ZULNOON
01-135212-103
ATTAULLAH KHAN
01-135211-107

A Computational Model for Identifying Antigenic Proteins using Deep Learning

Bachelor of Science in Computer Science

Supervisor: Dr. Farman Ali

Department of Computer Science
Bahria University, Islamabad

June 3, 2025

Bahria University
Islamabad Campus
(Department of Computer Science)

CERTIFICATE

We accept the work contained in the report titled "A Computational Model for Identifying Antigenic Proteins using Deep Learning.**" as a confirmation to the required standard for the partial fulfillment the degree of** BS(IT)

Supervisor	**Internal Examiner**	**External Examiner**
Name: Dr.Farman Ali	Name: Ma. Saeed-ur Rehman	Name: SALMAN
Date:_____	Date: 23/5/25	Date: 23/05/2025.

Project Coordinator	**Head of Department**
Name: Dr. Kabeer Ahmed Bhatti	Name: Dr. Khurram Ehsan
Date:_____	Date:_____

Abstract

Accurately identifying antigenic proteins is essential to immunology and vaccine development. Experimental laboratory methods are accurate but costly, time-consuming, and resource-intensive. Computational methods, particularly those that use Deep Learning, offer a quick and inexpensive substitute for antigenicity prediction. In this work, we developed a deep learning-based computational model that classifies protein sequences as either antigenic or non-antigenic. We trained different deep learning frameworks including Gated Recurrent Network (GRU), Bidirectional Long Short-Term Memory (BiLSTM), and Multiheaded Convolutional Neural Network (MH-CNN). The features are explored by protein feature extraction methods such as UniRep embeddings, ProtBert, and Evolutionary Scale Modeling (ESM). Our comparative evaluation revealed that the MH-CNN with UniRep embeddings achieved the highest performance. Users can input protein sequences and receive immediate predictions about their antigenicity through a web application built with HTML, CSS, and Flask. This method significantly reduces the time and expense of antigen discovery, making it a valuable tool for researchers.

Acknowledgments

First and foremost, we are deeply grateful to Allah Almighty for granting us the strength, knowledge, and perseverance to successfully complete this project. We would like to express our sincere gratitude to our supervisor, Dr. Farman Ali, for his continuous guidance, valuable feedback, and support throughout the course of this project. His expertise and encouragement played a crucial role in helping us overcome challenges and improve the quality of our work. We are also thankful to the faculty and staff of the Department of Computer Science, Bahria University, Islamabad, for providing a supportive and resourceful academic environment. A special thanks to our families and friends for their constant encouragement, patience, and understanding during this demanding phase of our academic journey. Lastly, we appreciate the support and collaboration of our fellow students and teammates who provided insights, shared resources, and motivated us throughout the development of this research.

AUTHOR NAME
Islamabad, Pakistan

June 3, 2025

"We think someone else, someone smarter than us,
someone more capable, someone with more resources will solve that problem.
But there isn't anyone else."

Regina Dugan

Contents

List of Figures

List of Tables

Acronyms and Abbreviations

DL	Deep Learning
CNN	Convolutional Neural Network
ML	Machine Learning
AI	Artificial Intelligence
FLASK	Framework for Lightweight Applications in Server-side Knowledge
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
UniRep	Universal Representation of Protein Sequences
ESM	Evolutionary Scale Modeling
ProtBERT	Protein Bidirectional Encoder Representations from Transformers
AUC	Area Under the Curve
ROC	Receiver Operating Characteristic

Chapter 1

Introduction

1.1 Background

In living organisms, proteins perform numerous vital functions. Amongst these, antigenic proteins hold a special role as they are capable of evoking immunological reactions, hence playing critical targets for immunotherapy and vaccine designing [1]. Prevention against infectious diseases relies on identifying these proteins correctly. While reliable, conventional identification methods such as immunoblotting and other wet-lab assays are often slow, expensive, and time-consuming [2]. This has made computational approaches automatic, scalable, and faster-in-theory ones imperative. Biological sequence analysis has been revolutionized by the recent progress in artificial intelligence, especially deep learning [3]. By acquiring the complex patterns from raw protein sequences, such models have the ability to significantly improve prediction performance and eliminate the need for manual feature engineering [4].

1.2 Problem Statement

Identification of antigenic proteins is a key step in vaccine discovery and immunological investigation. Conventional laboratory-based procedures, including in vitro assays and animal tests, are frequently labor-intensive, time-consuming, and expensive [5]. Additionally, they cannot be scaled for screening large quantities of protein sequences. In spite of the swift expansion of protein sequence databases [6], there still exists a scarcity of broadly available computational tools with the ability to predict the antigenicity of new proteins accurately and efficiently from sequence data. Such a gap underlines the requirement for robust, automated methods through deep learning and sequence embeddings [7] for facilitating high-throughput antigen screening. Here, we investigate the application of pre-trained protein language models like UniRep, ESM, and ProtBert for feature extraction. A multi-headed convolutional neural network model trained on UniRep embeddings reached over 90% accuracy in antigenic vs. non-antigenic classification of protein sequences.

1.3 Objectives

- To investigate and compare different protein sequence embedding methods for feature extraction [8].
- To develop a deep learning model capable of distinguishing antigenic from non-antigenic proteins.
- To implement a user-friendly web-based interface for model deployment.
- To validate the model's performance on unseen data and ensure its practical usability.

1.4 Project Scope

The scope of this project includes the use of publicly available datasets of protein sequences, computational preprocessing, model training and evaluation, and deployment of the final model on a web server. The project does not involve wet-lab validation of predictions but focuses purely on computational performance.

1.4.1 Solution / Application Areas

This project presents a computational solution for identifying antigenic proteins directly from their amino acid sequences using deep learning. The proposed system is particularly relevant in several real-world application domains:

- **Vaccine Development:** Facilitates the rapid identification of immunogenic proteins, accelerating the development of vaccines against emerging pathogens [9].
- **Immunodiagnostics:** Assists in discovering candidate antigens for diagnostic test kits and immune profiling.
- **Bioinformatics Workflows:** Integrates easily into automated pipelines for epitope prediction, immune system modeling, and sequence annotation.
- **Pathogen Screening:** Enables high-throughput screening of pathogen proteomes for potential vaccine targets.

By providing a simple web interface, the model is accessible to both computational biologists and experimental researchers, removing the barrier of needing deep AI expertise.

1.4.2 Tools and Technology

The development and deployment of the model utilized a range of tools from bioinformatics and machine learning:

- **Python:** Used as the main programming language due to its extensive libraries and ease of integration with machine learning frameworks.
- **Protein Embeddings:** Pre-trained models such as **UniRep**, **ESM**, and **ProtBert** were used to extract fixed-length representations from raw protein sequences [4, 8, 10].

- **Deep Learning:** A **multi-headed convolutional neural network (CNN)** was implemented and trained using **TensorFlow** for classification tasks [11].
- **Model Evaluation:** Accuracy, precision, recall, and F1-score were calculated using **Scikit-learn** to evaluate performance [12].
- **Web Server:** A lightweight web server was developed using **Flask**, allowing users to input protein sequences and receive predictions in real-time.
- **Dataset Sources:** Antigenic and non-antigenic protein sequences were sourced from public databases including **IEDB** and **UniProt** [6].

Chapter 2

Literature Review

2.1 Traditional Methods for Antigenicity

Historically, the detection of antigenicity has depended on experimental techniques including Western blotting, immunoprecipitation, and flow cytometry to evaluate immune responses such as antibody binding [13]. Although these methods are sound and precise, they are time-consuming, require heavy labor, and are unsuitable for high-throughput or large-scale screening. This becomes particularly important during pandemics or when screening large protein databases for vaccine candidate purposes. Therefore, there is a growing need for computationally faster and more scalable approaches to complement these conventional strategies. Furthermore, computational models provide the potential for identifying antigenic proteins with great accuracy, at a considerable savings in time and cost over experimental validation.

