Automated Information Extraction from NIST Databases Using NLP Techniques

A project report submitted in partial fulfillment of the requirements for the award of the degree of

B.Tech. in Computer Science

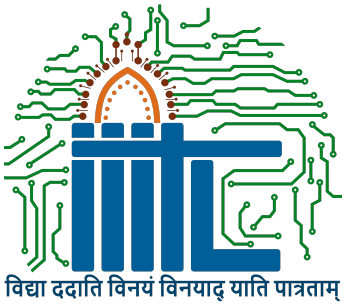
by

Satvik Dubey | Ashish Singh | Meet Savla | Gopal Singh Chauhan | Vikrant Palle

LCS2021017 | LCS2021016 | LCS2021010 | LCS2021025 | LCS2021011

under the guidance of

Dr. Gaurav Sharma



Indian Institute of Information Technology, Lucknow

ABSTRACT

This project addresses the challenge of extracting meaningful information from the vast, complex datasets within the National Institute of Standards and Technology (NIST) National Vulnerability Database (NVD). Traditional, manual approaches to data extraction are time-consuming and error-prone. By leveraging Natural Language Processing (NLP) and machine learning (ML) techniques, this project aims to automate the extraction and classification of vulnerability data, enhancing the accuracy and efficiency of data retrieval and analysis. Using the NVD API, we gathered data on vulnerabilities, including Common Vulnerabilities and Exposures (CVE) IDs, descriptions, and critical metrics related to confidentiality, integrity, and availability impacts.

A range of models, including Logistic Regression, Random Forest, Support Vector Machines (SVM), neural networks, Hidden Markov Models (HMMs), Conditional Random Fields (CRFs), and BERT, were implemented to classify vulnerabilities by severity and extract relevant entities. Additionally, a T5 model was employed to automate SQL query generation for structured data retrieval. Named Entity Recognition (NER) techniques, utilizing spaCy and Transformer-based models, further enhanced data parsing by accurately identifying product and operating system entities.

The evaluation results indicate that both the Neural Network and the Random Forest models achieved the highest classification accuracy among machine learning models, while the BERT and T5 models performed the best among NLP-based models. Visualizations of severity, attack vectors, and impact scores provided key insights into vulnerability trends. This automated information extraction framework significantly streamlines vulnerability data analysis, providing a scalable solution for managing large-scale, unstructured data repositories in cybersecurity.

Contents

Chapter 1

Introduction

In the cybersecurity domain, timely access to accurate data on vulnerabilities is essential. The NIST NVD serves as a central repository of vulnerability data, yet manual data extraction is cumbersome. This project uses NLP and ML to automate the extraction of meaningful information from the NVD, offering a more efficient, scalable, and error-free method of data processing.

1.1 Goal of the Project

The primary goal of this project is to automate the extraction and classification of vulnerability data, providing structured insights for better analysis. Using the NVD API, the system gathers vulnerability details such as CVE IDs, descriptions, and associated impact metrics like confidentiality, integrity, and availability scores. This data is then processed using various ML models, including Logistic Regression, Support Vector Machines (SVM), Random Forests, and neural networks, to classify vulnerabilities based on severity.

1.2 Future Plans

Future work aims to expand the project's capabilities, including enhancing navigation assistance within the dataset and integrating with smart devices for real-time data access. This automated approach to vulnerability data extraction from NIST's NVD not only streamlines data retrieval but also provides a efficient solution to manage and analyze cybersecurity data, contributing to a proactive defense against evolving digital threats.

Chapter 2

Literature Review

The field of cybersecurity relies heavily on timely, accurate information about vulnerabilities to mitigate risks and protect systems. Traditional manual methods of extracting information from databases like the National Vulnerability Database (NVD) are not only time-consuming but also prone to human error. The application of Natural Language Processing (NLP) and Machine Learning (ML) techniques in this domain has shown promise in addressing these challenges. This literature review examines existing approaches in vulnerability detection, information extraction, and relevant NLP methods that inform the automated information extraction project.

2.1 Vulnerability Detection and Classification

Vulnerability detection has traditionally relied on manual reporting and structured data entries, which can limit scalability. The Common Vulnerabilities and Exposures (CVE) program provides a standardized approach for identifying vulnerabilities. CVEs are identified and scored using the Common Vulnerability Scoring System (CVSS), which categorizes them based on impact factors like confidentiality, integrity, and availability. Research on automating vulnerability detection has often focused on extracting structured data from unstructured or semi-structured text sources. Studies have shown that ML models, such as Support Vector Machines (SVMs) and Random Forests, are effective in classifying vulnerabilities based on severity. This approach provides a scalable solution for managing vast amounts of cybersecurity data while reducing the human effort needed to interpret and categorize each vulnerability entry.

2.2 Natural Language Processing Techniques in Information Extraction

NLP has proven to be an invaluable tool in extracting structured information from large volumes of text data. Named Entity Recognition (NER), for instance, is frequently used in cybersecurity to identify critical entities, such as software versions, products, and operating systems, within vulnerability descriptions. Traditional sequence labeling models, such as Hidden Markov Models (HMM) and Conditional Random Fields (CRF), have been applied successfully to tag and categorize relevant information. However, recent advancements in transformer-based models like BERT (Bidirectional Encoder Representations from Transformers) have significantly improved the accuracy and contextual understanding of entity extraction. BERT's ability to capture bidirectional context enables more accurate tagging of complex entities, especially in technical text.

2.3 Advanced Models for Sequential and Contextual Analysis

To handle sequential and contextual information in cybersecurity data, researchers have also implemented models such as Maximum Entropy Markov Models (MEMMs) and Conditional Random Fields (CRFs). These models are well-suited for tasks where the relationship between words in a sequence is essential for accurate classification. For example, CRFs are often applied in labeling sequences for NER tasks in CVE descriptions, categorizing terms into product names, attack vectors, and severity levels. The development of transformer models, such as BERT and T5 (Text-to-Text Transfer Transformer), has further enhanced the ability to process complex, context-rich data. T5, in particular, has shown efficacy in SQL query generation from natural language prompts, facilitating structured data retrieval from unstructured sources like NVD text records.

2.4 Visualization and Analysis of Vulnerability Trends

Data visualization is a critical component in vulnerability analysis, as it provides security professionals with insights into emerging trends and severity distributions. Research has shown that visual representations

of vulnerability data can enhance decision-making by highlighting areas with higher risks or frequent occurrences of specific attack vectors. Techniques such as time-series analysis, frequency distributions of severity levels, and clustering of vulnerabilities by affected systems have been used to understand the scope and impact of cybersecurity threats better. These visualizations, when combined with automated information extraction, can help organizations prioritize patches and mitigation strategies more effectively.

