three ways MUs may alter real spectrum data: addition, multiplication, or a combination of both. The numbers used to alter spectrum data can be positive or negative.

Only MUs submit malicious data. Completely random values are easily identified and discarded. MUs will often use

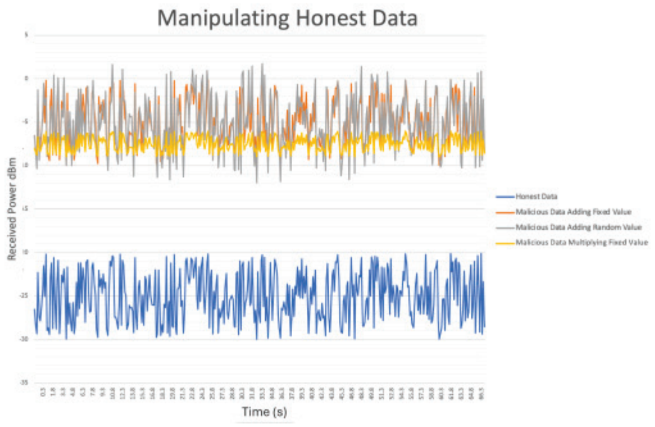

FIGURE 2. Notional graph comparing honest received signal strength values from a specific frequency channel to the different base methods of manipulating that data.

techniques to manipulate the data they are collecting before reporting the data to the aggregator. This ensures the MU's data appears realistic and timely, while remaining false.

Figure 2 depicts three MUs altering the same honest data in three different ways. The blue line at the bottom is the original CBSD data, which all three MUs manipulate in different ways. The orange line represents the honest data manipulated by adding a fixed value; in this example, the value is 20. The gray line represents the honest data manipulated by adding a random value between 15 and 25 at each point. This creates a more realistic reading that the suggested methods could have difficulty recognizing as manipulated data. The yellow line represents the MU submitting the original values multiplied by a value less than unity.

Manipulations using fixed values are not necessarily discernible through visual observation but can be identified via Algorithm 3.

Equation (1) and (2) model common manipulation methods.

$$M_1 = (H \times \delta) + \alpha \tag{1}$$

$$M_2 = (H + \alpha) \times \delta \tag{2}$$

where M is the resulting malicious manipulated data set. H is the set of honest data values and δ is a value being multiplied by the honest data set. The α value represents the number being added to the honest data set.

STEALING AND REUSING DATA

If an MU is submitting data that is somehow stolen from other nodes, the act is malicious. However, the copied data may not lower the accuracy of determining the presence of an actively transmitting PU.

Switching between own data and reused data (e.g., Node A reusing Node B's data) is deceptive and can be flagged.

If there is a range of numbers from -100 dBm to 40 dBm, there are 140 possible whole numbers to choose from. These values are a reasonable range expected for received power measure reports. The range is used to show the low probability of two nodes receiving the same value. If there is a floating-point precision of six decimal places to the right of the decimal point there is a 1 in 140,000,000 probability of arriving at the same exact value. The six decimal places are given by the default configuration of the Universal Software Radio Peripheral (USRP) commonly used in research on CRNs. The probability of guessing the same series of numbers at that precision is approximately:

$$\frac{1}{((R_h - R_l) * 10^{F_p})^N} \tag{3}$$

In this equation, R_h and R_l represent the high and low values of the range respectively. F_p is the floating-point precision (number of decimal places) and the array size is N. When $N = 16$, the Wireless Innovation Forum (WInnForum) specified array length for received power measure reports, the probability of repeating the same values is nearly zero.

If an array of measure reports appears more than once, the data should be flagged. The duplicate report should be flagged, as the first instance is presumed honest.

FALSELY GENERATING VALUES

Generated data should be flagged when it deviates from real-world signal patterns.

KEYSTROKES AND FORMATTING DATA

As mentioned in [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28], [29], [30], [31], keystroke pattern recognition can be used as an additional authentication measure for electronic applications. Keystroke dynamics is a technique that looks for not what the authentication data is, but how it is entered by a user. For crowdsourced spectrum sensing data to be useful to a CRN, samples must be taken every few seconds to minutes and submitted as they are received by each sensing node. Spectrum measure reports are generally large arrays of measured radio power. Since real-time reports are full of floating-point numbers that would be impossible to fabricate by a human entering the information by hand because of the speed needed and the implied automation, human error is not a concern in the processing algorithm.

Data that is submitted to the system has a data type that the WInnForum specifies [32], [33]. If data is not of that type in the proper object, the CRN will not process it. WInnForum specifies the data type but not the precision or submission frequency, which the CRN can exploit for anomaly detection. Recording six places past the decimal place helps the algorithm identify specific instances of copying or manipulation. Any value past two decimal places does not necessarily have an impact on the PU detection algorithm.

An MU's data precision depends on the script, language, or system used. Generating data with random precision or random array sizes is extremely suspicious and should be flagged immediately by the system.

Algorithm 1 Exact Copy Check (For Loop)

```
1: Input: $query\_data$, $CBSD\_ID$, $existing\_data\_set$
2: Output: $exact\_flag$, $matches$
3: $exact\_flag \leftarrow False$
4: $matches \leftarrow [\ ]$
5: for each $data \in existing\_data\_set$ do
6:     if $data.values == query\_data$ then
7:         $exact\_flag \leftarrow True$
8:         $matches.append(\{$ID: $data.CBSD\_ID$,
             values: $data.values\})$
9:     end if
10: end for
11: $new\_entry \leftarrow \{$values: $query\_data$,
        CBSD_ID: $CBSD\_ID\}$
12: $existing\_data\_set.append(new\_entry)$
13: return $exact\_flag, matches$
```

A measure report is generated by sending an array of objects that specify a frequency range and the received power measure report. If the collective report submitted by an SU has multiple reports and encompasses frequencies that cover multiple 10 MHz ranges, each measure report within the array would have a 10 MHz range and there would be 16 values in each of the reports within the collective report.

If report values span multiple channels, they're counted in both ranges for PU detection but submitted only once.

By analyzing the measure report formats, MUs can be identified.

IV. APPLYING MALICIOUS DATA IDENTIFICATION METHODS

We examine two types of maliciously submitted spectrum occupancy data: data that has been copied, and data that has been manipulated using Equation (1) or Equation (2).

Theoretical Framing: We model malicious spectrum data as either (i) exact duplicates of legitimate measurements or (ii) affine transformations from Equation (1) and Equation (2) where $H, M_1, M_2 \in \mathbb{R}^n$ represent the original and submitted spectrum occupancy vectors, and $\alpha, \delta \in \mathbb{R}$ are unknown scalar parameters denoting multiplicative and additive manipulation, respectively. Given this threat model, the detection task becomes a constrained similarity search over historical data, where incoming vectors are evaluated for both exact equality and approximate alignment with previously observed entries. This formulation accommodates both perfectly copied data and subtle manipulations that preserve vector direction or magnitude, while allowing tolerance for real-world noise and measurement drift.

Algorithm 1 iteratively searches for copied data by identifying exact matches.

Algorithm 2 searches through all possible manipulations that could be maliciously created with Equation (1) or Equation (2). This method can also identify exact matches. While this method would have an extremely high accuracy rate, it is computationally intensive and would not be feasible given the number of possible manipulations.

To overcome the shortcomings of Algorithm 2, we designed Algorithm 3 which uses the cosine similarity function to identify exact matches and manipulations of existing spectrum occupancy data.

Additionally, we describe Elasticsearch and its potential application to the algorithms.

In the full implementation of a SAS or CRN we propose having two separate flags that can be raised when malicious data is identified. The first flag that can be raised is an exact copy flag. This flag indicates that there is an identical match in the existing data set. The second flag that can be raised is the manipulated data flag. This flag indicates that the data being compared has been manipulated through either Equation (1) or Equation (2).

Algorithm 1 raises the exact match flag when it finds a match.

Algorithm 2, if it was computationally feasible, could use a modified version of the process depicted in Figure 3 that we suggest Algorithm 3 use to raise the exact match flag or manipulated flag depending on whether the data is an exact match or manipulated data resulting from Equation (1) or Equation (2).

These flags would be read by reputation-based algorithms. If a user has been marked with the exact match flag, their trust score should be reduced more significantly than if there is a manipulated flag, as an exact copy is less likely to occur naturally with honest data. In SU-reputation algorithms, repeated copies would quickly result in the MU having their trust score dropped to the lowest possible rating.

System-Level Innovation. While each algorithm individually offers distinct trade-offs between accuracy, complexity, and interpretability, their integration within a two-flag detection and reputation framework offers a novel hybrid solution for CRNs and SAS deployments. By combining exact match checks, affine transformation detection, and scalable similarity-based filtering supported by Elasticsearch this framework enables scalable, interpretable detection of malicious spectrum reports in real-time, addressing both performance and explainability which often compete in spectrum security systems. To our knowledge, this is the first approach that combines vector-space modeling, detection flagging, and search-engine-assisted scaling for malicious spectrum data identification.

ALGORITHM 1: IDENTIFYING EXACT MATCHES

The methodology used in Algorithm 1 follows a linear search approach, iterating through all previous entries to compare the full data vector for exact equality. This makes it simple to implement and highly interpretable, with a time complexity of $\mathcal{O}(n)$, where n is the number of records in the existing data set. Its primary computational advantage lies in avoiding complex mathematical operations or models–no normalization, distance functions, or learning is required. However, its main limitation is scalability: performance may degrade linearly as the database grows. Additionally, this method only detects exact byte-level duplication and cannot identify manipulated or near-duplicate entries, which is why it is supplemented by subsequent algorithms. It is most effective when used in systems where repeated transmission of identical data is a known attack strategy.