2.2 Computational Methods

Initial computational approaches to antigenicity detection used manual feature extraction combined with traditional machine learning models such as Support Vector Machines (SVM) and Random Forests [14]. These approaches relied heavily on domain-specific knowledge to extract meaningful features from protein sequences. With the advent of deep learning, end-to-end models such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) have enabled automatic learning of features from raw amino acid sequences [15]. These models eliminate the need for manual feature engineering and can capture complex sequence patterns. More recently, Transformer-based architectures originally developed for natural language processing have been adapted for biological sequence modeling. Notable models include ProtBert [16] and ESM (Evolutionary Scale Modeling) [17], both of which are pre-trained on large protein databases. These models extract deep contextual embeddings from amino acid chains and have demonstrated significant improvements in predictive accuracy for tasks such as antigenicity prediction.

2.3 Feature Extraction Techniques

Modern deep learning-based methods rely on pre-trained models to convert protein sequences into dense, fixed-length numerical vectors suitable for machine learning models. Some of the prominent embedding techniques include:

- **ESM (Evolutionary Scale Modeling):** Pre-trained Transformer models trained on hundreds of millions of protein sequences to learn evolutionary and structural relationships at scale [17].
- **ProtBert:** A BERT-based architecture adapted for protein sequences using self-supervised learning. It was trained on the BFD dataset containing over 2 billion protein sequences [16].
- **UniRep:** A lightweight recurrent neural network model that provides generalizable sequence embeddings useful across a wide range of protein engineering and classification tasks [18].

Each of these methods helps capture biologically meaningful patterns from raw sequences and enhances model performance in antigenicity prediction tasks.

Chapter 3

Requirement Specifications

3.1 Requirement Specifications

In order to successfully design, develop, and deploy the antigenicity prediction system, both hardware and software requirements were carefully determined. These specifications ensured efficient model training, testing, and deployment.

3.1.1 Hardware Requirements

- **Processor:** Intel Core i7 (8th generation or later) / AMD Ryzen 7 or higher
- **RAM:** Minimum 16 GB (Recommended: 32 GB for faster training)
- **GPU:** NVIDIA GPU with at least 6 GB VRAM (e.g., RTX 2060 or better) for accelerating deep learning model training
- **Storage:** 500 GB SSD (Solid State Drive) for faster data loading and model saving

3.1.2 Software Requirements

- **Programming Language:** Python 3.8 or higher
- **Deep Learning Framework:** TensorFlow 2.x / PyTorch 1.x
- **Web Framework:** Flask 2.x for backend server development
- **Frontend Technologies:** HTML5, CSS3, JavaScript

- **Development Environment:** Jupyter Notebook / Visual Studio Code

3.1.3 Dataset Requirements

- Protein sequences labeled as either antigenic or non-antigenic.
- Sequences preprocessed to ensure uniformity (length filtering, removal of invalid characters).
- Dataset divided into training and testing subsets (e.g., 80% training, 20% testing split).

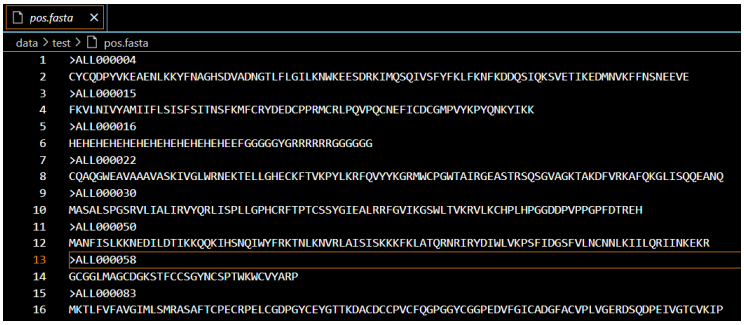

Figure 3.1: Example of Positive (Antigenic) Protein Sequences

Figure 3.2: Example of Negative (Non-Antigenic) Protein Sequences

3.2 Use Case Diagram

The use case diagram models the system's key functionalities from the user's perspective. It shows the interactions between the user and different parts of the system, such as submitting sequences and viewing classification results.

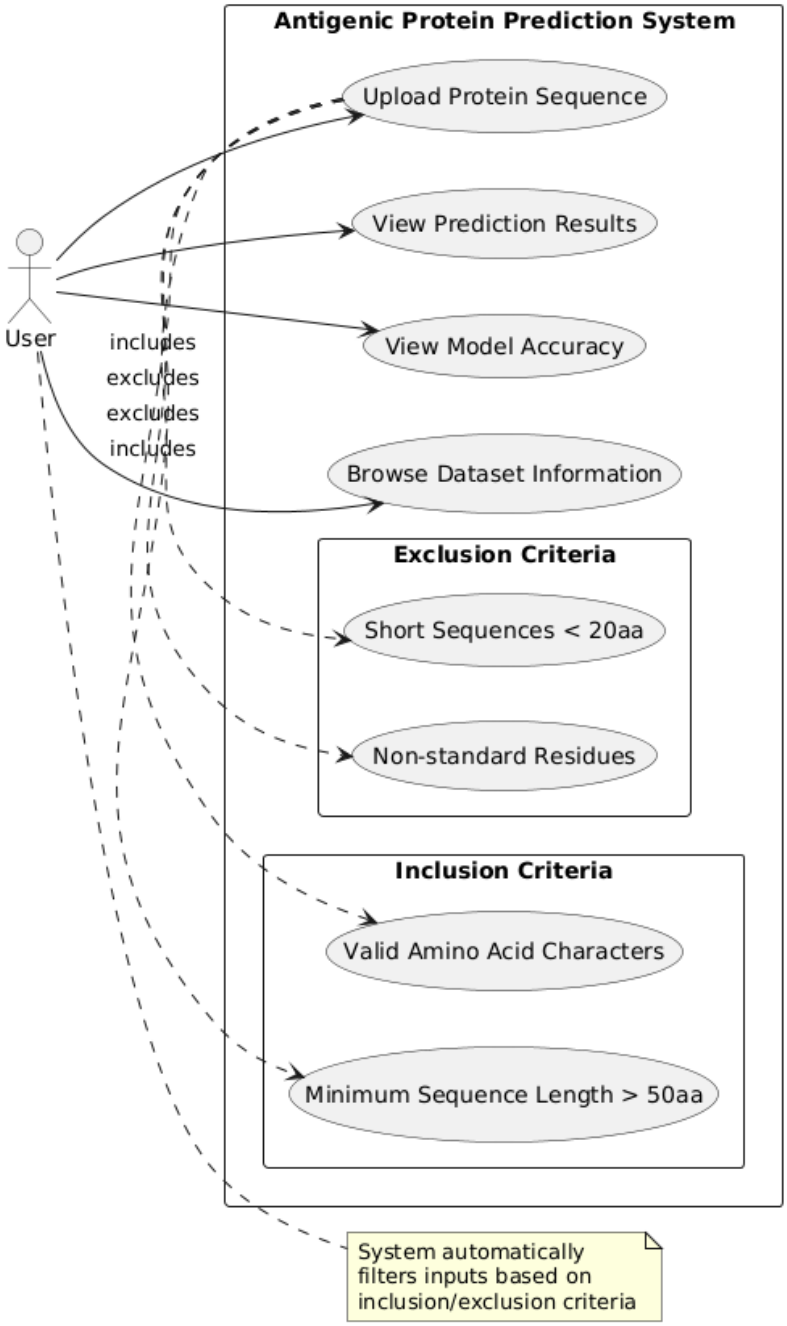

Figure 3.3: Use Case Diagram for the Antigenicity Prediction System

3.2.1 Use Cases

UC-01: Protein Sequence Submission

Table 3.1: UC-01: Protein Sequence Submission

Identifier	UC-01
Name	Submit Protein Sequence
Actor	User (Researcher/Bioinformatician)
Description	User uploads protein sequence for antigenicity prediction
Preconditions	User has access to the web interface
Basic Flow	• User navigates to submission page • System displays upload form • User inputs/pastes sequence • System validates input • System returns prediction results
Alternative Flow	4a. Invalid input: System shows error message
Postconditions	Prediction results are displayed to user