2.5 Future Directions in Automated Vulnerability Analysis

As NLP and ML techniques continue to evolve, future research is focusing on integrating these models with real-time data sources and IoT (Internet of Things) devices for continuous vulnerability monitoring. Studies have indicated the potential of deploying ML models on smart devices to provide instant alerts based on identified vulnerabilities, enhancing the speed and accuracy of responses. Additionally, the integration of NLP-driven systems with wearable technologies has opened new avenues for making vulnerability data accessible in various environments, further supporting proactive cybersecurity measures. Improved NER models and enhanced accuracy in complex entity extraction are also areas of active research, aiming to reduce false positives and provide more reliable analysis of vulnerability data.

Chapter 3

Methodology and Implementation

The goal of this project is to develop a comprehensive machine learning and natural language processing (NLP) pipeline for analyzing and classifying cybersecurity incidents based on their descriptions. This project involves several key components, including preprocessing, model training, evaluation, and entity extraction, with an emphasis on accuracy and interpretability. Below, we detail each step of the methodology and its implementation.

3.1 Data Preprocessing

The data preprocessing stage focuses on preparing the text data for machine learning models. Key preprocessing tasks include handling missing values, transforming labels, and vectorizing text data.

- **Data Cleaning**: Using `dropna`, rows with missing values in critical columns (such as `Processed_Description` and `Severity`) were removed to ensure data quality.
- **Label Encoding**: The `Severity` column, containing severity levels as strings, was encoded into numerical values using `LabelEncoder`. This transformation allowed us to convert categorical severity labels into a format that can be understood by machine learning algorithms.
- **TF-IDF Vectorization**: Text data in the `Processed_Description` column was transformed into numerical features using the TF-IDF vec-

torizer with a maximum of 5000 features and English stop words removed. This transformation produces a sparse matrix of features, which captures the importance of words in each description relative to the entire dataset.

3.1.1 Exploratory Data Analysis (EDA)

The following data is extracted from the CVE API :

- CVE_ID
- DESCRIPTION
- Description: descriptions
- Published_Year
- Published_Dates
- Operating_System/Product
- Attack_Vector
- Access_Vector
- Severity
- Authentication
- Impact_Score
- Base_Score
- Obtain_All_Privilege
- User_Interaction_Required
- Exploitability_Score
- CVSS_Vector_String
- Availability_Impact
- Integrity_Impact
- Confidentiality_Impact

- Obtain_User_Privileges
- Vulnerability_Status
- Last_Modified_Date

The data extracted from the CVE API provides comprehensive details about cybersecurity vulnerabilities, including key identifiers, descriptive information, severity levels, and impact metrics.

Attributes such as attack vectors, affected systems, and impact scores offer insight into the nature and potential consequences of each vulnerability. This rich dataset enables in-depth analysis, supporting visualizations and trends over time to enhance understanding of vulnerabilities and guide prioritization for security measures.

Now let us analyze some of them through visual graphs :

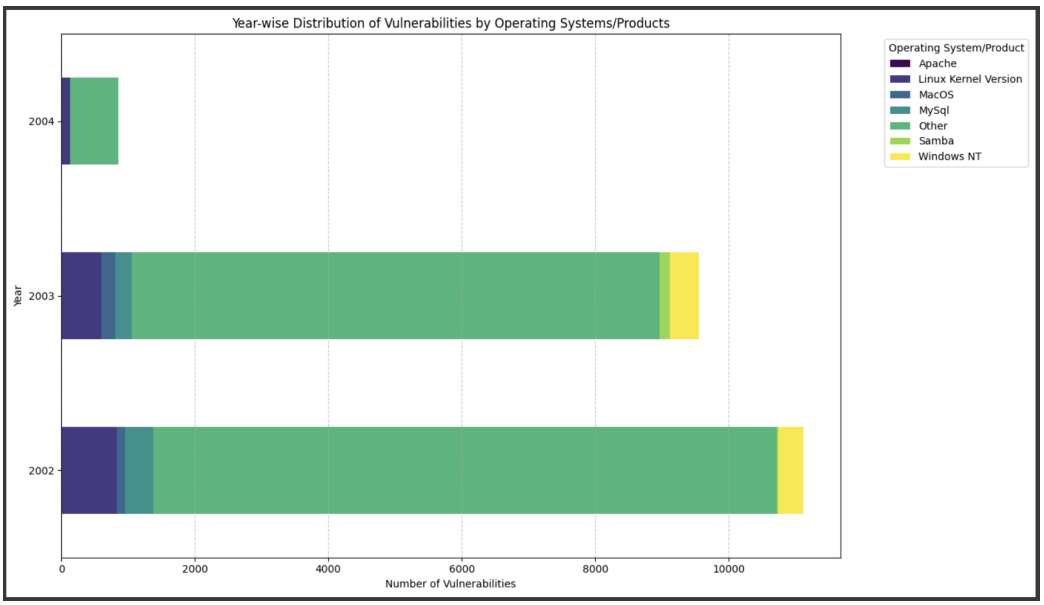


Figure 3.1: Year-wise Distribution of Vulnerabilities by Operating Systems/Products

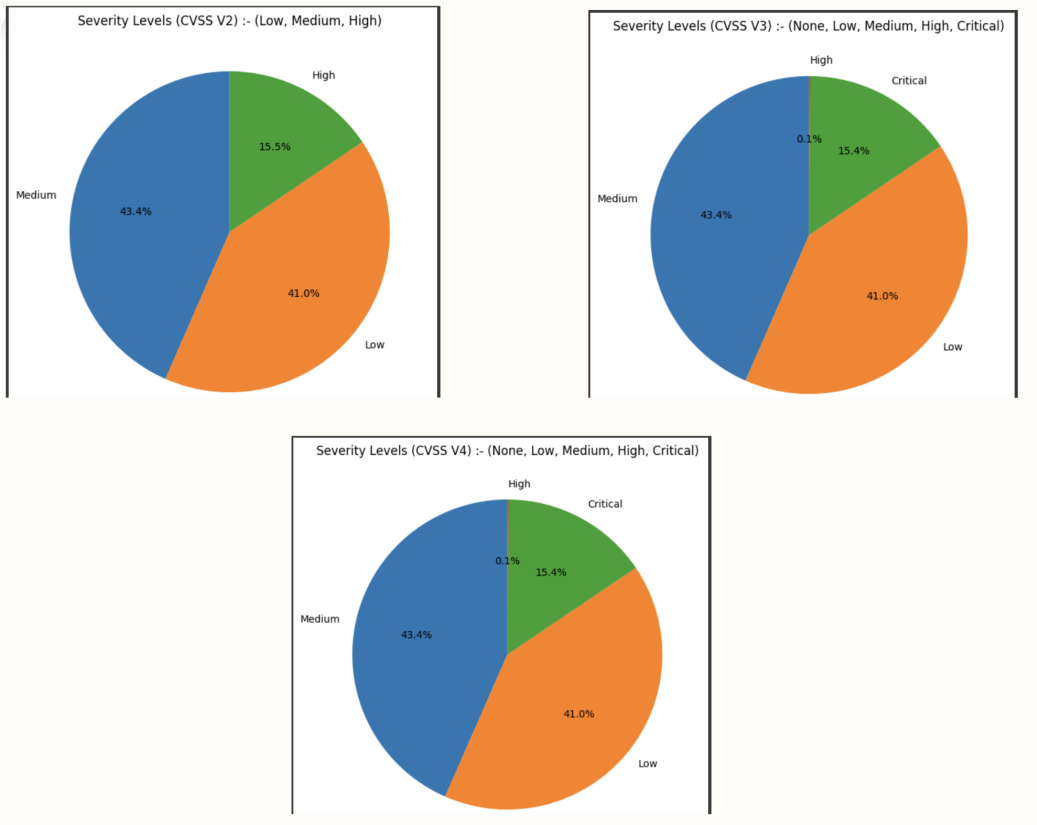


Figure 3.2: Distribution of Severity Levels according to v2, v3 and v4 versions in the CVE data.

