PROGRAM REPAIR AND RETRIEVAL ON THE PRUTOR PLATFORM - III

A Thesis Submitted

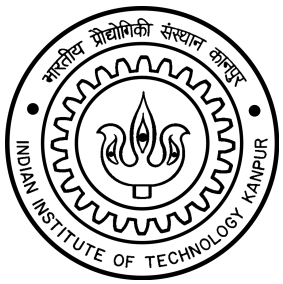
In Partial Fulfillment of the Requirements

For the Degree of M.Tech.

by

Utkarsh Srivastava

21111063



to the

Department of Computer Science and Engineering

Indian Institute of Technology Kanpur

May, 2024

Declaration

This is to certify that at the thesis titled **"PROGRAM REPAIR AND RETRIEVAL ON THE PRUTOR PLATFORM - III"** has been authored by me. It presents the research conducted by me under the supervision of **PROF. AMEY KARKARE AND PROF. PURUSHOTTAM KAR**.

To the best of my knowledge, it is an original work, both in terms of research content and narrative, and has not been submitted elsewhere, in part or in full, for a degree. Further, due credit has been attributed to the relevant state-of-the-art and collaborations (if any) with appropriate citations and acknowledgments, in line with established norms and practices.

Name: Utkarsh Srivastava (21111063)
Program: M.Tech.
Department of Computer Science and Engineering
Indian Institute of Technology Kanpur, Kanpur, 208016.
May, 2024

Declaration (To be submitted at DOAA Office)

I hereby declare that

1. The research work presented in the thesis titled **"PROGRAM REPAIR AND RETRIEVAL ON THE PRUTOR PLATFORM - III"** has been conducted by me under the guidance by my supervisor(s) **PROF. AMEY KARKARE AND PROF. PURUSHOTTAM KAR**.
2. The thesis has been formatted as per Institute guidelines.
3. The content of the thesis (text, illustration, data, plots, pictures etc.) is original and is the outcome of my research work. Any relevant material taken from the open literature has been referred and cited, as per established ethical norms and practices.
4. All collaborations and critiques that have contributed to giving the thesis its final shape have been duly acknowledged and credited.
5. Care has been taken to give due credit to the state-of-the-art in the thesis research area.
6. I fully understand that in case the thesis is found to be unoriginal or plagiarized, the Institute reserves the right to withdraw the thesis from its archive and also revoke the associated degree conferred. Additionally, the Institute also reserves the right to apprise all concerned sections of society of the matter, for their information and necessary action (if any).

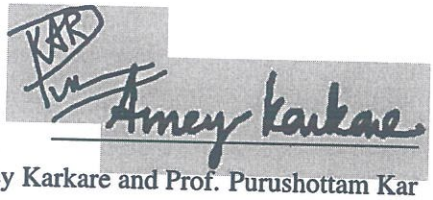
Name: Utkarsh Srivastava
Program: M.Tech.
Department of Computer Science and Engineering
Roll No.: 21111063
Indian Institute of Technology Kanpur, Kanpur, 208016.
May, 2024

Page intentionally left blank

POST GRADUATE OFFICE
THESIS SUBMITTED
ON 17/05/24
I.I.T. KANPUR

Certificate

It is certified that the work contained in the thesis titled **"PROGRAM REPAIR AND RETRIEVAL ON THE PRUTOR PLATFORM - III"** by **UTKARSH SRIVASTAVA** has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.



Prof. Amey Karkare and Prof. Purushottam Kar
Department of Computer Science and Engineering
Indian Institute of Technology Kanpur
Kanpur, 208016.
May, 2024

Page intentionally left blank

Abstract

Name of student: **Utkarsh Srivastava** Roll no: **21111063**
Degree for which submitted: **M.Tech.**
Department: **Department of Computer Science and Engineering**
Thesis title: **PROGRAM REPAIR AND RETRIEVAL ON THE PRUTOR PLATFORM - III**

Name of thesis supervisor: **Prof. Amey Karkare and Prof. Purushottam Kar**
Month and year of thesis submission: **May, 2024**

In recent years, the field of education has experienced a significant surge in the utilization of machine learning and artificial intelligence applications. Particularly, there has been a noticeable increase in the development of AI-assisted solutions within the education sector. One notable example is ESC101, an introductory programming course available to students at IIT Kanpur. Among the tools developed to support instructors teaching this course is PRIORITY.

PRIORITY provides users with access to an online platform where they can find programming problems sourced from previous iterations of the ESC101 course at IIT Kanpur.

Initially, PRIORITY existed as a standalone application requiring users to log in separately to access and copy the required problems for setting up the ESC101 course programming assignments. However, it has now been seamlessly integrated into PRUTOR, eliminating the need for separate login. Additionally, a new functionality has been introduced to log user interactions on PRIORITY, facilitating the creation of a database of user preferences. This data will be utilized to enhance the machine learning algorithms powering PRIORITY, based on user feedback.

PRIORITY operates within a distinct docker container as a server built on NodeJS, offering two endpoints to the PRUTOR application. When users wish to browse problems, they can select relevant labels, generating an API request containing these labels. In response, PRIORITY provides a ranked list of problems from its database. If a user wishes to view a specific problem in detail, including the problem statement, solution code, and solution template, they can request it from the PRIORITY server. Both of these interactions are logged to support the aforementioned improvement process.

Acknowledgments

First and foremost, I want to express my sincerest gratitude to my thesis supervisor Prof. Purushottam Kar and co-supervisor Prof. Amey Karkare, for their constant guidance and support. Without their mentorship and encouragement, it would have been impossible for me to complete this thesis.

Secondly, I want to express my gratitude to Ayush Sahni and Jeet Sarangi, with whom I've had numerous discussions and brainstorming sessions. Their problem solving approach and out-of-the-box ideas helped me to complete this thesis.

I would also like to express my gratitude to all of the professors in the CSE Department for providing me with invaluable technical knowledge throughout my tenure at IIT Kanpur. The academic knowledge and critical thinking that I learned here will always be a crucial aspect in shaping my personality as an engineer as well as a human being.

I would also like to thank all my friends and batchmates who have been an integral part of my academic journey during M.Tech at IIT Kanpur.

Last but not the least, I would like to take this opportunity to express my gratitude and love for my mom, dad, and brothers (Ashish & Umang), whose love, affection and constant motivation guide me like a North-Star everyday.

This thesis was compiled using a template graciously made available by Olivier Commowick `http://olivier.commowick.org/thesis_template.php`. The template was suitably modified to adapt to the requirements of the Indian Institute of Technology Kanpur.

Utkarsh Srivastava
May, 2024

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Contents

In the era of digital revolution, software systems have been at the core of innovation. This technology advancement has significantly changed the way of modern education. One such remarkable feat of software engineering was the development of PRUTOR - a tutoring system platform for teaching introductory programming courses developed at Department of Computer Science & Engineering, IIT Kanpur published in [7].

Indian Institute of Technology, Kanpur offers ESC101 as an introductory programming course to all the fresher undergraduate students using PRUTOR [7]. Setting up questions for such a large number of students needs proper scrutiny and standards to maintain a balance between different batches. To overcome this problem, [8] developed PRIORITY (PRoblem IndicatOr ReposITorY) as a separate web portal where instructors and tutors can access a large repository of question from previous course offerings and get an idea of the difficulty level and concepts asked in different chapters and topics.

Over the course of years, it proved to be a tedious task to log in to PRUTOR and PRIORITY separately and then copy-paste the question from one portal to another. In this thesis, we have developed a new PRIORITY module with added features and integrated it within the PRUTOR platform. This will make it convenient for the problem setters (Instructors and tutors) to access the repository and reuse the questions without having to log-in separately and copy-paste. We have added features for logging user actions and also provided a secure authentication method to keep it secure [10].

The purpose of logging user actions can be attributed to the fact that it will provide a direct correlation between the user search parameters (in our case, problem labels) and the preferred question selected [10]. This relation will serve as a dataset to further improve the label tagging ML algorithm's performance based on user experience. Earlier, it was a simple feedback mechanism in which the user had to manually provide feedback on the quality of search results. In actual deployment, it was observed that users were reluctant to provide feedback to save time. Now, with the advent of the logging mechanism, the user does not need to explicitly spend time. Another benefit of this feature is that it helps the course instructors to easily identify if some user (tutor) has directly taken the question from priority without any modification of the problem statement. This helps to maintain innovation as the course can be run with a fresh set of questions in every offering. The details on logging are present in [10].