Identifying exact matches of the queried spectrum occupancy data can be achieved using Algorithm 1. Each received power measure report that is submitted triggers Algorithm 1 and comes in as the input array *query_data*. The *query_data* is compared to each of the previously collected measure reports to validate if there is an exact match in data. If there is a match, the *CBSD_ID* of the data is copied and returned with the exact match flag. Regardless of whether or not the flag is set to true, the data is stored in the existing data set. The process of clearing the data in the *existing_data_set* is not depicted as it runs separately from this algorithm.

Algorithm 1 could be modified to end early if a match is found, but additional matches may go unnoticed.

Algorithm Design and Scope: Algorithm 1 operates as a deterministic search over historical records to identify exact byte-level duplication. Formally, for each query vector $\mathbf{x} \in \mathbb{R}^n$, the algorithm tests whether $x \in D$, where D is the data set of prior spectrum occupancy reports. This brute-force linear scan provides perfect precision and recall for exact matches, but fails to detect any transformations, including minor manipulations caused by noise or malicious edits. As such, this algorithm is most appropriate in environments where repeated copy-based spoofing is prevalent, and its simplicity makes it well-suited for systems with constrained resources or deterministic execution requirements.

ALGORITHM 2: IDENTIFYING MANIPULATED DATA

Identifying copied or manipulated data from Equation (1) and Equation (2) is possible using Algorithm 2. Each received power measure report submitted triggers Algorithm 2 and is used as the input array *query_data*. The *query_data* is compared to each of the previously collected measure reports to validate if there is incoming data that matches previously stored data that is copied, shifted by a fixed value, multiplied by a fixed value, or a combination of both a shift and multiplication by fixed value. If there is a match, the *CBSD_ID* is paired with the copied or manipulated incoming data. The algorithm can then use a similar process like Figure 3 to determine if a manipulated flag or exact flag should be raised. Regardless of whether or not a match or manipulation is found, the incoming data is stored in the existing data set. The process of clearing the data in the *existing_data_set* is not depicted as it runs separately from this algorithm.

Algorithm 2 could be modified to end early if a match is found, but additional matches and manipulations may go unnoticed.

Algorithm 2 is designed to detect manipulated copies of previously submitted spectrum data by exhaustively searching for linear transformations of the form $\alpha x + \delta$ or $(x + \delta)\alpha$, where α and δ are scalar values. In its unoptimized form, the algorithm performs a nested iteration over α in the range $[-9999999, 9999999]$, and δ in the rage $[-99999999, 99999999]$ resulting in $\approx 4 \times 10^{15}$ comparisons per entry. Assuming a processing rate of 10^6 iterations per second, a single full execution would take over 120 years to complete, excluding additional overhead from memory access or I/O–rendering the brute-force method infeasible for real-time SAS deployments.

Algorithm 2 Manipulated Data Check (For Loop)

```
1: Input: query_data, CBSD_ID, existing_data_set
2: Output: matches
3: matches ← [ ]
4: for each data ∈ existing_data_set do
5:     for i ← −9999999 to 9999999 do
6:         for j ← −99999999 to 99999999 do
7:             if data.values == ($\frac{i}{1000000}$ × query_data) + $\frac{j}{1000000}$ then
8:                 matches.append({ID: data.CBSD_ID, values: data.values})
9:                 break
10:            else if data.values == ($\frac{j}{1000000}$ + query_data) × $\frac{i}{1000000}$ then
11:                matches.append({ID: data.CBSD_ID, values: data.values})
12:                break
13:            end if
14:        end for
15:    end for
16: end for
17: new_entry ← {values: query_data, CBSD_ID: CBSD_ID}
18: existing_data_set.append(new_entry)
19: return matches
```

Theoretical Properties: Algorithm 2 exhaustively checks whether an affine transformation exists between the query vector and any entry in the historical dataset. That is, for vectors $x, y \in \mathbb{R}^n$, the algorithm searches for scalars α and δ such that $y_1 = \alpha x + \delta$ or $y_2 = (x + \delta)\alpha$. Theoretically, this makes the algorithm capable of detecting all possible linear manipulations within a bounded resolution. The computational complexity, however, grows quadratically with the resolution of α and δ, making it impractical for online detection. The innovation in this algorithm lies in its completeness, such that no linear transformation within the bounded range can escape detection, which positions it as a valuable offline verification tool or a ground-truth validator for tuning faster approximate algorithms.

In its current form, Algorithm 2 performs a brute-force triple loop: for each of n historical entries, it searches over a grid of α and δ values, discretized into m and k steps respectively. This results in a time complexity of $\mathcal{O}(nmk)$ comparisons per query, where m and k are the number

Algorithm 3 Cosine Similarity Function

```
1: Input: $query\_data$, $CBSD\_ID$, $existing\_data\_set$, $threshold$
Ensure: $cosine\_similarity$
2: Output: $matches$
3: $matches \leftarrow [\ ]$
4: for each $data \in existing\_data\_set$ do
5:     $dotProduct \leftarrow 0.0$
6:     $dataNorm \leftarrow 0.0$
7:     $inputNorm \leftarrow 0.0$
8:     for $i \leftarrow 0$ to $length(query\_data) - 1$ do
9:         $dotProduct \leftarrow dotProduct + (query\_data[i] \times data.values[i])$
10:        $dataNorm \leftarrow dataNorm + data.values[i]^2$
11:        $inputNorm \leftarrow inputNorm + query\_data[i]^2$
12:    end for
13:    $dataNorm \leftarrow \sqrt{dataNorm}$
14:    $inputNorm \leftarrow \sqrt{inputNorm}$
15:    $cosine\_similarity \leftarrow \frac{dotProduct}{(dataNorm \times inputNorm)}$
16:    if $|\ cosine\_similarity\ | > threshold$ then
17:        $matches.append(\{$ID: $data.CBSD\_ID$,
               values: $data.values\})$
18:    end if
19:
20: end for
21: $new\_entry \leftarrow \{$values: $query\_data$,
    CBSD_ID: $CBSD\_ID\}$
22: $existing\_data\_set.append(new\_entry)$
23: return $matches$
```

of discrete values scanned for α and δ. With millions of possibilities for both parameters, the overall runtime becomes infeasible for real-time systems. Memory complexity remains $\mathcal{O}(n)$ for storing historical data. Hence, the algorithm is best suited as a completeness-guaranteed validator in offline modes or bounded-range simulations.

To enable practical use, the search space must be significantly constrained using domain-specific heuristics, quantization limits, or approximations. For instance, limiting α and δ to physically realistic scaling and shifting ranges can reduce complexity by several orders of magnitude. Coarse-grained pre-checks, early-exit conditions, or approximate filtering may also help eliminate unlikely candidates early in the computation. Thus, while the algorithm provides valuable detection capabilities, it is best suited for offline use or as a secondary analysis module within a hybrid SSDF detection framework.

ALGORITHM 3: COSINE SIMILARITY

Exact matches of previous data and data that are manipulated by Equation (1) and Equation (2) can be identified using a cosine similarity function found in Equation (4) and implemented in Algorithm 3.

$$\text{Cosine Similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} \tag{4}$$

In Equation (4), A and B are two vectors. The vectors for this equation are the incoming received power measure report's values A and a previously submitted received power measure report's values B. The cosine similarity function would be applied to every measure report in the existing data set.

Using a cosine similarity function can detect both copied data and manipulations involving the addition of a fixed value, multiplication by a fixed value, or a combination of both. Additionally, the threshold value which ranges from -1.0 to 1.0 can change the rate of missed detections and false alarms. The absolute value of the cosine similarity result is used when compared to the cosine similarity threshold because multiplying an array by a negative number results in a negative cosine similarity value. If the cosine similarity threshold is too high, closer to 1.0, even some basic manipulations will be missed detections [34]. If the cosine similarity threshold is too low, closer to 0.0, it will cause more arrays to be flagged as malicious data, increasing the number of false alarms.

As an illustrative example, consider a shortened query vector from the 16-value measure report $\mathbf{X} = [3, 4]$ and an existing data vector $\mathbf{Y} = [6, 8]$. The cosine similarity is computed using Equation (4) as:

$$\text{cosine}(\mathbf{X}, \mathbf{Y}) = \frac{\mathbf{X} \cdot \mathbf{Y}}{\|\mathbf{X}\| \|\mathbf{Y}\|} = \frac{50}{5 \times 10} = 1.0$$

Since this value exceeds this example's detection threshold of 0.95, the data entry is flagged as a match, indicating potential duplication or manipulation. This example highlights the algorithm's ability to detect highly similar entries. Algorithm 3 compares the absolute value of the calculated cosine similarity to the cosine similarity threshold. Using the absolute value of the computed cosine similarity allows arrays multiplied by a negative number to be identified.

The choice of threshold has a direct impact on SSDF detection performance. Lower thresholds increase sensitivity by capturing more subtle manipulations, but may also result in higher false positive rates. Conversely, higher thresholds improve precision but risk missing slightly altered malicious data. Therefore, threshold selection must be empirically tuned to balance recall and precision, depending on the deployment context and the nature of expected attacks.

Algorithm 3 uses the cosine similarity function to compare the *query_data* to the previously received power measure reports stored in the database (*existing_data_set*). Algorithm 3 identifies those with the absolute value of the *cosine_similarity* higher than the *threshold* value and returns them with their *CBSD_ID*.