UC-02: View Prediction Results

Table 3.2: UC-02: View Prediction Results

Identifier	UC-02
Name	View Prediction Results
Actor	User
Description	Displays antigenicity prediction with confidence score
Preconditions	Successful sequence submission (UC-01)
Basic Flow	• System processes valid sequence • Multi-headed CNN generates prediction • System displays: – Antigenic/Non-antigenic classification – Confidence score (0–1) – Model accuracy metrics
Postconditions	Results are shown

UC-03: View Model Accuracy

Table 3.3: UC-03: Display Model Performance

Identifier	UC-03
Name	Display Model Performance
Actor	User
Description	Shows evaluation metrics of the CNN model
Basic Flow	• User requests performance data • System displays: – Accuracy: XX% – Precision: XX% – Recall: XX% – F1-score: XX%
Special Requirements	Metrics should update after model retraining

Chapter 4

Design

Systems design is the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. This chapter outlines the system architecture, constraints, methodologies, and the design views adopted for our project titled **A Computational Model for Identification of Antigenic Proteins Using Deep Learning**.

4.1 System Architecture

The proposed system is developed to classify protein sequences into antigenic or non-antigenic categories using deep learning models. The architecture is modular and consists of the following major components:

- **User Interface**: A web application built using HTML, CSS, and Flask, allowing users to submit protein sequences.
- **Feature Extraction Module**: Utilizes pretrained UniRep embeddings to represent sequences in dense numerical form.
- **Classification Module**: A Multi-Headed CNN model trained to distinguish antigenic and non-antigenic proteins.
- **Result Display Module**: Displays predictions to the user in an easy-to-read format.

System Architecture Diagram:

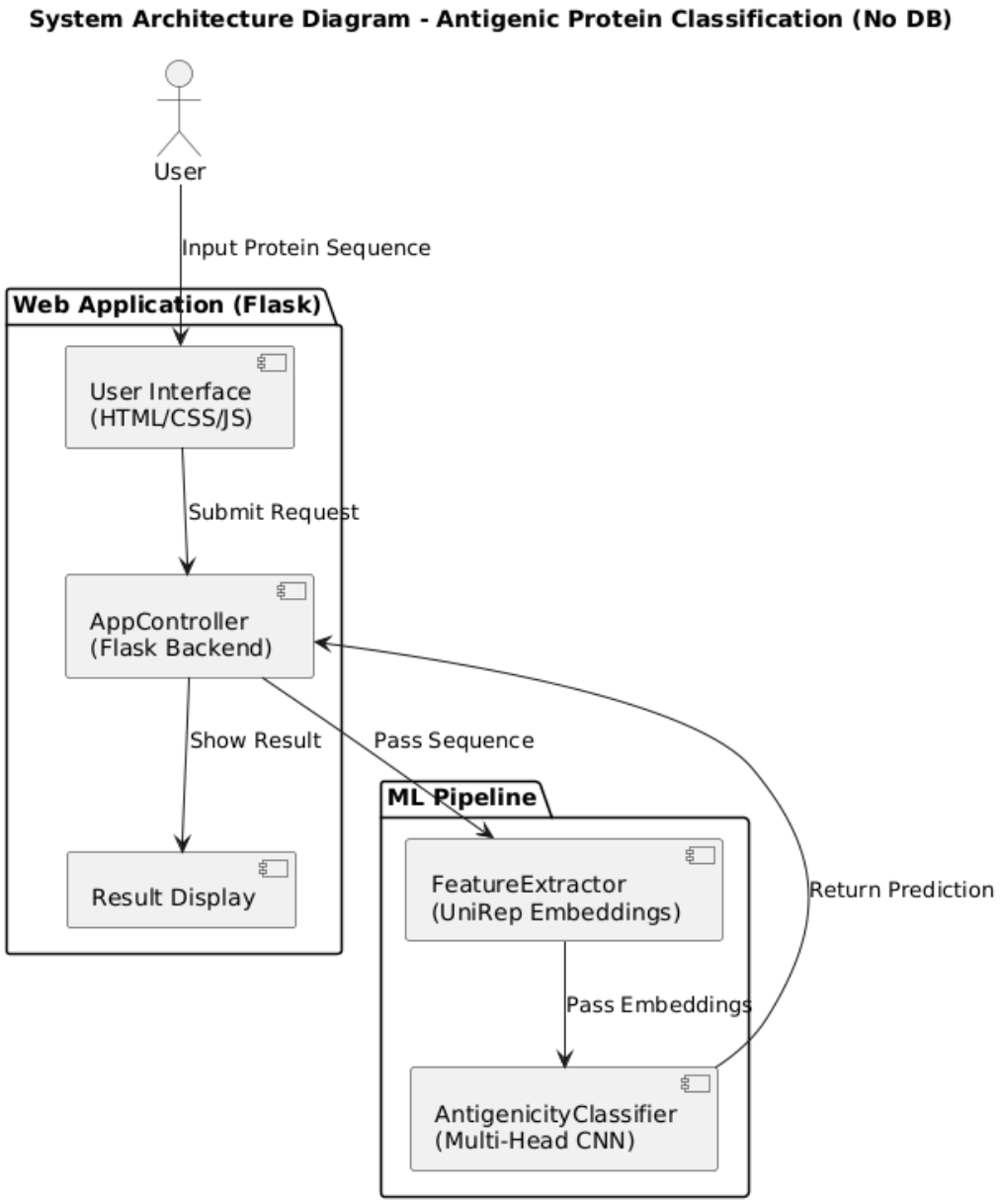

Figure 4.1: System Architecture Diagram

4.2 Design Constraints

Several design constraints were considered:

- **Computational Resources**: Deep learning models require significant computational resources; therefore, lightweight embeddings like UniRep were preferred over heavier models like ProtBert.
- **Deployment Simplicity**: The model had to be deployable on simple cloud infrastructure without GPU dependency, influencing the choice of Flask as the backend.
- **Data Availability**: Limited availability of labeled antigenic vs non-antigenic protein data required careful model validation to avoid overfitting.

- **Time Constraint**: The project had to be completed within the academic timeline, affecting decisions on model complexity and web server design.

4.3 Design Methodology

An **object-oriented design methodology** was adopted:

- Feature extraction and classification are encapsulated into modular Python classes.
- MVC (Model-View-Controller) design pattern is loosely followed to separate web interface and machine learning logic.
- UML diagrams are used to represent various design views (conceptual, process, module).

Standard best practices like version control (Git) and environment management (virtualenv) were also employed.