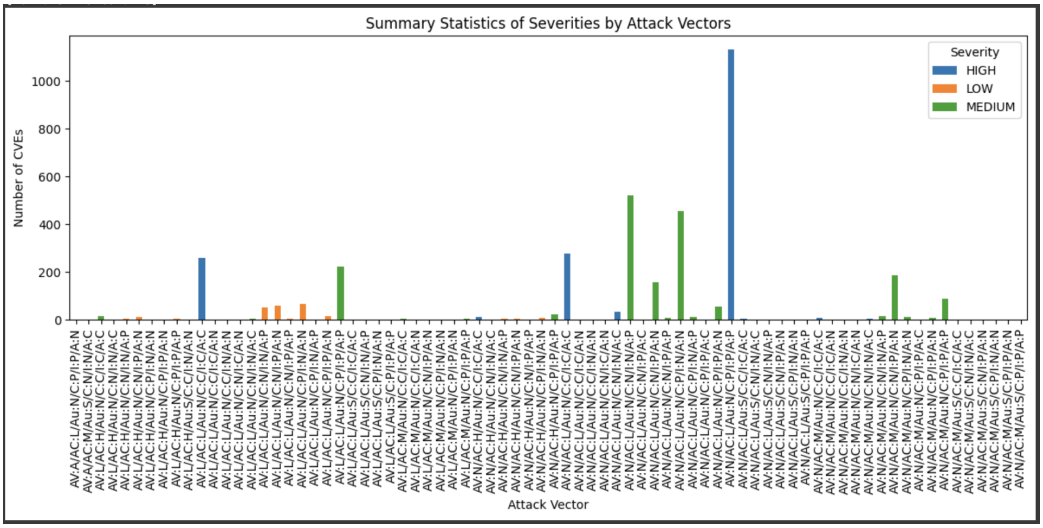
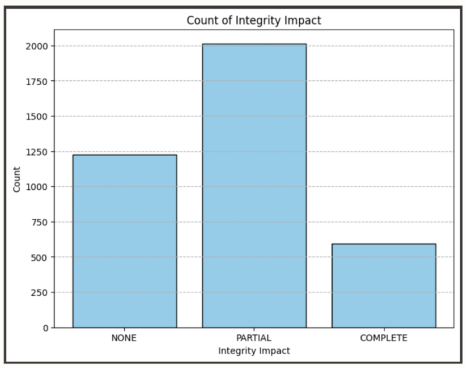
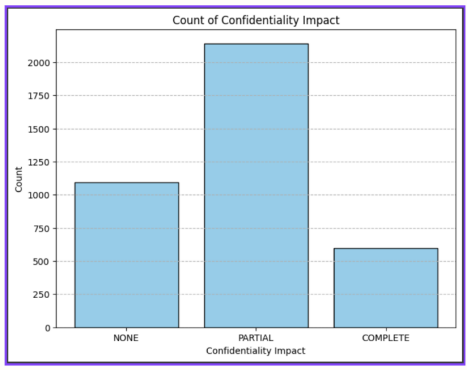


Figure 3.3: Summary Statistics of Severities by Attack Vectors



Integrity Impact	Count
NONE	1224
PARTIAL	2014
COMPLETE	596

Figure 3.4: Count of Integrity Impact



Confidentiality Impact	Count
NONE	1095
PARTIAL	2142
COMPLETE	597

Figure 3.5: Count of Confidentiality Impact

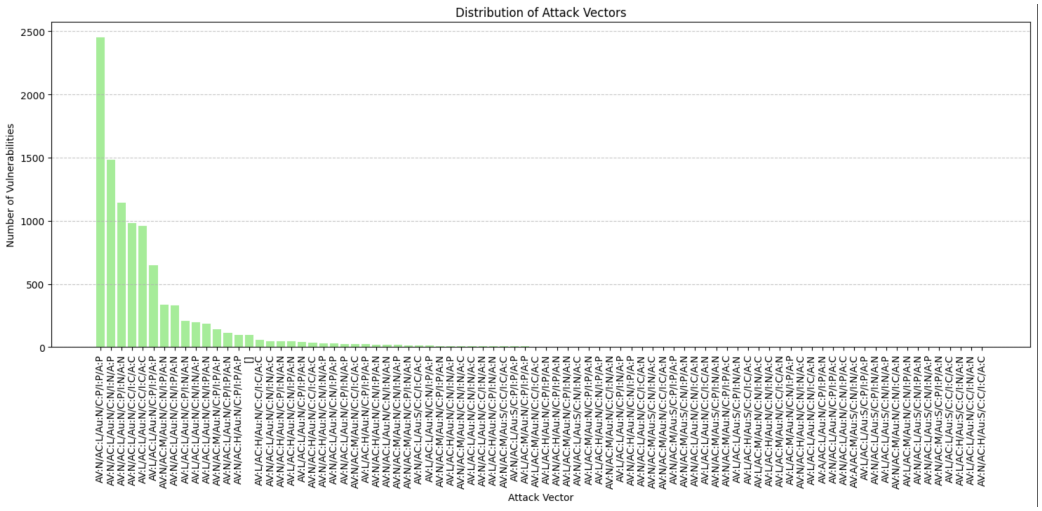


Figure 3.6: Distribution of Attack Vectors vs Frequency of Vulnerabilities.

3.2 Model Training and Evaluation

A set of machine learning classifiers were chosen for severity classification, including Logistic Regression, Random Forest, and Support Vector Machine (SVM). Additionally, a neural network model was developed to compare performance across traditional models and deep learning.

