**Developer Documentation:**

**CVE to Apache Commit Mapping System**

Contents

[**1.** **Project Overview** 2](#_Toc175471497)

[**2. Setup and Installation** 2](#_Toc175471498)

[2.1 Prerequisites 2](#_Toc175471499)

[2.2 Project Setup 2](#_Toc175471500)

[2.3 Running the CLI 3](#_Toc175471501)

[**3. Code Structure and Explanation** 4](#_Toc175471502)

[4. System Design and Architecture 5](#_Toc175471503)

[4.1 Component Overview 5](#_Toc175471504)

[4.2 Data Flow Diagram 5](#_Toc175471505)

[4.3 System Component Diagram 6](#_Toc175471506)

[4.4 Sequence Diagram 6](#_Toc175471507)

[**5. Detailed Functionality** 7](#_Toc175471508)

[5.1 Data Loading 7](#_Toc175471509)

[5.2 Data Preprocessing 7](#_Toc175471510)

[5.3 Cosine Similarity Calculation 7](#_Toc175471511)

[5.4 CVE Data Pull (NIST & MITRE) 7](#_Toc175471512)

[5.5 Output Generation 8](#_Toc175471513)

[**6. Extending the Project** 8](#_Toc175471514)

[6.1 Adding New Data Sources 8](#_Toc175471515)

[6.2 Improving Matching Accuracy 8](#_Toc175471516)

[**7. Future Enhancements** 9](#_Toc175471517)

[7.1 Evaluation Metrics for System Performance 9](#_Toc175471518)

[7.2 Web Interface 9](#_Toc175471519)

[**8. Conclusion** 9](#_Toc175471520)

# **Project Overview**

The **CVE to GitHub Repository Apache Commit Mapping System** is a tool designed to map **Common Vulnerabilities and Exposures (CVE)** to commit messages in the **Apache repository**. The system uses a **semantic similarity model** (SentenceTransformer) to determine which commits are most relevant to a given CVE description by calculating **cosine similarity** between the CVE descriptions and commit messages and ID’s.

The main goal of this project is to help developers and security researchers quickly identify which code changes (commits) are associated with specific vulnerabilities (CVEs).

# **2. Setup and Installation**

## 2.1 Prerequisites

Ensure that the following dependencies are installed:

* **Python 3.7 or later**: You can download Python [here](https://www.python.org/downloads/).
* **pip**: Package manager for installing Python packages.

## 2.2 Project Setup

1. Clone the GitHub repository:

```bash

git clone https://github.com/yourusername/cve-commit-mapping.git

cd cve-commit-mapping

```

1. Create and activate a virtual environment (recommended):

```bash

python -m venv venv

```

source venv/bin/activate # For Linux/Mac

venv\Scripts\activate # For Windows

1. Install the project dependencies:

```bash

pip install -r requirements.txt

```

## 2.3 Running the CLI

You can run the main **CLI** as follows:

```bash

python cli.py --output\_file output.csv

```

The system will prompt for a **CVE ID**, perform the mapping, and output the results to the specified CSV file.

# **3. Code Structure and Explanation**

The project is organized into several key files and folders, each handling specific responsibilities within the system.

project-root/

│

├── cli.py # Main CLI interface for the project

├── requirements.txt # Python package dependencies

├── datasets/ # Contains example data files (e.g., Apache commit data, CVE data)

├── source code files/ # Source code folder for core logic

│ ├── data\_loader.py # Handles data loading from CSV files

│ ├── cve\_mapping.py # Main logic for mapping CVEs to commits using cosine similarity

│ ├── apache\_commit.py # Handles Apache commit-specific data processing

│ └── mitre\_nist\_api.py # Handles CVE data fetching from MITRE/NIST APIs

└── docs/ # Documentation folder

├── usage\_guide.md # Usage instructions for the CLI

└── Developer Documentation # Detailed system design and architecture

# 4. System Design and Architecture

The system follows a modular architecture where different components handle different tasks like data loading, text processing, similarity computation, and output generation. Additionally, it pulls fresh CVE data from MITRE and NIST **on a monthly basis** to ensure the CVE database is up to date. Below is a comprehensive explanation of the design and architecture.

## 4.1 Component Overview

1. **CLI (Command-Line Interface)**:
   * The main entry point for the user. It takes the CVE ID as input and outputs the mapped CVE-commit pairs to a CSV file.
2. **Data Loader**:
   * Loads data from MITRE and NIST CSV files for CVE data and from Apache repositories for commit data. This component is responsible for data extraction, cleaning, and preprocessing.
3. **Cosine Similarity Model**:
   * This is the core of the system. It uses **SentenceTransformer** to convert textual data (CVE descriptions and commit messages) into vector embeddings and then calculates the **cosine similarity** between these embeddings.
4. **CVE Data Pull (Monthly)**:

* Fetches fresh CVE data from **MITRE** and **NIST** APIs every month to ensure the system is always up to date.

