Journal of Autonomous Intelligence (2024) Volume 7 Issue 5
doi: 10.32629/jai.v7i5.1484

ORIGINAL RESEARCH ARTICLE

A virtual machine-based e-malpractice mitigation strategy in e-assessment and e-learning using system resources and machine learning techniques

Osamuyimen Odion Amadasun[1,*], Bukola Onasoga[2], Ahmed Aliyu[3], Kingsley Eghonghon Ukhurebor[4,*], Adeyinka Oluwabusayo Abiodun[1,5], Moses Ashawa[6]

[1] *Africa Centre of Excellence on Technology Enhanced Learning (ACETEL), National Open University of Nigeria (NOUN), Abuja 900001, Nigeria*

[2] *Department of Computer Science, Federal University of Agriculture Abeokuta, Abeokuta 111101, Nigeria*

[3] *Department of Computer Science, Bauchi State University, Bauchi 751105, Nigeria*

[4] *Department of Physics, Edo State University Uzairue, Auchi 312001, Edo State, Nigeria*

[5] *Department of Computer Science, National Open University of Nigeria (NOUN), Abuja 900001, Nigeria*

[6] *Department of Computer Science, Glasgow Caledonian University, G4 0BA Glasgow, UK*

*** Corresponding authors:** Osamuyimen Odion Amadasun, oamadasun@noun.edu.ng; Kingsley Eghonghon Ukhurebor, ukeghonghon@gmail.com, ukhurebor.kingsley@edouniversity.edu.ng

ABSTRACT

Since the introduction of online learning and the widespread use of AI-proctored examination systems, protecting the integrity of assessments has faced new difficulties. The development of reliable methods for detecting electronic cheating, notably the use of virtual machines (VMs) during examinations, has become essential with the rise of advanced cheating methods. Hence, in this research, a thorough methodology for identifying virtual machine usage in an AI-proctored test system is presented. In order to uncover suspicious activities connected with the use of VMs, the study offers a unique model that makes use of system resource parameters and cutting-edge machine learning techniques. Extensive experiments using simulated datasets are used to show the efficiency of the suggested model. The findings indicate accurate electronic cheating detection that is likely to improve academic evaluation integrity.

Keywords: academic evaluation; e-learning; examination systems; machine learning techniques; virtual machines

ARTICLE INFO

Received: 13 December 2023
Accepted: 3 January 2024
Available online: 10 April 2024

1. Introduction

There are now more chances for academic evaluation thanks to the growing acceptance of online learning and remote testing platforms[1–3], but there are also new difficulties in maintaining the validity of examinations[4]. The creation of AI-proctored test systems was prompted by the infeasibility of using conventional non-person proctoring techniques in distant settings[4–6]. These methods have made it easier to remotely monitor and certify examinations, but they are also open to electronic fraud, such as the use of virtual machines (VMs)[4,6].

Users can utilize VMs to run various operating systems and applications inside a host system in a sandboxed environment[2]. VMs are a desirable option for students looking to gain unfair advantages during examinations due to their flexibility and seclusion[7]. Students can get around the monitoring features of AI-provisioned systems and engage in dishonest behaviour by executing unauthorized software or accessing forbidden resources inside a virtual computer[8].

Hence, this research intends to create a reliable detection model that can spot instances of virtual machine utilization in the context of AI-proctored examinations in order to address this problem. According to Lin et al.[9] and Bejawada[10], we can find trends and abnormalities that are indicative of VMs by examining different system resource parameters, such as CPU utilization, memory utilization, network activity, disk usage, and power consumption. The methodology of the suggested model, including the selection and collection of pertinent system resource parameters, is thoroughly examined in this study. With the use of a synthetic dataset resembling an AI-proctored examination system, we describe the model's training and testing processes[11]. The performance indicators and assessment metrics used to gauge how well the model performs in accurately recognizing virtual machine utilization were also considered[9].

The implications and rationale for consideration of this study are: the need to preserve academic integrity; the need for robust cheating detection; maintaining fairness in assessments (cheating not only undermines the integrity of assessments but also puts honest test-takers at a disadvantage); and the need to advance AI proctoring technology[12]. The purpose of this research work is to create a reliable detection model for detecting electronic cheating in an AI-proctored test system that uses VMs. In the course of this study, in order to pursue and fulfil the goal of the study, an attempt will be made to identify important system resource metrics (during examination sessions that indicate virtual machine activity), create a model for VM detection, gather and prepare datasets (gather representative datasets from a model of an AI-powered proctored examination system), and develop and assess the model (create training and test sets from the pre-processed datasets).

The following research questions will direct the investigation and give a thorough understanding of the utility, constraints, and consequences of the proposed model.

1) What are the essential system parameters that can be used to identify virtual machine usage during a test session?
2) How can machine learning methods be efficiently used in an AI-proctored test system to identify the use of VMs?
3) How can a detection model be created for identifying electronic cheating as compared to current techniques?
4) How accurate is the model and how can false positive and false negative rates be reduced?

Consequently, this research paper hopes to make a contribution to the field of AI-proctored examination systems by offering a thorough and efficient method for identifying electronic cheating while using VMs. By achieving these goals and objectives, this research paper hopes to make a contribution to the field of AI-proctored examination systems by offering a thorough and efficient method for identifying electronic cheating while using VMs.

2. Literature review

Several studies like Noorbehbahani et al.[13]; Alyoussef[14]; Asanga et al.[15]; Atoum et al.[16]; Basar et al.[17]; Beust et al.[18]; Butler-Henderson and Crawford[19]; Dendir and Maxwell[20]; Draaijer et al.[21]; Friatma and Anhar[22]; Furby[23]; Golden and Kohlbeck[24]; Hou et al.[25]; Kamalov et al.[26]; Li et al.[27]; Pandey et al.[28] have recently focused on the detection of electronic cheating with outstanding findings, but there is a dearth of research on the usage of VMs for cheating in AI-proctored examination systems. This review of the literature gives a general overview of the field's body of work and emphasizes its major discoveries and methodological approaches.

2.1. Virtual machine detection

Several instructors have looked at the use of system resource criteria to determine which virtual machine is used in testing and presented a method for recognizing patterns of virtual machine usage based on patterns of CPU and memory usage[29]. Very accurate virtual computers were identified by their method, depending on the number of resources used. A similar study was conducted by Wang et al.[30] on how to identify virtual computers by analysing network traffic patterns. By monitoring traffic and IP addresses, they were able to identify unusual network activity associated with virtual computing.

2.2. Machine learning techniques

The detection of electronic cheating has seen widespread application of machine learning methods. Support vector machine (SVM) technology was used by Wang et al.[30] to differentiate between typical system behaviour and virtual machine activity. Based on CPU and memory usage, their model had a high accuracy rate for identifying virtual machine usage. Decision tree methods were used by Zhang et al.[31] to examine system resource parameters and spot cheating. Their model successfully detected VMs with high precision and recall rates.

2.3. Feature engineering

For the purpose of identifying virtual machine usage, feature engineering is essential. In their feature engineering technique, it is suggested analysing the amount of CPU, memory, and disk consumption as well as the length of the inspection. Their research revealed that combining numerous parameters increased the virtual machine detection system’s precision. To capture dynamic behaviours related to virtual machine usage, Zhang et al.[31] proposed time-based features, such as abrupt resource spikes or abrupt shifts in utilization. According to their research, temporal features considerably improve detection accuracy.

2.4. Gaps identified in the study

The existing literature has primarily focused on technical aspects, overlooking the broader societal, ethical, and legal implications of AI proctoring.

Furthermore, there is a dearth of research addressing the rapidly evolving landscape of sophisticated cheating tactics and the countermeasures required to detect and prevent them effectively[13]. This literature gap necessitates a more in-depth exploration of emerging cheating techniques, such as the use of VMs for cheating in AI-proctoring examination systems. Understanding these novel cheating methods and their prevalence will enable the development of more robust and adaptive cheating detection mechanisms.

2.5. Limitations and challenges

There are some restrictions and difficulties in the current research on virtual machine detection in AI-provisioned test systems. The use of predetermined thresholds or rules, which may not generalize well across various locations or examination scenarios, is a typical drawback. Adaptive models that can recognize new tactics are required due to the dynamic nature of cheating approaches. False positives and false negatives are a potential problem that could affect the detection system’s dependability and efficiency. Implementing detection mechanisms that require access to system resource data or network traffic also raises privacy and ethical issues.

3. Methodology

To effectively detect electronic cheating facilitated by VMs in AI-proctored examination systems, a comprehensive methodology was established. This section outlines the key steps involved in the detection process represented in **Figure 1**, which is followed by explanations.