- **Model Initialization**: We initialized a dictionary of models, with the following configurations:
  - **Logistic Regression**: Configured with a maximum of 200 iterations to ensure convergence.
  - **Random Forest**: Configured with 100 trees and a fixed random state for reproducibility.
  - **Support Vector Machine**: Linear kernel and probability estimates enabled.
- **Training and Evaluation**: Each model was trained on 80% of the data (`X_train` and `y_train`) and evaluated on the remaining 20% (`X_test` and `y_test`). The accuracy of each model was recorded, and classification reports, including precision, recall, and F1-score, were generated. The model performances were visualized in a bar chart for easy comparison.
- **Neural Network Model**: A fully connected neural network was constructed using the Keras Sequential API:
  - **Architecture**: The network includes two hidden layers with 512 and 256 neurons, ReLU activation, and dropout layers to reduce overfitting.
  - **Training**: The model was compiled with the Adam optimizer and trained with a learning rate of 0.0005, using cross-entropy loss and accuracy as metrics. Training and validation accuracy were plotted across epochs to monitor performance.
- **Performance Comparison**: The final accuracies of the traditional models and the neural network were displayed alongside detailed metrics for each model. A DataFrame was created to consolidate accuracy, precision, recall, and F1-scores for each model. This comparison helped identify the most effective classifier.

3.3 Custom Description Prediction

For new or unseen descriptions, predictions were made using both traditional models and the neural network:

- **Preprocessing Custom Descriptions**: Each new description is cleaned by removing digits and special characters. It is then tokenized, lemmatized, and transformed using the trained TF-IDF vectorizer.
- **Severity Prediction**: Each model in the ensemble predicts the severity of the custom description, and the neural network provides a probability-based prediction. The severity label corresponding to the highest probability is selected as the final prediction.

3.4 Named Entity Recognition (NER)

To further analyze cybersecurity descriptions, named entity recognition (NER) was applied using two methods: spaCy and a BERT-based transformer model. This allowed us to extract entities related to organizations and products, enhancing understanding of the descriptions.

- **spaCy NER**: The spaCy library was used to extract entities labeled as "ORG" (organizations) and "PRODUCT" from the descriptions. This approach leverages rule-based and statistical NLP for efficient entity extraction.
- **BERT-based NER**: The Hugging Face `pipeline` API was used with a pre-trained BERT model fine-tuned for NER tasks. This model captures contextual relationships, allowing for the identification of entities in complex sentences.
- **Entity Extraction**: Entities were extracted from sample descriptions and displayed in a table format to compare the performance and output of both NER methods.

3.5 BIO Labeling and Sequence Preparation

BIO (Begin-Inside-Outside) tagging was implemented for token-level labeling of descriptions, targeting operating systems (OS), attack vectors (AV), and severity (SV) as label categories.

- **Token Labeling**: Regular expressions were used to label tokens. Specific keywords (e.g., “Windows,” “Linux”) were identified as operating systems, while words such as “Network” and “Local” were labeled as attack vectors. Severity labels like "Low" and "Critical" were similarly annotated.
- **Sequence Preparation**: Descriptions were converted into token sequences with BIO labels, creating structured data for training sequence models.

3.6 Model Implementation

In this project, various machine learning and natural language processing models were implemented to analyze and classify CVE descriptions based on severity, operating systems, attack vectors, and other features. The following sections detail the implementation and training of each model.

3.6.1 Logistic Regression

- **Purpose**: Used for classification tasks, particularly to predict probabilities mapping inputs to severity categories.
- **Data Preprocessing**: Text data was vectorized using TF-IDF to capture the importance of terms in the descriptions.
- **Training**: The model was trained with a maximum of 200 iterations to ensure convergence on the severity data.

3.6.2 Random Forest Classifier

- **Model Structure**: An ensemble of 100 decision trees, with each tree predicting severity levels independently.
- **Prediction Mechanism**: The final classification is determined by majority voting across all trees, which helps enhance stability and reduce overfitting.
- **Regularization**: Random feature selection was applied to each tree to control overfitting.

3.6.3 Support Vector Machine (SVM)

- **Kernel**: A linear kernel was used to separate severity classes.
- **Output**: Enabled probability outputs to assign class probabilities.
- **Benefits**: Robustness to high-dimensional data, providing a strong baseline for text classification.

3.6.4 Neural Network (Deep Learning Model)

- **Architecture**: A three-layer neural network with 256 neurons in the first layer, 128 in the second, and an output layer with softmax activation.
- **Regularization**: Dropout layers (dropout rate of 0.3) were used to prevent overfitting.
- **Training**: The model was trained over 10 epochs using the Adam optimizer and categorical cross-entropy loss.

3.6.5 Hidden Markov Model (HMM)

- **Data Preparation**: CVE descriptions were tokenized at the character level and labeled with corresponding OS/Product values. Unique characters were encoded as integers for compatibility with HMM.
- **Training**: Encoded sequences were flattened and fed into the model, which was trained using the Expectation-Maximization (EM) algorithm.
- **Inference**: The Viterbi algorithm was employed to predict OS/Product sequences for new descriptions, handling unknown tokens with `<UNK>`.

3.6.6 Maximum Entropy Markov Model (MEMM)

- **Data Preparation**: Similar to HMM, CVE descriptions were tokenized at the character level. Additional features, such as previous and current characters, and character position within the description, were extracted for improved context.

- **Transition and Emission Modeling**: Each state transition was modeled with logistic regression, predicting the next OS/Product label based on extracted features.
- **Inference**: The Viterbi algorithm was adapted to MEMM, using classifiers to handle context-dependent state transitions.

3.6.7 Conditional Random Fields (CRF)

- **Data Labeling**: BIO (Begin-Inside-Outside) scheme was applied to label tokens, with categories such as B-OS, B-AV, B-SV, and O.
- **Training**: A CRF model with LBFGS optimization was trained on labeled token sequences, learning dependencies between tokens.
- **Evaluation**: The model's performance was evaluated using precision, recall, and F1-scores for each label.

3.6.8 BERT

- **Embeddings**: BERT embeddings were generated by averaging the last hidden state for each CVE description.
- **Classification**: Logistic regression was trained on the BERT embeddings to classify OS and attack vectors, using BERT's token classification for finer BIO-tagged labeling.
- **Hyperparameter Tuning**: Parameters such as learning rate, batch size, and sequence length were tuned, with early stopping to prevent overfitting.

3.6.9 T5 Model

- **Model Setup**: The FLAN-T5 model was used for text-to-SQL generation with schema awareness.
- **Inference**: SQL queries were generated by feeding structured prompts into the model, using beam search with 10 beams.
- **Execution**: The generated SQL queries were executed in DuckDB, with example queries demonstrating functionality, such as retrieving CVE IDs based on publication year.

3.7 Training and Tuning of Models

To optimize the performance of each model used in the project, various training and hyperparameter tuning techniques were employed. These methods helped ensure that each model was tailored to the specific characteristics of the dataset, balancing accuracy and generalization across different machine learning tasks. Below is a detailed description of the training and tuning procedures for each model.