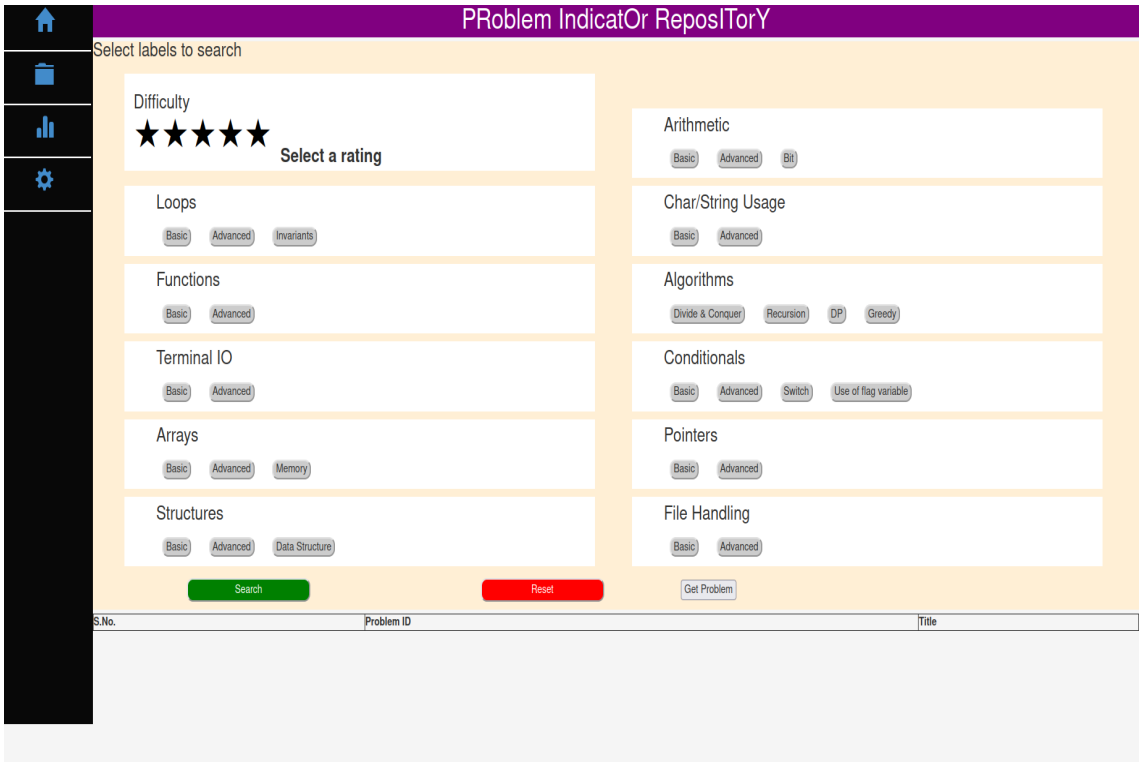


Figure 1.1: PRIORITY Homepage

.

The purpose of providing separate authentication is to serve as an additional security. We know that PRIORITY contains questions from all the previous offerings of the ESC101 course. If any new student gets access to this set, it would hamper the learning and thinking process. To discourage this kind of events, it becomes necessary to shield the set of questions from unauthorized access.

In this thesis, we have developed PRIORITY as a NodeJS module with logging and authentication feature. Thereafter, it is deployed in a docker container to maintain portability. Thereafter, it has been integrated within PRUTOR platform. Now we will briefly describe these implementations in section 1.2.

1.1 Our Contributions

- The previous version of PRIORITY was developed as a separate Web Portal using Angular [9]. It had separate credentials for logging as compared to PRUTOR. Now, it has been developed as a NodeJS module which has been integrated within PRUTOR. Any authorised user can directly access the questions repository without having to spend considerable amount of time in browsing separately. The existing PRUTOR credentials and access control rules define the access scope of PRIORITY. In this case, it has been set to Course instructors and tutors. In the end, PRIORITY has been deployed as a docker container. This makes it a lightweight and portable module which can be added in any existing PRUTOR installations. The complete description of web module of PRIORITY is given in chapter 2.

- Previously, there was a manual feedback option in PRIORITY which allowed the users to

provide rating to the quality of search results [9]. Now, we have added logging functionality for 2 separate events which will automate the previous tasks and also provided added benefits of creating a dataset of user preferences [10]. The first logging event occurs when a user performs search. User selects one or more labels and difficulty rating and gets the results in the form of a list sorted in descending order of relevance. This incident is logged as [Timestamp, User, {Label1, Label2...LabelM, Difficulty}, {Result1, Result2...ResultN}]. Some improvements in automated problem labelling were done in [6] and [12]. Second logging event occurs when a user selects a problem to be moved to PRUTOR. This incident is logged as [Timestamp, User, PRUTOR Problem ID, PRIORITY Problem ID].

- Earlier, PRIORITY had separate log-in credentials for every user [9]. Now, it has been built-up and integrated within PRUTOR. Hence, there would not be any need to create separate log-in credentials for PRIORITY. Also, PRIORITY module has been made secure by creating JSON web token based authentication for each query made from PRUTOR to PRIORITY. This will provide an additional layer of security apart from the usual Access Control List (ACL) feature present in PRUTOR by default [7].

1.2 Challenges

PRIORITY was developed in [9]. With user feedback, it was observed that the users faced some challenges which included switching browsers and simultaneously logging to 2 different portals (PRUTOR and PRIORITY) separately. Moreover, there was no mechanism to collect the data which could be helpful in improving problem labelling algorithms. In order to overcome the above mentioned challenges, a new PRIORITY module was needed to be developed. While developing such a system, we faced numerous challenges which are listed below.

- **Working with Legacy Systems:** PRUTOR was developed in [7]. With the passage of time, technologies have changed significantly. It was a difficult task to understand the architecture and functioning of PRUTOR in detail.

- **Outdated Libraries and Installation:** As we know that PRUTOR was designed to be installed in any flavour of Linux. Since the release of PRUTOR, Linux libraries have been updated numerous times. This created a problem of dependencies while installation. It took a considerable amount of time to rectify those errors and proceed with development of PRIORITY as a module.

- **Understanding Angular Framework:** PRIORITY was initially designed using Angular framework [9]. While integrating it with PRUTOR, it had to be redesigned in NodeJS. This process needed a deep understanding of Angular framework.

1.3 Companion Theses

This thesis has been completed in conjunction with [11] and [10] which are a part of group of companion theses. It is recommended to refer to [10] for Logging Events and their implementation,

which are mentioned briefly in this thesis. Also, it is recommended to refer to [11] for chapter 5. For a better and comprehensive understanding, it is advisable to read it vis-a-vis [10] and [11] as the group of companion theses provide the entire information and research methodologies adopted for completion.

CHAPTER 2

PRIORITY: Web Application

Contents

We have discussed briefly about PRIORITY in chapter 1. In this chapter, we will discuss about it in detail. PRIORITY is a web-module inside PRUTOR application which allows user to browse and import C programming questions for ESC101 course at IIT Kanpur [9]. Since the strength of students is large in ESC101, it becomes essential to provide the problem setters an automated way to setup questions. Moreover, it also helps them to understand the difficulty levels and nature of problems that were asked in previous offerings of this course. This helps to maintain a balance of difficulty level and relevance of concepts in ESC101 offerings over the course of years. Now we will go into the web module implementation in detail and provide architectural explanation as well as the application flow of this module inside PRUTOR.

2.1 Goals

PRIORITY is a web module developed using NodeJS and integrated in PRUTOR. The primary aim of this module is to provide assistance to course instructors and tutors to browse through a large repository of problems that were offered in ESC101 earlier. Initially, it was developed as an independent Web portal using Angular. But, with the advent of time, it was observed that it was causing overhead for users to open both of these web applications separately and then copy-paste the required problems. The solution that we propose in this thesis is to develop PRIORITY from scratch as a NodeJS module and then integrate it within PRUTOR itself, so that it becomes convenient for course administrators to browse through questions and add them to current ESC101 offerings in PRUTOR.

The newly developed PRIORITY module provides added functionality of Event Logging and API authentication which are explained in [10] and chapter 3 respectively.

2.2 Architecture

In this section, we will discuss the architecture of this module in detail.

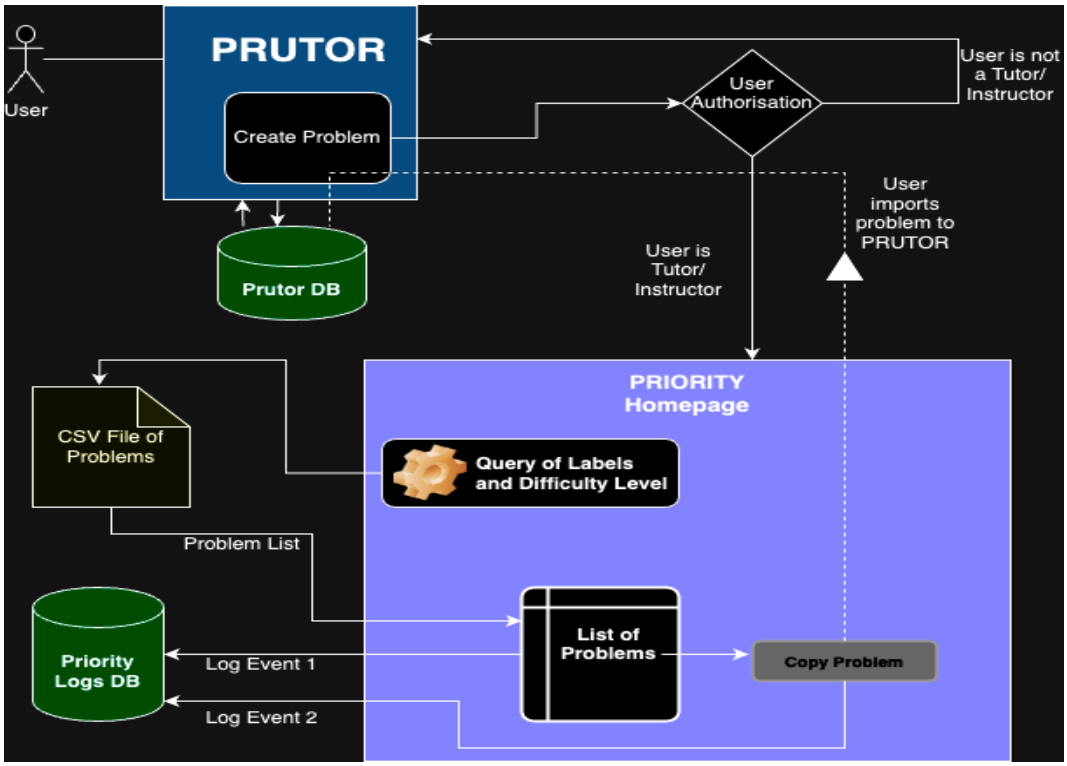


Figure 2.1: Architecture and Flow of PRIORITY.