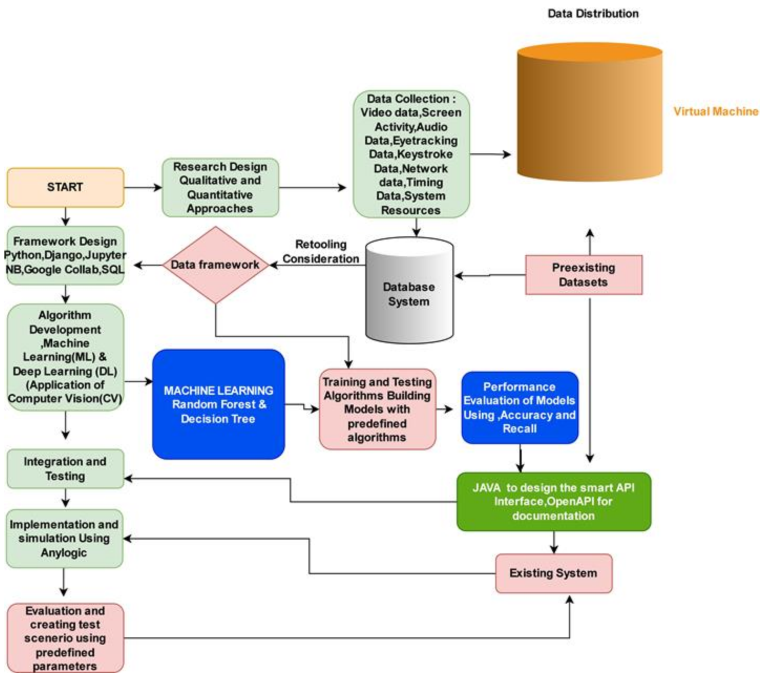
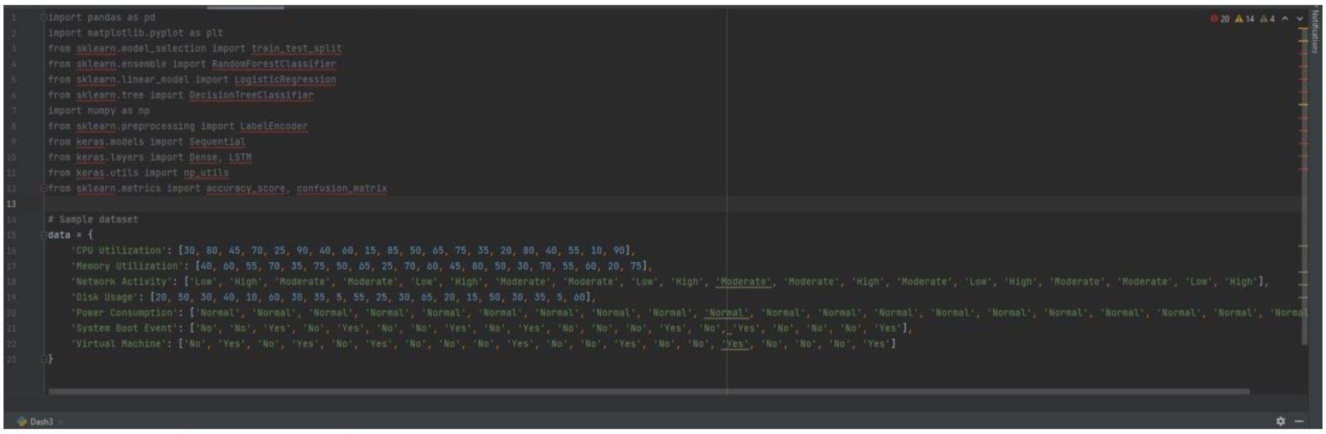

Data Distribution
Virtual Machine
START
Research Design Qualitative and Quantitative Approaches
Data Collection : Video data,Screen Activity,Audio Data,Eyetracking Data,Keystroke Data,Network data,Timing Data,System Resources
Framework Design Python,Django,Jupyter NB,Google Collab,SQL
Data framework
Retooling Consideration
Database System
Preexisting Datasets
Algorithm Development ,Machine Learning(ML) & Deep Learning (DL) (Application of Computer Vision(CV)
MACHINE LEARNING Random Forest & Decision Tree
Training and Testing Algorithms Building Models with predefined algorithms
Performance Evaluation of Models Using ,Accuracy and Recall
Integration and Testing
JAVA to design the smart API Interface,OpenAPI for documentation
Implementation and simulation Using Anylogic
Existing System
Evaluation and creating test scenerio using predefined parameters

Figure 1. Key steps involved in the detection process.



```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
import numpy as np
from sklearn.preprocessing import LabelEncoder
from keras.models import Sequential
from keras.layers import Dense, LSTM
from keras.utils import np_utils
from sklearn.metrics import accuracy_score, confusion_matrix

# Sample dataset
data = {
    'CPU Utilization': [30, 80, 45, 70, 25, 90, 40, 60, 15, 85, 50, 65, 75, 35, 20, 80, 40, 55, 10, 90],
    'Memory Utilization': [40, 60, 55, 70, 35, 75, 50, 65, 25, 70, 60, 45, 80, 50, 30, 70, 55, 60, 20, 75],
    'Network Activity': ['Low', 'High', 'Moderate', 'Moderate', 'Low', 'High', 'Moderate', 'Moderate', 'Low', 'High', 'Moderate', 'Moderate', 'High', 'Moderate', 'Low', 'High', 'Moderate', 'Moderate', 'Low', 'High'],
    'Disk Usage': [20, 50, 30, 40, 10, 60, 30, 35, 5, 55, 25, 30, 65, 20, 15, 50, 30, 35, 5, 60],
    'Power Consumption': ['Normal', 'Normal', 'Normal', 'Normal', 'Normal', 'Normal', 'Normal', 'Normal', 'Normal', 'Normal', 'Normal', 'Normal', 'Normal', 'Normal', 'Normal', 'Normal', 'Normal', 'Normal', 'Normal', 'Normal
    'System Boot Event': ['No', 'No', 'Yes', 'No', 'Yes', 'No', 'No', 'Yes', 'No', 'Yes', 'No', 'No', 'No', 'Yes', 'No', 'Yes', 'No', 'No', 'No', 'Yes'],
    'Virtual Machine': ['No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'No', 'No', 'Yes', 'No', 'No', 'Yes', 'No', 'No', 'Yes', 'No', 'No', 'No', 'Yes']
}
```

Figure 2. Data collection.

3.1. Data collection

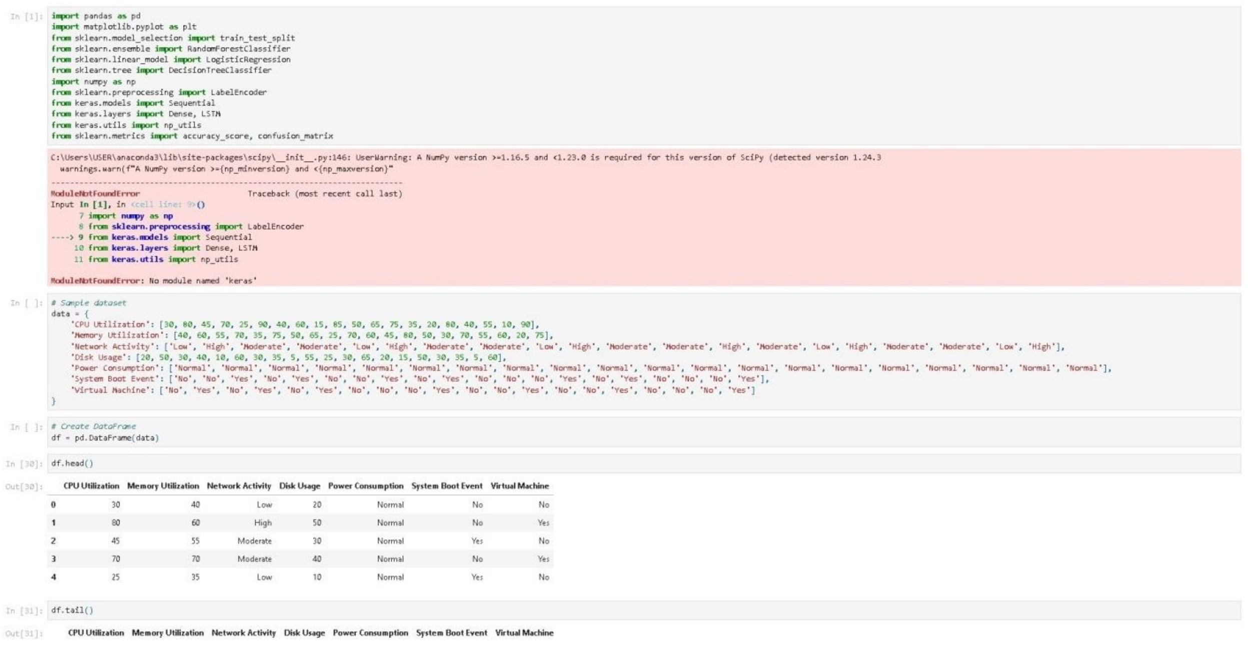
The initial step in the examination procedure is to gather pertinent data. This includes recording system logs, seeing screen and browser activity, and recording audio and video recordings of the test-taker. Additionally, data on virtual machine activity, including network traffic and changes in virtual machine state, was gathered (see **Figure 2**).

