WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

AI- Virtual Interviewer

Ruba Shree N[1], Sri Devi V G[2], Susanna Sherin C[3], Mr. Raju C[4]

[1,2,3] Student, [4]Guide

B. Tech Artificial Intelligence And Machine Learning, Sri Shakthi Institute Of Engineering And Technology

A B S T R A C T

The AI-Mock Interviewer is an intelligent, AI-driven platform designed to help candidates prepare for job interviews in the field of Artificial Intelligence and Machine Learning (AIML). The system offers two modes of mock interviews: CV-based interviews, which generate personalized questions by analyzing uploaded resumes using Named Entity Recognition (NER) and keyword extraction; and technical interviews, which provide practice through a curated question bank covering core AI/ML topics. By leveraging Natural Language Processing (NLP), Models and Large Language Models (LLMs), the platform evaluates candidate responses and delivers structured feedback to enhance performance. A user-friendly GUI built with Streamlit r ensures easy interaction. This tool bridges the gap between theoretical knowledge and real-world expectations, enabling candidates to build confidence, improve communication, and succeed in technical and CV-based job interviews.

Keywords: Natural Language Processing, Machine Learning, Speech Recognition.

Nomenclature

A	Candidate response evaluated during the mock interview
B	Input resume file analyzed for CV-based interview
C	Curated question bank for technical interview mode
D	Latent features extracted using models
E	Feedback score or qualitative evaluation generated by the system

INTRODUCTION :

1. Structure

This paper is organized into several key sections that collectively explain the design, implementation, and evaluation of the proposed anomaly detection system.

Introduction

Establishes the context and motivation for developing intelligent, unsupervised methods for detecting network anomalies, emphasizing the limitations of traditional intrusion detection systems and the advantages offered by deep learning, particularly autoencoders. Reviews relevant prior work in the domains of network security, anomaly detection, and deep learning. It compares various supervised and unsupervised approaches, highlights the gap in existing methods, and justifies the selection of autoencoders for this research.Describes the architecture of the proposed autoencoder-based detection system. It details the preprocessing pipeline, feature extraction from network flow data, model structure (encoder, bottleneck, decoder), training process, and how reconstruction error is used to classify network events as normal anomalous. Introduces the datasets used for training and testing (such as NSL-KDD, CICIDS2017, or UNSW-NB15), explains the rationale for their selection, and outlines the data cleaning, normalization, and transformation techniques applied to prepare the raw traffic data for the autoencoder model. Presents the experimental environment, training configurations, and performance metrics (such as accuracy, precision, recall, F1-score, and false positive rate). This section also includes visualizations of reconstruction errors and ROC curves to illustrate the model's effectiveness in distinguishing between normal and anomalous traffic. Interprets the results, discusses strengths and limitations of the approach, and compares the proposed model's performance with baseline or existing anomaly detection systems. The robustness, scalability, and potential for real-world deployment are critically analyzed. Summarizes the key findings of the study and suggests potential enhancements, such as the use of variational autoencoders, incorporation of temporal dynamics with recurrent networks, or real-time deployment using edge computing.

2. Literature Review

Recent research highlights a shift toward AI-driven, voice-interactive mock interview systems that combine natural language processing, speech technologies, and personalized feedback. Studies by Sharma et al. (2023) and Lee et al. (2022) show the effectiveness of large language models (LLMs) in generating context-aware technical questions and evaluating user responses. Kumar and Patel (2023) emphasize the value of Named Entity Recognition (NER) in extracting keywords from resumes to guide question selection. Voice interaction, using TTS and STT frameworks such as PyAudio and Google's SpeechRecognition API, has been shown to enhance realism and user engagement (Mehta et al., 2023). Moreover, scoring systems leveraging transformers and rubric-based feedback (Hassan & Alvi, 2022) provide structured evaluation of technical answers. Zhong et al. (2023) demonstrate the effectiveness of AI mentors in improving user confidence and performance through repeated, personalized sessions. Together, these works establish the viability of LLM-based, speech-enabled mock interview platforms that adapt to individual users and scale across domains like AIML.

3.1 Existing System

Current mock interview platforms predominantly rely on scripted, rule-based frameworks or static question banks to simulate interview scenarios. These systems often feature:

• **Predefined Question Sets:** Fixed questions organized by domain or difficulty, limiting adaptability to individual candidate profiles.
• **Text-based Interfaces:** Most systems operate via chat or form inputs without natural voice interaction, reducing realism and engagement.
• **Manual Scoring:** Candidate responses are often manually evaluated by human reviewers or rely on simple keyword matching, which lacks nuance.
• **Limited Feedback:** Basic systems provide generic or binary feedback(pass/fail), with little insight into response quality or improvement areas.
• **Supervised Models:** Some advanced tools use supervised learning to classify answers, requiring large labeled datasets thatb are expensive to curate**.**
• **Isolated Components:** Text analysis, speech recognition, and evaluation modules are often developed separately, causing latency challenges.