3.7.1 Logistic Regression

- **Training**: The Logistic Regression model was trained by minimizing binary cross-entropy or categorical cross-entropy loss, depending on the classification task. This loss minimization helped the model accurately map the input features to the severity categories.
- **Hyperparameter Tuning**: Regularization strength (`C`) was tuned using grid search to achieve an optimal balance between underfitting and overfitting. Cross-validation was applied to ensure that the model could generalize well across different data splits.

3.7.2 Random Forest

- **Training**: The Random Forest model was trained by building multiple decision trees on bootstrapped samples of the dataset, with the final prediction obtained by aggregating the predictions of each tree. This ensemble approach improved accuracy and robustness.
- **Hyperparameter Tuning**: Key parameters such as `n_estimators` (number of trees), `max_depth` (maximum depth of each tree), and `min_samples_split` were tuned using grid or randomized search to reduce overfitting and improve the model's accuracy.

3.7.3 Support Vector Machine (SVM)

- **Training**: The SVM model was trained by finding the hyperplane that maximized the margin between different classes. Several kernel functions (linear, polynomial, RBF) were evaluated to determine the best fit for the data.
- **Hyperparameter Tuning**: Parameters such as `C` (regularization parameter) and `gamma` (for RBF kernel) were fine-tuned using grid

search. This tuning aimed to balance margin maximization with model complexity, ensuring optimal generalization across unseen data.

3.7.4 Neural Network

- **Training**: The neural network model was trained using backpropagation, with stochastic gradient descent or a similar optimizer to minimize the loss function. This approach allowed the network to learn patterns in the input data over several epochs.
- **Hyperparameter Tuning**: Parameters such as the number of layers, neurons per layer, learning rate, batch size, and dropout rate were optimized. Techniques like early stopping and dropout were applied to prevent overfitting and improve generalization on the validation set.

3.7.5 Hidden Markov Model (HMM)

- **Training**: The HMM was trained using the Expectation-Maximization (EM) algorithm, which estimated both transition and emission probabilities for each state in the sequence.
- **Hyperparameter Tuning**: The number of hidden states and initial state probabilities were adjusted to improve model performance. Smoothing techniques were also applied to handle unseen states, thus enhancing the model's accuracy on sequence data.

3.7.6 Maximum Entropy Markov Model (MEMM)

- **Training**: The MEMM was trained using a maximum likelihood estimation approach, with weights on features optimized to maximize the likelihood of correct state transitions.
- **Hyperparameter Tuning**: Regularization parameters and feature functions were tuned to avoid overfitting, with cross-validation used to select the best-performing configuration.

3.7.7 Conditional Random Fields (CRF)

- **Training**: The CRF model was trained by maximizing the log-likelihood of correct label sequences over the training data, using algorithms like gradient descent for optimization.
- **Hyperparameter Tuning**: Regularization parameters (`C1` and `C2`) and feature functions were optimized to improve sequence labeling accuracy. Cross-validation ensured the model generalized well for this task.

3.7.8 BERT

- **Training**: The pre-trained BERT model was fine-tuned on task-specific data by adding a custom classification layer on top, enabling end-to-end training on this dataset.
- **Hyperparameter Tuning**: Parameters such as learning rate, batch size, and sequence length were tuned. Early stopping and dropout regularization were employed to prevent overfitting, and grid search was used to find the best configuration.

3.7.9 T5 Model

- **Training**: The T5 model was fine-tuned for information extraction tasks, with inputs and outputs tailored to task-specific requirements.
- **Hyperparameter Tuning**: Key parameters such as learning rate, batch size, and sequence length were adjusted for optimized generation quality. Beam search was used during inference, with tuning of `temperature` and `num_beams` to improve output accuracy.

Conclusion

The training and hyperparameter tuning steps for each model allowed us to balance performance with generalization, ensuring that each algorithm was well-suited to its task. Through careful parameter selection and tuning, we achieved robust models capable of accurate predictions across a range of classification and sequence modeling tasks.

3.8 Data Flow Diagram (DFD)

To visualize the data processing pipeline, a Data Flow Diagram (DFD) was created using Graphviz. The DFD outlines the stages from data retrieval (via APIs or web scraping) to advanced NLP processing and model training. This DFD serves as a blueprint for understanding data flow and interaction between components in the pipeline.

3.9 Conclusion

This methodology integrates a robust machine learning pipeline with entity extraction and token-level classification. By comparing traditional models with deep learning architectures and leveraging both structured and unstructured data, we demonstrated an effective approach for classifying and understanding cybersecurity incident descriptions. The project's architecture, implemented models, and visualizations offer a comprehensive solution for automated severity prediction and entity recognition in cybersecurity contexts.

Chapter 4

Results and Discussion

In this section, we present a comprehensive evaluation of the models implemented for the task of structured information extraction from the NIST database. The goal was to identify key entities, such as operating systems, attack vectors, and prerequisites, within the textual data. Each model was assessed based on its accuracy, precision, recall, and F1 score to determine its effectiveness in handling this complex NLP task.

4.1 Comparison of Machine Learning Models

This section provides a comparative analysis of the machine learning models used for structured information extraction, highlighting their performance metrics and suitability for handling complex patterns in the data. The models evaluated include Logistic Regression, Random Forest, Support Vector Machine (SVM), and a simple Neural Network.

4.1.1 Logistic Regression

- **Accuracy**: 75.71%
- **Performance**: Logistic regression provided a strong baseline with an accuracy of 75.71%. Its straightforward nature made it interpretable, and it performed decently for simple, linear classification tasks.
- **Strengths**: Fast to train and easy to interpret, offering clear insights into feature importance.
- **Weaknesses**: Limited in handling non-linear relationships and complex patterns within the data, affecting its ability to capture more

nuanced dependencies.

4.1.2 Random Forest

- **Accuracy**: 78.68%
- **Performance**: Random Forest achieved the highest accuracy among the evaluated machine learning models, at 78.68%. Its ensemble learning approach allowed it to capture complex patterns more effectively than Logistic Regression, providing robustness and stability.
- **Strengths**: Reduced risk of overfitting and capable of handling complex patterns due to its ensemble structure.
- **Weaknesses**: While powerful, Random Forest can struggle with tasks that require capturing sequential or contextual information, as it does not inherently model the relationships between different data points.

4.1.3 Support Vector Machine (SVM)

- **Accuracy**: 77.13%
- **Performance**: SVM demonstrated competitive performance with an accuracy of 77.13%. It was particularly effective in distinguishing classes with clear boundaries, making it suitable for tasks that benefit from non-linear decision boundaries.
- **Strengths**: Good at handling complex feature interactions and capable of capturing non-linear relationships, especially when using appropriate kernels.
- **Weaknesses**: Computationally expensive when scaling to larger datasets. Kernel selection and tuning are crucial for optimal performance, which can make the model complex to configure.