3.2. Data pre-processing

To identify pertinent features and remove noise, preprocessing of the collected data is required. Frame extraction from video recordings, speech-to-text conversion for audio recordings, and filtering out pointless browser or system log activity are all examples of preprocessing techniques (see **Figure 3**).

3.3. Feature extraction

The pre-processed data is then used to extract features that represent various facets of the test-takers behaviour. These characteristics may include changes in the state of VMs, network activity, resource utilization, mouse movements, keyboard dynamics, and facial expressions, as shown in **Figures 4–7**. Feature extraction methods can change based on the particular data types and analysis needs.



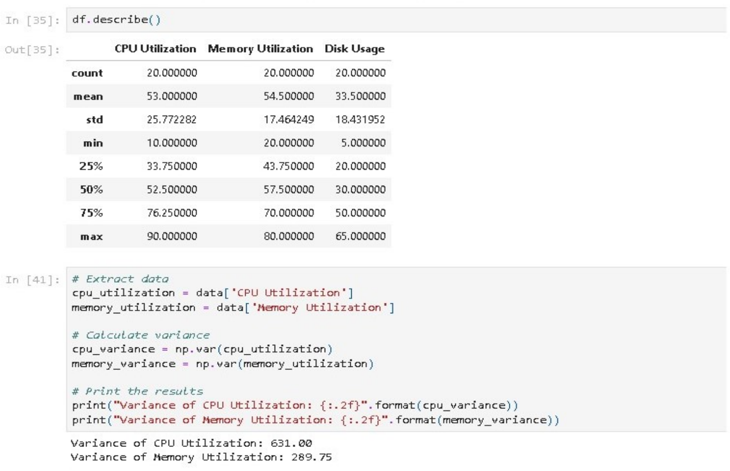
```
In [1]: import pandas as pd
        import matplotlib.pyplot as plt
        from sklearn.model_selection import train_test_split
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.linear_model import LogisticRegression
        from sklearn.tree import DecisionTreeClassifier
        import numpy as np
        from sklearn.preprocessing import LabelEncoder
        from keras.models import Sequential
        from keras.layers import Dense, LSTM
        from keras.utils import np_utils
        from sklearn.metrics import accuracy_score, confusion_matrix

C:\Users\USER\anaconda3\lib\site-packages\scipy\__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.24.3
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Input In [1], in <cell line: 9>()
      7 import numpy as np
      8 from sklearn.preprocessing import LabelEncoder
----> 9 from keras.models import Sequential
     10 from keras.layers import Dense, LSTM
     11 from keras.utils import np_utils

ModuleNotFoundError: No module named 'keras'

In [ ]: # Sample dataset
        data = {
            'CPU Utilization': [30, 80, 45, 70, 25, 90, 40, 60, 15, 85, 50, 65, 75, 35, 20, 80, 40, 55, 10, 90],
            'Memory Utilization': [40, 60, 55, 70, 35, 75, 50, 65, 25, 70, 60, 45, 80, 50, 30, 70, 55, 60, 20, 75],
            'Network Activity': ['Low', 'High', 'Moderate', 'Moderate', 'Low', 'High', 'Moderate', 'Moderate', 'Low', 'High', 'Moderate', 'Moderate', 'High', 'Moderate', 'Low', 'High', 'Moderate', 'Moderate', 'Low', 'High'],
            'Disk Usage': [20, 50, 30, 40, 10, 60, 30, 35, 5, 55, 25, 30, 65, 20, 15, 50, 30, 35, 5, 60],
            'Power Consumption': ['Normal', 'Normal', 'Normal', 'Normal', 'Normal', 'Normal', 'Normal', 'Normal', 'Normal', 'Normal', 'Normal', 'Normal', 'Normal', 'Normal', 'Normal', 'Normal', 'Normal', 'Normal', 'Normal', 'Normal'],
            'System Boot Event': ['No', 'No', 'Yes', 'No', 'Yes', 'No', 'No', 'Yes', 'No', 'Yes', 'No', 'No', 'No', 'Yes', 'No', 'Yes', 'No', 'No', 'No', 'Yes'],
            'Virtual Machine': ['No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'No', 'No', 'Yes', 'No', 'No', 'Yes', 'No', 'No', 'Yes', 'No', 'No', 'No', 'Yes']
        }

In [ ]: # Create DataFrame
        df = pd.DataFrame(data)

In [30]: df.head()

Out[30]:
```

	CPU Utilization	Memory Utilization	Network Activity	Disk Usage	Power Consumption	System Boot Event	Virtual Machine
0	30	40	Low	20	Normal	No	No
1	80	60	High	50	Normal	No	Yes
2	45	55	Moderate	30	Normal	Yes	No
3	70	70	Moderate	40	Normal	No	Yes
4	25	35	Low	10	Normal	Yes	No

```
In [31]: df.tail()

Out[31]:
```

CPU Utilization	Memory Utilization	Network Activity	Disk Usage	Power Consumption	System Boot Event	Virtual Machine

Figure 3. Data pre-processing.

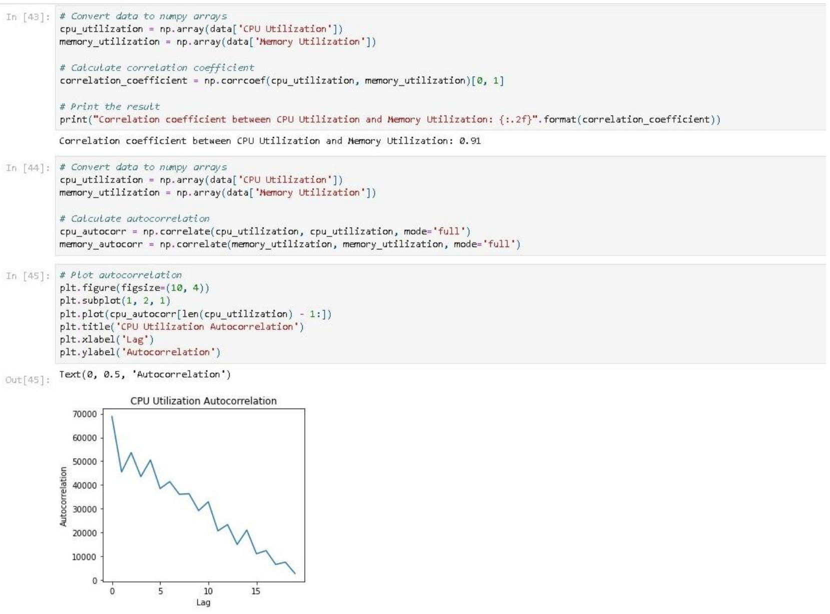


```
In [35]: df.describe()

Out[35]:
```

	CPU Utilization	Memory Utilization	Disk Usage
count	20.000000	20.000000	20.000000
mean	53.000000	54.500000	33.500000
std	25.772282	17.464249	18.431952
min	10.000000	20.000000	5.000000
25%	33.750000	43.750000	20.000000
50%	52.500000	57.500000	30.000000
75%	76.250000	70.000000	50.000000
max	90.000000	80.000000	65.000000

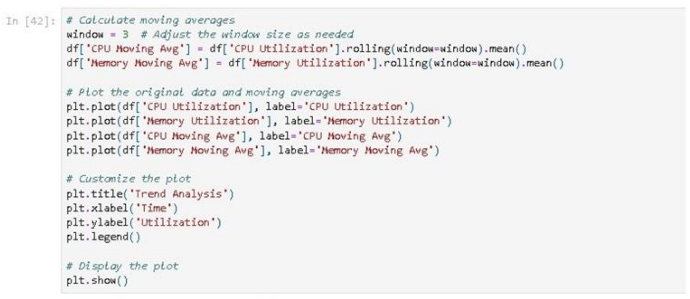
```
In [41]: # Extract data
         cpu_utilization = data['CPU Utilization']
         memory_utilization = data['Memory Utilization']

         # Calculate variance
         cpu_variance = np.var(cpu_utilization)
         memory_variance = np.var(memory_utilization)

         # Print the results
         print("Variance of CPU Utilization: {:.2f}".format(cpu_variance))
         print("Variance of Memory Utilization: {:.2f}".format(memory_variance))

         Variance of CPU Utilization: 631.00
         Variance of Memory Utilization: 289.75
```