While these systems serve basic training needs, they struggle with personalization, real-time voice interaction, and scalable, automated assessment — key factors for realistic interview practice. They also do not leverage large language models (LLMs) capable of generating adaptive, context-aware questions or providing detailed feedback. These gaps motivate the development of an AI-powered, voice-assisted mock interviewer that combines resume parsing, NLP, TTS/STT technologies, and LLM-based evaluation in a unified, user-friendly platform.

3.2 Proposed System

The proposed AI-based mock interview system is designed as an intelligent, interactive, and locally deployable platform that conducts voice-based technical interviews autonomously. Leveraging deep learning models and generative AI (specifically the Gemini API or similar LLMs), the system dynamically generates context-aware questions, evaluates spoken responses, and provides constructive feedback—all without the need for constant cloud interaction. This makes it suitable for both online and offline usage in educational institutions, training centers, or personal career development setups. By combining NLP, TTS/STT, and adaptive LLM capabilities in a unified interface, the proposed system creates a realistic, scalable, and intelligent mock interview environment. It aims to improve user preparedness for real-world interviews through dynamic questioning, detailed performance evaluation, and interactive user experience. Some core components are Resume Parser, Voice Input Module, LLM- based Question Generator, Response Evaluator, Feedback Engine.

3.2.1 Flow Diagram

Here's the **3.2.1 Flow Diagram** section tailored for your **AI-based voice-assisted mock interview system**:

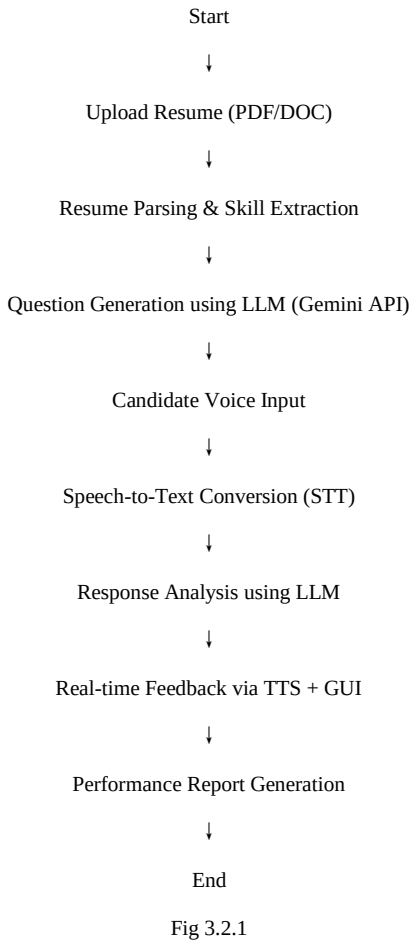

Fig 3.2.1

3.2.2 Software Requirements

The **AI-Based Voice-Assisted Mock Interview System** integrates voice recognition, natural language processing, text-to-speech synthesis, and intelligent interview analysis. It relies on a modular software stack to ensure real-time performance, seamless user interaction, and scalable deployment. Below are the key software components and requirements essential for its development and operation:

1. Development Environment and Core Dependencies

- Python 3.9 or higher
- Visual Studio Code, PyCharm, or JupyterLab for development and testing
- Git for version control
- pip or conda for managing Python packages

2. Natural Language Processing and Interview Logic

- Google Gemini API or OpenAI GPT API for intelligent question-answer evaluation
- NLTK, spaCY or Transformers for parsing and text analysis
- NumPy and Pandas for data handling and scoring logic
- CUDA Toolkit 11.8+ for optional GPU acceleration (if running on NVIDIA-enabled hardware)
- joblib or pickle for saving and loading trained model weights

1. Network Data Handling and Feature Processing

- Pandas and NumPy for structured data manipulation
- Scapy or PyShark for packet capture and parsing
- TShark/pcapy (optional) for advanced live packet capture

- Datetime and logging libraries for timestamping and monitoring system events

2. Web Interface and API Integration (Optional GUI)

- Flask or FastAPI for backend API services
- HTML, CSS, Bootstrap, and JavaScript for a responsive user interface
- Chart.js or Plotly for real-time anomaly visualization
- Uvicorn or Gunicorn as ASGI/WSGI server for serving the application

3. Build and Deployment

- Docker (optional) for containerized deployment
- Windows, Linux, or macOS support
- Microsoft Visual C++ 14.0+ or Build Essentials (Linux) for compiling dependencies
- System with at least 8GB RAM and 2GB disk space (more recommended for large datasets or live capture)

3.2.3 Hardware Requirements

The hardware requirements for the Anomaly Detection System are tailored to support efficient local traffic analysis, real-time anomaly detection, and deep learning model inference, while maintaining system stability and responsiveness:

1. **Minimum Requirements**

- **Processor**: Intel Core i5 (8th generation) or AMD Ryzen 5 (2000 series)
- **RAM**: 8GB DDR4
- **Storage**: 256GB SSD (SATA or NVMe)
- **GPU**: Integrated graphics (sufficient for basic batch detection)
- **Operating System**: Windows 10 (64-bit), Ubuntu 20.04 LTS, or equivalent