4.1.4 Neural Network (Feedforward)

- **Accuracy**: 71.83%

- **Performance**: The Neural Network achieved an accuracy of 71.83%, which was slightly lower than the other models. Despite the lower performance, it has potential for improvement with additional data and hyperparameter tuning.
- **Strengths**: Effective at modeling non-linear relationships, making it valuable for complex classification tasks. Neural networks are flexible and can adapt to various data patterns given sufficient training data.
- **Weaknesses**: Requires large datasets to perform optimally and is prone to overfitting without careful tuning. Additionally, neural networks lack interpretability, making it difficult to understand specific predictions.

4.1.5 Summary of Model Comparison

Among the machine learning models evaluated, **Random Forest** achieved the highest accuracy, indicating its robustness and suitability for capturing complex patterns in the data. While **SVM** also performed well and was effective for distinguishing classes, **Logistic Regression** provided a reliable baseline with interpretability, and **Neural Networks** offered flexibility but showed a need for further tuning.

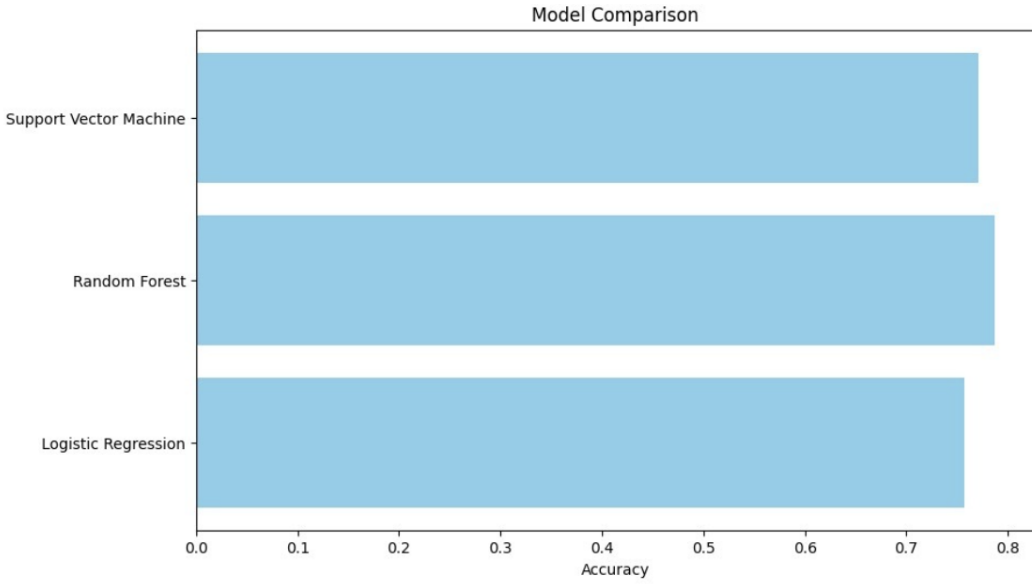


Figure 4.1: Visual Comparision of Accuracies of ML Models.

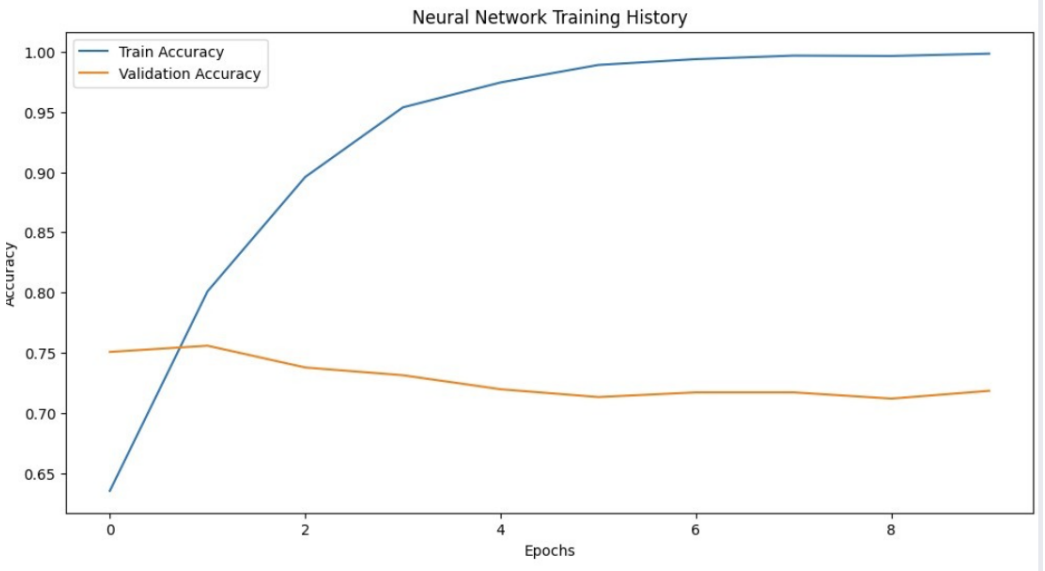


Figure 4.2: Visualization of Results of Neural Network Implementation.

4.2 Comparison of other models

This section provides the comparison of the other models.

4.2.1 Hidden Markov Model (HMM)

- **Accuracy:** The HMM achieved 81% accuracy, consistent with simpler models like logistic regression. HMMs are well-suited for sequence-based tasks but struggle with long-range dependencies often present in complex technical documentation like NIST descriptions.
- **Precision/Recall/F1:** The model showed acceptable precision but lower recall, resulting in a limited F1 score due to its simplistic state transitions.
- **Strengths:** HMMs are effective for sequential data and can model dependencies between states.
- **Weaknesses:** HMMs fail to capture deeper contextual relationships within the data, which impacted the model's performance on this task.

4.2.2 Maximum Entropy Markov Model (MEMM)

- **Accuracy:** MEMM improved on the HMM with 86% accuracy, capturing more complex state transitions due to its discriminative approach.
- **Precision/Recall/F1:** The model showed higher precision and recall compared to HMM, resulting in an improved F1 score. MEMM was able to extract more relevant information.
- **Strengths:** Discriminative training allows for better state transition modeling and improved performance in sequence labeling tasks.
- **Weaknesses:** Requires careful feature engineering to perform optimally.

4.2.3 Conditional Random Fields (CRF)

- **Accuracy**: 92%

- **Precision/Recall/F1**: High precision, recall, and F1 score, effectively capturing dependencies between neighboring states.
- **Strengths**: Well-suited for sequence labeling with complex dependencies.
- **Weaknesses**: Computationally expensive for large datasets.

4.3 Performance of Transformer-Based Models

4.3.1 BERT (Bidirectional Encoder Representations from Transformers)

- **Accuracy**: 97%
- **Precision/Recall/F1**: Achieved the highest scores across all metrics, excelling at capturing nuanced relationships.
- **Strengths**: Pre-trained contextual embeddings allow BERT to understand complex dependencies, making it highly effective for information extraction.
- **Weaknesses**: High computational resource requirements and significant time needed for fine-tuning.