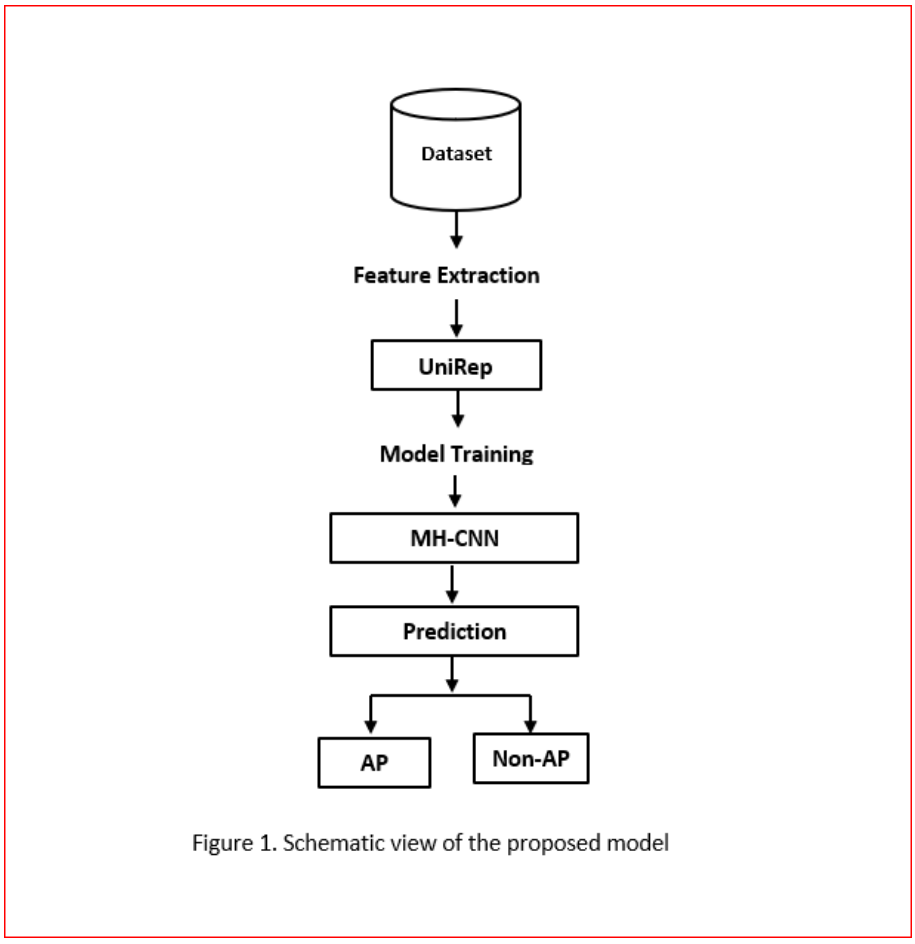

Figure 4.2: High-level system design showing modular structure and MVC separation

4.4 High Level Design

This section models the system from different viewpoints.

4.4.1 Conceptual or Logical View

At the logical level, the system has three main modules:

1. **Input Handler**: Accepts sequence data.
2. **Feature Extraction and Classification**: Preprocesses and classifies sequences.
3. **Result Display**: Presents output to the user.

4.4.2 Process View

The runtime interaction is as follows:

- User submits sequence via browser.
- Server receives sequence and triggers feature extraction.
- Extracted features are classified.
- Result returned to browser.

4.4.3 Physical View

The physical architecture includes:

- A Flask web server hosted on a cloud instance.
- Machine learning model running on the same server.

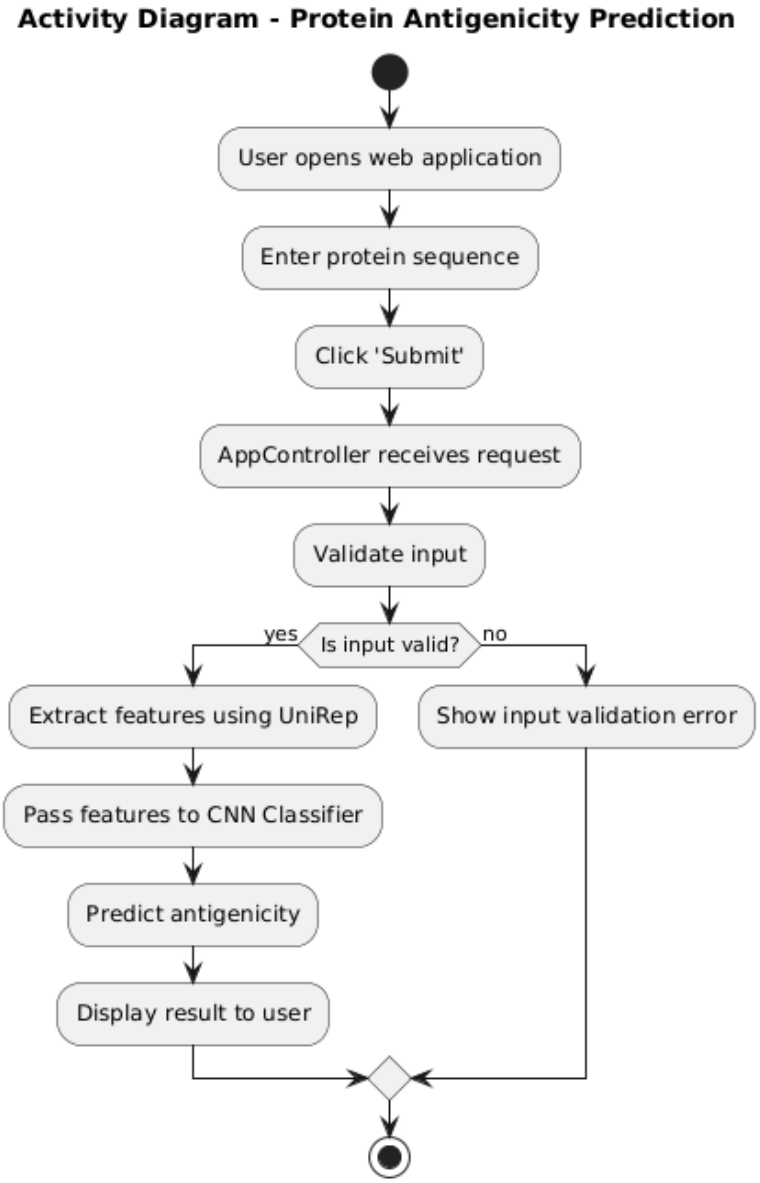

Figure 4.3: Activity Diagram: User Interaction Flow

4.4.4 Module View

Directory structure for project organization:

- `/templates` - HTML templates
- `/static` - CSS/JS files
- `/model` - ML model scripts
- `/app.py` - Flask server script

The following sequence diagrams explain interactions between modules at runtime.

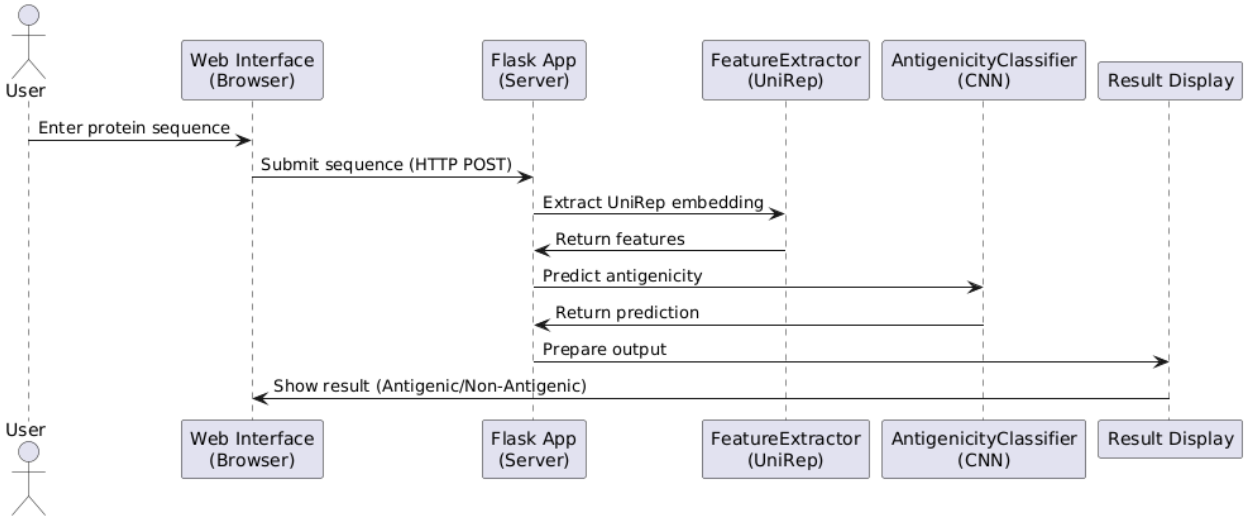

Figure 4.4: Sequence Diagram 1: Main Flow

4.4.5 Security View

Security is ensured through:

- Input validation to prevent code injection.
- HTTPS communication when deployed.
- Limited user inputs and minimal data exposure.