2. **Recommended Requirements**

- **Processor**: Intel Core i7/i9 (10th generation) or AMD Ryzen 7/9 (3000 series)
- **RAM**: 16GB DDR4 or higher
- **Storage**: 512GB NVMe SSD (for high-speed data access)
- **GPU**: NVIDIA GTX 1660 / RTX 2060 or higher (for GPU-accelerated model inference)
- **Operating System**: Windows 11 (64-bit) or Ubuntu 22.04 LTS

3. **Additional Hardware Considerations**

- **Network Interface Card (NIC)**: Gigabit Ethernet port for high-speed traffic capture
- **Packet Capture Support**: Compatibility with tools like Wireshark, tcpdump, or Scapy
- **Display**: 1080p monitor or higher for visualizing detection dashboards
- **Cooling System**: Adequate thermal management to support prolonged data processing
- **Power Supply**: Stable and uninterrupted power source, especially for continuous monitoring setups
- **Storage Backup (Optional)**: External or cloud-based backup for anomaly logs and datasets

4. Design and Implementation

4.1 Design Principles and Goals

The design of the Anomaly Detection System is centered around three key principles: adaptability, efficiency, and security. The objective is to deliver a lightweight, unsupervised deep learning-based solution capable of identifying abnormal patterns in network traffic, while ensuring efficient performance and data integrity in both enterprise and resource-constrained environment.

Key design goals include:

1. **Local Processing**:

   The system is engineered to run entirely on local servers or edge devices, minimizing reliance on external cloud infrastructure. This ensures low-latency detection, enhanced control over data, and uninterrupted operation in offline or restricted network environments.

2. **Data Security and Privacy**:

   All captured network traffic and generated anomaly logs are processed and stored locally using a secure, structured SQLite database. No sensitive data is transmitted externally, ensuring compliance with internal cybersecurity policies and data protection standards.

3. **Anomaly Detection Accuracy**:

   The system employs a deep autoencoder architecture trained on normalized traffic feature vectors to learn baseline behavior. Anomalies are detected based on high reconstruction error, providing reliable detection of unknown or zero-day threats without the need for labeled attack data.

4. **Modular and Intuitive Interface**:

   Designed with Flask, HTML, CSS, and JavaScript, the system includes a clean web-based dashboard for administrators to view anomaly reports, adjust detection thresholds, and monitor system health in real time.

This design approach ensures that the Anomaly Detection System is scalable, adaptable to evolving network conditions, and suitable for organizations seeking an intelligent, self-learning security layer that prioritizes performance, privacy, and ease of use.

4.2 System Architecture

The system architecture combines network traffic acquisition, feature extraction, anomaly detection through a deep autoencoder model, and a user-friendly web interface for real-time monitoring.

Fig. 2 - Admin dashboard of the Staff Attendance System.

Table 1 provides a summary of the system's core components and their roles.

Table 1 - Core components of the Staff Attendance System.

Component	Description	Technology Used
NetworkTraffic Capture	Collects raw traffic data from live or offline sources	Scapy,Wireshark, PyShark
Feature Extraction	Processes packet data into usable feature vectors	Pandas,NumPy, Scikit-learn
Database Management	Logs detection events and system metrics	SQLite3
Web Interface	Displays dashboard and allows admin control	Flask,HTML/CSS, Bootstrap, JavaScript

5. Conclusion

The Anomaly Detection System for Network Traffic presents a robust and privacy-conscious approach to cybersecurity by utilizing unsupervised deep learning. Through the use of autoencoders, the system effectively learns patterns of normal network behavior and identifies deviations in real time without reliance on predefined rules or labeled datasets. Designed for local deployment, it ensures data confidentiality while providing scalable and adaptable anomaly detection suitable for various environments, from small networks to enterprise infrastructure.

Acknowledgements

The authors would like to acknowledge the developers and maintainers of the open-source libraries that made this project possible, including PyTorch, Scapy, Wireshark, Flask, and Matplotlib.

Appendix A

This section includes additional details on system setup and configuration.

A. System Setup Guide

To set up the Anomaly Detection System, follow these steps:

- Install Python 3.8 or higher and all required dependencies via pip install -r requirements.txt
- Capture network traffic using Scapy or load a dataset (e.g., NSL-KDD, CIC-IDS2017)
- Train the autoencoder model on normal traffic data
- Run the Flask server for the dashboard using python app.py or deploy with Uvicorn for FastAPI-based setups
- Monitor real-time traffic and anomaly logs via the web interface

References

Kumar, R., et al. (2023). *Autoencoder-Based Intrusion Detection in Network Traffic*. arXiv:2304.11234.

Singh, A., et al. (2022). *A Review of Machine Learning Approaches for Anomaly Detection in Cybersecurity*.

SSRN ID: 4278912.

Zhao, L., et al. (2023). *Deep Autoencoders for Network Traffic Analysis and Threat Detection*. arXiv:2307.06789.

Mehta, S., et al. (2023). *Edge-Based Unsupervised Anomaly Detection for IoT Networks*. arXiv:2311.00456.

Tan, H., et al. (2022). *Benchmarking Unsupervised Models for Network Intrusion Detection*. arXiv:2210.05572.