1. **Output Generator**:
   * Generates a CSV file containing the mapped CVE-commit pairs along with similarity scores and other relevant information like **commit SHA** or **commit ID**.

## 4.2 Data Flow Diagram

A **Data Flow Diagram** (DFD) illustrates the flow of data through the system. Here’s a breakdown:

* **Input**: User inputs a CVE ID.
* **Data Loading**: The system loads CVE data from MITRE/NIST and commit data from Apache.
* **Preprocessing**: The raw data is cleaned and processed.
* **Cosine Similarity Calculation**: The system computes the similarity between CVE descriptions and commit messages.
* **Output**: The system generates a CSV file containing the mappings.

## 4.3 System Component Diagram

A diagram explains how the different components of the system interact with each other:

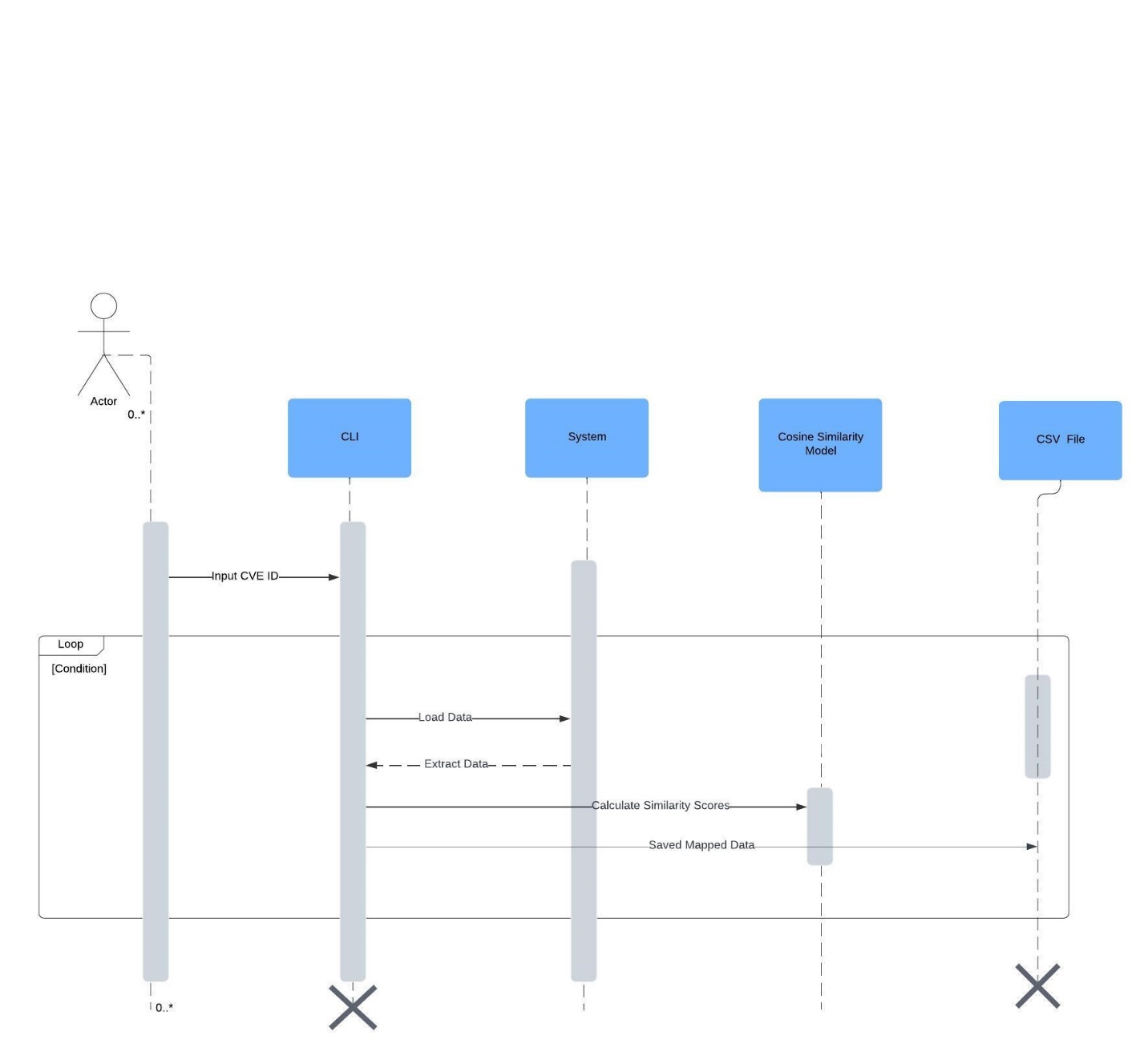
* **CLI**: The user interacts with the CLI.
* **Data Loader**: Loads the necessary data.
* **Cosine Similarity Model**: Handles the similarity computation.
* **Output Generator**: Writes the final output.