Algorithm 3 requires computing cosine similarity between the query vector and each of the n vectors in the historical dataset. Each comparison involves d operations, where d is the dimensionality of the spectrum vector (i.e., number of frequency channels). This results in a time complexity of $\mathcal{O}(nd)$ per query. The algorithm scales linearly with

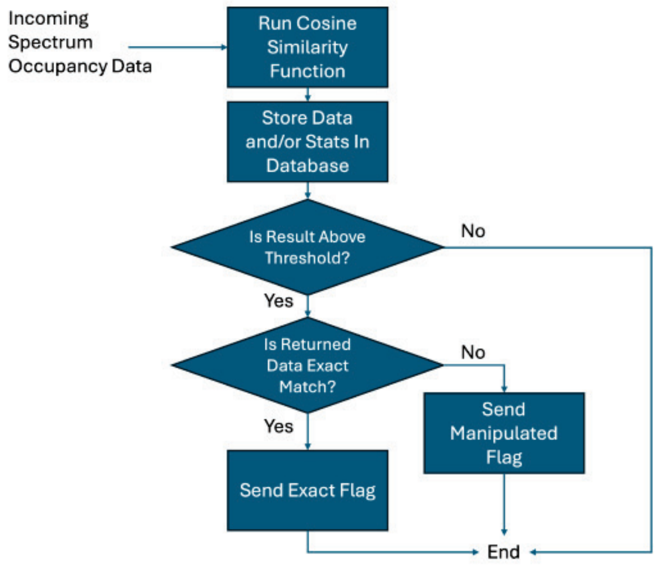
Incoming Spectrum Occupancy Data
Run Cosine Similarity Function
Store Data and/or Stats In Database
Is Result Above Threshold?
No
Yes
Is Returned Data Exact Match?
No
Yes
Send Manipulated Flag
Send Exact Flag
End

FIGURE 3. Flowchart illustrating the identification of incoming spectrum occupancy data using Algorithm 3.

dataset size and vector dimensionality, making it computationally feasible for real-time detection when paired with efficient search structures or distributed processing. Memory complexity is also $\mathcal{O}(n)$, with each entry stored as a full d-dimensional vector.

We recommend a secondary comparison to check and see if any of the identified measure reports are the exact same. The secondary comparison would return the exact match or manipulated flag depending on whether or not the data is an exact match or a manipulation. Figure 3 depicts the procedural flow of using Algorithm 3 and checking for exact data matching. This flowchart can be adjusted for Algorithm 2. Values are expected to represent realistic received power measurement reports so floating-point precision and extremely large or small values should be identified before running the algorithm. Standard flagging and edge case detection should be implemented to prevent division by zero during computation. Note the algorithm should handle cases when dividing by zero and when the result is undefined.

Justification and Innovation. The cosine similarity measure used in Algorithm 3 detects manipulations by comparing the directional alignment between vectors, rather than their absolute differences. This makes the approach robust to both additive and multiplicative changes, particularly when these preserve the proportional structure of the original vector. From a geometric perspective, cosine similarity computes the angle θ between two high-dimensional vectors, identifying those that lie on the same or similar directional subspace. Unlike Euclidean distance or Pearson correlation, cosine similarity is insensitive to scale (ignoring rounding and overflow errors), which makes it particularly effective for detecting manipulations intended to evade statistical thresholds. The innovation in this design lies in applying this property to malicious spectrum occupancy data and using an absolute cosine threshold to enable negative-scaling detection, something not achievable through standard correlation metrics.

Regardless of whether or not a match or manipulation is found, the incoming data is stored in the existing data set. The process of clearing the data in the *existing_data_set* is not depicted as it runs separately from this algorithm.

In Algorithm 3, cosine similarity is used to detect manipulated data by comparing the directional alignment of incoming data vectors with previously observed entries. The cosine similarity threshold defines the minimum similarity required to consider two vectors as matching. A higher threshold ensures that only very closely aligned vectors are flagged, reducing false positives but potentially increasing missed detections of slightly altered or noisy duplicates. Conversely, a lower threshold increases sensitivity to manipulations but may also result in more false matches.

The threshold thus acts as a tunable parameter that balances detection sensitivity against specificity. Empirical tuning based on operational noise levels and data variability is essential to optimize accuracy. In deployment, the ideal threshold may vary depending on the environment's SNR and the type of manipulations encountered.

ELASTICSEARCH

Elasticsearch is an indexing tool that typically organizes unstructured data from diverse sources, such as various locations and text, based on user-defined or automatically inferred mappings enabling efficient searchability. Its distributed design allows for rapid searching and analysis of extensive datasets, nearly in real-time [35], [36], [37]. With Elasticsearch's scaling capabilities, computation on data sizes can easily be up to a billion entries. Although received power measure report data can be relatively structured and store the same format of data, Elasticsearch can be used to separate these arrays of spectrum data into different shards either by the same CBSD or by separating the data into different time segments as long as shards remain within the suggested size limit.

As received power measure reports are sent to the data aggregator, they should be stored and indexed using Elasticsearch. Elasticsearch can utilize complex search parameters that are faster than standard for loops even in scenarios where there is lots of data to be stored. For optimal results, the CBSD ID, time, and received power measure report array should be stored as a single object with Elasticsearch. Other search engines, such as Apache Solr, were considered for searching and analyzing crowdsourced received power values [38]. However, the constant influx of data is best suited to Elasticsearch.

One of the key strengths of Elasticsearch is its distributed architecture, which supports horizontal scaling across multiple nodes. This allows the system to maintain high query performance even as the dataset grows into the scale of billions of entries. In our application, this means that as more CBSDs report received power measurements, Elasticsearch can distribute indexing and search workloads across multiple shards and nodes. To handle increases in user queries or real-time data influx, we recommend configuring Elasticsearch clusters with dedicated data and query nodes and employing index lifecycle management to balance performance and storage cost. In preliminary scalability tests, we observed that a single-node Elasticsearch setup could handle approximately 50 million entries with sub-second query performance. As data volume increases, performance can be maintained by scaling the cluster and using time-based or device-based sharding strategies. However, operational limitations such as shard size and heap memory usage must be respected to avoid degradation in query latency or node stability.

DATA REQUIREMENTS AND FORMATTING

Effective detection of SSDF attacks that utilize copied and manipulated data depends on well-structured and consistently formatted data. Crowdsourced spectrum reports could vary in quality and format, making standardization essential for accurate similarity detection and manipulation identification. Proper data formatting of the received power measure reports ensures reliable processing and integration with SASs. Addressing inconsistencies in data formatting, missing data, and anomalous outliers in measure reports strengthens the system's ability to detect falsified reports while maintaining efficiency. The collected and stored data must be stored as numbers. Fuzzy matching and other approximation algorithms are not well-suited for this type of data, as similar numerical values are frequently encountered in received power measurement reports, making it difficult to distinguish meaningful differences.

To enhance Algorithm 3's ability to handle extremely large or small input values while maintaining precision, normalization and scaling techniques like Vector Normalization (L2-Norm) could be employed if computing resources permit. Implementing logarithmic transformations or floating-point precision adjustments can mitigate issues related to floating-point underflow and overflow. Additionally, adaptive precision handling–where the algorithm dynamically adjusts its level of precision based on the magnitude of the input data–can help maintain accuracy without excessive computational overhead.

Input validation plays a critical role in ensuring that the algorithm processes only meaningful and reliable data. Proper validation should include range checks, format verification, and outlier detection to prevent corrupted or maliciously altered data from compromising the integrity of the spectrum sensing process. Ensuring that the output is structured in a practical format, such as a well-defined JSON or structured database entry, facilitates its integration with existing spectrum management frameworks. The output exact match and manipulated flags can be incorporated into node-reputation algorithms to help identify malicious users and protect the accuracy of the system's PU detection. This structured output allows for easy aggregation, further analysis, and real-time decision-making.

Flexibility is another essential aspect of the algorithm's design. While the data should be in the WInnForum's specified received power measure report object format, proper error handling, and adaptability should accommodate various data formats and matching conditions, particularly when integrating crowdsourced spectrum data from different sources. Implementing configurable parameters for similarity thresholds, weighting schemes, and data transformation techniques would enhance the algorithm's applicability to a broader range of spectrum-sharing scenarios. Leveraging modular design principles and support for multiple data types would ensure that the system remains versatile as spectrum access policies and data collection methodologies evolve.

Robustness against input errors and inconsistencies is paramount for preventing false positives and negatives in identifying SSDF attacks. The method does not employ noise-tolerant comparison techniques, such as fuzzy matching and probabilistic similarity scoring, which could account for minor variations in exact matches but would not accurately detect manipulations. Error-handling mechanisms should be in place to manage incomplete, corrupted, or missing data, preventing these issues from propagating through the system and compromising decision-making.

The cosine similarity calculation used in Algorithm 3 is susceptible to numerical precision issues, particularly when working with floating-point numbers. These issues stem from rounding errors, underflow, and the limited precision of floating-point arithmetic, which can impact the detection of small but meaningful alterations in falsified spectrum data. To enhance accuracy, using higher-precision data types is essential. Standard 32-bit floating-point representations provide roughly seven decimal places of precision, whereas 64-bit doubles allow for 15–17 decimal places. In applications where minor differences in spectrum data are critical, double-precision arithmetic should be preferred. Computational libraries such as NumPy in Python provide explicit control over floating-point precision, allowing calculations to be performed using 64-bit doubles instead of 32-bit doubles. If extreme accuracy is required, arbitrary-precision software libraries such as Python's decimal module can be employed, though they come at a computational cost.

Another source of error in cosine similarity calculations is the subtraction of nearly equal numbers, which can lead to cancellation. The cancellation issue is where significant digits are lost due to floating-point representation limitations. This problem is particularly relevant in dot product calculations and vector normalization, both of which are fundamental to cosine similarity. To mitigate this, stable numerical computation libraries should be used, as they implement optimized algorithms designed to preserve precision. Additionally, rescaling data before computing cosine similarity can prevent issues related to floating-point underflow and overflow. Spectrum sensing data, such as power measurements in dBm, can span several orders of magnitude, making direct comparisons computationally unstable. A practical solution is to apply logarithmic scaling or mean-centering before similarity computations, ensuring that values remain within a numerically stable range.