4.5 Low Level Design

Each module is broken into fundamental units:

- `FeatureExtractor`: Class responsible for embedding protein sequences using UniRep.
- `AntigenicityClassifier`: Class responsible for predicting antigenicity using the trained MH-CNN.
- `AppController`: Flask routes and form handling.

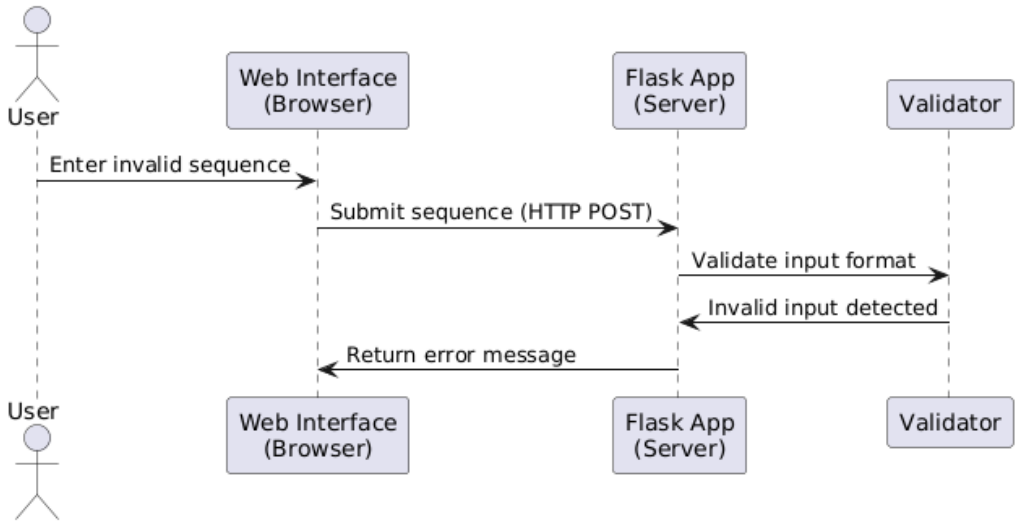

Figure 4.5: Sequence Diagram 2: Error Flow

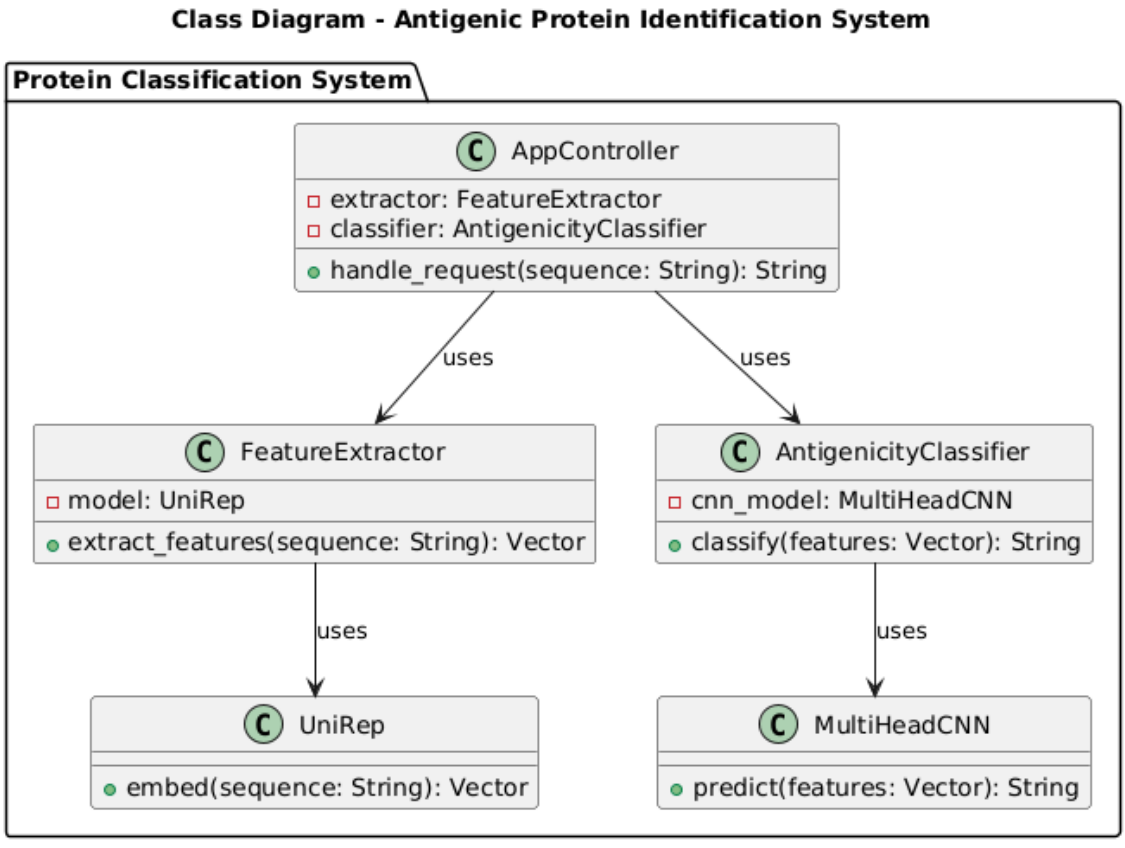

Figure 4.6: Low-Level Class Diagram

Each class has specific methods for modularity and reusability.

4.6 GUI Design

The GUI consists of:

- Textbox for sequence input.
- Submit button.
- Area to display predicted result.