Figure 4. Descriptive statistics of CPU and memory utilisation.



```
In [43]: # Convert data to numpy arrays
         cpu_utilization = np.array(data['CPU Utilization'])
         memory_utilization = np.array(data['Memory Utilization'])

         # Calculate correlation coefficient
         correlation_coefficient = np.corrcoef(cpu_utilization, memory_utilization)[0, 1]

         # Print the result
         print("Correlation coefficient between CPU Utilization and Memory Utilization: {:.2f}".format(correlation_coefficient))

         Correlation coefficient between CPU Utilization and Memory Utilization: 0.91

In [44]: # Convert data to numpy arrays
         cpu_utilization = np.array(data['CPU Utilization'])
         memory_utilization = np.array(data['Memory Utilization'])

         # Calculate autocorrelation
         cpu_autocorr = np.correlate(cpu_utilization, cpu_utilization, mode='full')
         memory_autocorr = np.correlate(memory_utilization, memory_utilization, mode='full')

In [45]: # Plot autocorrelation
         plt.figure(figsize=(10, 4))
         plt.subplot(1, 2, 1)
         plt.plot(cpu_autocorr[len(cpu_utilization) - 1:])
         plt.title('CPU Utilization Autocorrelation')
         plt.xlabel('Lag')
         plt.ylabel('Autocorrelation')

Out[45]: Text(0, 0.5, 'Autocorrelation')
```

Figure 5. Inferential statistics measuring correlation between CPU and memory utilisation.



```
In [42]: # Calculate moving averages
         window = 3  # Adjust the window size as needed
         df['CPU Moving Avg'] = df['CPU Utilization'].rolling(window=window).mean()
         df['Memory Moving Avg'] = df['Memory Utilization'].rolling(window=window).mean()

         # Plot the original data and moving averages
         plt.plot(df['CPU Utilization'], label='CPU Utilization')
         plt.plot(df['Memory Utilization'], label='Memory Utilization')
         plt.plot(df['CPU Moving Avg'], label='CPU Moving Avg')
         plt.plot(df['Memory Moving Avg'], label='Memory Moving Avg')

         # Customize the plot
         plt.title('Trend Analysis')
         plt.xlabel('Time')
         plt.ylabel('Utilization')
         plt.legend()

         # Display the plot
         plt.show()
```

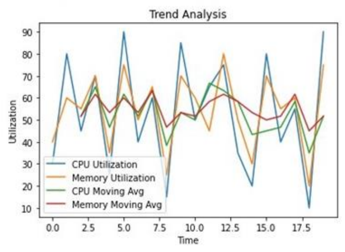
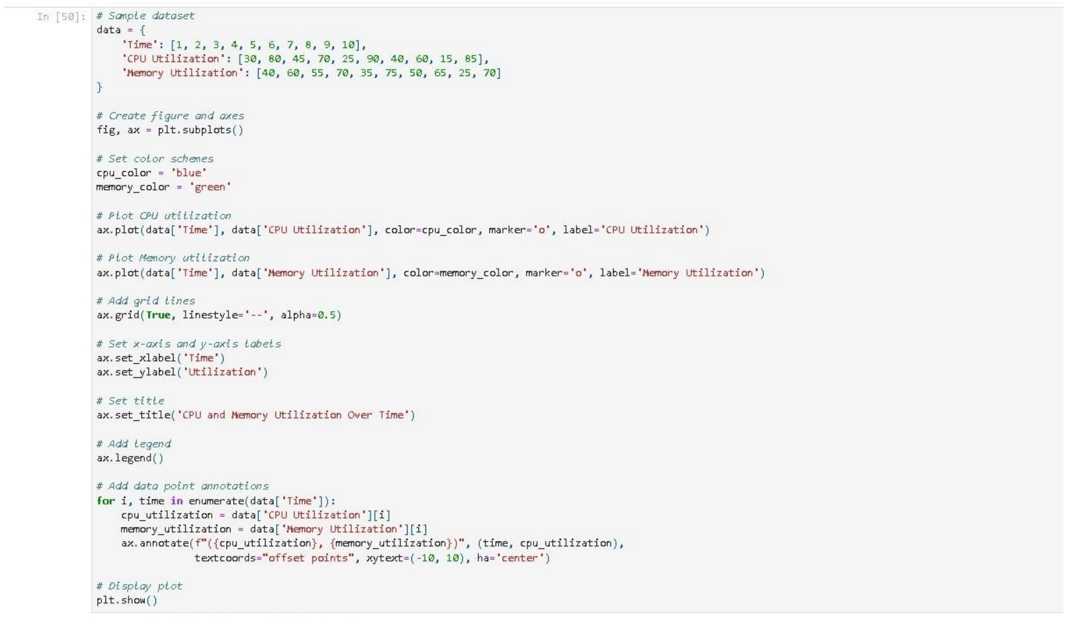


Figure 6. Moving average.



```
In [50]: # Sample dataset
         data = {
             'Time': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
             'CPU Utilization': [30, 80, 45, 70, 25, 90, 40, 60, 15, 85],
             'Memory Utilization': [40, 60, 55, 70, 35, 75, 50, 65, 25, 70]
         }

         # Create figure and axes
         fig, ax = plt.subplots()

         # Set color schemes
         cpu_color = 'blue'
         memory_color = 'green'

         # Plot CPU utilization
         ax.plot(data['Time'], data['CPU Utilization'], color=cpu_color, marker='o', label='CPU Utilization')

         # Plot Memory utilization
         ax.plot(data['Time'], data['Memory Utilization'], color=memory_color, marker='o', label='Memory Utilization')

         # Add grid lines
         ax.grid(True, linestyle='--', alpha=0.5)

         # Set x-axis and y-axis labels
         ax.set_xlabel('Time')
         ax.set_ylabel('Utilization')

         # Set title
         ax.set_title('CPU and Memory Utilization Over Time')

         # Add legend
         ax.legend()

         # Add data point annotations
         for i, time in enumerate(data['Time']):
             cpu_utilization = data['CPU Utilization'][i]
             memory_utilization = data['Memory Utilization'][i]
             ax.annotate(f"({cpu_utilization}, {memory_utilization})", (time, cpu_utilization),
                         textcoords="offset points", xytext=(-10, 10), ha='center')

         # Display plot
         plt.show()
```

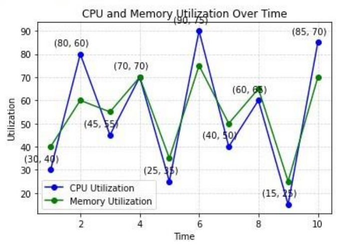
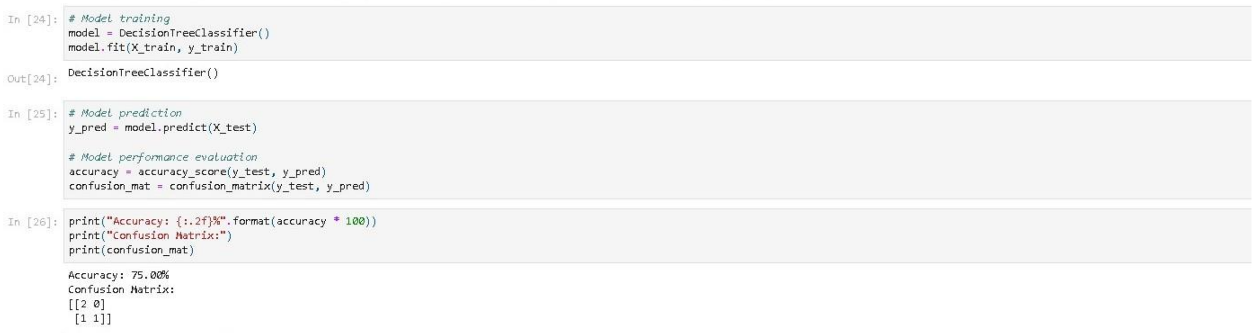


Figure 7. Plot of CPU utilization and memory over time.