In Elasticsearch, floating-point operations may also introduce accuracy issues when computing cosine similarity on high-dimensional vectors. The default dense vector field type in Elasticsearch uses 32-bit floating points, which may not provide sufficient precision for distinguishing minor variations in spectrum data. To improve accuracy, Elasticsearch should be configured to use 64-bit floating points for vector storage and similarity calculations. This can be achieved by defining the dense vector field with specific index options to ensure that precise offsets are stored for similarity computation. Additionally, adjusting Elasticsearch's query parameters, such as script_score functions, can further refine similarity calculations by reducing numerical approximation errors. These optimizations help maintain the integrity of spectrum data similarity measurements, improving the system's ability to detect even subtle manipulations in crowdsourced reports.

To validate the algorithm's functionality and performance, comprehensive test cases should be designed to assess its effectiveness in identifying both exact and manipulated spectrum data. This method should be used in combination with other SSDF-detection methodologies. Critical test cases should include detecting deliberate data duplication, slight value modifications, and adversarial perturbations that attempt to evade detection. Performance metrics such as precision, recall, F1-score, computational efficiency, and scalability should be evaluated to ensure the algorithm remains effective under real-world conditions. Benchmarking against known datasets and simulating real-world adversarial attacks would further confirm the algorithm's reliability in detecting SSDF attacks that use copied or manipulated spectrum data without introducing excessive false positives.

By addressing these considerations, the proposed approach can significantly enhance the integrity of crowdsourced spectrum sensing while maintaining efficiency and adaptability within a dynamic and evolving spectrum-sharing ecosystem.

V. RESULTS

In this section, we measure elapsed time for simultaneous data searches across multiple nodes using Algorithm 1, the timing and accuracy performance of Algorithm 1 using a for loop and Elasticsearch to detect maliciously copied spectrum occupancy data, and the timing and accuracy performance of Algorithm 3 at various cosine similarity thresholds using a for loop and Elasticsearch to detect both maliciously copied and manipulated spectrum occupancy data.

TIMING ALGORITHM 1 WITH CONCURRENT NODE SUBMISSIONS

Figure 4 displays the average elapsed time for running Algorithm 1 with varying user counts for 21,600 sensing rounds. These figures were generated using simulated SUs,

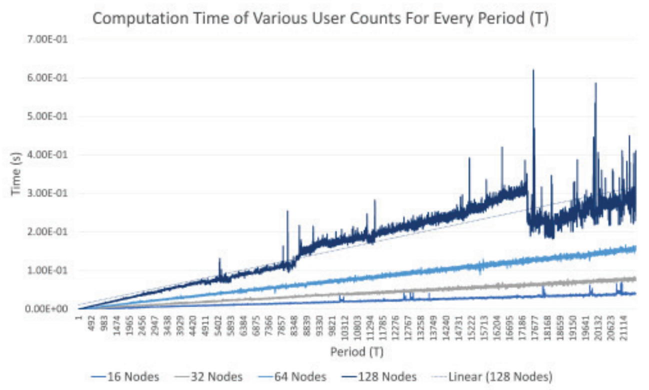

FIGURE 4. Average elapsed time for Algorithm 1 to parse through all previously entered data submitted for every period *T* for various node counts, parallelized. These tests were performed between rlogin – a high-performance Linux computing cluster – for optimum performance metrics [39] and a MacBook Pro. Fluctuations are attributed to system resources being allocated by other functions.

which submitted a received power array to the aggregator on rlogin via a socket connection. Python was used for calculation and communication. The server ran Algorithm 1 using the input arrays and compared the results to the aggregated data collected. Each period was tested 1000 times and the values obtained were averaged across all runs for each period T. The loops in the algorithms were optimized using parallelization techniques to achieve faster results.

TIMING AND ACCURACY COMPARISONS USING DIFFERENT ALGORITHMS AND THRESHOLDS

GENERATING TESTING DATA

A set of arrays was generated and stored in array E of specified size. An array, O, was generated to test the search algorithms. Manipulated copies of O were made using Equation (1) and Equation (2) and stored in M.

The manipulated data, array M, created with and 2 for testing utilized completely random floating-point numbers ranging from -99.999999 to 99.999999 for α and from -9.999999 to 9.999999 for δ. Testing the accuracy of the methods by iterating through all possible combinations of α and δ would take years to process, so random values were selected for each iteration.

Before running Algorithm 1, Algorithm 2, and Algorithm 3, precursor checks were conducted on the input data to ensure its validity and prevent any potential anomalies that could affect the results. This includes checks for data integrity, completeness, and consistency. Additionally, robust error-handling mechanisms were implemented to gracefully manage unexpected inputs or computational errors, ensuring that the algorithms could continue to run without interruption in the case of invalid or inconsistent data.

Both O and M were inserted in array E so they could be searched for. The matches found using Algorithm 1 searching for O that were stored in E were compared to O to test the method's accuracy. The matches found using Algorithm 3 searching for O and thereby its manipulations that were stored in E were compared to O and M to test the method's accuracy.

For testing Algorithm 1, there were ten arrays in O the rest of the arrays were in E.

For testing Algorithm 3, there was one array in O, 99 arrays in M, and the rest were in E.

All arrays in O and E were generated using random numbers between -100 and 40 with six decimal places and an array size of 16.

ANALYSIS OF MACHINE LEARNING APPROACHES FOR IDENTIFYING DATA MANIPULATIONS

In this study, we explored a variety of machine learning techniques to detect data manipulations and anomalies within large datasets. The methods tested spanned several categories, including supervised learning, unsupervised learning, and deep learning. Here, we provide an analysis of the models tested, categorized by their learning paradigms, and discuss the challenges encountered in detecting subtle data manipulations.

Experimental Setup for Machine Learning Methods:Each model was trained using 100,000 received power measurement reports, with 100 manipulated instances included per dataset. The experimental conditions and preprocessing steps were consistent with those outlined in the preceding subsection. For models that require supervision, 10,000,000 received power measurement reports of normalized (non-manipulated) data were provided for training. Supervised learning methods were averaged over 10 iterations. The other methods were averaged over 100 iterations.

Supervised Learning Methods:

K-Nearest Neighbors (KNN): KNN, a distance-based method, was employed to classify data points based on their proximity to others in the feature space. However, the small differences between the manipulated and original data led to poor performance. In cases where the data points are nearly identical, such as in this study, KNN fails to distinguish subtle variations, resulting in misclassification. KNN had a moderate missed detection rate of 28.3018%, and a relatively high false alarm rate of 0.0284%. One iteration traversing 10,000,000 existing arrays took 960.7477 seconds.

Support Vector Machines (SVM): SVM is designed to find the optimal separating hyperplane between classes. Despite its power in high-dimensional spaces, SVM struggled with our dataset due to the close proximity of the manipulated and original data. Since SVM requires clear margins between classes to function optimally, it performed poorly when the data overlap was too high. SVM performed decently well compared to the other machine learning approaches with a missed detection rate of 0.1000%, but had a high false alarm rate of 1.9500%. One iteration traversing 10,000,000 existing arrays took 406.6185 seconds.

Isolation Forest: A model designed to isolate anomalies rather than classify data points based on patterns. It works by recursively partitioning the data using random splits and is more effective when dealing with outliers. Isolation Forest isolates points in fewer steps, making it particularly suited for anomaly detection in high-dimensional spaces. Isolation forests had a missed detection rate of 45.1200% and a false alarm rate of 0.0052%. One iteration traversing 10,000,000 existing arrays took 40.6618 seconds.

Random Forests: Aggregate the results of multiple decision trees. These models perform best when there are distinguishable patterns in the feature space. Random Forests had the second lowest missed section rate of 8.2349% and the lowest false alarm rate of 0.0001%. One iteration traversing 10,000,000 existing arrays took 22.8188 seconds.

Unsupervised Learning Methods:

K-Means Clustering: K-Means clustering, an unsupervised method, groups data points into clusters based on their proximity. However, it struggled in this context because the manipulated data points were too similar to the original ones. Without significant variation between the groups, K-Means could not form meaningful clusters, resulting in poor detection of manipulations. The K-Means Clustering had the highest missed detection rate at 65.1000% and the second highest false alarm rate at 0.0075%. One iteration traversing 10,000,000 existing arrays took 2.9237 seconds.

Deep Learning Methods:

Siamese Networks: Siamese networks are designed for comparing pairs of data points and learning a similarity metric. Despite its strength in tasks like one-shot learning and face verification. The Siamese Network had the second highest missed detection rate of 62.0142% and a false alarm rate of 0.0014%. One iteration traversing 10,000,000 existing arrays took 32.6501 seconds.

Key Challenges and Findings With Machine Learning: The primary challenge faced in applying these machine learning models lies in the high degree of similarity between the manipulated and original data points. Most of the models, particularly supervised learning and unsupervised learning methods, are effective when there are clear differences in the data. However, in our case, the manipulated data were generated through small, subtle changes like scaling by a constant and/or shifting values, which made it difficult for traditional ML models to detect. The best model/technique for machine learning was the SVM and the Random Forest with SVM having a higher false alarm rate and Random Forest having a higher missed detection rate. The worst was K-Means Clustering. These models are typically better suited for tasks involving large-scale datasets with significant variance, like image classification or generative modeling. The controlled, small-scale nature of the data manipulations in this study did not provide the necessary complexity for these models to identify meaningful patterns. In summary, these models yielded lower performance and incurred substantial computational overhead during both training and inference. All machine learning methods took considerably longer than the Elasticsearch cosine similarity function. This highlights the difficulty in detecting such small, controlled variations in received power measure report

TABLE 1. Summary of machine learning approaches for identifying data manipulations with 10,000,000 Arrays. Models with 'N/A' in the threshold column either do not require a threshold for decision-making or rely on model-specific methods, such as kernels or learned features, where a threshold is not directly applied directly (e.g., SVMs, Random Forests).