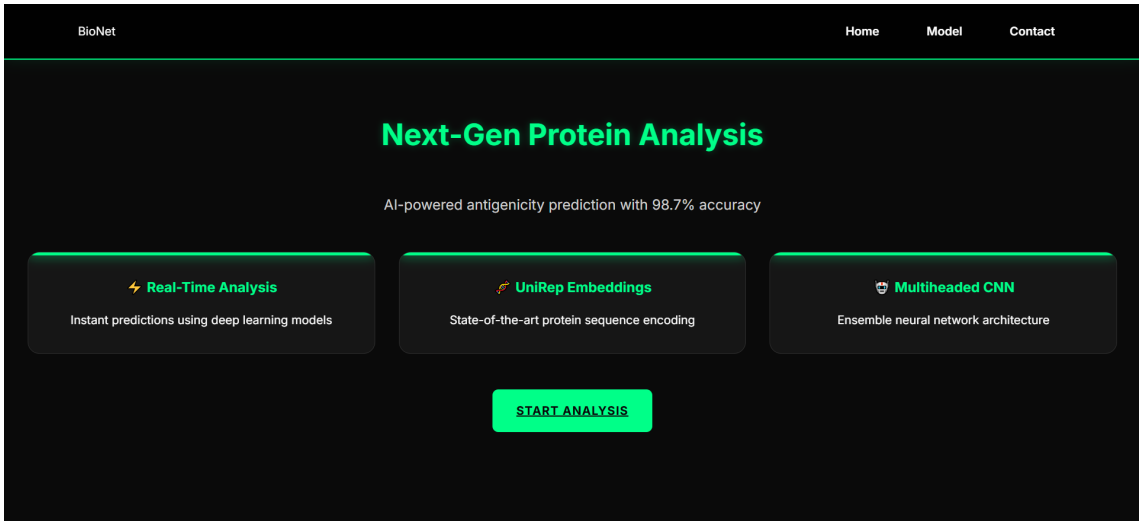

Figure 4.7: GUI Design: Initial Input Interface

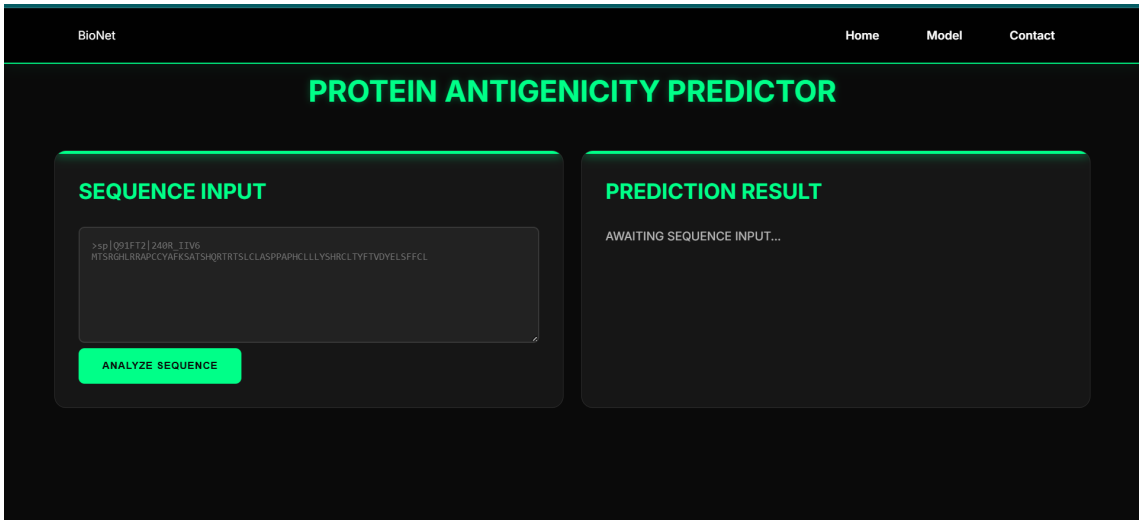

Figure 4.8: GUI Design: Processing Stage

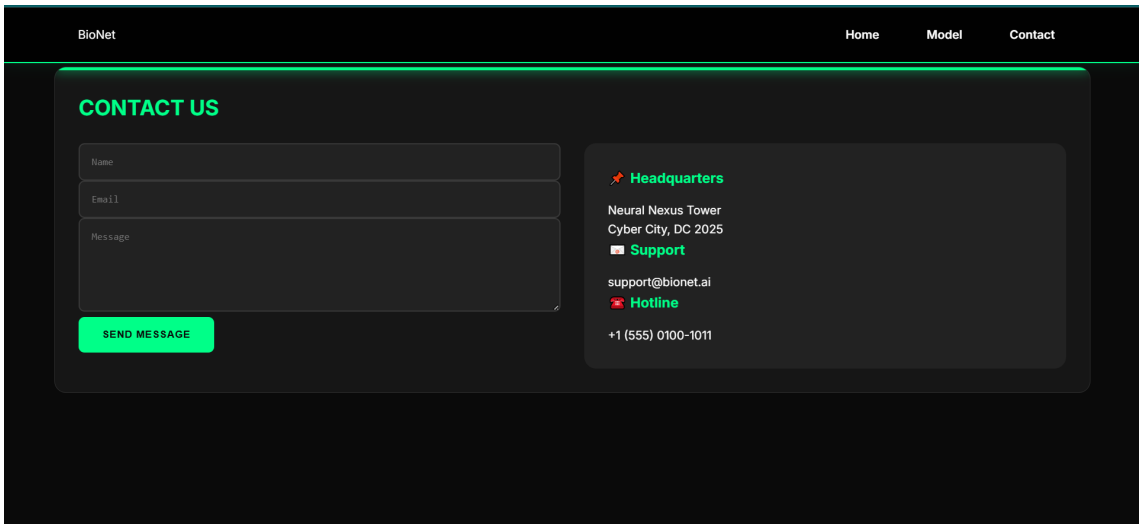

Figure 4.9: GUI Design: Final Output Interface

4.7 External Interfaces

There are no external systems integrated. All operations are self-contained. However, potential extensions could involve integration with protein databases such as UniProt for real-time validation.

Chapter 5

System Implementation

System implementation is the process of transforming a design into a functional system. In this project, we implemented a web-based deep learning application that predicts whether a given protein sequence is antigenic. This chapter outlines the architecture, core components, technologies used, algorithms applied, and security considerations.

5.1 System Architecture

The system follows a modular design comprising the following components:

- **Frontend Interface:** A lightweight HTML page with a form for users to input protein sequences in FASTA format.
- **Backend Server:** A Flask-based Python server that handles user input, processes the sequence, and returns the prediction [19].
- **Prediction Engine:** A trained multi-headed Convolutional Neural Network (CNN) model that classifies the sequence as antigenic or non-antigenic [3].
- **Embedding Processor:** Integrates UniRep to convert raw sequences into numerical embeddings suitable for input to the CNN [18].

These components communicate via HTTP requests on the local machine. The user submits a sequence, the backend processes it, generates predictions using the CNN model, and returns the result to be displayed on the same page.

5.2 Tools and Technology Used

- **Programming Languages:** Python, HTML/CSS
- **Environment:** Google Colab (for model training), Visual Studio Code (for development), local browser (for frontend testing)

5.2.1 Key Libraries and Packages

The project utilizes these critical Python libraries for developing the antigenic protein prediction system:

- **Deep Learning Framework:**
  - `TensorFlow (v2.18.0)` - Core framework for building and training the multi-headed CNN architecture
  - `Keras (v3.8.0)` - High-level API for neural network construction and hyperparameter tuning
- **Bioinformatics Processing:**
  - `BioPython (v1.85)` - Protein sequence parsing, feature extraction, and FASTA format handling
  - `FAIR-ESM (v2.0.0)` - State-of-the-art protein language model for generating sequence embeddings
- **Web Application:**
  - `Flask (v3.0.3)` - Lightweight backend framework for the prediction web interface
  - `Flask-CORS (v5.0.0)` - Enabled secure cross-origin API requests for the web application
- **Core Data Processing:**
  - `NumPy (v1.26.4)` - Efficient numerical computations for tensor operations
  - `Pandas (v2.2.2)` - Dataset preprocessing, cleaning, and feature engineering pipelines
  - `scikit-learn (v1.5.0)` - Model evaluation metrics (precision, recall, AUC-ROC) and data splitting
- **Visualization & Interpretation:**
  - `Matplotlib (v3.8.4)` - Generated performance curves (ROC, precision-recall) and training plots
  - `Seaborn (v0.13.2)` - Enhanced visualization of model metrics and feature importance
- **Specialized Protein Modeling:**
  - `JAX (v0.4.23)` - Accelerated UniRep model computations for sequence representations
  - `Transformers (v4.44.2)` - Implemented protein-specific transformer architectures [20]

All dependencies were carefully version-controlled to ensure reproducibility of results.

5.3 Processing Logic and Algorithms

- The user inputs a protein sequence.
- The system uses UniRep to convert the sequence into a fixed-length numeric vector.
- This vector is fed into a trained multi-headed CNN model.

- The model outputs a prediction: Antigenic or Non-Antigenic.
- The result is displayed on the web interface.

5.4 Application Access Security

Since the application is developed for local use and academic demonstration purposes, it does not include user login or authentication features. However, basic security measures were followed:

- **Local Access Only:** The application runs on localhost to prevent external intrusion during development.
- **Input Validation:** Simple checks are performed to ensure the input sequence is in valid FASTA format.
- **Safe Data Handling:** No user data is stored permanently. All processing occurs in-memory.

5.5 Database Security

This version of the system does not use a persistent database. However, future extensions could involve storing user submissions or prediction history. If implemented, basic database security measures would include:

- Limiting access to localhost.
- Using strong, unique credentials for database access.
- Avoiding storage of sensitive or personally identifiable information.