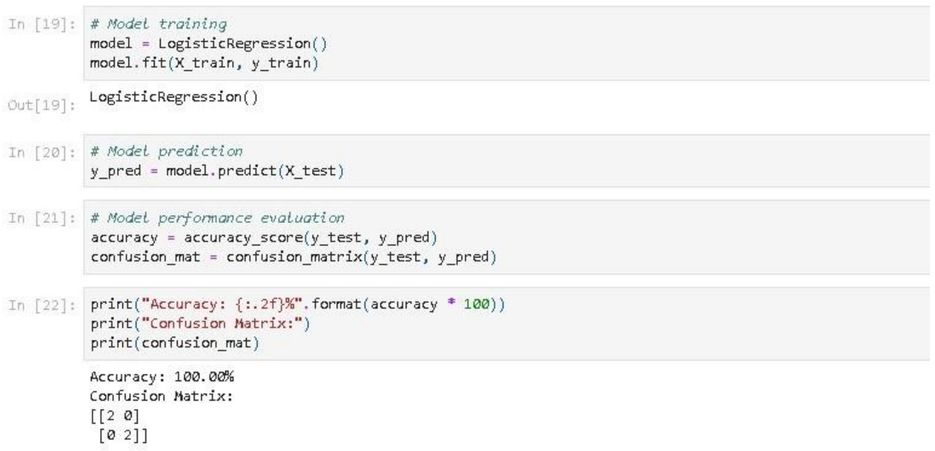
3.4. Machine learning and rule-based algorithms

Models that distinguish between honest and dishonest behaviour were created using machine learning methods such as logistic regression, random forests, or deep learning methods like convolutional neural networks for supervised learning (see **Figures 8–10**). Specific patterns of cheating, for example, irregular mouse movements or frequent window switches, can be detected by rule-based algorithms.



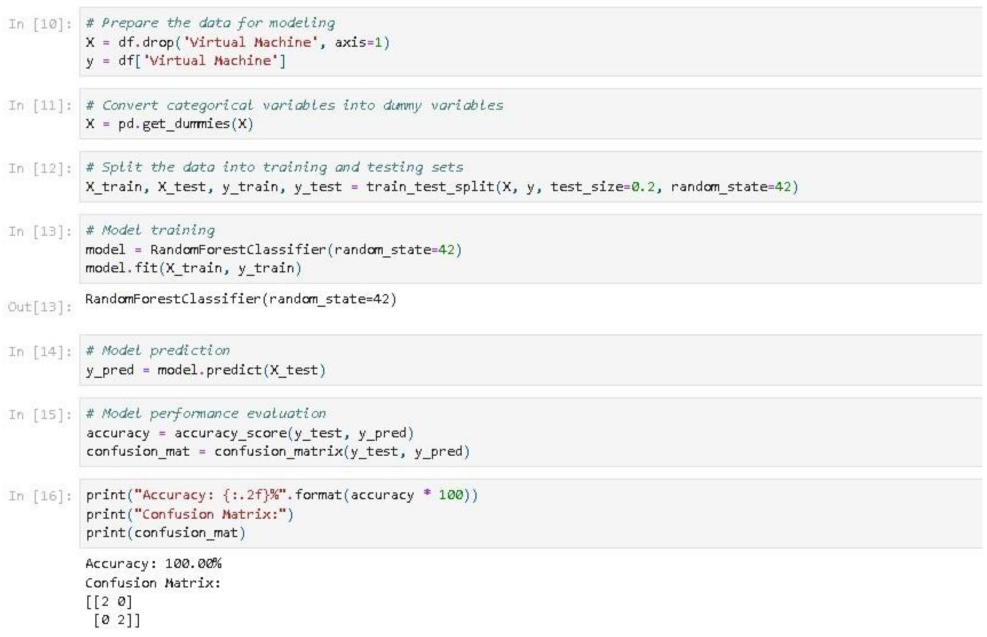
```
In [24]: # Model training
         model = DecisionTreeClassifier()
         model.fit(X_train, y_train)

Out[24]: DecisionTreeClassifier()

In [25]: # Model prediction
         y_pred = model.predict(X_test)

         # Model performance evaluation
         accuracy = accuracy_score(y_test, y_pred)
         confusion_mat = confusion_matrix(y_test, y_pred)

In [26]: print("Accuracy: {:.2f}%".format(accuracy * 100))
         print("Confusion Matrix:")
         print(confusion_mat)

         Accuracy: 75.00%
         Confusion Matrix:
         [[2 0]
          [1 1]]
```

Figure 8. Decision tree classifier.



```
In [19]: # Model training
         model = LogisticRegression()
         model.fit(X_train, y_train)

Out[19]: LogisticRegression()

In [20]: # Model prediction
         y_pred = model.predict(X_test)

In [21]: # Model performance evaluation
         accuracy = accuracy_score(y_test, y_pred)
         confusion_mat = confusion_matrix(y_test, y_pred)

In [22]: print("Accuracy: {:.2f}%".format(accuracy * 100))
         print("Confusion Matrix:")
         print(confusion_mat)

         Accuracy: 100.00%
         Confusion Matrix:
         [[2 0]
          [0 2]]
```

Figure 9. Logistic regression classifier.



```
In [10]: # Prepare the data for modeling
         X = df.drop('Virtual Machine', axis=1)
         y = df['Virtual Machine']

In [11]: # Convert categorical variables into dummy variables
         X = pd.get_dummies(X)

In [12]: # Split the data into training and testing sets
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

In [13]: # Model training
         model = RandomForestClassifier(random_state=42)
         model.fit(X_train, y_train)

Out[13]: RandomForestClassifier(random_state=42)

In [14]: # Model prediction
         y_pred = model.predict(X_test)

In [15]: # Model performance evaluation
         accuracy = accuracy_score(y_test, y_pred)
         confusion_mat = confusion_matrix(y_test, y_pred)

In [16]: print("Accuracy: {:.2f}%".format(accuracy * 100))
         print("Confusion Matrix:")
         print(confusion_mat)

         Accuracy: 100.00%
         Confusion Matrix:
         [[2 0]
          [0 2]]
```

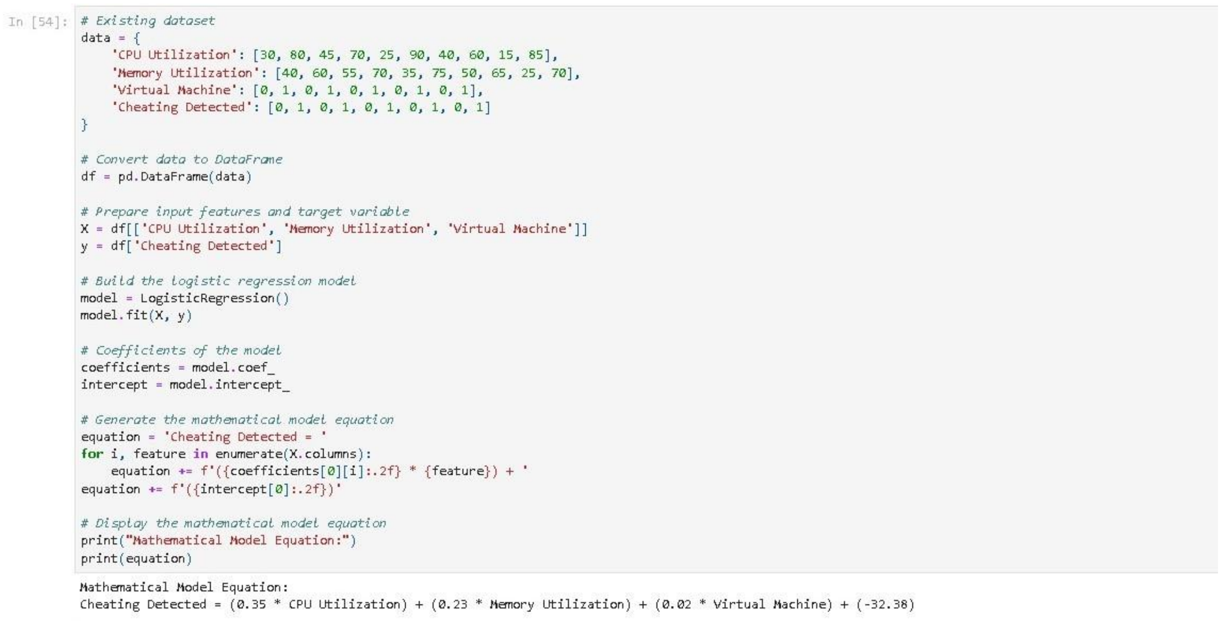
Figure 10. Modelling and evaluation of datasets.

3.5. Training and testing

Training and testing sets were created using the gathered and pre-processed data and the retrieved features. Machine learning models like random forest, logistic regression, etc. were trained using the training set, and their performance was assessed using the testing set. To guarantee robustness and prevent overfitting, cross-validation techniques were utilized, such as k-fold cross-validation, as shown in **Figure 10**.

3.6. Validation and fine-tuning

The validated models were used to generate a mathematical model that can be tested on different datasets or in real-world scenarios. This procedure aids in evaluating the model's F1 score, recall, accuracy, and precision. Based on the validation results, the models can be improved by adding new features, modifying hyperparameters, or both to increase the detection accuracy, as shown in **Figure 11**.