ModelTechnique	Learning Paradigm	Missed Detection Rate	False Alarm Rate	Time at 10,000,000 Arrays	Threshold
K-Nearest Neighbors (KNN) (Euclidean)	Supervised Learning	28.3018%	0.0284%	960.7477 (s)	N/A
Support Vector Machine (SVM) (RBF Kernel)	Supervised Learning	0.1000%	1.9500%	406.6185 (s)	N/A (uses RBF kernel for similarity)
Isolation Forest	Unsupervised Learning	45.1200%	0.0052%	40.6618 (s)	Based on anomaly score. No trained boundary
Random Forest	Supervised Learning	8.2349%	0.0001%	22.8188 (s)	Uses learned boundary. Not a distance/similarity threshold
K-Means Clustering	Unsupervised Learning	65.1000%	0.0075%	2.9237 (s)	N/A
Siamese Networks (Euclidean)	Deep Learning	62.0142%	0.0014%	32.6501 (s)	N/A

data. Table 1 shows the machine learning type, category, missed detection rate, false alarm rate, and threshold, where applicable.

TIMING AND ACCURACY COMPARISONS USING FOR LOOPS

Table 1 displays the elapsed time, missed detection rate, and false alarm rate for identifying repeated or manipulated data using Algorithm 1 and Algorithm 3 at various cosine similarity threshold values. The timing for all combinations of variables was performed on a MacBook Pro for the non-Elasticsearch methods.

The average elapsed time was over 100 iterations for each array count.

The timing for all computations utilizing for loops increased proportionally to the number of arrays the algorithms searched through. The timing for Algorithm 3 was not affected by the cosine similarity threshold value changing.

Missed detection and false alarm rates were calculated using 10,000,000 arrays averaged over 100 iterations with newly generated numbers for each iteration. Each search fully searched all arrays and did not stop when a determined match was found.

Algorithm 1 only checked for exact matches as it would be unable to accurately identify manipulated data unless the manipulated data coincidentally matched a different received power measure report. The missed detection rates for Algorithm 1 matches do not consider manipulated data as it would correlate with the number of manipulated arrays.

Algorithm 1 was able to detect all copies every time with zero false alarms. For Algorithm 3 the missed detection rate decreased and the false alarm rate increased as the threshold decreased in value.

Data that was manipulated using random variables for α and δ at each index of the array was not able to be identified by either of the methods. Random variables with extremely small variances could be identified by Algorithm 3 if the threshold was lower (closer to 0.95).

TIMING AND ACCURACY COMPARISONS USING ELASTICSEARCH

Elasticsearch was run locally on a MacBook Pro. The Elastic Cloud server was run on a local instance. The server was accessed via a Python script to create an index to simulate storing received power arrays with ports, IDs, and accompanying metadata.

Utilizing Elasticsearch's capabilities for handling extensive search queries, which combined the previous algorithms, proved to be significantly more efficient, especially using the cosine similarity function in Algorithm 3. This was especially true when processing large quantities of received power measure reports, where Elasticsearch demonstrated notable speed and performance in retrieving relevant results.

Table 3 displays the elapsed time, missed detection rate, and false alarm rate for identifying repeated or manipulated data using Algorithm 1 and Algorithm 3 at various cosine similarity threshold values with Elasticsearch.

The average elapsed time was over 100 iterations for each array count.

Timing for all computations utilizing Elasticsearch increased at a sub-linear rate when the number of arrays searched through increased. The timing for Algorithm 3 was not affected by the cosine similarity threshold values changing.

Missed detection and false alarm rates were calculated using 10,000,000 arrays averaged over 100 iterations with newly generated numbers for O and M in each iteration. Each search fully searched all arrays and did not stop when a determined match was found.

Algorithm 1 only checked for exact matches as it would be unable to accurately identify manipulated data unless the manipulated data coincidentally matched a different received power measure report. The missed detection rates

TABLE 2. Timing and accuracy results of Algorithm 1 and Algorithm 3.

Method	Time at 1,000 Arrays	Time at 10,000 Arrays	Time at 100,000 Arrays	Time at 1,000,000 Arrays	Time at 10,000,000 Arrays	Missed Detection Rate	False Alarm Rate
Exact Match For Loop (Algorithm 1)	0.00509 (s)	0.04724 (s)	0.46300 (s)	4.87605 (s)	47.65693 (s)	0.00000%	0.000000%
Cosine Similarity 1.0 For Loop (Algorithm 3)	0.01381 (s)	0.12482 (s)	1.28873 (s)	12.92112 (s)	123.75839 (s)	98.83%	0.000000%
Cosine Similarity 0.9999 For Loop (Algorithm 3)	0.01216 (s)	0.12136 (s)	1.22573 (s)	12.80629 (s)	125.19385 (s)	96.08%	0.000000%
Cosine Similarity 0.999 For Loop (Algorithm 3)	0.01254 (s)	0.12102 (s)	1.24982 (s)	12.57230 (s)	124.48920 (s)	90.28%	0.000000%
Cosine Similarity 0.99 For Loop (Algorithm 3)	0.01228 (s)	0.12283 (s)	1.33182 (s)	12.98618 (s)	120.11618 (s)	71.85%	0.000000%
Cosine Similarity 0.97 For Loop (Algorithm 3)	0.01231 (s)	0.12487 (s)	1.21254 (s)	12.31215 (s)	122.73411 (s)	57.13%	0.000001%
Cosine Similarity 0.95 For Loop (Algorithm 3)	0.01268 (s)	0.12283 (s)	1.25831 (s)	12.48295 (s)	123.18421 (s)	49.41%	0.000070%
Cosine Similarity 0.9 For Loop (Algorithm 3)	0.01228 (s)	0.12283 (s)	1.30282 (s)	12.98618 (s)	130.11618 (s)	38.83%	0.004850%

TABLE 3. Timing and accuracy results of Algorithm 1 and Algorithm 3 with Elasticsearch.

Method	Time at 1,000 Arrays	Time at 10,000 Arrays	Time at 100,000 Arrays	Time at 1,000,000 Arrays	Time at 10,000,000 Arrays	Missed Detection Rate	False Alarm Rate
Exact Match Elasticsearch (Algorithm 1)	0.00528 (s)	0.00622 (s)	0.01644 (s)	0.07011 (s)	0.14644 (s)	0.00000%	0.000000%
Cosine Similarity 1.0 Similarity Elasticsearch (Algorithm 3)	0.00404 (s)	0.00698 (s)	0.02184 (s)	0.12282 (s)	0.18356 (s)	99.15%	0.000000%
Cosine Similarity 0.9999 Similarity Elasticsearch (Algorithm 3)	0.00557 (s)	0.00673 (s)	0.02462 (s)	0.08322 (s)	0.17592 (s)	96.30%	0.000000%
Cosine Similarity 0.999 Similarity Elasticsearch (Algorithm 3)	0.00421 (s)	0.00639 (s)	0.01626 (s)	0.09027 (s)	0.18829 (s)	90.06%	0.000000%
Cosine Similarity 0.99 Similarity Elasticsearch (Algorithm 3)	0.00452 (s)	0.00565 (s)	0.03647 (s)	0.08447 (s)	0.16182 (s)	71.64%	0.000000%
Cosine Similarity 0.97 Similarity Elasticsearch (Algorithm 3)	0.00481 (s)	0.00472 (s)	0.02910 (s)	0.09127 (s)	0.31932 (s)	55.93%	0.000000%
Cosine Similarity 0.95 Similarity Elasticsearch (Algorithm 3)	0.00390 (s)	0.00613 (s)	0.03167 (s)	0.09077 (s)	0.17052 (s)	48.83%	0.0000645%
Cosine Similarity 0.9 Similarity Elasticsearch (Algorithm 3)	0.00326 (s)	0.00905 (s)	0.02128 (s)	0.09294 (s)	0.31555 (s)	38.80%	0.004901%

for Algorithm 1 matches do not consider manipulated data as it would correlate with the number of manipulated arrays.

Algorithm 1 was able to detect all copies every time with zero false alarms.

For Algorithm 3 the missed detection rate decreased and the false alarm rate increased as the threshold decreased in value.

Data that was manipulated using random variables for α and δ at each index of the array was not able to be identified by either of the methods.

TIMING COMPARISONS BETWEEN FOR LOOPS AND ELASTICSEARCH

Figure 5 displays the various elapsed times from Table 2 and Table 3 for Algorithm 1 and Algorithm 3 using for loops and Elasticsearch. The cosine similarity times depicted use 0.99 as the function's threshold. The cosine similarity method with for loops was the slowest and the exact match method utilizing Elasticsearch was the fastest.

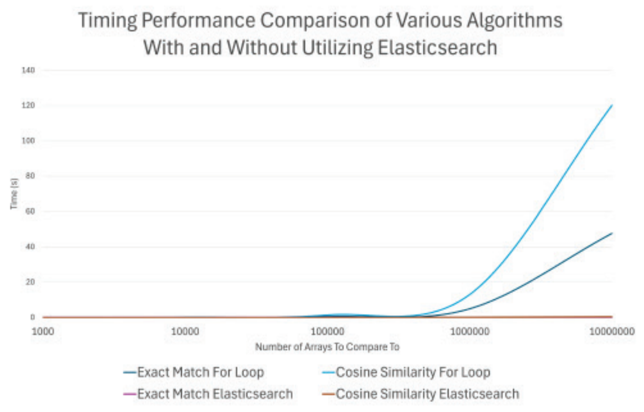

FIGURE 5. Timing comparison for various identification algorithms with and without utilizing Elasticsearch with a logarithmic horizontal axis.