## 4.4 Sequence Diagram

A **Sequence Diagram** shows the step-by-step interactions between components from the time the user inputs a CVE ID to the generation of the final CSV file.

Steps:

1. User inputs CVE ID via CLI.
2. The system loads the required data.
3. The system processes the data (extracts relevant parts).
4. The cosine similarity model calculates the similarity scores.
5. The final mapped data is saved to a CSV file.



# **5. Detailed Functionality**

## 5.1 Data Loading

* **MITRE and NIST CVE Data**: Data is loaded from CSV files that contain CVE descriptions from the **MITRE** and **NIST** databases.
* **Apache Commit Data**: This includes commit messages and commit SHAs from the Apache repository.

The system uses **Pandas** to load CSV files containing **CVE data** (from MITRE and NIST) and **Apache commit data**. The load\_csv() function reads the CSV files, handles any missing data, and skips invalid rows.

## 5.2 Data Preprocessing

Once the data is loaded, it undergoes preprocessing to:

* Extract the **CVE IDs** from commit messages.
* Normalize the text by removing unnecessary characters and splitting the **CVE descriptions** and **commit messages** into useful fields.
* Clean and normalize the text for semantic similarity analysis.

## 5.3 Cosine Similarity Calculation

The **SentenceTransformer** model is used to convert both the CVE descriptions and commit messages into vector embeddings. The **cosine similarity** is calculated between these embeddings, which produces a similarity score between 0 and 1:

* A score closer to 1 means the texts are highly similar.
* A score closer to 0 means the texts are dissimilar.

This score is used to map the CVEs to commits. The similarity score threshold (e.g., 0.5) can be adjusted to control how strict the mapping is.

## 5.4 CVE Data Pull (NIST & MITRE)

The system includes scripts that run **monthly** to automatically pull new CVE data from the **MITRE** and **NIST** APIs. This ensures that the system is always working with the most up-to-date vulnerability information. The process involves:

* Fetching data via API calls.
* Storing the updated data in CSV files.
* Integrating the new data into the mapping process.

## 5.5 Output Generation

The system generates a **CSV file** containing:

* **CVE ID**: The ID of the CVE.
* **Commit SHA**: The unique identifier of the commit (if available).
* **Commit Message**: The commit message text.
* **Similarity Score**: The calculated cosine similarity score between the CVE description and the commit message.

# **6. Extending the Project**

## 6.1 Adding New Data Sources

If you want to expand the project to include additional data sources (e.g., more repositories or CVE databases), you can:

* Add new data loader functions to fetch or parse new datasets.
* Extend the preprocessing and mapping logic to handle the new datasets.

## 6.2 Improving Matching Accuracy

To improve the accuracy of the mappings, you can:

* Experiment with different pre-trained models for **semantic similarity**.
* Tune the **threshold** used in cosine similarity to better balance precision and recall.

# **7. Future Enhancements**

## 7.1 Evaluation Metrics for System Performance

An important enhancement is the introduction of **evaluation metrics** to assess the performance of the mapping system. The evaluation metrics will include:

* **Precision**: Measures the proportion of correct CVE-commit mappings.
* **Recall**: Measures the proportion of relevant CVE-commit mappings retrieved.
* **F1 Score**: Balances precision and recall.

We will develop a separate evaluation script to compute these metrics using a **test dataset** that contains ground truth mappings of CVEs to commits.

## 7.2 Web Interface

A future enhancement could be building a **web interface** to make the system more accessible for non-technical users. The interface could provide search functionality for CVEs and visualize the mappings.

# **8. Conclusion**

The **CVE to GitHub Repository Apache Commit Mapping System** is a powerful tool that links CVEs to relevant code changes using semantic similarity. This documentation provides a deep dive into the system architecture, the inner workings of the key components, and how developers can extend and enhance the project in the future.

This modular design makes it easy for developers to understand the structure, extend the system, and contribute to its ongoing development.