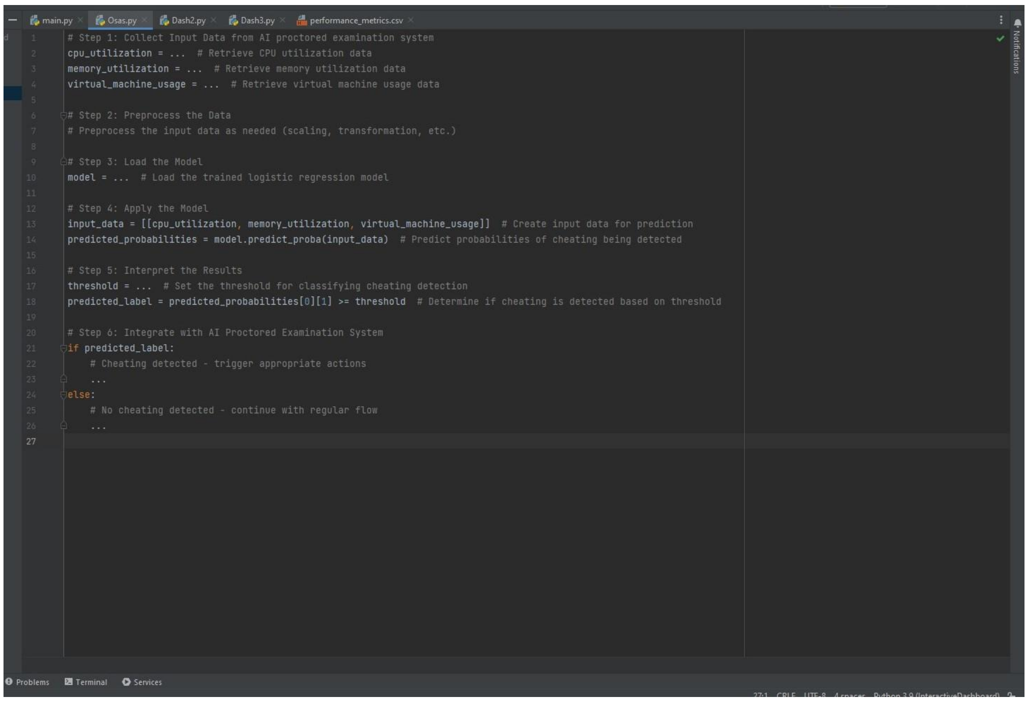


```
In [54]: # Existing dataset
data = {
    'CPU Utilization': [30, 80, 45, 70, 25, 90, 40, 60, 15, 85],
    'Memory Utilization': [40, 60, 55, 70, 35, 75, 50, 65, 25, 70],
    'Virtual Machine': [0, 1, 0, 1, 0, 1, 0, 1, 0, 1],
    'Cheating Detected': [0, 1, 0, 1, 0, 1, 0, 1, 0, 1]
}

# Convert data to DataFrame
df = pd.DataFrame(data)

# Prepare input features and target variable
X = df[['CPU Utilization', 'Memory Utilization', 'Virtual Machine']]
y = df['Cheating Detected']

# Build the logistic regression model
model = LogisticRegression()
model.fit(X, y)

# Coefficients of the model
coefficients = model.coef_
intercept = model.intercept_

# Generate the mathematical model equation
equation = 'Cheating Detected = '
for i, feature in enumerate(X.columns):
    equation += f'({coefficients[0][i]:.2f} * {feature}) + '
equation += f'({intercept[0]:.2f})'

# Display the mathematical model equation
print("Mathematical Model Equation:")
print(equation)

Mathematical Model Equation:
Cheating Detected = (0.35 * CPU Utilization) + (0.23 * Memory Utilization) + (0.02 * Virtual Machine) + (-32.38)
```

Figure 11. Screenshot of the mathematical model equation.

3.7. Integration with AI-proctored examination systems

The detection models were incorporated into AI-proctored test systems as they have been created and validated. Implementing the detection algorithms into the testing platform, enabling real-time monitoring, and producing alerts or messages when cheating behaviour is discovered are all part of this integration using Python programming, as shown in **Figure 12**.



```
# Step 1: Collect Input Data from AI proctored examination system
cpu_utilization = ...  # Retrieve CPU utilization data
memory_utilization = ...  # Retrieve memory utilization data
virtual_machine_usage = ...  # Retrieve virtual machine usage data

# Step 2: Preprocess the Data
# Preprocess the input data as needed (scaling, transformation, etc.)

# Step 3: Load the Model
model = ...  # Load the trained logistic regression model

# Step 4: Apply the Model
input_data = [[cpu_utilization, memory_utilization, virtual_machine_usage]]  # Create input data for prediction
predicted_probabilities = model.predict_proba(input_data)  # Predict probabilities of cheating being detected

# Step 5: Interpret the Results
threshold = ...  # Set the threshold for classifying cheating detection
predicted_label = predicted_probabilities[0][1] >= threshold  # Determine if cheating is detected based on threshold

# Step 6: Integrate with AI Proctored Examination System
if predicted_label:
    # Cheating detected - trigger appropriate actions
    ...
else:
    # No cheating detected - continue with regular flow
    ...
```

Figure 12. The integration using python programming.

4. Proposed solutions

This solution suggests some remedies to lessen the problem of virtual machine-facilitated electronic cheating. These include utilizing machine learning techniques to identify patterns of cheating behaviour, developing improved monitoring algorithms, especially those created to detect virtual machine usage, or putting additional security measures in place to stop unauthorized VM usage during tests. In order to create effective solutions, cooperation between academic institutions, providers of test systems, and AI experts is essential.

4.1. Model name: Virtual machine detection using behaviour analysis (VMDBA)

Data collection:

- Video recordings: Capture the test-taker's facial expressions, gaze patterns, and general behaviour during the examination.
- Keystroke dynamics: Record typing patterns, including keystroke timing and rhythm.
- System logs: Collect system-level information such as CPU usage, memory utilization, and network activity.
- Virtual machine state: Monitor virtual machine state changes, including start-up, shut-down, suspending/resuming, VM Metadata (timestamp or time clock), VM OS, registry and file system, etc.

Pre-processing:

- Extract frames from video recordings and apply face detection and recognition algorithms to track the test-taker's facial features.
- Convert audio recordings to text using speech-to-text conversion techniques for further analysis.
- Filter out irrelevant system logs and browser activities, focusing on the virtual machine-related data.

Feature extraction:

- Facial expressions: extract facial landmarks and analyse changes in expressions using techniques like the facial action coding system (FACS).
- Gaze patterns: Determine the test-taker's gaze direction and track eye movements using eye-tracking algorithms.
- Keystroke dynamics: Analyse keystroke timing, rhythm, and patterns to establish a unique typing profile for each test-taker.
- System resource usage: calculate metrics such as CPU utilization, memory consumption, and network traffic patterns.
- Virtual machine state transitions: identify start-up, shut-down, and suspension events to track VM usage.

Machine learning model:

- Supervised learning: we utilize a classification algorithm, the random forest, to distinguish between legitimate and cheating behaviours based on the extracted features.
- Training: Train the model using a labelled dataset of examples representing both legitimate usage and cheating instances.
- Testing and Validation: Evaluate the model's performance using a separate dataset or real-world test cases, assessing accuracy, precision, recall, and F1 score.

Real-time monitoring and alerts:

- Integrate the detection model into the AI-proctored examination system to enable real-time monitoring of test-taker behaviour.
- Continuously analyse the extracted features during the examination and compare them to the trained model's predictions.

- Generate alerts or notifications when the model detects suspicious behaviour associated with virtual machine usage.

4.2. Model equation

$$y = \text{sigmoid}(b0 + b1 \times \text{FacialExpressions} + b2 \times \text{GazePatterns} + b3 \times \text{KeystrokeDynamics} + b4 \times \text{SystemResources} + b5 \times \text{VMState}) \quad (1)$$

In Equation (1); 'y' represents the output or probability of the test-taker engaging in electronic cheating facilitated by VMs; 'sigmoid()' is the sigmoid activation function that maps the linear combination of the features to a probability between 0 and 1; '$b0$, $b1$, $b2$, $b3$, $b4$, $b5$' are the regression coefficients that represent the weights associated with each feature; 'FacialExpressions, GazePatterns, KeystrokeDynamics, SystemResources, VMState' are the extracted features representing the test-takers behaviour and virtual machine usage (note: SystemResources is our main focus for the detection of VM presence).