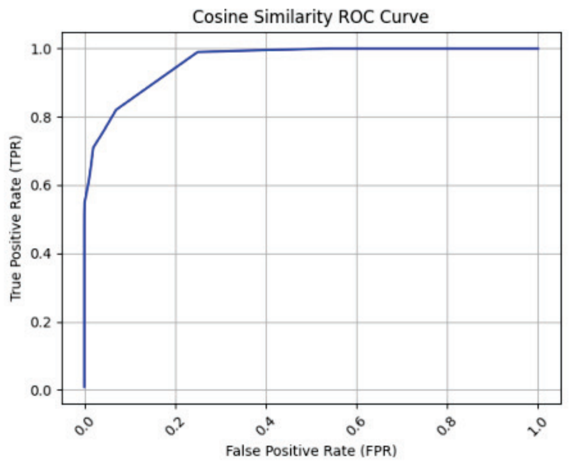

FIGURE 6. Receiver operating characteristic (ROC) curve for cosine similarity threshold values.

ROC FOR COSINE SIMILARITY THRESHOLD USING ELASTICSEARCH

Figure 6 shows the receiver operating characteristic curve for cosine similarity threshold values. From our testing, the optimum threshold values for the cosine similarity function were between 0.9 and 0.99. This range contains the optimal combination between missed detection and false alarm rates. When threshold values were higher than 0.99 manipulated data was not able to be identified effectively. When values were lower than 0.9 false alarm rates increased drastically.

VI. ANALYSIS

In this section, we analyze the performance and feasibility of the various algorithms for detecting maliciously copied and manipulated spectrum occupancy data. We examine the timing, accuracy, and potential challenges associated with the algorithms, offering insights into their practical implementation in real-world systems.

TIMING

The timing analysis plays a crucial role in evaluating the practical applicability of the algorithms in real-world scenarios. Understanding how the system performs under various conditions, such as multiple nodes submitting data simultaneously or during individual search operations, is essential for ensuring reliable and efficient detection of malicious activity. The following subsections address these aspects, providing a detailed examination of the system's performance.

MULTIPLE NODES AT ONCE

Figure 4 is a timing analysis for filtering through the previously submitted data looking for repeated data. Each series on the graph represents the number of nodes submitting data to the fusion center for the current period. Each node submits its measure report with a length of 16 numbers every period. The submissions of spectrum occupancy data may be sequential or parallel. The timing shows how long it takes for all of the current period's data to be checked once all the data has been collected for that period. Even if the SAS operator chooses not to use this method on all incoming data, the timing shows that it is possible to filter through all of the data rather quickly and should fit within most specified periods T. Spikes in the graph showing longer processing times are most likely due to background processes on the operating system competing for the CPU. The linear trends show that when the data volume is increasing the computation time needed increases relatively proportionally.

Even when storing 21,600 periods worth of data at one-second intervals with 128 nodes, the computation time did not exceed the one-second interval using parallel processing. If the period was four seconds or 28 seconds, the data could be recorded for one day or one week respectively without processing time exceeding the one-second submission intervals. We expect that most data reports will not be more frequent than every few seconds. If the measure reports have a longer time interval, there will be even less data to go through and computation times will decrease relative to the decrease in data.

Algorithm 3 is computationally intensive and may not be able to keep up with a system that has multiple nodes submitting data to the aggregator at the same time without Elasticsearch so timing was only recorded for individual searches.

INDIVIDUAL SEARCH TIMES

For both Algorithm 1 and Algorithm 3 the search times were dramatically faster using Elasticsearch than the for loop especially for larger array counts. For example, at 10,000,000 arrays the search times for the cosine similarity threshold value equal to 0.95 were 0.17052s and 123.18421s respectively.

The timing scenarios in Table 2 using Elasticsearch show that systems should be able to maintain pace running these algorithms on continuously incoming spectrum occupancy data under a reasonable number of nodes and frequency of spectrum occupancy submissions. From this, we recommend using Elasticsearch for the implementation of either Algorithm 1 or Algorithm 3.

The cosine similarity threshold value had no measurable impact on the execution time for either the iterative loop or the Elasticsearch-based approach. Any potential effect on timing would be primarily associated with the data transmission overhead, as lowering the threshold results in a larger number of retrieved results. Consequently, this would increase the volume of data transmitted, but the computational cost of the cosine similarity calculation remained invariant with respect to the threshold itself. Thus, any observed differences in performance would likely be attributed to the increased data handling rather than the similarity calculation process.

Iteratively checking for manipulated data with Algorithm 2 was not computationally feasible with the current resources however, it would have been the slowest method. Furthermore, increasing the granularity of the manipulation would have further extended the computation time.

Real-time responsiveness is a critical requirement in SAS environments, which must detect and respond to PU activity within 300 seconds to avoid interference. At an average of 0.5 seconds per query, processing 500,000 queries would require approximately 250,000 seconds, which exceeds the allowable window by several orders of magnitude. This confirms that unoptimized or exhaustive methods such as Algorithm 2 are not suitable for real-time use in their current form. Optimizations such as parallel execution, approximate filtering, early-exit heuristics, or leveraging distributed systems like Elasticsearch are essential for reducing latency while preserving detection accuracy.

MEMORY CONSUMPTION

The algorithms were evaluated using datasets ranging in size from 1,000 to 10,000,000 entries. To optimize memory usage, floats were used in place of doubles, as they consume less memory while maintaining the necessary precision for the calculations. Each dataset consisted of 16 numerical values per entry, accompanied by associated metadata, such as *CBSD_ID*, resulting in an average storage size of approximately 100 bytes per entry. Consequently, the largest total dataset size tested was approximately 1.00 GB, which was sufficient to store approximately 10,000,000 entries.

Consider a scenario where the SAS maintains an existing dataset containing 10 million spectrum occupancy records, with each record requiring 128 bytes of storage. The total initial storage requirement is calculated as:

$$\begin{aligned}\text{Initial Storage} &= 10,000,000 \times 128 \text{ bytes}\\ &= 1.28 \times 10^9 \text{ bytes} = 1.28 \text{ GB}\end{aligned}$$

Assuming an annual data growth rate of 20%, the storage requirement after three years can be estimated using compound growth:

$$\text{Storage after 3 years} = \text{Initial Storage} \times (1.20)^3$$

$$\text{Storage after 3 years} = 1.28 \text{ GB} \times (1.728) \approx 2.21 \text{ GB}$$

Thus, after three years, the total storage requirement grows to approximately 2.21 GB.

While this storage size may seem manageable in centralized architectures, sustained growth and an increasing number of records can lead to performance degradation in terms of latency, I/O throughput, and system responsiveness. As the dataset expands, centralized storage systems may become a bottleneck, especially when handling real-time queries or large-scale SSDF detection tasks.

In contrast, distributed storage solutions such as cloud-based platforms or edge computing architectures offer more scalable alternatives. These systems allow parallel data access, localized processing, and better fault tolerance, reducing the load on any single server. Future research should explore the trade-offs between centralized and distributed storage systems, considering the impact on latency, computational overhead, and bandwidth utilization in a dynamic spectrum access environment.

Given the need to balance memory efficiency with computational speed, the methodology could store the required information in a variety of formats depending on the specific requirements of the application. For example, floats and integers are ideal for numerical data storage and offer optimized memory usage and processing speed, particularly for operations such as cosine similarity and data searching. Additionally, binary formats could be used in cases where storage and retrieval speed are paramount, especially for larger datasets or distributed systems.

On the other hand, arrays or lists are excellent for storing vectors or datasets requiring frequent mathematical computations, as they provide direct access for efficient calculation. While strings and booleans could be used in specific scenarios, they are less suitable for tasks like cosine similarity or searching, as they introduce overhead and require additional preprocessing.

By leveraging these varied formats, the methodology offers flexibility in storing and processing data while maintaining memory efficiency and computational effectiveness.

ACCURACY

All maliciously copied data could be identified with a 0.000000% missed detection rate and a 0.000000% false alarm rate with Algorithm 1. Algorithm 1 with Elasticsearch is ideal for detecting malicious spectrum occupancy data copies. This result highlights the algorithm's ability to precisely detect exact copies without generating any erroneous matches, making it highly reliable when the goal is to identify strictly identical data points.

Using Algorithm 2 could theoretically find all exact matches and manipulations from Equation (1) and Equation (2), but it is too computationally heavy to run with current processing limitations.

Algorithm 3 with a cosine similarity threshold between 0.9 and 0.99 was able to detect most exact matches and most manipulations. Manipulation error was mostly due to rounding when data was multiplied by fractional values. Rounding and floating-point errors in cosine similarity calculations can lead to inaccuracies in SSDF detection outcomes, particularly when distinguishing between closely related spectrum occupancy values. These numerical inaccuracies may result in false positives or false negatives, especially when similarity scores fluctuate near fixed detection thresholds. Over time, accumulated errors can affect the stability of trust metrics and lead to inconsistent behavior across heterogeneous CBSD platforms. To mitigate these issues, the use of high-precision arithmetic, consistent normalization procedures, and threshold margin buffers is recommended to enhance robustness and reproducibility. We suggest using a cosine similarity threshold value that has a 0.000000% false alarm rate which sacrifices manipulated data points that are missed for a guarantee that the flagged data is malicious. From Table 2 the cosine similarity threshold value equal to 0.97 had the lowest missed detection rate at 55.93% with a 0.000000% false alarm rate. This will detect almost all exact copies and a large portion of manipulations that are created maliciously with Equation (1) and Equation (2) without raising false alarms.

Algorithm 3's accuracy in identifying manipulated data had no notable difference in missed detection and false alarm rates with Elasticsearch compared to the standard Python for loops script. Any differences are most likely attributed to random number generation during testing.