2.2.1 NodeJS Module

PRIORITY is a module written in NodeJS, a JavaScript runtime environment [4]. As a NodeJS module, PRIORITY likely encapsulates specific functionalities or features that can be imported and used in NodeJS applications. It may include functions, classes, or other resources that provide services related to priority management or other relevant tasks. Adding it as a module in PRUTOR provides the following advantages listed below.

1. **Component Breakdown**: Dividing the application into smaller, modular parts enhances comprehension, maintenance, and expansion. Each module can concentrate on a specific function, enhancing organization and code clarity.

2. **Reuse Potential**: Once created, modules can be utilized across various sections of the application or even in different projects. This saves development time and fosters uniformity across the application.

3. **Scalability Support**: Modular design facilitates scalability by enabling the addition of new features or expansion of existing ones without impacting other areas of the application. This simplifies accommodating growth and adapting to evolving needs.

4. **Testing Convenience**: Modules can be tested independently, streamlining the testing process and enhancing overall code quality. Unit tests can be developed for each module to ensure its isolated functionality.

5. **Functionality Isolation**: Modules encapsulate related functions, aiding in maintaining clear interfaces between different sections of the application. This mitigates the risk of unintended

consequences and facilitates code comprehension.

6. **Structural Organization**: Introducing new features as modules maintains a well-organized and structured codebase. Developers can easily locate and address specific features without navigating through a cumbersome monolithic codebase.

7. **Enhanced Collaboration**: Modular design fosters collaboration among team members by enabling them to work on distinct modules concurrently without disrupting one another. This boosts productivity and accelerates development cycles.

8. **Dependency Handling**: Node.js's module system inherently manages dependencies between modules. This simplifies handling complex dependencies and ensures the application's stability and reliability.

2.2.2 Front End

PRIORITY homepage is built with dot View Engine in NodeJS. The dot view engine, commonly referred to as doT.js, is a lightweight and efficient templating engine for JavaScript. Developed by Laura Doktorova, doT.js provides a simple yet powerful solution for generating dynamic HTML content in Node.js applications [1]. It provides the following features which make it a suitable choice for this thesis work.

1. **Performance**: doT.js is renowned for its speed and efficiency. It compiles templates into highly optimized JavaScript functions, resulting in fast rendering times and improved performance, particularly beneficial for applications with high rendering requirements.

2. **Simplicity**: The syntax of doT.js templates is minimalistic and easy to learn. It follows a straightforward approach, enabling developers to write and understand templates with ease, without unnecessary complexity.

3. **Flexibility**: doT.js offers flexibility in terms of usage. It supports both server-side and client-side rendering, allowing templates to be shared between the server and client, facilitating isomorphic or universal JavaScript applications.

4. **Customization**: While providing a simple default syntax, doT.js also offers options for customization. Developers can extend the templating engine with custom filters, helpers, or logic to meet the specific requirements of their applications.

5. **Security**: doT.js templates support automatic HTML escaping by default, helping to mitigate cross-site scripting (XSS) attacks. This feature ensures that user-generated content or data from external sources is properly sanitized before being rendered, enhancing the security of the application.

6. **Compatibility**: doT.js is compatible with various frameworks and environments commonly used in the Node.js ecosystem. It can be seamlessly integrated into popular frameworks like Express.js or used standalone in any Node.js project.

7. **Community and Documentation**: The doT.js templating engine benefits from an active community of developers and comprehensive documentation. This facilitates access to support, contribution opportunities, and learning resources, enhancing its usability and adoption.

PRIORITY front-end serves 3 web-pages which are explained below.

1. **Home Page** This is the default page that every user will see firstly while using PRIORITY. A snapshot of this page is given in Figure (mention it and paste image here). The page contains 2 different types of inputs that the user can provide. Difficulty rating input captures the difficulty level of questions that the user want to query. With 1 star denoting the minimum difficulty to 5 star denoting maximum difficulty. Second type of input present on this page is the Radio Buttons. These buttons are multi-click multi select type buttons which means that a user can select multiple inputs at once. These are listed below.

    - **TerminalIO Basic:** Challenges involving straightforward input and output operations, often employing `printf` and `scanf` statements.
    - **Terminal Advanced:** Tasks demanding intricate handling of input/output formatting, frequently necessitating the use of sophisticated format specifiers within `printf` and `scanf` statements.
    - **Arithmetic Basic:** Exercises focusing on fundamental arithmetic operations like addition, subtraction, multiplication, and division.
    - **Arithmetic Advanced:** Problems requiring more complex arithmetic expressions, including explicit type conversions, utilization of `math.h` functions, and intricate arithmetic computations.
    - **Arithmetic Bit:** Challenges centered around manipulation of individual bits, utilizing bitwise operators such as AND, OR, XOR, and bit shifting.
    - **Conditionals Basic:** Problems necessitating the use of simple `if/else` conditional constructs to make decisions based on specified conditions.
    - **Conditionals Switch:** Tasks specifically requiring the application of switch-case constructs for decision-making based on multiple potential values of a variable.
    - **Conditionals Advanced:** Challenges involving advanced conditional logic such as ternary operators, nested `if/else` constructs, and complex condition evaluations.
    - **Conditionals Flag:** Problems where flag variables are employed to manage program flow or indicate particular states.
    - **Loops Basic:** Exercises focusing on basic iterative or loop constructs, often utilizing `for`, `while`, or `do-while` loops.
    - **Loops Advanced:** Challenges necessitating more sophisticated loop concepts, including nested loops, loop optimizations, and the use of control flow statements like `break` and `continue`.
    - **Loops In-variants:** Tasks involving identification and utilization of non-trivial loop invariants to solve computational problems efficiently.

- **Arrays Basic:** Problems requiring the use of simple one-dimensional arrays to store and manipulate data.
- **Arrays Advanced:** Challenges involving multi-dimensional arrays and more complex array manipulation operations.
- **Arrays Memory:** Exercises necessitating understanding of memory management concepts, including dynamic memory allocation and deallocation using functions such as `malloc()`, `calloc()`, `realloc()`, `sizeof()`, and `free()`.
- **Pointers Basic:** Problems centered around basic manipulation of pointers, including dereferencing, pointer arithmetic, and basic memory operations.
- **Pointers Advanced:** Challenges requiring more advanced utilization of pointers, such as arrays of pointers, pointers to functions, or pointers to pointers.
- **Char-String Basic:** Tasks focusing on basic operations involving characters and strings, including string manipulation, comparison, and searching.
- **Char-String Advanced:** Challenges involving advanced concepts related to characters and strings, such as encoding/decoding schemes, regular expressions, or string algorithms.
- **Functions Basic:** Problems necessitating the definition and application of basic functions, typically with simple parameter types and return values.
- **Functions Advanced:** Challenges involving more complex functions, including functions with arrays, pointers, or references as parameters and return types.
- **Structures Basic:** Problems focusing on basic utilization of structures to organize related data fields.
- **Structures Advanced:** Challenges requiring more advanced use of structures, such as nested structures, utilization of pointers to structures, or implementation of complex data structures.
- **Structures DS:** Tasks involving utilization of structures to implement fundamental data structures such as linked lists, stacks, queues, graphs, trees, etc.
- **Algorithms DC:** Problems requiring the application of algorithms based on the divide and conquer approach, which entails breaking down a problem into smaller subproblems, solving them recursively, and combining their solutions.
- **Algorithms Recursion:** Challenges specifically involving the utilization of recursive algorithms, where a function calls itself to solve smaller instances of the same problem until reaching a base case.
- **Algorithms Greedy:** Problems requiring the application of algorithms based on the greedy approach, which involves making locally optimal choices at each step with the aim of achieving a global optimum.
- **Algorithms DP:** Challenges requiring the application of algorithms based on dynamic programming, which involves breaking down a problem into simpler sub-problems and storing their solutions to avoid redundant calculations and improve efficiency.