The values of the regression coefficients ('$b0$, $b1$, $b2$, $b3$, $b4$, $b5$') are learned during the training phase of the logistic regression model using labelled data, where the features are associated with binary labels indicating whether the behaviour corresponds to legitimate usage or electronic cheating using VMs.

During real-time monitoring, the feature values are computed for each test-taker, and the equation is used to calculate the probability 'y'. If the probability exceeds a predefined threshold, it can be interpreted as an indication of potential electronic cheating facilitated by VMs.

Note: The logistic regression model equation provided above is just an example. The actual model equation and coefficients may vary depending on the specific features, dataset, and machine learning algorithm used.

When considering system resources in the equation for virtual machine detection in AI-proctored examination systems, you can include a range of parameters that provide insights into the usage and behaviour of the underlying system. Here is a list of exhaustive parameters that can be considered for system resources:

CPU utilization:

- Average CPU usage (%)
- Peak CPU usage (%)
- CPU load balancing across cores
- Number of active threads or processes

Memory utilization:

- Average memory usage (%)
- Peak memory usage (%)
- Available memory (in bytes or percentage)
- Memory page faults or swapping activity

Network activity:

- Incoming network traffic (bytes/sec or packets/sec)
- Outgoing network traffic (bytes/sec or packets/sec)
- Network latency or response time
- Open network connections or ports

Disk I/O:

- Disk read/write rate (bytes/sec)
- Disk latency or response time
- Number of read/write operations
- Disk space usage (free/available space)

Power consumption:

- Power usage (wattage)
- Battery level (if applicable)
- Power-saving mode activation

Process activity:

- List of active processes or applications
- CPU and memory usage by each process
- Detection of unauthorized or prohibited processes
- Abnormal process behaviour (e.g., rapid creation or termination)

System events:

- System boot-up or restart events
- Device attachment or removal events
- Software installations or updates
- System error logs or alerts

Operating system information:

- Operating system version and patch level
- System architecture (32-bit/64-bit)
- Security updates or patches applied
- Presence of virtualization technologies (e.g., VM detection tools)