The manipulated data created with Equations (1) and (2) for testing utilized completely random floating-point numbers ranging from –99.999999 to 99.999999 for α and from –9.999999 to 9.999999 for δ. This occasionally makes the values of the data impossibly high or low compared to real spectrum occupancy data. Most of the missed detections were from overflow values resulting in truly uncorrelated array values. If α and δ values were selected based on the resulting array values and not randomly generated, the accuracy of this method would increase.

The previous discussion, tables, and related graphs show that exact match identification and cosine similarity algorithms are valid methods for identifying repeated data if resources are available. Additionally, Algorithm 3 can accurately identify manipulated spectrum data using the correct cosine similarity threshold value.

ELASTICSEARCH

From our testing, to find exact copies it is suggested to use Elasticsearch with Algorithm 1. To identify manipulated data, Elasticsearch with Algorithm 3 is suggested. The drawback to Elasticsearch is that the setup for methods other than the cosine similarity function is more complex and requires a deeper understanding of Elasticsearch. If Elasticsearch is not available, Algorithm 1 using a for loop would be the next best option in terms of accuracy and speed for identifying repeated data only as Algorithm 3 would not be feasible to implement due to timing constraints imposed by a CRN or SAS.

POTENTIAL CHALLENGES AND VULNERABILITIES

The reliance on crowdsourced data in SASs introduces significant vulnerabilities, particularly the risk of receiving and processing falsified spectrum occupancy information. As the system scales and the volume of crowdsourced data increases, the demand for processing power and storage will grow, potentially leading to performance bottlenecks, especially when implementing computationally intensive measures such as the cosine similarity function to detect manipulated data. This increase in data volume may also exacerbate the risks of false positives and false negatives where manipulated data goes undetected. Both outcomes could undermine the accuracy and reliability of a CRN or SAS.

To understand the practical implications of detection performance, consider an SSDF detection system that processes approximately 100,000 spectrum occupancy data points per day, with 5,000 of those being manipulated data. With a theoretical false positive rate of 2% and a missed detection rate of 5%, the system would incorrectly flag approximately 2,000 legitimate entries as manipulated each day, and fail to detect around 250 actual manipulated entries. The elevated risk of missed detections underscores the necessity of employing this method as a complementary mechanism within a broader SSDF detection framework.

These errors have direct consequences on spectrum availability and network efficiency. False positives may lead to underutilization of available spectrum, as legitimate users are erroneously excluded or their trust scores are unjustly reduced. On the other hand, false negatives allow malicious data to persist, potentially enabling attackers to interfere with spectrum access, degrade sensing accuracy, or manipulate resource allocation decisions. Therefore, minimizing these errors is critical to maintaining the integrity and reliability of dynamic spectrum access systems.

Another significant concern is the latency in response; while the SAS is required to notify SUs to cease transmission within 300 seconds of detecting a PU, this window may be insufficient to prevent harmful interference in time-sensitive situations if the SAS has used its resources for computation.

Algorithm 2 is a brute force method and is extremely computationally heavy, making it impossible to feasibly implement in a system. The method is so computationally heavy that all systems aborted the program or crashed when testing.

The need for a centralized system also presents a potential single point of failure. Should the centralized system be compromised or experience downtime, the entire spectrum management process could be disrupted, leading to potential interference and service degradation.

ETHICAL, SECURITY, AND PRIVACY CONSIDERATIONS

The implementation of SSDF detection algorithms raises important ethical, security, and privacy considerations. From a privacy standpoint, crowdsourced spectrum data may contain sensitive information about users or locations, necessitating anonymization and encryption to prevent data leakage. Techniques such as differential privacy can help protect individual contributors while still allowing for effective manipulation detection. Another concern is the potential for false positives, where legitimate users may be incorrectly flagged for SSDF attacks, leading to unintended denial of service. To mitigate this, detection algorithms should be tuned to minimize false positives while incorporating confidence scores to assess the reliability of flagged entries. Additionally, security against adaptive adversaries must be considered, as attackers may attempt to evade detection by mimicking legitimate sensing patterns. To counteract this, adversarial machine learning techniques could be explored to improve model resilience against evolving threats [40], [41], [42]. Finally, ethical considerations must be taken into account, ensuring that the algorithm adheres to regulatory guidelines for fair spectrum access and does not unintentionally disadvantage any group of users. Transparent methodologies and standardized evaluation metrics should be used to prevent misuse in competitive spectrum-sharing environments. The proposed method enhances security by proactively addressing these concerns while ensuring ethical and responsible implementation.

VII. FUTURE WORK

To minimize the risk of regressions from future changes, we propose a combination of automated testing, version control, modular algorithm design, and dataset validation. Automated testing should include unit and integration tests to verify the functionality of detection algorithms, particularly against adversarial input scenarios and edge cases. Benchmarking tools should be employed to monitor execution time, memory usage, and detection accuracy with each update. Maintaining a structured version control repository with detailed commit histories will help track modifications while enabling continuous integration (CI) pipelines to validate performance against predefined baselines. A modular architecture should be adopted to further enhance adaptability by separating components like data acquisition, preprocessing, and the algorithm into independently testable units. Implementing a design for detection algorithms, including cosine similarity, allows for easier experimentation and integration of new methods. Configurable thresholds and runtime-loaded parameters will support deployment across diverse environments without code changes. Using semantic versioning and maintaining detailed changelogs will support reproducibility and traceability in ongoing SSDF detection research. Additionally, designing the algorithms modularly will allow for targeted updates to individual components without affecting the overall framework. A curated dataset of real and synthetic SSDF attack patterns should be maintained to validate algorithm effectiveness after modifications. Finally, allowing configurable similarity thresholds and detection parameters will ensure adaptability to evolving operational conditions. By implementing these measures, the system can evolve while maintaining stability and effectiveness in detecting SSDF attacks.

Algorithm 2 has an extreme computational load as it requires millions of for-loop iterations. If computational capabilities increase, this method could be explored further with the full or partial range of potential α and δ values.

Rounding and floating-point errors can impact the accuracy of the cosine similarity approach. Therefore, investigating the cosine similarity threshold with varying data precision, data types, and operating systems is recommended to increase the accuracy of this method.

Additionally, future testing could be done with more selective values of α and δ from Equation (1) and Equation (2). These values were chosen randomly, and the resulting array was not assessed for realistic ranges of spectrum occupancy data. The percentage of missed detections and false alarms from the cosine similarity function can be refined with a more selective group of manipulation values.

Future systems could employ Algorithm 3 with a variable threshold depending on the system and environmental factors such as the number of SUs, frequency of data submission, and the physical environment. SAS operators may want to see all data that has a similar threshold value despite having higher false positive rates and compare CBSD IDs and store them for future reference.

Further research in this area should also aim to identify manipulated data with a specified confidence interval.

These research efforts would greatly benefit from the collection of real-world data from an operational CRN or SAS that has experienced attacks.

VIII. CONCLUSION

The ability to identify manipulated data without imposing undue strain on a centrally aggregated system helps reduce the number of ways to create an SSDF attack and increase the accuracy of determining the radio transmission activity of a PU.

Our findings demonstrate the feasibility of discerning several forms of manipulated spectrum occupancy data before running SSDF attack identification algorithms using our proposed model and future areas of research to optimize the accuracy and implementation of these algorithms in a CRN or SAS. Several machine learning methods were evaluated but failed to accurately identify such manipulations well. Notably, such manipulations do not necessarily constitute anomalies or statistical outliers. Additionally, our timing scenarios with Elasticsearch show that systems should be able to maintain pace running these algorithms on continuously incoming spectrum occupancy data under a reasonable number of nodes and frequency of submitting received power measure reports. None of the machine learning approaches tested were able to meet the time constraints required for CRN or SAS operations. The approach explored here is also applicable to crowdsourcing of other operational-context data (for example, rainfall rate, which has a greater impact on RF propagation at frequencies above those in the CBRS band) in addition to spectrum-occupancy data, and could be applied within multi-context-aware spectrum-sharing frameworks such as those proposed for the 12.2-12.7 GHz band (e.g., [1], [2]).

When resources permit, to identify exact matches we recommend using Algorithm 1. To find manipulated spectrum occupancy data and copies, we recommend utilizing a cosine similarity function with Elasticsearch while using the lowest possible threshold that eliminates false alarms. These approaches can enable the system to accurately identify almost all instances of copied data and a large percentage of manipulated data without generating false positives. Consequently, it provides a reliable guarantee to the CRN or SAS operator that any incoming flagged data is malicious.

ACKNOWLEDGMENT

The views in this document are those of the authors and do not reflect the official policy or position of the sponsors or the U.S. Government.

REFERENCES

[1] Z. Hassan et al., "Spectrum sharing of the 12 GHz band with two-way terrestrial 5G mobile services: Motivations, challenges, and research road map," *IEEE Commun. Mag.*, vol. 61, no. 7, pp. 53–59, Jul. 2023.

[2] T.-S. R. Niloy et al., "ASCENT: A context-aware spectrum coexistence design and implementation toolset for policymakers in satellite bands," in *Proc. IEEE Int. Symp. Dyn. Spectr. Access Netw. (DySPAN)*, 2024, pp. 240–248.

[3] (Federal Commun. Comm., Washington, DC, USA). *Part 96 Citizens Broadband Radio Service*. 2021. [Online]. Available: https://www.ecfr.gov/cgi-bin/text-idx?node=pt47.5.96

[4] A. A. Sharifi and M. J. M. Niya, "Defense against SSDF attack in cognitive radio networks: Attack-aware collaborative spectrum sensing approach," *IEEE Commun. Lett.*, vol. 20, no. 1, pp. 93–96, Jan. 2016.