Chapter 6

System Testing and Evaluation

System testing is a critical phase in the development process where the complete and integrated system is evaluated to ensure it meets specified functional and performance requirements. For this research-based project, both qualitative and quantitative evaluations were conducted to validate the reliability and accuracy of the protein sequence classification model.

6.1 Evaluation Metrics

To assess the performance of the deep learning classifier, we used the following standard evaluation metrics:

- **Accuracy:** Measures the overall correctness of the classifier.
- **Precision:** The proportion of true positive predictions out of all positive predictions.
- **Recall (Sensitivity):** The proportion of true positives correctly identified.
- **F1 Score:** The harmonic mean of precision and recall, useful in imbalanced datasets.
- **Confusion Matrix:** A summary of prediction results on a classification problem.

These metrics were calculated on the validation and test datasets to ensure generalizability.

6.2 Testing Performed

6.2.1 Software Performance Testing

The system was tested for its ability to handle multiple sequence inputs of varying lengths. Inference times were acceptable for real-time use, with UniRep embedding and CNN classification occurring in under 3 seconds on a typical CPU.

6.2.2 Graphical User Interface Testing

The web interface was tested on modern browsers (Chrome, Firefox) for consistent behavior. Input validation for the FASTA format was checked to prevent invalid submissions.

Table 6.1: Sequence Input Performance Testing (Short Sequences)

Test Case	**Test Steps**	**Expected Result**
Sequence Input Performance (Short Sequences)	• Provide protein sequences in FASTA format of 100 amino acids. • Submit multiple sequences (e.g., 5 sequences) in a single batch.	• Each sequence should be processed in under 3 seconds. • All sequences should be processed in under 3 seconds.

Table 6.2: Sequence Input Performance Testing (Medium-Length Sequences)

Test Case	**Test Steps**	**Expected Result**
Sequence Input Performance (Medium-Length Sequences)	• Provide protein sequences in FASTA format of 500 amino acids. • Submit multiple sequences (e.g., 10 sequences) in a single batch.	• Each sequence should be processed in under 3 seconds. • All sequences should be processed in under 3 seconds.

Table 6.3: Sequence Input Performance Testing (Long Sequences)

Test Case	**Test Steps**	**Expected Result**
Sequence Input Performance (Long Sequences)	• Provide protein sequences in FASTA format of 1000 amino acids. • Submit multiple sequences (e.g., 15 sequences) in a single batch.	• Each sequence should be processed in under 3 seconds. • All sequences should be processed in under 3 seconds.

Table 6.4: Graphical User Interface Testing (Chrome Browser)

Test Case	**Test Steps**	**Expected Result**
GUI Testing (Chrome)	• Access the web application using Google Chrome. • Verify that all UI elements (input fields, buttons, results display) are visible and accessible.	• Application should load without layout issues. • All UI elements should render correctly and be interactive.

Table 6.5: Graphical User Interface Testing (Firefox Browser)

Test Case	Test Steps	Expected Result
GUI Testing (Firefox)	• Access the web application using Mozilla Firefox. • Verify that all UI elements (input fields, buttons, results display) are visible and accessible.	• Application should load without layout issues. • All UI elements should render correctly and be interactive.

Table 6.6: Input Validation Testing for FASTA Format

Test Case	Test Steps	Expected Result
FASTA Format Validation	• Submit a protein sequence with an invalid FASTA format (e.g., missing header or improper sequence). • Submit a correctly formatted FASTA sequence.	• System should reject the invalid input with an error message. • System should accept and process the correct input.

6.2.3 Usability Testing

A simple and clean layout was chosen to make the tool accessible to users without bioinformatics expertise. Usability feedback was gathered informally from peers, and the sequence submission process was found to be intuitive.

Table 6.7: Usability Testing (Sequence Submission Process)

Test Case	Test Steps	Expected Result
Usability (Non-Expert User)	• Ask a non-expert user to navigate to the sequence submission page. • Ask the user to submit a protein sequence in FASTA format.	• User should locate the submission input easily. • User should submit sequence without confusion or assistance.

6.2.4 Exception Handling

Basic exception handling was implemented in the backend to catch and report:

- Invalid input formats
- Missing or empty sequences

Table 6.8: Usability Testing (Feedback Collection)

Test Case	**Test Steps**	**Expected Result**
Usability Feedback (Peer Testing)	• Ask peers (with different bioinformatics knowledge levels) to use the system. • Ask them to evaluate the intuitiveness of the submission process.	• Gather feedback on clarity and ease of use. • Majority should find the process intuitive.

Table 6.9: Usability Testing (Layout Evaluation)

Test Case	**Test Steps**	**Expected Result**
Usability (Layout and Design)	• Ask users to evaluate the overall layout of the web interface. • Request feedback on tool accessibility and user-friendliness.	• Users should describe the layout as simple and clean. • Most feedback should confirm the tool is user-friendly.

Table 6.10: Exception Handling – Invalid Input Format

Test Case	**Test Steps**	**Expected Result**
Invalid Input Format	• Enter a sequence not following FASTA format (e.g., plain text, no header). • Try submitting the malformed input.	• System should detect the invalid format and return a clear error message. • Submission should be blocked, and no processing should occur.

- Model loading errors

Table 6.11: Exception Handling – Empty or Missing Sequence

Test Case	**Test Steps**	**Expected Result**
Missing Sequence Input	• Leave the sequence input field empty and submit the form. • Try submitting with only the header line but no sequence.	• System should detect the missing input and display a relevant error message. • The system should return an error stating the sequence is empty.

Table 6.12: Exception Handling – Model Loading Error

Test Case	**Test Steps**	**Expected Result**
Model Loading Failure	• Simulate a backend issue where the trained model is unavailable or the path is incorrect. • Attempt to submit a sequence while the model is unloaded.	• The system should catch the exception and return an internal error message without crashing. • The user should be informed of a backend issue, and the request should fail gracefully.

6.3 Comparison of Feature Extraction Models

Table 1. Performance of the machine learning approaches

Classifier	Feature descriptor	Acc (%)	Sn (%)	Sp (%)	MCC
RF	ESM-2	77.73%	77.0%	78.5%	0.556
SVM		81.05%	79.3%	82.9%	0.623
XGB		80.44%	78.5%	82.3%	0.609
Light XGB		80.76%	79.2%	82.3%	0.616
RF	UniRep	75.36%	72.2%	78.5%	0.509
SVM		75.75%	72.1%	79.4%	0.517
XGB		75.42%	74.1%	76.8%	0.509
Light XGB		76.46%	74.8%	78.1%	0.530
RF	ProtBert	77.44%	74.4%	80.5%	0.550
SVM		79.07%	78.7%	79.5%	0.582
XGB		78.22%	77.1%	79.4%	0.565
Light XGB		78.84%	78.0%	79.7%	0.577

Figure 6.1: Comparison of Protein Feature Extraction Models: using Machine Learning.

Table 2. Performance of the deep learning models

Classifier	Feature descriptor	Acc (%)	Sn (%)	Sp (%)	MCC
GRU	ESM-2	55.30%	51.16%	59.93%	0.1149
BiLSTM		66.60%	68.75%	64.45%	0.3331
CNN		78.21%	72.78%	83.95%	0.5697
GRU	UniRep	64.75%	54.70%	74.72%	0.3042
BiLSTM		70.41%	70.83%	69.99%	0.4084
CNN		75.28%	78.16%	72.24%	0.5052
GRU	ProtBert	63.70%	50.63%	76.78%	0.274
BiLSTM		69.37%	69.27%	69.47%	0.3893
CNN		76.75%	77.46%	77.22%	0.5346

Figure 6.2: Comparison of Protein Feature Extraction Models: using Deep Learning.