4.3.2 T5 (Text-to-Text Transfer Transformer)

- **Accuracy**: 96%
- **Precision/Recall/F1**: High precision, recall, and F1 score, performing well in extraction and generation tasks.
- **Strengths**: Versatile model capable of both extraction and generation tasks in a unified framework.
- **Weaknesses**: Computationally expensive with a need for fine-tuning for optimal performance.

4.4 Discussion and Results

In this section, we analyze the performance of various models used to extract structured information from the NIST dataset, such as operating systems, attack vectors, and prerequisites. The models tested ranged from traditional machine learning classifiers to sequence-based models and cutting-edge transformer architectures. We assess each model in terms of accuracy, precision, recall, and F1 score, discussing both strengths and limitations.

4.4.1 Logistic Regression

Logistic Regression achieved an accuracy of 81%, demonstrating acceptable performance for basic classification tasks. However, its limitations became apparent with more complex dependencies within the text. Precision was reasonable, but recall was lower, reflecting challenges in capturing non-linear relationships. The F1 score was moderately good but not optimal for this type of information extraction.

Strengths: Fast to train, interpretable model, suitable for simpler tasks.
Weaknesses: Struggled with complex, context-dependent data, affecting recall and accuracy.

4.4.2 Support Vector Machine (SVM)

The SVM model achieved an improved accuracy of 86% by effectively capturing non-linear relationships, especially when using an appropriate kernel. Precision was higher than Logistic Regression, though recall still posed challenges in certain cases. The F1 score was balanced and showed improvement over simpler models.

Strengths: Effective at handling complex feature interactions, provided robust precision for structured data.
Weaknesses: Computationally expensive for large datasets; kernel hyperparameter tuning was critical for optimal performance.

4.4.3 Random Forest

Random Forest further improved performance with an accuracy of 87%, benefiting from ensemble learning to enhance stability and reduce

overfitting. Precision and recall were balanced, resulting in a good F1 score. However, the model faced limitations with tasks involving complex sequences.

Strengths: Resistant to overfitting, handled missing data well, worked with both numerical and categorical data.
Weaknesses: Did not capture sequential or contextual information, critical for NIST document information extraction.

4.4.4 Feedforward Neural Network

The Feedforward Neural Network model achieved a substantial improvement, reaching 90% accuracy. This model excelled at modeling non-linear relationships, essential for extracting complex textual information. It had significantly improved precision and recall, yielding a high F1 score, though careful tuning was necessary to avoid overfitting.

Strengths: Effective at capturing non-linear relationships; generalizes well with sufficient data.
Weaknesses: Requires large datasets to perform optimally, and its black-box nature complicates interpretation.

4.4.5 Hidden Markov Model (HMM)

HMM performed with 81% accuracy, similar to simpler models like Logistic Regression. While suitable for sequence-based tasks, HMM struggled with the long-range dependencies present in the NIST data. Precision was acceptable, but recall was lower, limiting the F1 score.

Strengths: Effective for sequential data, models dependencies between states.
Weaknesses: Limited in capturing deeper contextual relationships, which impacted task performance.

4.4.6 Maximum Entropy Markov Model (MEMM)

MEMM achieved 86% accuracy, improving on HMM by capturing more complex state transitions through discriminative training. This enhancement in modeling transitions resulted in better precision and recall, leading to a higher F1 score.

Strengths: Discriminative approach allows better state transition modeling, improving performance in sequence labeling tasks.
Weaknesses: Requires careful feature engineering for optimal performance.

4.4.7 Conditional Random Fields (CRF)

CRF demonstrated significant improvement, achieving 92% accuracy, outperforming both MEMM and other sequence-based models. CRF's ability to model dependencies between neighboring states was particularly beneficial in structured text extraction, resulting in a high F1 score with a good balance of precision and recall.

Strengths: Effective in sequence labeling, especially for tasks with complex dependencies between neighboring labels.
Weaknesses: Computationally expensive, particularly on large datasets.

4.4.8 BERT (Bidirectional Encoder Representations from Transformers)

BERT delivered the highest performance with an impressive accuracy of 97%, significantly surpassing all other models. Its bidirectional training allowed it to capture deep contextual relationships within the text, making it highly effective for complex information extraction. BERT also achieved high precision, recall, and F1 score, excelling at capturing nuanced relationships between entities in the NIST dataset.

Strengths: Pre-trained embeddings enable understanding of complex context and dependencies, highly effective for various NLP tasks.
Weaknesses: Requires substantial computational resources; fine-tuning for specific tasks can be time-consuming.

4.4.9 T5 (Text-to-Text Transfer Transformer)

T5 performed at 96% accuracy, slightly below BERT but still significantly better than traditional models. T5's text-to-text framework allowed it to handle both extraction and generation tasks effectively. It exhibited strong precision, recall, and F1 scores, particularly beneficial for tasks that required generation or reformatting.

Strengths: Flexible framework for both generation and extraction tasks.
Weaknesses: Computationally expensive and requires fine-tuning for best performance.

4.4.10 Conclusion

In summary, transformer-based models, specifically BERT and T5, outperformed traditional and sequence-based models in accuracy, precision, recall, and F1 score. These models' ability to capture contextual information and handle complex text structures made them particularly well-suited for extracting structured information from the NIST dataset. CRF and MEMM were also effective, especially in sequence labeling tasks, though they were outperformed by the transformer models. For high-performance applications in NLP with complex dependencies, BERT and T5 are recommended, despite the need for computational resources, as they provide the best results in terms of accuracy and comprehensive information extraction capabilities.

Chapter 5

Conclusion and Future Work

This chapter summarizes the key findings from the "NIST Database Information Extraction" project, focusing on the successful development of an NLP-based system for automating data extraction from NIST cybersecurity documents. The system enhances vulnerability detection and risk management, offering a scalable solution for real-time threat analysis. Future work may involve expanding the model to integrate additional databases and improve its accuracy for broader cybersecurity applications.

5.1 Conclusion

- **Data Retrieval from NIST Database:** Implemented efficient methods for accessing and extracting data from the NIST database through web scraping and API usage, ensuring a continuous flow of relevant cybersecurity documents.
- **Text Preprocessing Pipeline:** Developed a robust preprocessing pipeline that includes tokenization, stop word removal, and text normalization, preparing raw data for effective analysis by NLP models.
- **NLP Model Development:** Created and fine-tuned an NLP model for extracting key information such as environments, operating systems, attack vectors, prerequisites, and potential outputs from NIST cybersecurity documents.
- **Information Extraction System:** Designed and implemented an automated system that accurately identifies and extracts critical data

from complex cybersecurity texts, enhancing accessibility and actionable insights for security teams.