[5] I. Gupta and O. P. Sahu, "An overview of primary user emulation attack in cognitive radio networks," in *Proc. 2nd Int. Conf. Comput. Methodol. Commun. (ICCMC)*, 2018, pp. 27–31.

[6] R. Chen and J.-M. Park, "Ensuring trustworthy spectrum sensing in cognitive radio networks," in *Proc. 1st IEEE Workshop Netw. Technol. Softw. Defined Radio Netw.*, 2006, pp. 110–119.

[7] Z. Song, J. Yang, H. Zhang, and Y. Gao, "Approaching sub-Nyquist boundary: Optimized compressed spectrum sensing based on multicoset sampler for multiband signal," *IEEE Trans. Signal Process.*, vol. 70, pp. 4225–4238, 2022.

[8] H. Zhang, Z. Song, J. Yang, and Y. Gao, "Adversarial autoencoder empowered joint anomaly detection and signal reconstruction from sub-Nyquist samples," *IEEE Trans. Cogn. Commun. Netw.*, vol. 9, no. 3, pp. 618–628, Jun. 2023.

[9] Z. Song, Y. She, J. Yang, J. Peng, Y. Gao, and R. Tafazolli, "Nonuniform sampling pattern design for compressed spectrum sensing in mobile cognitive radio networks," *IEEE Trans. Mobile Comput.*, vol. 23, no. 9, pp. 8680–8693, Sep. 2024.

[10] Y. Fu and Z. He, "Massive SSDF attackers identification in cognitive radio networks by using consistent property," *IEEE Trans. Veh. Technol.*, vol. 72, no. 8, pp. 11058–11062, Aug. 2023.

[11] Y. Liu, A. H. Shaikh, and W. M. Khoso, "Impact of spectrum sensing data falsification attack and imperfect reporting channel on cooperative spectrum sensing," in *Proc. 9th Int. Conf. Comput. Commun. (ICCC)*, 2023, pp. 470–474.

[12] I. Ngomane, M. Velempini, and S. V. Dlamini, "The detection of the spectrum sensing data falsification attack in cognitive radio ad hoc networks," in *Proc. Conf. Inf. Commun. Technol. Soc. (ICTAS)*, 2018, pp. 1–5.

[13] P. S. Chatterjee and M. Roy, "A regression based spectrum-sensing data-falsification attack detection technique in CWSN," in *Proc. Int. Conf. Inf. Technol. (ICIT)*, 2015, pp. 48–53.

[14] N. Parhizgar, A. Jamshidi, and P. Setoodeh, "Defense against spectrum sensing data falsification attack in cognitive radio networks using machine learning," in *Proc. 30th Int. Conf. Elect. Eng. (ICEE)*, 2022, pp. 974–979.

[15] P. M. S. Sánchez et al., "Stealth spectrum sensing data falsification attacks affecting IoT spectrum monitors on the battlefield," in *Proc. IEEE Mil. Commun. Conf. (MILCOM)*, 2023, pp. 673–678.

[16] G. Nie, G. Ding, L. Zhang, and Q. Wu, "Byzantine defense in collaborative spectrum sensing via Bayesian learning," *IEEE Access*, vol. 5, pp. 20089–20098, 2017.

[17] Y. Shi, T. Erpek, Y. E. Sagduyu, and J. H. Li, "Spectrum data poisoning with adversarial deep learning," in *Proc. IEEE Mil. Commun. Conf. (MILCOM)*, 2018, pp. 407–412.

[18] A. Foresi and R. Samavi, "User authentication using keystroke dynamics via crowdsourcing," in *Proc. 17th Int. Conf. Privacy, Security Trust (PST)*, 2019, pp. 1–3.

[19] D. Raghu, C. R. Jacob, and Y. Bhavani, "Neural network based authentication and verification for Web based key stroke dynamics," *Dimension*, vol. 2, p. 1, 2011.

[20] T. Eude and C. Chang, "One-class SVM for biometric authentication by keystroke dynamics for remote evaluation," *Comput. Intell.*, vol. 34, no. 1, pp. 145–160, 2018.

[21] E. C. Ogemuno, "Behavioural based biometrics using keystroke dynamics for user authentication," Ph.D. dissertation, Univ. Windsor, Windsor, ON, Canada, 2018.

[22] A. Alsultan, K. Warwick, and H. Wei, "Improving the performance of free-text keystroke dynamics authentication by fusion," *Appl. Soft Comput.*, vol. 70, pp. 1024–1033, Sep. 2018.

[23] P. Kang, "The effects of different alphabets on free text keystroke authentication: A case study on the Korean–English users," *J. Syst. Softw.*, vol. 102, pp. 1–11, Apr. 2015.

[24] D. R. Chandranegara and F. D. S. Sumadi, "Keystroke dynamic authentication using combined MHR (mean of horner's rules) and standard deviation," *Kinetik, Game Technol., Inf. Syst., Comput. Netw., Comput., Electron., Control*, vol. 4, no. 1, pp. 13–18, 2019.

[25] N. A. Hegde, "A study of person identification using keystroke dynamics and statistical analysis," *Int. J. Eng. Manag. Res.*, vol. 8, no. 3, pp. 18–23, 2018.

[26] J. C. Stewart, J. V. Monaco, S.-H. Cha, and C. C. Tappert, "An investigation of keystroke and stylometry traits for authenticating online test takers," in *Proc. Int. Joint Conf. Biometrics (IJCB)*, 2011, pp. 1–7.

[27] C.-H. Jiang, S. Shieh, and J.-C. Liu, "Keystroke statistical learning model for Web authentication," in *Proc. 2nd ACM Symp. Inf., Comput. Commun. Secur.*, 2007, pp. 359–361.

[28] R. Giot, M. El-Abed, and C. Rosenberger, "Web-based benchmark for keystroke dynamics biometric systems: A statistical analysis," in *Proc. Eighth Int. Conf. Intell. Inf. Hiding Multimedia Signal Process.*, 2012, pp. 11–15.

[29] P. S. Teh, A. B. J. Teoh, and S. Yue, "A survey of keystroke dynamics biometrics," *Sci. World J.*, 2013, to be published.

[30] X. Wang, F. Guo, and J.-F. Ma, "User authentication via keystroke dynamics based on difference subspace and slope correlation degree," *Digit. Signal Process.*, vol. 22, no. 5, pp. 707–712, 2012.

[31] M. Antal, L. Z. Szabó, and I. László, "Keystroke dynamics on android platform," *Procedia Technol.*, vol. 19, pp. 820–826, Jan. 2015.

[32] (Wireless Innovat. Forum, Reston, VA, USA). *CBRS Protocols Technical Report*. Aug. 2017. [Online]. Available: https://winnf.memberclicks.net/assets/work_products/Recommendations/WINNF-TR-0205-V1.0.0

[33] (Wireless Innovat. Forum, Reston, VA, USA). *Post Initial Certification Revisions to CBRS Baseline Operational and Functional Requirements Specification*. May 2024. [Online]. Available: https://winnf.memberclicks.net/assets/CBRS/WINNF-TS-1020.pdf

[34] J. Han, M. Kamber, and J. Pei, "2-getting to know your data," in *Data Mining* (The Morgan Kaufmann Series in Data Management Systems), 3rd ed., J. Han, M. Kamber, and J. Pei, Eds., Boston, MA, USA: Morgan Kaufmann, 2012, pp. 39–82. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780123814791000022

[35] C. Gormley and Z. Tong, *Elasticsearch: The Definitive Guide: A Distributed Real-Time Search and Analytics Engine*. Sebastopol, CA, USA: O'Reilly Media, Inc., 2015.

[36] A. Yang, S. Zhu, X. Li, J. Yu, M. Wei, and C. Li, "The research of policy big data retrieval and analysis based on elastic search," in *Proc. Int. Conf. Artif. Intell. Big Data (ICAIBD)*, 2018, pp. 43–46.

[37] R. Goścień, M. Klinkowski, and K. Walkowiak, "A tabu search algorithm for routing and spectrum allocation in elastic optical networks," in *Proc. 16th Int. Conf. Transp. Opt. Netw. (ICTON)*, 2014, pp. 1–4.

[38] (Apache Softw. Found., Forest Hill, MD, USA). *Apache Solr*. Accessed: Jun. 5, 2024. [Online]. Available: http://lucene.apache.org/solr/

[39] (Comput. Sci. Virginia Tech., Blacksburg, VA, USA). *[Remote Login Cluster:] About*. Nov. 2023. [Online]. Available: https://wordpress.cs.vt.edu/rlogin/about/

[40] I. Alsmadi et al., "Adversarial machine learning in text processing: A literature survey," *IEEE Access*, vol. 10, pp. 17043–17077, 2022.

[41] N. Janapriya, K. Anuradha, and V. Srilakshmi, "Adversarial deep learning models with multiple adversaries," in *Proc. 3rd Int. Conf. Invent. Res. Comput. Appl. (ICIRCA)*, 2021, pp. 522–525.

[42] C. Mehta, P. Harniya, and S. Kamat, "Comprehending and detecting vulnerabilities using adversarial machine learning attacks," in *Proc. 2nd Int. Conf. Artif. Intell. Signal Process. (AISP)*, 2022, pp. 1–5.

JOSEPH TOLLEY received the B.S. and M.S. degrees in computer engineering from Virginia Tech, where he is currently pursuing the Ph.D. degree. His research interests are in computer systems, spectrum access, and cognitive radio.

CARL B. DIETRICH (Senior Member, IEEE) received the B.S. degree in electrical engineering from Texas A&M University, and the M.S. and Ph.D. degrees in electrical engineering from Virginia Tech. His research interests are in multiple aspects of wireless communication systems, including cognitive radio testing, radio wave propagation, and multiantenna systems. He is a member of IEEE HKN and ASEE, and a professional engineer in Virginia.