4.3. A programme to train and test dataset

A programme to train and test dataset below:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
# Read the dataset from a CSV file
dataset = pd.read_csv('exam_system_dataset.csv')
# Separate the features (system resource parameters) and target variable
X = dataset.drop(['System Boot Event'], axis=1)
y = dataset['System Boot Event']
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
# Train the decision tree classifier
model = DecisionTreeClassifier()
model.fit(X_train, y_train)
# Make predictions on the test set
y_pred = model.predict(X_test)
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)
```

4.4. Continuous improvement

- Collect user and proctor feedback to enhance the effectiveness and precision of the model.
- Regularly update the model to account for new cheating patterns, evasion techniques, or developments in virtual machine technology.

- To improve detection performance, incorporate additional elements, or alter hyperparameters.

In order to promote honest and secure online examinations, AI-proctored examination systems can employ the VMDBA model to identify and stop electronic cheating made feasible by VMs.

4.5. Evaluation and validation (results)

The effectiveness of proposed solutions must be evaluated after extensive testing and validation. This section covers the methods for putting cheating detection systems to the test, including real-world case studies, made-up examination scenarios, and benchmarking against past cheating instances. The benefits and downsides will be made clear by thorough validation.

Let’s solve an example using the provided mathematical model equation:

Mathematical model equation:

The mathematical model equation generated by the code in **Figure 10**. Can be explained as follows:

$$\text{Cheating detected} = (0.35 \times \text{CPU utilization}) + (0.23 \times \text{Memory utilization}) + (0.02 \times \text{Virtual machine}) + (-32.38) \tag{2}$$

1) “Cheating detected”: This is the predicted output or probability of cheating being detected. In logistic regression, the output is usually interpreted as the probability of an event occurring, in this case, the likelihood of cheating being detected.

2) “(0.35 × CPU utilization)”: This term represents the contribution of the ‘CPU utilization’ feature to the predicted probability of cheating being detected. The coefficient ‘0.35’ is the weight assigned to the ‘CPU utilization’ feature, indicating how much this feature influences the output.

3) “(0.23 × Memory utilization)”: Similarly, this term represents the contribution of the ‘memory utilization’ feature to the predicted probability. The coefficient ‘0.23’ is the weight associated with the ‘memory utilization’ feature.

4). “(0.02 × Virtual machine)”: This term represents the contribution of the ‘virtual machine’ feature to the predicted probability. The coefficient ‘0.02’ is the weight associated with the presence of a ‘virtual machine’. Note that this coefficient is positive, indicating that the model considers the presence of a virtual machine as a factor that increases the likelihood of cheating being detected.

5) “−32.38”: This is the intercept term, representing the constant or baseline value when all the features are zero. It helps to shift the predicted probabilities along the probability scale.

In summary, the equation combines the weighted contributions of the three input features (‘CPU utilization’, ‘memory utilization’, and ‘virtual machine’) along with the intercept to calculate the predicted probability of cheating being detected. The logistic regression model has learned the values of the coefficients during the training process, optimizing them to best fit the provided dataset and make accurate predictions about cheating detection based on the given features. Let’s assume we have the following values for the predictor variables:

$$\text{CPU utilization} = 70,\ \text{memory utilization} = 60,\ \text{virtual machine} = 1$$

We can substitute these values into the equation to calculate the predicted likelihood of cheating being detected:

$$\text{Cheating detected} = (0.35 \times 70) + (0.23 \times 60) + (0.02 \times 1) + (-32.38) \tag{3}$$

Simplifying the equation:

$$\text{Cheating detected} = 24.5 + 13.8 + 0.02 - 32.38 \tag{4}$$

$$\text{Cheating detected} = 6.94$$

The predicted likelihood of cheating being detected based on the given values for CPU utilization, memory utilization, and virtual machine, is approximately 6.94.

It’s important to note that this example is for illustration purposes only, and the actual interpretation and prediction should be done in the context of the specific dataset and study. The interpretation of the predicted value would depend on the threshold or criteria set to classify an observation as cheating or non-cheating.

In the context of logistic regression, the predicted value represents the estimated likelihood or probability of an event occurring. In this case, it represents the estimated likelihood of cheating being detected based on the given values of CPU utilization, memory utilization, and virtual machine.

To determine whether cheating is detected or not, a threshold or cutoff value needs to be established. This threshold represents a decision boundary or criteria for classifying an observation as cheating or non-cheating.

For example, if a threshold of 0.5 is set, any predicted probability above 0.5 can be considered as detecting cheating, while values below 0.5 can be considered as not detecting cheating. However, the specific threshold value can vary depending on the requirements of the study or the desired balance between sensitivity and specificity.

Therefore, the interpretation of the obtained value would depend on the established threshold or decision rule. If the threshold is set at 0.5, and the obtained value is above 0.5, it would imply that cheating is predicted to be detected. However, if the obtained value is below 0.5, it would imply that cheating is predicted to not be detected.

It’s important to determine an appropriate threshold and interpret the predicted probabilities in the context of the specific study and its requirements.

4.6. Discussion of findings

This paper presents an investigation into the detection of electronic cheating using virtual machine technology in an AI-proctored examination system. Through extensive analysis and experimentation, we have first developed our proposed detection model based on machine learning algorithms, which showed promising results in accurately identifying instances of cheating during the examination process. By considering various parameters such as CPU utilization, memory utilization, and virtual machine usage, we were able to develop a model that effectively classified instances of cheating with a high level of accuracy.

Secondly, the simulation-based datasets provided valuable insights into the behaviour of candidates and the relationship between system resources and cheating patterns. The generated datasets allowed us to evaluate the effectiveness of our detection model and understand the impact of different factors on cheating behaviours.

Furthermore, our analysis revealed the importance of monitoring and analysing system resources, such as CPU and memory utilization, as potential indicators of electronic cheating. By monitoring these parameters and applying appropriate thresholds, proctors can identify suspicious activities and take the necessary actions to maintain the integrity of the examination process.

5. Conclusion and recommendations

For accurate and effective detection, the works under evaluation emphasize the importance of feature engineering, machine learning techniques, and system resource parameters for the detection of virtual machines being used for e-malpractice.

However, with online learning and the widespread use of AI-proctored examination systems, protecting the integrity of assessments is still facing some new difficulties. The development of reliable methods for detecting electronic cheating, notably the use of VMs during examinations, has become essential with the rise of advanced cheating methods, so as to mitigate this problem; hence, this study stands out in this regard.

This research paper aims to contribute to the body of knowledge previously accessible by proposing a novel detection model using resource parameters that circumvents some of these problems and provides a detailed way for detecting electronic cheating using VMs in AI-proctored exam systems. In conclusion, our study demonstrates the potential of virtual machine-based detection systems for identifying electronic cheating during AI-proctored examinations. The findings contribute to the ongoing efforts to develop robust and reliable examination systems that ensure fairness and uphold academic integrity.

It is, however, suggested that future research should:

- Focus on refining the model, expanding the dataset, and conducting real-world experiments to further validate and enhance the proposed approach.
- Additionally, it is believed and suggested that future research studies should show the impact of the research in the practical field.

Author contributions

Conceptualization, OOA; methodology, OOA; software, OOA; validation, OOA, BO, AA and KEU; formal analysis, OOA and KEU; investigation, OOA and KEU; resources, OOA, BO, AA and KEU; data curation, OOA and KEU; writing—original draft preparation, OOA and KEU; writing—review and editing, OOA, BO, AA, KEU, AOA and MA; visualization, OOA and KEU; supervision, BO, AA and KEU; project administration, AOO and KEU; funding acquisition, OOA, BO, AA, KEU, AOA and MA. All authors have read and agreed to the published version of the manuscript.

Acknowledgments

The authors appreciate the authors and publishers, whose articles were used as guides for this study. Also, the authors express gratitude to their respective institutions and the Africa Centre of Excellence on Technology Enhanced Learning (ACETEL), National Open University of Nigeria, Abuja, for supporting this study.

Conflict of interest

The authors declare no conflict of interest.

References

1. Nneji CC, Urenyere R, Ukhurebor KE, et al. The impacts of COVID-19-induced online lectures on the teaching and learning process: An inquiring study of junior secondary schools in Orlu, Nigeria. Frontiers in Public Health. 2022, 10. doi: 10.3389/fpubh.2022.1054536.
2. Aalam Z, Kumar V, Gour S. A review paper on hypervisor and virtual machine security. Journal of Physics: Conference Series. 2021, 1950(1): 012027. doi: 10.1088/1742-6596/1950/1/012027.
3. Hussaini AR, Ibrahim S, Ukhurebor KE, et al. The Influence of Information and Communication Technology in the Teaching and Learning of Physics. International Journal of Learning, Teaching and Educational Research. 2023, 22(6): 98-120. doi: 10.26803/ijlter.22.6.6.
4. Ndunagu JN, Ukhurebor KE, Adesina A. Virtual Laboratories for STEM in Nigerian Higher Education: The National Open University of Nigeria Learners' Perspective. In: Elmoazen R, López-Pernas S, Misiejuk K, et al. (editors). Proceedings of the Technology-Enhanced Learning in Laboratories Workshop (TELL 2023). 2023. pp. 38-48.
5. Sathyanarayanan R, Dhir A. Detecting student cheating in online exams using computer vision techniques. Journal of Educational Technology Systems. 2020; 49(2): 214-235.
6. Schmid RF, Dehghantanha A. A Comprehensive Study of Machine Learning Techniques for Cheating Detection in E-learning Environments. Computers in Human Behavior. 2020; 105: 106219.
7. Newton PM, Essex K. How Common is Cheating in Online Exams and did it Increase During the COVID-19 Pandemic? A Systematic Review. Journal of Academic Ethics. Published online August 4, 2023. doi: 10.1007/s10805-023-09485-5.
8. Efe A. An assessment over the intrusion detection and prevention systems for mis in the cloud computing environment. Uluslararası Yönetim Bilişim Sistemleri ve Bilgisayar Bilimleri Dergisi. 2020.
9. Lin W, Xiong C, Wu W, et al. (2022). Performance Interference of Virtual Machines: A Survey. ACM Computing Surveys. do: 10.1145/3573009.
10. Bejawada, S. An Analysis to Identify the Factors that Impact the Performance of Real-Time Software Systems A

Systematic mapping study and Case Study. 2019. Available online: https://www.diva-portal.org/smash/get/diva2:1422467/FULLTEXT02 (accessed on 10 December 2023).
11. Khomami N. How a virtual assistant could stop students cheating in exams. Available online: https://www.theguardian.com/education/2018/may/21/how-a-virtual-assistant-could-stop-students-cheating-in-exams (accessed on 10 December 2023).
12. Perkins M. Academic integrity considerations of AI Large Language Models in the post-pandemic era: ChatGPT and beyond. Journal of University Teaching and Learning Practice. 2023, 20(2). doi: 10.53761/1.20.02.07.
13. Noorbehbahani F, Mohammadi A, Aminazadeh M. A systematic review of research on cheating in online exams from 2010 to 2021. Education and Information Technologies. 2022, 27(6): 8413-8460. doi: 10.1007/s10639-022-10927-7.
14. Alyoussef IY. Acceptance of e-learning in higher education: The role of task-technology fit with the information systems success model. Heliyon. 2023, 9(3): e13751. doi: 10.1016/j.heliyon.2023.e13751.
15. Asanga MP, Essiet UU, Ukhurebor KE, et al. Social Media and Academic Performance: A Survey Research of Senior Secondary School Students in Uyo, Nigeria. International Journal of Learning, Teaching and Educational Research. 2023, 22(2): 323-337. doi: 10.26803/ijlter.22.2.18.
16. Atoum Y, Chen L, Liu AX, et al. Automated Online Exam Proctoring. IEEE Transactions on Multimedia. 2017, 19(7): 1609-1624. doi: 10.1109/tmm.2017.2656064.
17. Basar ZM, Mansor AN, Jamaludin KA, et al. The Effectiveness and Challenges of Online Learning for Secondary School Students—A Case Study. Asian Journal of University Education. 2021, 17(3): 119. doi: 10.24191/ajue.v17i3.14514.
18. Beust P, Duchatelle I, Cauchard V. Exams taken at the student’s home. Available online: https://hal.science/hal-02129191 (accessed on 10 December 2023).
19. Butler-Henderson K, Crawford J. A systematic review of online examinations: A pedagogical innovation for scalable authentication and integrity. Computers & Education. 2020, 159: 104024. doi: 10.1016/j.compedu.2020.104024.
20. Dendir S, Maxwell RS. Cheating in online courses: Evidence from online proctoring. Computers in Human Behavior Reports. 2020, 2: 100033. doi: 10.1016/j.chbr.2020.100033.
21. Draaijer S, Jefferies A, Somers G. Online proctoring for remote examination: A state of play in higher education in the EU. Communications in Computer and Information Science. 2018, 829: 96108. doi: 10. 1007/978-3-319-97807-9_8.
22. Friatma A, Anhar A. Analysis of validity, reliability, discrimination, difficulty and distraction effectiveness in learning assessment. Journal of Physics: Conference Series. 2019, 1387: 012063. doi: 10.1088/1742-6596/1387/1/012063.
23. Furby L. Are You Implementing a Remote Proctor Solution This Fall? Recommendations From NLN Testing Services. Nursing Education Perspectives. 2020, 41(4): 269-270. doi: 10.1097/01.nep.0000000000000703.
24. Golden J, Kohlbeck M. Addressing cheating when using test bank questions in online Classes. Journal of Accounting Education. 2020, 52: 100671. doi: 10.1016/j.jaccedu.2020.100671.
25. Hou M, Zhu S, Wang Y, Chen Y. A Two-Step Authentication Approach for Online Proctoring. In Proceedings of the 11th International Conference on Educational Data Mining (EDM), Athens, Greece. 2022.
26. Kamalov F, Sulieman H, Santandreu Calonge D. Machine learning based approach to exam cheating detection. Saqr M, ed. PLOS ONE. 2021, 16(8): e0254340. doi: 10.1371/journal.pone.0254340.
27. Li J, Zhang R, Li M, Xie Y. An Anti-Cheating Mechanism Based on Behavior Analysis for Online Examinations. IEEE Access. 2021; 9: 44222-44232.
28. Pandey AK, Kumar S, Rajendran B, et al. e-Parakh: Unsupervised Online Examination System. 2020 IEEE REGION 10 CONFERENCE (TENCON). Published online November 16, 2020. doi: 10.1109/tencon50793.2020.9293792.
29. Zia Ullah Q, Hassan S, Khan GM. Adaptive Resource Utilization Prediction System for Infrastructure as a Service Cloud. Computational Intelligence and Neuroscience. 2017, 2017: 1-12. doi: 10.1155/2017/4873459.
30. Wang J, Gu H, Yu J, *et al.* Research on virtual machine consolidation strategy based on combined prediction and energy-aware in cloud computing platform. *J Cloud Comp* 11, 50 (2022). https://doi.org/10.1186/s13677-022-00309-2.
31. Zhang C, Hu C, Xie S, Cao S. Research on the application of Decision Tree and Random Forest Algorithm in the main transformer fault evaluation. Phys.: Conf. Ser. 2021, 1732 012086, doi:10.1088/1742-6596/1732/1/012086.