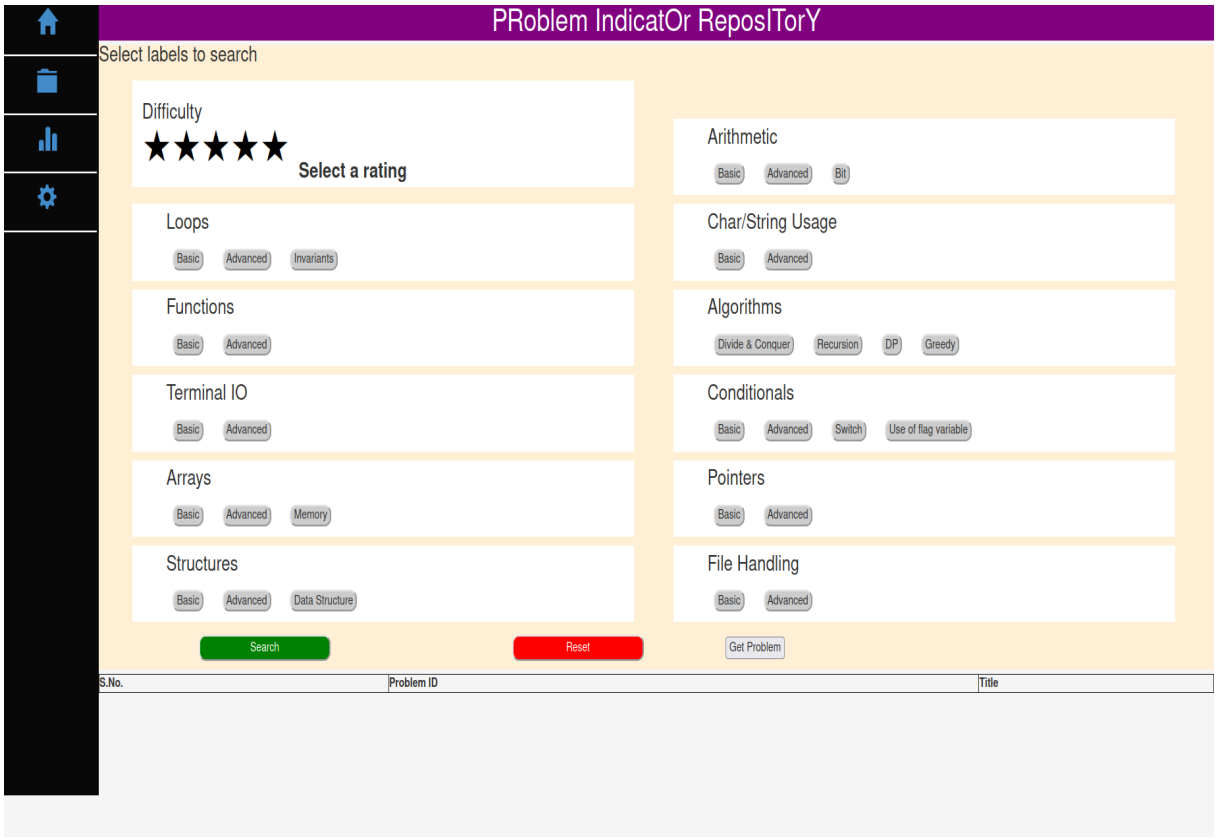


Figure 2.2: PRIORITY Homepage.

As said earlier, user can select multiple labels for a single query.

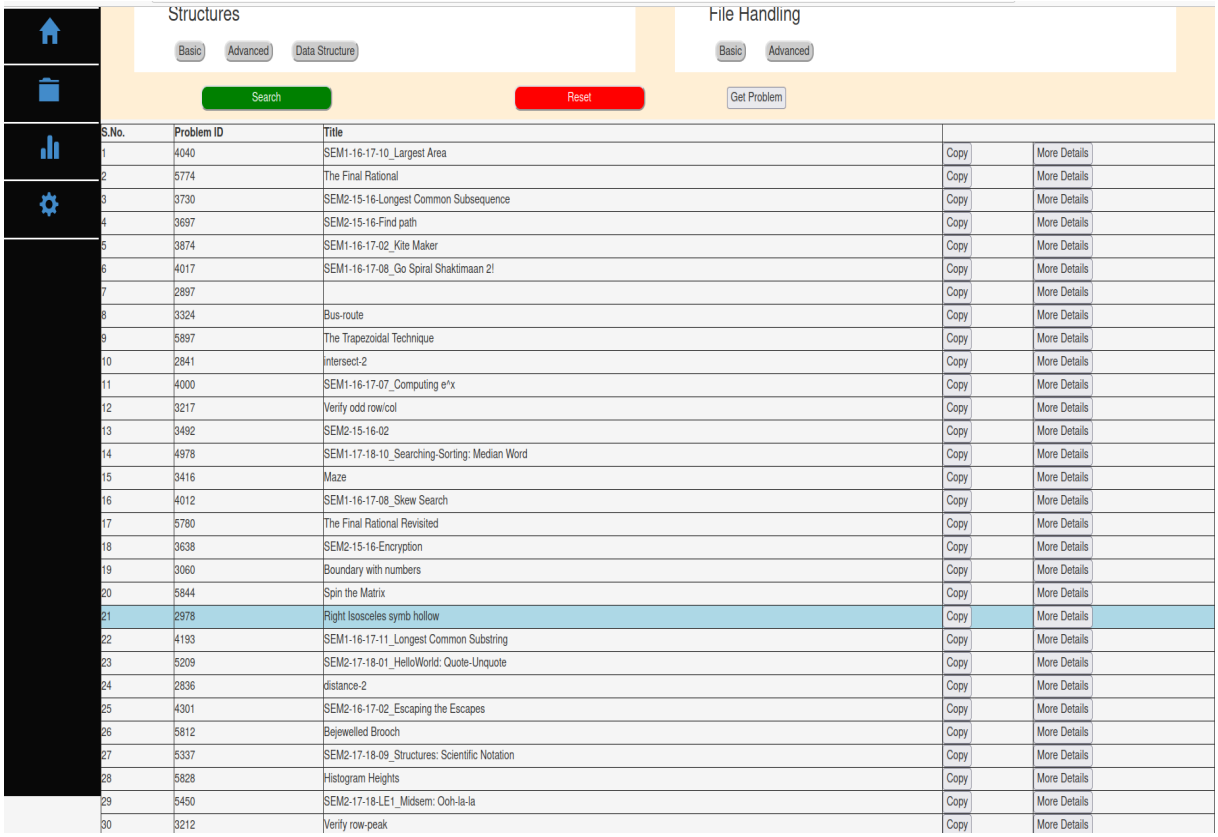
2. **Problem Search Page** When a user performs a query, an API request is generated for PRIORITY backend which is duly authenticated by methods mentioned in chapter 3. After authentication, the query is processed and a list of relevant problem id is given back which in turn is displayed on the bottom section of the homepage. A snapshot of the same is shown in Figure 2.3

3. **Problem Display** After getting the list of questions, a user can click on any relevant problem and it opens the problem details. It is shown as a sample in figure (mention it). It contains Problem Id, problem ID, problem solution. On the button of this page, there is a simpkle move to prutor button. On clicking this button, this problem is automatically copied to PRUTOR.

2.2.3 Back End

The backend architecture of the PRIORITY module comprises several key components:

1. **Node.js Server**: The PRIORITY module is constructed using Node.js, which serves as a runtime environment for executing JavaScript code. This module functions as the backend server, managing HTTP requests from the PRUTOR web application and interacting with the SQL database.



S.No.	Problem ID	Title		
1	4040	SEM1-16-17-10_Largest Area	Copy	More Details
2	5774	The Final Rational	Copy	More Details
3	3730	SEM2-15-16-Longest Common Subsequence	Copy	More Details
4	3697	SEM2-15-16-Find path	Copy	More Details
5	3874	SEM1-16-17-02_Kite Maker	Copy	More Details
6	4017	SEM1-16-17-08_Go Spiral Shaktimaan 2!	Copy	More Details
7	2897		Copy	More Details
8	3324	Bus-route	Copy	More Details
9	5897	The Trapezoidal Technique	Copy	More Details
10	2841	intersect-2	Copy	More Details
11	4000	SEM1-16-17-07_Computing e^x	Copy	More Details
12	3217	Verify odd row/col	Copy	More Details
13	3492	SEM2-15-16-02	Copy	More Details
14	4978	SEM1-17-18-10_Searching-Sorting: Median Word	Copy	More Details
15	3416	Maze	Copy	More Details
16	4012	SEM1-16-17-08_Skew Search	Copy	More Details
17	5780	The Final Rational Revisited	Copy	More Details
18	3638	SEM2-15-16-Encryption	Copy	More Details
19	3060	Boundary with numbers	Copy	More Details
20	5844	Spin the Matrix	Copy	More Details
21	2978	Right Isosceles symb hollow	Copy	More Details
22	4193	SEM1-16-17-11_Longest Common Substring	Copy	More Details
23	5209	SEM2-17-18-01_HelloWorld: Quote-Unquote	Copy	More Details
24	2836	distance-2	Copy	More Details
25	4301	SEM2-16-17-02_Escaping the Escapes	Copy	More Details
26	5812	Bejewelled Brooch	Copy	More Details
27	5337	SEM2-17-18-09_Structures: Scientific Notation	Copy	More Details
28	5828	Histogram Heights	Copy	More Details
29	5450	SEM2-17-18-LE1_Midsem: Ooh-la-la	Copy	More Details
30	3212	Verify row-peak	Copy	More Details

Figure 2.3: PRIORITY Search Results List.

2. **Express.js Framework**: For the backend server development, the module employs Express.js, a minimalist web framework designed for Node.js. Express.js facilitates the creation of the server by providing features like routing, middleware, and request handling. This simplifies the implementation of RESTful APIs, enabling smooth communication with the PRUTOR frontend [2].

3. **MySQL Database**: The PRIORITY module utilizes a MongoDB database to store and manage logs. It's connected to a MySQL instance comprising tables. The database schema is designed to accommodate various question types and their metadata. MySQL's adaptable schema eases adjustment to changes in data structure and supports storage of complex data types. Moreover, its scalability and performance make it apt for managing large data volumes [5].

4. **API Endpoints**: The backend exposes a set of API endpoints allowing PRUTOR to execute Read operations on the question bank repository. These endpoints follow RESTful principles and employ authentication mechanisms to ensure data integrity and user privacy.

The backend architecture of the PRIORITY module offers several advantages:

1. **Scalability**: The backend's modular design supports scalability, enabling it to effectively handle an increasing number of users and questions. Node.js and Express.js facilitate horizontal scaling by allowing deployment of multiple instances behind a load balancer.