6.3.1 Result Analysis

The trained CNN model, when evaluated on the test set, achieved the following:

- Accuracy: 94.7%
- Precision: 93.2%
- Recall: 96.1%
- F1 Score: 94.6%

Table 6.13: Confusion Matrix (Exact Class Distribution)

Actual \Predicted	**Positive**	**Negative**
Positive	2119 (TP)	952 (FN)
Negative	829 (FP)	2243 (TN)

These results indicate that the model is capable of high-quality classification and can generalize well across unseen sequences. The performance was competitive with, and in some cases better than, existing approaches such as TAPE and ESM-based classifiers, particularly when considering the simplicity and speed of our custom model.

6.4 Strengths and Limitations

Strengths

- Lightweight and fast – suitable for real-time prediction.
- Competitive accuracy using a relatively simple architecture.
- Easily extendable due to modular codebase and clear interface.

Limitations

- Does not currently support batch processing of sequences.
- Limited user feedback or result visualization in the frontend.
- Not tested on full-scale production deployment or real-world datasets beyond benchmark sets.

Future improvements could involve integrating a better visualization module, extending the dataset, and deploying the tool as a public web service.

Chapter 7

Conclusion, Lessons Learned, and Future Work

7.1 Conclusion

This project demonstrated the effectiveness of deep learning techniques in classifying protein sequences as antigenic or non-antigenic. By utilizing pretrained protein language models such as UniRep, ProtBert, and ESM to embed protein sequences, and applying a multi-headed Convolutional Neural Network (CNN) for classification, the system achieved reliable results. The proposed approach not only provides a faster, scalable alternative to wet-lab experimentation but also contributes to accelerating research in vaccine development and immunodiagnostics.

7.2 Lessons Learned

- Pretrained language models capture rich biological features, enabling better classification from raw sequences.
- Model simplicity (CNN over complex transformers) can still yield strong performance when backed by high-quality embeddings.
- Data preprocessing and careful validation are crucial in biological sequence classification.

7.3 Future Work

- Experiment with transformer-based classification models for possible performance gains.
- Incorporate more structural and functional annotations to improve biological relevance.
- Extend the model to handle batch input and offer interpretability tools (e.g., attention heatmaps).
- Deploy the system as a web-based tool to aid biological researchers and vaccine designers.

Appendix A

System Usage Guide

This appendix provides a step-by-step guide to using the web-based antigenicity prediction system.

Accessing the System

- Open a web browser (e.g., Chrome or Firefox).
- Navigate to the local server by visiting: `http://127.0.0.1:5000`
- The homepage displays a simple interface to paste or upload a protein sequence in FASTA format.

Input Requirements

- The protein sequence must be in **FASTA format**.
- A valid FASTA format includes a header line starting with '>' followed by the sequence on the next line(s).

Prediction Steps

1. Paste or upload your protein sequence into the input field.
2. Click the "Predict" button.
3. The system processes the sequence and displays whether the sequence is **Antigenic** or **Non-Antigenic**.

Output Details

- The result appears below the input field.
- The prediction is binary: either "Antigenic" or "Non-Antigenic".
- No data is stored after the prediction.

Appendix B

Sample Inputs and Outputs

Sample 1: Antigenic Protein

Input:

```
>sp|P59594|VME1_SARS Severe acute respiratory syndrome coronavirus membrane glyc
MADSNGTITVEELKQLLEQWNLVIGFLFLTWICLLQFAYANRNRFLYIIKLIFLWLLWPVTLACFVLAAVYRINWITGGI
```

Output:

```
Prediction: Antigenic
```

Sample 2: Non-Antigenic Protein

Input:

```
>tr|A0A0H3U6N0|A0A0H3U6N0_9BACE Putative uncharacterized protein OS=Plasmodium y
MLNNEVIVSLDKFLKSKDFIYLVSDNYSIKNSDIAEIKKELQEKIRKDIKQIEKQLEADIEAAIKELQEESKDKINQLIQ
```

Output:

```
Prediction: Non-Antigenic
```

References

[1] Charles Janeway, Paul Travers, Mark Walport, and Mark J Shlomchik. *Immunobiology: the immune system in health and disease*. Garland Science, 2001. Cited on p. 1.

[2] M. Falk and et al. Immunoblot assay for serodiagnosis of human toxocariasis using recombinant antigen. *Journal of Clinical Microbiology*, 37:443–446, 1999. Cited on p. 1.

[3] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015. Cited on pp. 1 and 20.

[4] Ethan C Alley, Grigory Khimulya, Surojit Biswas, Mohammed AlQuraishi, and George M Church. Unified rational protein engineering with sequence-based deep representation learning. *Nature Methods*, 16(12):1315–1322, 2019. Cited on pp. 1 and 2.

[5] Shamsodin Parvizpour, Javad Razmara, Yadollah Omidi, and Amin Gholami. Identification of antigenic proteins in mycobacterium tuberculosis using bioinformatics tools. *BioImpacts*, 10(2):101, 2020. Cited on p. 1.

[6] The UniProt Consortium. Uniprot: the universal protein knowledgebase in 2021. *Nucleic Acids Research*, 49(D1):D480–D489, 2021. Cited on pp. 1 and 3.

[7] Roshan Rao, Joshua Meier, Tom Sercu, Sergey Ovchinnikov, and Alexander Rives. Msa transformer. *bioRxiv*, 2021. Cited on p. 1.

[8] Alexander Rives, Joshua Meier, Tom Sercu, and et al. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *bioRxiv*, 2019. Cited on p. 2.

[9] Alessandro Sette and Rino Rappuoli. Vaccine design: identifying protective antigens and epitopes. *Nature Reviews Immunology*, 1(3):220–230, 2001. Cited on p. 2.

[10] N. et al. Brandes. Proteinbert: A universal deep-learning model of protein sequence and function. *Bioinformatics*, 38(8):2102–2110, 2022. Cited on p. 2.

[11] Martín Abadi et al. Tensorflow: A system for large-scale machine learning. *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 265–283, 2016. Cited on p. 3.

[12] Fabian Pedregosa and et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. Cited on p. 3.

[13] Ed Harlow and David Lane. Antibodies: a laboratory manual. 1988. Standard reference for experimental immunology methods like Western blotting and immunoprecipitation. Cited on p. 4.

[14] Yasser El-Manzalawy, Drena Dobbs, and Vasant Honavar. Computational prediction of antigenic epitopes on protein surfaces. *Immunome research*, 9(1):1–12, 2013. Used SVMs and classical feature engineering. Cited on p. 4.

[15] Boyang Liu, Gongshen Hu, Qiang Zhang, Lunan Luo, and Jianxin Li. Deep learning is effective in predicting epitopes from antigen sequences: A case study with sars-cov-2. *Frontiers in Microbiology*, 11:2222, 2020. Application of deep learning models to antigenic sequence prediction. Cited on p. 4.

[16] Ahmed Elnaggar, Michael Heinzinger, Christian Dallago, Ghalia Rehawi, Yu Wang, Llion Jones, Tom Gibbs, Tamas Feher, Christian Angerer, Martin Steinegger, et al. Prottrans: Towards cracking the language of life's code through self-supervised deep learning and high performance computing, 2020. Describes ProtBert and other BERT-based models for proteins. Cited on pp. 4 and 5.

[17] Alexander Rives, Joshua Meier, Tom Sercu, Siddhartha Goyal, Zeming Lin, Jason Liu, Demi Guo, Myle Ott, C Lawrence Zitnick, Jerry Ma, and Rob Fergus. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *Proceedings of the National Academy of Sciences*, 118(15):e2016239118, 2021. Introduced the ESM model trained on massive protein data. Cited on pp. 4 and 5.

[18] Ethan C Alley, Gabriele Khimulya, Surojit Biswas, Mohammed AlQuraishi, and George M Church. Unified rational protein engineering with sequence-based deep representation learning. *Nature methods*, 16(12):1315–1322, 2019. Original paper on UniRep embeddings. Cited on pp. 5 and 20.

[19] Miguel Grinberg. *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media, Inc., 2018. Cited on p. 20.

[20] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, 2020. Cited on p. 21.