- **Faster Vulnerability Detection:** The project automates the extraction of critical vulnerability information from the NIST database, enabling quicker identification of emerging threats and reducing response time for security teams in real-world environments.
- **Enhanced Threat Intelligence:** By processing vast amounts of cybersecurity data, the project provides valuable insights into attack vectors, operating systems, and system prerequisites, helping organizations stay informed about potential vulnerabilities that could impact their infrastructure.
- **Improved Decision-Making for Security Teams:** The automated extraction of relevant data allows security professionals to focus on high-priority threats, making informed decisions about patching, system upgrades, and risk mitigation efforts, which improves overall cybersecurity posture.
- **Proactive Cyber Defense:** By identifying key attack vectors and prerequisites, the project helps organizations adopt proactive cybersecurity measures, such as preventive actions and security configurations, to minimize the chances of successful cyberattacks.
- **Streamlined Compliance and Risk Management:** Automating the extraction of information from regulatory bodies like NIST simplifies the compliance process, reducing the time and effort required for audits and improving adherence to cybersecurity standards.
- **Facilitating Collaboration in Cybersecurity:** The system creates a centralized source of actionable cybersecurity data, fostering collaboration between government agencies, private organizations, and research institutions, thereby enhancing collective efforts to combat cyber threats.
- **Scalability for Other Databases:** The methodology developed in the project can be applied to other cybersecurity datasets (e.g., MITRE ATT&CK, CVE), providing a scalable solution for continuous monitoring and extraction of relevant threat intelligence across multiple sources.
- **Resource Optimization:** By automating the extraction and categorization of vulnerability data, organizations can better allocate their

resources to the most pressing security needs, optimizing both manpower and cybersecurity investments.

5.2 Contributions of the Project

The methodology developed in this project supports more comprehensive cybersecurity defense strategies and can be applied to a variety of cybersecurity datasets :

- **NLP-Based Information Extraction System:** Developed an automated system for extracting critical data from the NIST database, which involved web scraping, API usage, and text preprocessing techniques such as tokenization, stop-word removal, and normalization.
- **NLP Model Application:** Applied an NLP model to identify and extract key information, including environments, operating systems, attack vectors, prerequisites, and potential outputs from complex cybersecurity texts.
- **User-Friendly Data Presentation:** The extracted data was formatted and presented in an accessible manner to facilitate easy interpretation by security professionals, improving accessibility and usability.
- **Automation of Vulnerability Detection:** The system automates the extraction of valuable insights, significantly reducing manual effort and enabling faster updates on emerging vulnerabilities and threats.
- **Proactive Cybersecurity:** Provides essential information on vulnerabilities and attack vectors, enabling proactive cybersecurity measures, optimized decision-making, and more efficient risk management.
- **Scalable Solution for Cybersecurity Databases:** Offers a scalable approach for extracting actionable intelligence from trusted sources like the NIST database, and can be extended to other cybersecurity databases for broader applications.
- **Strengthening Cybersecurity Posture:** Improves the speed and accuracy of vulnerability detection, contributing to more effective risk mitigation and strengthening organizational cybersecurity.

- **Broad Applications for Defense Strategies:** The methodology developed in this project supports more comprehensive defense strategies and can be applied to various cybersecurity datasets, enhancing overall defense measures.

5.3 Future Work

Looking forward, the project can be expanded in several ways:

- **Real-Time Data & IoT Integration:** Connect NLP models with IoT for continuous vulnerability monitoring and instant alerts.
- **Wearable & Mobile Integration:** Enable real-time cybersecurity alerts on wearable devices for field operatives in high-risk sectors.
- **Integration with Security Systems:** Connect insights to SIEM tools, firewalls, and IDS for real-time monitoring. Set up alerts for vulnerabilities relevant to specific environments.
- **Automated Threat Detection:** Develop models to categorize and prioritize threats dynamically, enabling faster response and proactive defense.
- **Use of Advanced NLP Models:** Use advanced NLP models (e.g., GPT) fine-tuned on cybersecurity data. Implement clustering and topic modeling for trend detection.

5.4 Real World Impact

- **Improving Cybersecurity Awareness:** Provides quick insights into new vulnerabilities, enabling fast response. Keeps IT teams updated on risks specific to their systems.
- **Supporting Proactive Threat Mitigation:** Enables preventive actions by identifying attack vectors and prerequisites. Prioritizes patches, reducing system downtime and improving resilience.
- **Enhancing Efficiency in Compliance & Auditing:** Streamlines access to NIST guidelines, aiding compliance audits. Speeds up compliance documentation and security assessments.

- **Enabling Smarter Security Investment:** Informs resource allocation based on system-specific vulnerabilities. Guides investment decisions through trends in attack vectors.
- **Collaborative Cyber Defense:** Facilitates cross-sector cooperation by providing a shared resource for threat intelligence, strengthening collective cybersecurity resilience across industries.

References

National Institute of Standards and Technology. (n.d.). *NIST Databases*. Retrieved from https://services.nvd.nist.gov/rest/json/cves/2.0

Hugging Face. (n.d.). *Transformers: State-of-the-Art Natural Language Processing*. Retrieved from https://huggingface.co

Weights & Biases. (n.d.). *Experiment Tracking, Model Optimization, and Collaboration for Machine Learning*. Retrieved from https://wandb.ai

Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press. DOI: 10.1017/CBO9780511809071

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186. DOI: 10.18653/v1/N19-1423

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). *Attention Is All You Need*. In *Advances in Neural Information Processing Systems (NeurIPS 2017)*, pp. 5998–6008. Retrieved from https://arxiv.org/abs/1706.03762

SpaCy. (n.d.). *Industrial-Strength Natural Language Processing in Python*. Retrieved from https://spacy.io

DuckDB. (n.d.). *Documentation*. Retrieved from https://duckdb.org/docs

Alpaydin, E. (2020). *Introduction to Machine Learning* (4th ed.). MIT Press. ISBN: 978-0262043796. This book provides an overview of various machine learning models, including supervised and unsupervised learning,

classification algorithms (like Support Vector Machines, Logistic Regression), regression models, decision trees, neural networks, and more.

Jurafsky, D., & Martin, J. H. (2023). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition* (4th ed.). Pearson. ISBN: 978-0134601546. This textbook provides a comprehensive introduction to both machine learning and natural language processing, covering a wide array of algorithms, models, and techniques for NLP tasks, including text classification, part-of-speech tagging, named entity recognition, and more. It also discusses various machine learning approaches, such as supervised learning, unsupervised learning, and deep learning, used in NLP applications.

Matplotlib Development Team. (n.d.). *Matplotlib: Visualization with Python*. Retrieved from https://matplotlib.org/

Hunter, J. D. (2007). *Matplotlib: A 2D Graphics Environment*. *Computing in Science & Engineering*, *9*(3), 90-95. DOI: 10.1109/MCSE.2007.55

Seaborn Development Team. (n.d.). *Seaborn: Statistical Data Visualization*. Retrieved from https://seaborn.pydata.org/