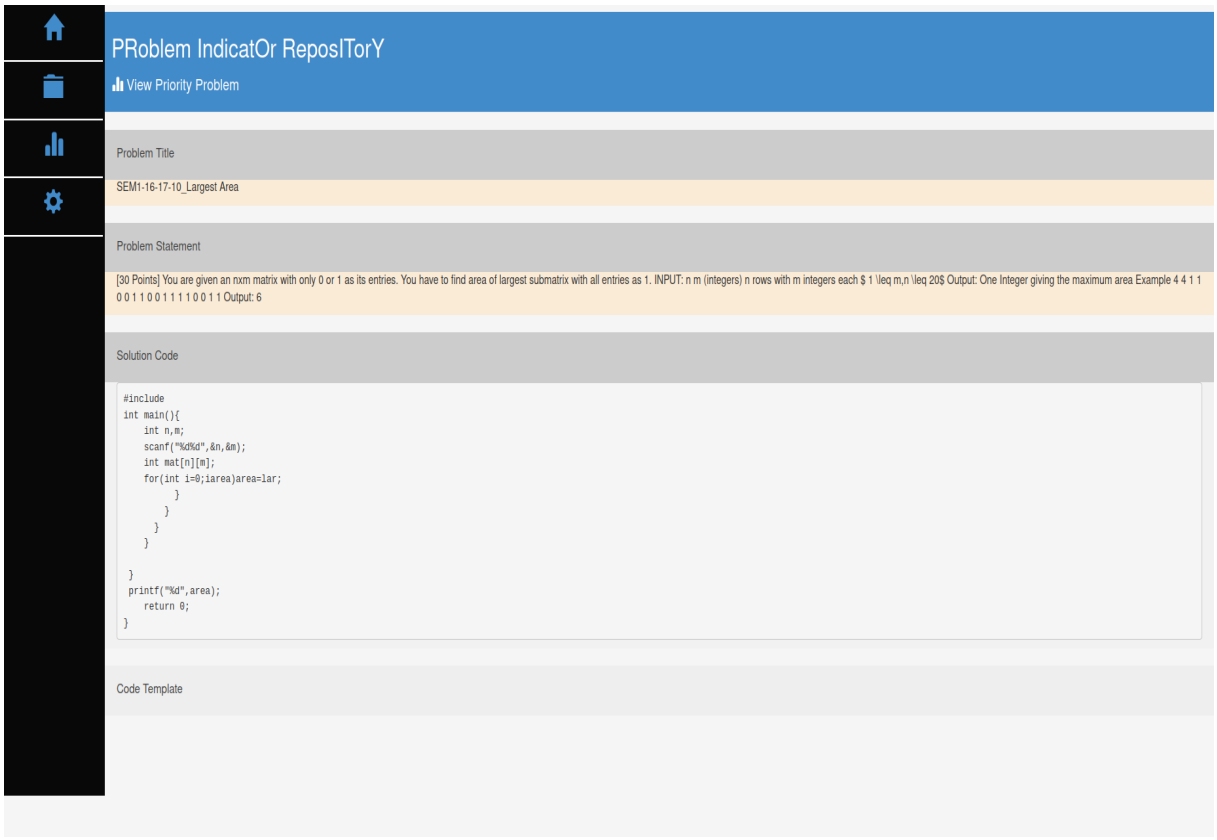


Figure 2.4: PRIORITY Homepage.

2. **Performance**: Thanks to Node.js's non-blocking I/O model and Express.js's lightweight nature, the PRIORITY module's backend achieves high performance. This results in responsive API responses and minimal latency for users accessing the question bank repository.

3. **Reliability**: The backend ensures data reliability through the use of a SQL database with transaction support and data integrity constraints. It maintains ACID properties, preventing data corruption or inconsistencies.

4. **Flexibility**: With a SQL database, the backend offers flexibility in querying and filtering questions based on various criteria such as difficulty level, topic, or assessment type. This enables customization of coding practice sessions to meet specific needs and preferences.

5. **Security**: Security measures like authentication and authorization are implemented to safeguard sensitive data in the question bank repository. Access controls are enforced to prevent unauthorized access or modification of confidential information.

6. **Ease of Maintenance**: ORM libraries simplify database management, reducing reliance on raw SQL queries and lowering the risk of SQL injection vulnerabilities. The modular architecture enhances code maintainability by isolating concerns and facilitating incremental updates or enhancements.

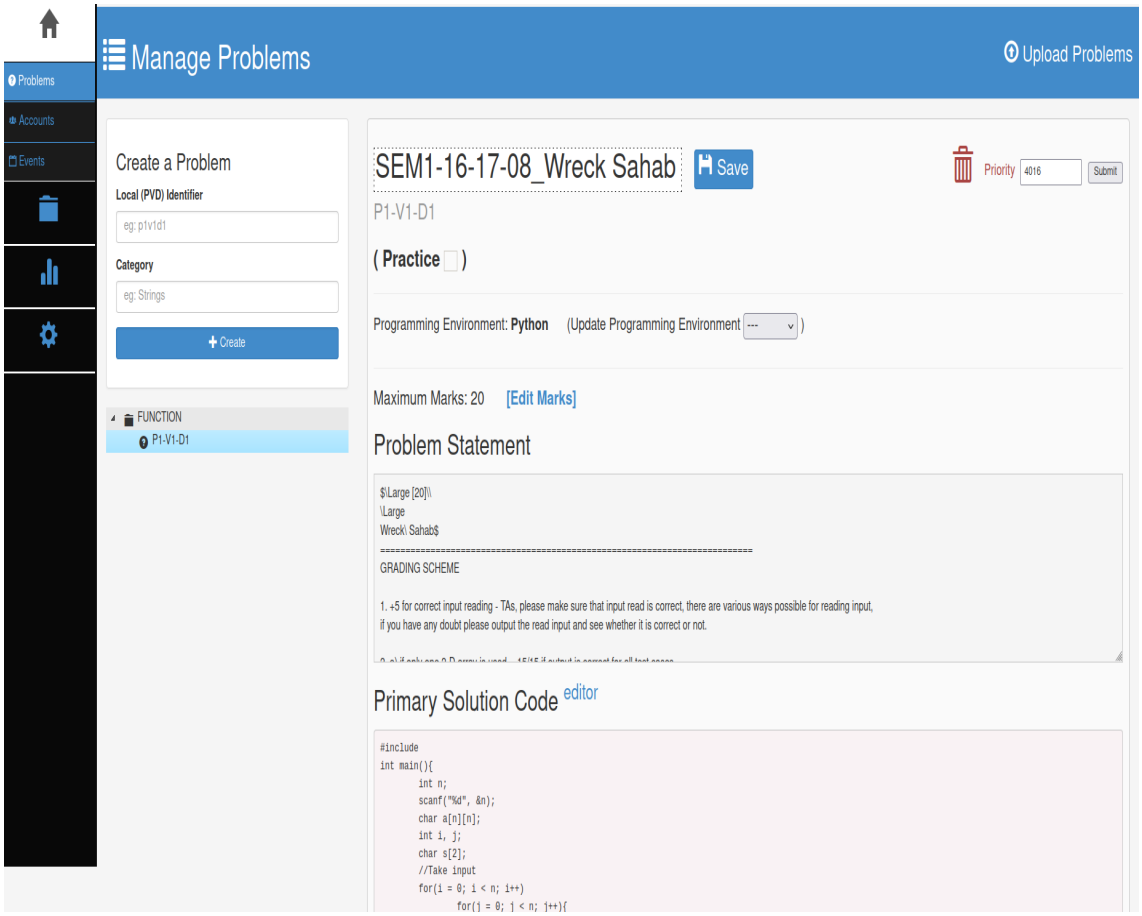


Figure 2.5: PRIORITY Homepage.

2.3 Application Flow and User Actions

In this section, we outline a common user workflow for utilizing the portal. Figure 7.15 illustrates the user's interactions with the PRIORITY frontend, summarizing the typical journey through the platform.

1. To initiate their experience with the PRIORITY, users must first log in to PRUTOR via a web browser. Access to the platform requires prior registration, with users provided credentials in advance. It's important to note that only tutors and course instructors have access to PRIORITY within this thesis context. This limitation is in place to prevent students from accessing question banks, thereby preserving the integrity of the ESC101 foundational programming course.

2. After logging in to PRUTOR, users will click on Problems->Create New Problem button. Here they will see the Create Problem page as shown in Figure. On the top there is a Open Priority Button. After clicking on this button, user will be redirected to PRIORITY homepage.

3. On this page, user will select a number of labels and difficulty rating and click on Search button.

4. User will get a list of questions. User can open any of them to view the problem statement, problem solution etc.

5. After viewing the problem, if user wants to import this problem in PRUTOR, user can click on Copy button and will be redirected to create problem page of PRUTOR where the imported question will be visible.

Figure 2.1 shows the conceptual view of application flow which has been described above.

CHAPTER 3

PRIORITY: API And Authentication

Contents

As discussed in chapter 1, Priority fetches data using API calls, which can be hacked and intercepted in between. To prevent that from happening we have used several measures.

3.1 Access Control Lists:

Access Control Lists (ACLs) are mechanisms in Node.js used to manage permissions and control access to resources within applications. They specify which users or roles can perform certain actions (e.g., reading, writing, deleting) on specific resources based on their roles or identities.

3.1.1 Key Components

In a standard ACL system, several key components exist:

- **Roles**: Represent groups of users with similar permissions or privileges, such as "instructor," "tutor," or "student." Roles help categorize users based on their responsibilities and access levels.
- **Permissions**: Define actions that users in specific roles can perform on particular resources, like read, write, delete, or update. Permissions are linked to roles and dictate each role's access level to resources.

- **Resources**: Entities or objects users interact with within the application, such as URLs, files, or database records. Resources can be categorized by type and functionality, with access control applied to ensure proper authorization.
- **Access Control Rules**: Specify which roles have permission to perform specific actions on designated resources. These rules govern access control decisions and are based on the relationships between roles, permissions, and resources.

3.1.2 Implementation Techniques

Node.js applications can implement ACLs using various techniques:

1. **Using npm Packages**: Utilizing npm packages like `acl` or `casbin` provides pre-built functionalities for ACL management. These packages offer APIs to define roles, permissions, and access control rules, as well as methods to check permissions during runtime.
2. **Custom Implementation**: Developers can implement ACLs with custom middleware or logic tailored to their application's needs. This involves defining roles, permissions, and rules in the application code and enforcing them based on user roles and requested actions.
3. **Database-based Approach**: ACL information can be stored in a database for dynamic management of roles and permissions. Role-permission mappings and access control rules are stored in the database, with custom logic querying and enforcing these rules during runtime.

3.1.3 Conclusion

ACLs are critical for security and access control in Node.js applications, restricting access to sensitive resources and functionalities based on user roles and permissions. Effective ACL implementation ensures data and functionality integrity and confidentiality. The choice of implementation technique depends on factors like application complexity and scalability needs. Regardless of the approach, robust access control mechanisms are essential for secure and reliable Node.js applications.

3.2 Cross-Origin Resource Sharing (CORS)

Cross-Origin Resource Sharing (CORS) serves as a security feature in web browsers, governing access to resources hosted on domains separate from the requesting web page. It allows web servers to specify which origins can access their resources across different domains, ensuring secure cross-origin communication in web applications.

3.2.1 How CORS Works

When a web page initiates a cross-origin HTTP request (to a different domain, protocol, or port), the browser enforces the Same-Origin Policy, limiting resource access based on the requesting

page's origin. However, CORS provides a means for servers to relax this restriction under controlled conditions.

CORS functions by appending additional HTTP headers to the server's response, indicating whether the requested resource can be shared with the requesting domain. These headers include:

- **Access-Control-Allow-Origin**: Specifies permitted origins for accessing the resource, either a specific origin or "*" to allow access from any origin.
- **Access-Control-Allow-Methods**: Defines allowed HTTP methods for resource access (e.g., GET, POST, PUT, DELETE).
- **Access-Control-Allow-Headers**: Specifies permitted HTTP headers for the request.
- **Access-Control-Allow-Credentials**: Determines whether the browser should include credentials in the request.
- **Access-Control-Expose-Headers**: Identifies headers exposed to the client in the response.
- **Access-Control-Max-Age**: Sets the caching duration for preflight requests (OPTIONS).

3.2.2 Implementation

CORS is typically implemented server-side by configuring the web server to include the appropriate CORS headers in cross-origin request responses. Web frameworks and platforms often offer built-in support for CORS configuration, allowing developers to define allowed origins, methods, headers, and other parameters.

3.2.3 Preflight Requests

For certain cross-origin requests, the browser initiates a preflight request (via the OPTIONS method) to ascertain the safety of the actual request. The preflight request includes CORS-specific headers to inform the server of the intended cross-origin request, and the server responds with CORS headers indicating permission status.

3.2.4 Security Considerations

While CORS facilitates web application interoperability, it introduces security considerations. Allowing cross-origin requests from any domain ("*") may expose sensitive resources to malicious sites. Hence, configuring CORS policies meticulously is crucial to mitigate security risks and prevent unauthorized data access.

Security best practices for CORS implementation include:

- Restricting Access: Specify trusted origins explicitly instead of permitting access from any origin.
- Request Validation: Verify origin and CORS headers in incoming requests to ensure they originate from trusted sources and remain unaltered.

- Limited Credential Use: Exercise caution when allowing cross-origin requests with credentials to prevent unauthorized data access.
- Rate Limiting: Enforce rate limits on cross-origin requests to prevent abuse and potential denial-of-service attacks.
- Monitoring and Logging: Monitor CORS activities and log violations to identify suspicious behavior and security incidents.

3.2.5 Conclusion

Cross-Origin Resource Sharing (CORS) is vital for secure resource access across diverse origins in web applications. Effective CORS policy implementation, coupled with adherence to security best practices, ensures secure cross-origin communication and safeguarding of sensitive data against unauthorized access and exploitation.

3.3 Token Authentication

Token authentication is a technique used to verify users in web applications by assigning a unique token to each user after a successful login. This token is then utilized in subsequent requests to authenticate the user's identity and access permissions. Compared to traditional session-based authentication, token-based authentication offers advantages like statelessness, scalability, and improved security.

3.3.1 How Token Authentication Functions

In token-based authentication, once a user logs in with valid credentials, the server generates a unique token (often a JSON Web Token or JWT) and sends it back to the client. The client then stores this token locally, typically in the browser's local storage or session storage.

For subsequent requests to secured resources, the client includes the token in the request header (typically in the `Authorization` header) to validate its identity. The server validates the token and grants access to the requested resource if the token is valid and carries the necessary permissions.

3.3.2 Advantages of Token Authentication

Token-based authentication provides several benefits:

- **Statelessness**: Unlike session-based authentication, token authentication doesn't require the server to retain session data on the server-side. This inherent statelessness simplifies scalability and load balancing across multiple servers.
- **Scalability**: Since token authentication doesn't rely on server-side session management, it scales more efficiently, making it suitable for distributed and microservices-based architectures.

- **Enhanced Security**: Tokens can be digitally signed and encrypted, rendering them tamper-proof and resistant to unauthorized alterations. Additionally, token-based authentication enables fine-grained access control and the ability to revoke access tokens if necessary.
- **Cross-Origin Authentication**: Tokens can be easily transmitted across different domains or origins, enabling cross-origin authentication in single-page applications (SPAs) and APIs.

3.3.3 Implementation

Implementing token authentication in web applications involves several steps:

1. **Authentication**: Authenticate users using their credentials and generate a unique token upon successful authentication.
2. **Token Issuance**: Issue a signed and optionally encrypted token containing user information, access permissions, and expiration time.
3. **Token Storage**: Securely store the token on the client-side for subsequent use.
4. **Token Validation**: Validate the token on each incoming request to secured resources by verifying its signature, expiration, and permissions.
5. **Access Control**: Implement access control logic based on the user's roles and permissions encoded in the token.
6. **Token Revocation**: Optionally implement mechanisms to revoke or invalidate tokens if compromised or no longer needed.

3.3.4 Security Considerations

While token authentication offers significant security advantages, it also introduces potential risks if implemented incorrectly. Developers should consider the following security considerations:

- **Token Expiration**: Set appropriate expiration times for tokens to minimize the risk of token theft and unauthorized access.
- **Token Storage**: Securely store tokens on the client-side and transmit them over HTTPS to prevent interception and tampering.
- **Token Revocation**: Implement mechanisms to revoke or invalidate tokens if compromised or if a user's access permissions change.
- **Cross-Site Request Forgery (CSRF)**: Guard against CSRF attacks by including anti-CSRF tokens alongside authentication tokens and validating them on the server.
- **Token Payload**: Refrain from including sensitive information (e.g., passwords or personally identifiable information) in token payloads to prevent data exposure in the event of token leakage.

By addressing these security considerations and adhering to best practices for token authentication, developers can harness the benefits of token-based authentication while minimizing security risks and ensuring the integrity and confidentiality of user data.

CHAPTER 4

Deployment & Conclusion

In this chapter, we will explain about the deployment of our PRIORITY app inside PRUTOR. In section 4.1, we explain about docker and how our app is deployed using docker followed by section 4.2, where we provide the conclusion of our project. In the end, we provide the provide the future scope of this thesis in section 4.3

4.1 Docker

Docker stands as a widely-used platform for the development, transportation, and execution of applications within containers. These containers offer a streamlined and adaptable method to bundle software along with its dependencies, ensuring consistent operation across diverse environments [3].

4.1.1 Fundamental Concepts

Docker introduces several fundamental concepts:

- **Container**: Representing a lightweight, self-contained package, a container encompasses all components necessary for software execution, including code, runtime, libraries, and dependencies. Containers facilitate application isolation and promote deployment uniformity.
- **Image**: Serving as a blueprint for container creation, an image encapsulates the filesystem and configuration required for application execution. These images are crafted from Dockerfiles, which specify the instructions for image construction.
- **Dockerfile**: A text-based document, a Dockerfile contains directives for building a Docker image. It outlines the base image, environment settings, dependencies, and commands essential for configuring the application environment.
- **Registry**: Functioning as a centralized hub, a registry stores and disseminates Docker images. While Docker Hub serves as the primary public registry, organizations may establish private registries for internal usage.
- **Docker Engine**: Serving as the runtime environment for containers, Docker Engine encompasses the Docker daemon, responsible for container and image management, along with the Docker CLI, facilitating user interaction with Docker.

4.1.2 Operational Workflow

The operational workflow for Docker typically involves:

1. **Dockerfile Creation**: Crafting a Dockerfile tailored to the application's requirements, specifying the base image, dependencies, and directives for image construction.
2. **Image Building**: Employing the Docker CLI to construct an image from the Dockerfile, resulting in a snapshot of the application environment.
3. **Container Execution**: Utilizing the Docker CLI to initiate a container from the built image, enabling seamless container management operations such as starting, stopping, and restarting.
4. **Image Publication**: Optionally, publishing the generated image to a registry for widespread accessibility, allowing others to retrieve and execute the image in their respective environments.

4.1.3 Advantages

Docker offers several advantages for both application development and deployment:

- **Portability**: Containers encapsulate all dependencies, ensuring application portability and consistency across diverse environments, from development setups to production environments.
- **Isolation**: Containers isolate applications, preventing mutual interference and safeguarding against changes in one application affecting others or the underlying infrastructure.
- **Resource Efficiency**: Containers leverage shared resources from the host system, leading to lightweight and efficient deployment, in contrast to traditional virtual machines.
- **Scalability**: Containers can be effortlessly scaled to accommodate varying workload demands, making them well-suited for dynamic and scalable applications.
- **DevOps Integration**: Docker seamlessly integrates with DevOps practices, facilitating continuous integration, delivery, and deployment workflows.

4.1.4 Usage Scenarios

Docker finds applications across various scenarios, including:

- **Microservices**: Docker streamlines the development and deployment of microservices-based architectures, fostering agility and scalability.
- **Continuous Integration/Continuous Deployment (CI/CD)**: Docker is integral to CI/CD pipelines, automating build, test, and deployment processes to streamline software delivery.
- **Hybrid Cloud**: Docker ensures consistent deployment across hybrid cloud environments, enabling seamless transitions between on-premises and cloud infrastructures.
- **Development Environments**: Docker offers reproducible development environments, enabling developers to work in isolated containers with consistent dependencies.

4.1.5 Security Considerations

Despite its numerous benefits, Docker introduces security considerations:

- **Image Security**: Verifying that Docker images originate from trustworthy sources and are devoid of vulnerabilities, necessitating regular vulnerability scans using tools like Clair or Trivy.
- **Container Isolation**: Ensuring proper isolation of containers from each other and from the host system, preventing unauthorized access and potential privilege escalation.
- **Runtime Security**: Enforcing best practices for securing container runtimes, such as employing non-root users within containers, limiting resource usage, and enabling security profiles like seccomp or AppArmor.
- **Registry Security**: Securing Docker registries through authentication and access controls, alongside periodic audits to detect unauthorized images.
- **Networking and Firewall**: Implementing network segmentation and firewall rules to govern traffic between containers and external networks, thereby reducing the attack surface.

By addressing these security considerations and adhering to Docker best practices, organizations can harness the benefits of containerization while mitigating associated security risks.

4.2 Conclusion

This thesis presents the development of PRIORITY, an extensive application designed to tackle the challenges encountered by educators in courses like ESC101 at IIT Kanpur. We've implemented diverse machine learning models and crafted a NodeJS module within PRUTOR to streamline the setup of questions. Additionally, we've integrated a logging feature to capture events, aiding in the refinement of ML models.

Throughout this research endeavor, we've explored various approaches to integrate a novel module into PRUTOR, a legacy system. Notably, we've incorporated logging and API authentication functionalities into the newly developed PRIORITY module.

Moreover, we've seamlessly integrated the PRIORITY application with PRUTOR, a pivotal tutoring platform for courses such as ESC101, facilitating the effortless creation of problem sets within educational institutions. Leveraging Docker for containerization ensures the application's swift deployment and scalability across diverse environments.

Overall, the PRIORITY application signifies a remarkable stride in modern education and pedagogical methodologies, offering a pragmatic solution for effortlessly accessing and importing questions from an extensive question bank. With further refinement and validation, this application holds immense potential to bolster foundational learning mechanisms in classrooms grappling with a large student cohort.

4.3 Future Scope

While the PRIORITY application showcases promising outcomes, several avenues for future enhancement and expansion exist:

- **Feature Enrichment**: Explore additional features and data sources to enhance the predictive capabilities of the models.
- **Model Fine-tuning**: Refine the hyperparameters of the machine learning models and delve into advanced methodologies like ensemble learning and deep learning architectures to bolster accuracy and resilience.
- **Semantic Query Enhancement**: Implement semantic search techniques such as Elastic Search to execute more nuanced queries within PRIORITY.
- **User Feedback Incorporation**: Gather feedback from academic experts and stakeholders to iteratively refine the application's functionality, ensuring its practicality and usability in real-world educational contexts.
- **Scalability and Performance Optimization**: Streamline the scalability and performance of the application to effectively handle extensive problem datasets and concurrent user requests. This may involve harnessing cloud computing resources and implementing distributed computing strategies.

Addressing these areas of future work will enable the continual evolution of the PRIORITY application, transforming it into an indispensable tool for academic professionals seeking to enhance teaching efficacy in foundational programming courses with large student cohorts.

CHAPTER 5

Late Fusion

Contents

This chapter on Late Fusion is an extension of the work carried out in [11]. Before reading this chapter, it is advised to refer to [11] for basic concepts, problem statements and research methodologies to have a complete idea and understanding. Late fusion, also known as post-processing fusion, is a technique that is used to combine the outputs of several models or sources of information after those models or sources have created their individual predictions. Late fusion is also known as post-processing fusion. This method is utilized in situations in which multiple models, each of which is based on a unique set of characteristics or modalities, independently produce their own outputs. The major goal of late fusion is to produce a comprehensive answer by combining the results of several separate forecasts into a single comprehensive whole.

The late fusion process involves several steps:

1. **Model Prediction**:Each model creates its own prediction based on the data that is currently available. These forecasts may be presented in the form of probabilities, class labels, or any other output format that is deemed appropriate for the particular endeavor.

2. **Combination Strategy**:When combining the outputs, a specified strategy or algorithm is utilized as the method of choice. This tactic can be as straightforward as taking an average or voting, or it might involve more complicated procedures such as weighted aggregation or ensemble methods.

3. **Final Prediction**:The final product of the late fusion process is the combined prediction, which is considered to be the final output of the process. This output is often more robust and reliable than the predictions of individual models because it leverages the strengths of each model and compensates for their deficiencies. This is because the output takes into account the interplay between the strengths and limitations of each model.

Late fusion provides a number of benefits, including the following:

- **Improved Robustness**: Late fusion can improve the robustness of predictions by lowering the influence of errors or biases in individual models, which is accomplished by combining many models at a later time.

- **Flexibility**: It allows for adaptability to a variety of circumstances and data characteristics by providing flexibility in the selection of the combination approach. Flexibility is provided.

Label	Accuracy	Precision	Recall	F1_score
Arithmetic_Bit	0.9692	0.0	0.0	0.0
Algorithms_Greedy	0.9846	0.0	0.0	0.0
Arithmetic_Basic	0.7692	0.6	0.3529	0.4444
Loops_Basic	0.6769	0.3684	0.4375	0.4
TerminalIO_Advanced	0.8615	0.4286	0.375	0.4
Arrays_Basic	0.8462	0.4444	0.4444	0.4444
Pointers_Advanced	0.9692	0.0	0.0	0.0
Conditionals_Flags	0.8769	0.0	0.0	0.0
Structures_Basic	0.9692	0.0	0.0	0.0
Char-String_Basic	0.8769	0.4	0.6667	0.5
Conditionals_Basic	0.6615	0.381	0.4706	0.4211
Functions_Advanced	0.8308	0.6667	0.5333	0.5926
Arithmetic_Advanced	0.9692	0.5	0.5	0.5
Conditionals_Switch	0.9846	0.5	1.0	0.6667
Algorithms_Recursion	0.9385	0.8333	0.625	0.7143
Structures_Advanced	0.9077	0.0	0.0	0.0
Algorithms_DP	1.0	1.0	1.0	1.0
Conditionals_Advanced	0.8923	0.0	0.0	0.0
Loops_Advanced	0.8308	0.9167	0.5238	0.6667
Arrays_Advanced	0.9077	0.5	0.1667	0.25
Char-String_Advanced	0.9231	0.0	0.0	0.0
TerminalIO_Basic	0.8308	0.0	0.0	0.0
Pointers_Basic	0.8615	0.2222	0.5	0.3077
Structures_DS	1.0	1.0	1.0	1.0
Arrays_Memory	0.9538	0.6667	0.5	0.5714
Algorithms_DC	1.0	1.0	1.0	1.0
Functions_Basic	0.7077	0.3333	0.2667	0.2963
Loops_Invariants	0.9385	0.0	0.0	0.0
Overall	0.8907	0.4226	0.4104	0.4164
Micro	0.8907	0.4172	0.4329	0.3866

Table 5.1: Late Fusion using Logistic Regression

5.1 Model Prediction

The first level of model prediction involves training and using two distinct models: the Statement-Based Model and the AST (Abstract Syntax Tree) Model. These models are trained individually to make predictions. For a given data point d, we have:

$P_{\text{statement}}(d)$ - Predicted probability by the Statement-Based Model for the positive class,

$P_{\text{AST}}(d)$ - Predicted probability by the AST Model for the positive class,

$P_{\text{statement}}(d, 0)$ - Predicted probability by the Statement-Based Model for the negative class,

$P_{\text{AST}}(d, 0)$ - Predicted probability by the AST Model for the negative class.

To perform late fusion, we introduce two weights: α and β. These weights determine the contribution of each model to the final prediction. The final prediction score $P_{\text{final}}(d)$ for data point d is calculated as:

Label	Accuracy	Precision	Recall	F1_score
Arithmetic_Bit	0.9846	0.5	1.0	0.6667
Algorithms_Greedy	0.9846	0.0	0.0	0.0
Arithmetic_Basic	0.7846	0.6364	0.4118	0.5
Loops_Basic	0.6769	0.3684	0.4375	0.4
TerminalIO_Advanced	0.8462	0.375	0.375	0.375
Arrays_Basic	0.8615	0.5	0.5556	0.5263
Pointers_Advanced	0.9077	0.0	0.0	0.0
Conditionals_Flags	0.8615	0.0	0.0	0.0
Structures_Basic	0.9846	0.0	0.0	0.0
Char-String_Basic	0.8462	0.3333	0.6667	0.4444
Conditionals_Basic	0.6462	0.35	0.4118	0.3784
Functions_Advanced	0.8154	0.6154	0.5333	0.5714
Arithmetic_Advanced	0.9077	0.1667	0.5	0.25
Conditionals_Switch	0.9385	0.2	1.0	0.3333
Algorithms_Recursion	0.9385	0.8333	0.625	0.7143
Structures_Advanced	0.8769	0.0	0.0	0.0
Algorithms_DP	0.9846	0.0	0.0	0.0
Conditionals_Advanced	0.8462	0.0	0.0	0.0
Loops_Advanced	0.8	0.7	0.6667	0.6829
Arrays_Advanced	0.9385	0.75	0.5	0.6
Char-String_Advanced	0.9077	0.0	0.0	0.0
TerminalIO_Basic	0.8308	0.1111	0.25	0.1538
Pointers_Basic	0.8462	0.2	0.5	0.2857
Structures_DS	1.0	1.0	1.0	1.0
Arrays_Memory	0.9385	0.5	0.5	0.5
Algorithms_DC	1.0	1.0	1.0	1.0
Functions_Basic	0.7077	0.3333	0.2667	0.2963
Loops_Invariants	0.8923	0.0	0.0	0.0
Overall	0.8769	0.3768	0.4509	0.4105
Micro	0.8769	0.4018	0.4944	0.4054

Table 5.2: Late Fusion using SVC

$$P_{\text{final}}(d) = \alpha \cdot P_{\text{statement}}(d) + \beta \cdot P_{\text{AST}}(d)$$

This represents the weighted linear combination of the probability scores from the two models and will be used for further final prediction. In the next we section we discuss strategies on how this α and β values are determined.

5.1.1 Combining Strategies

For combining there are certainly 2 approaches which we experimented with are as follows:

- Let $P_{\text{statement}}(d)$ and $P_{\text{AST}}(d)$ represent the predicted probabilities by the Statement-Based Model and AST Model, respectively, for data point d. We introduce two static weights α and β to combine these probabilities:

$$P_{\text{final}}(d) = \alpha \cdot P_{\text{statement}}(d) + \beta \cdot P_{\text{AST}}(d)$$

Label	Accuracy	Precision	Recall	F1_score
Arithmetic_Bit	0.9692	0.0	0.0	0.0
Algorithms_Greedy	0.9846	0.0	0.0	0.0
Arithmetic_Basic	0.7692	0.5625	0.5294	0.5455
Loops_Basic	0.6769	0.381	0.5	0.4324
TerminalIO_Advanced	0.8462	0.4	0.5	0.4444
Arrays_Basic	0.8615	0.5	0.5556	0.5263
Pointers_Advanced	0.9692	0.0	0.0	0.0
Conditionals_Flags	0.8	0.0	0.0	0.0
Structures_Basic	0.9846	0.0	0.0	0.0
Char-String_Basic	0.8615	0.3636	0.6667	0.4706
Conditionals_Basic	0.6462	0.3636	0.4706	0.4103
Functions_Advanced	0.8	0.5833	0.4667	0.5185
Arithmetic_Advanced	0.9077	0.1667	0.5	0.25
Conditionals_Switch	0.9538	0.25	1.0	0.4
Algorithms_Recursion	0.9385	0.8333	0.625	0.7143
Structures_Advanced	0.8462	0.0	0.0	0.0
Algorithms_DP	1.0	1.0	1.0	1.0
Conditionals_Advanced	0.8154	0.0	0.0	0.0
Loops_Advanced	0.7385	0.5909	0.619	0.6047
Arrays_Advanced	0.9231	0.6	0.5	0.5455
Char-String_Advanced	0.9231	0.0	0.0	0.0
TerminalIO_Basic	0.8	0.0909	0.25	0.1333
Pointers_Basic	0.8154	0.1667	0.5	0.25
Structures_DS	1.0	1.0	1.0	1.0
Arrays_Memory	0.9538	0.6667	0.5	0.5714
Algorithms_DC	1.0	1.0	1.0	1.0
Functions_Basic	0.7077	0.375	0.4	0.3871
Loops_Invariants	0.8769	0.0	0.0	0.0
Overall	0.8703	0.3624	0.4798	0.4129
Micro	0.8703	0.3655	0.5137	0.3966

Table 5.3: Late Fusion using Random Forest

Here, α and β are fixed values that determine the contribution of each model to the final prediction.

- For this strategy, we calculate separate weights α_i for each label i using a model-based approach. For each data point d, we have $P_{\text{statement}}(d, i)$ and $P_{\text{AST}}(d, i)$ as the predicted probabilities by the Statement-Based Model and AST Model for label i. We create a vector of dimension 2 for each label:

$$\mathbf{P}_i = \begin{bmatrix} P_{\text{statement}}(d, i) \\ P_{\text{AST}}(d, i) \end{bmatrix}$$

We then use a machine learning model, one for each label, to determine the optimal weights α_i for that label. Let Θ_i represent the parameters of the model for label i. The predicted optimal weights $\hat{\alpha}_i$ are obtained as:

$$\hat{\alpha}_i = \text{Model}(\mathbf{P}_i, \Theta_i)$$

For each label, this process gives us a set of optimal weights $\hat{\alpha}_i$. To make a binary prediction for data point d and label i, we combine the probabilities using these optimal weights:

$$P_{\text{final}}(d, i) = \hat{\alpha}_i \cdot P_{\text{statement}}(d, i) + (1 - \hat{\alpha}_i) \cdot P_{\text{AST}}(d, i)$$

Again, this final probabilities can be used to make probability based prediction using ranking or we can directly give out the results by comaparing $P_{\text{final}}(d, i)$ with a threshold τ to make a binary prediction for label i.

Label	accuracy	precision	recall	f1_score
Arithmetic_Bit	0.8615	0.0	0.0	0.0
Algorithms_Greedy	0.9846	0.0	0.0	0.0
Arithmetic_Basic	0.6923	0.4286	0.5294	0.4737
Loops_Basic	0.6769	0.4	0.625	0.4878
TerminalIO_Advanced	0.8462	0.4286	0.75	0.5455
Arrays_Basic	0.8	0.3571	0.5556	0.4348
Pointers_Advanced	0.5692	0.0	0.0	0.0
Conditionals_Flags	0.8	0.0	0.0	0.0
Structures_Basic	0.9692	0.0	0.0	0.0
Char-String_Basic	0.8615	0.4	1.0	0.5714
Conditionals_Basic	0.6615	0.4138	0.7059	0.5217
Functions_Advanced	0.6923	0.4	0.6667	0.5
Arithmetic_Advanced	0.9077	0.1667	0.5	0.25
Conditionals_Switch	0.9231	0.1667	1.0	0.2857
Algorithms_Recursion	0.9385	0.75	0.75	0.75
Structures_Advanced	0.8615	0.0	0.0	0.0
Algorithms_DP	0.9846	0.5	1.0	0.6667
Conditionals_Advanced	0.8308	0.0909	0.5	0.1538
Loops_Advanced	0.6923	0.5185	0.6667	0.5833
Arrays_Advanced	0.8769	0.375	0.5	0.4286
Char-String_Advanced	0.9077	0.0	0.0	0.0
TerminalIO_Basic	0.8154	0.1	0.25	0.1429
Pointers_Basic	0.7538	0.125	0.5	0.2
Structures_DS	0.9846	0.6667	1.0	0.8
Arrays_Memory	0.9385	0.5	0.75	0.6
Algorithms_DC	0.9846	0.5	1.0	0.6667
Functions_Basic	0.7231	0.4	0.4	0.4
Loops_Invariants	0.8769	0.0	0.0	0.0
overall	0.8363	0.3077	0.578	0.4016
Micro	0.8363	0.3077	0.6154	0.3944

Table 5.4: Late fusion using Random Forest and Multi-label Ranking prediction

5.2 Models

Following are the Models that were used for the late fusion experimentation

- **Linear Regression:** In the context of late fusion experimentation, Linear Regression serves as a fundamental model for combining predictions from multiple sources. The essence of linear regression lies in its simplicity and interpretability. For each label, a linear regression model is trained using the predicted probabilities from both the Statement-Based Model and AST Model as input features. The model learns coefficients that represent the weights assigned to each source of information, essentially determining how much influence each model has on the final prediction. This approach aligns with the static weighting strategy, where fixed coefficients regulate the combination of probabilities. Linear Regression is advantageous for its transparency, enabling insights into the contribution of each model to the overall prediction, facilitating a clear understanding of the fusion process.

- **Support Vector Machines:** Support Vector Machines (SVM) play a crucial role in late fusion experiments, offering a robust approach to combining predictions from different models. In the late fusion paradigm, SVM is employed as a model for each label, and it excels in handling non-linear relationships and complex decision boundaries. SVM operates by transforming the input probabilities from both the Statement-Based Model and AST Model into a higher-dimensional space, where it seeks to find the optimal hyperplane that maximally separates data points belonging to different classes. This approach aligns with the dynamic weighting strategy, allowing SVM to adaptively assign different weights to each model's prediction for different labels. SVM's ability to handle high-dimensional data and intricate decision boundaries makes it a valuable component in the late fusion experimentation, contributing to enhanced predictive performance and adaptability to varying label complexities.

- **Random Forest:** Random Forest, a versatile ensemble learning method, is a pivotal component in the late fusion experimentation process. In the late fusion framework, Random Forest is employed as a model for each label, capitalizing on its ability to handle high-dimensional data and capture complex relationships. Random Forest operates by constructing multiple decision trees and combining their outputs through an ensemble approach. This ensemble nature allows Random Forest to adaptively assign different weights to the predictions from the Statement-Based Model and AST Model for each label. The inherent diversity in decision trees helps mitigate overfitting and enhances the overall robustness of the late fusion model. Random Forest's capability to capture non-linear relationships and handle large datasets contributes significantly to the success of late fusion experiments, ensuring accurate predictions across diverse programming concepts and skill levels.

5.3 Results:

While evaluating the late fusion results, we can assess it in two ways based the analysis we did which are model-wise assessment and another is method wise assessment. The summary of both is as follows:

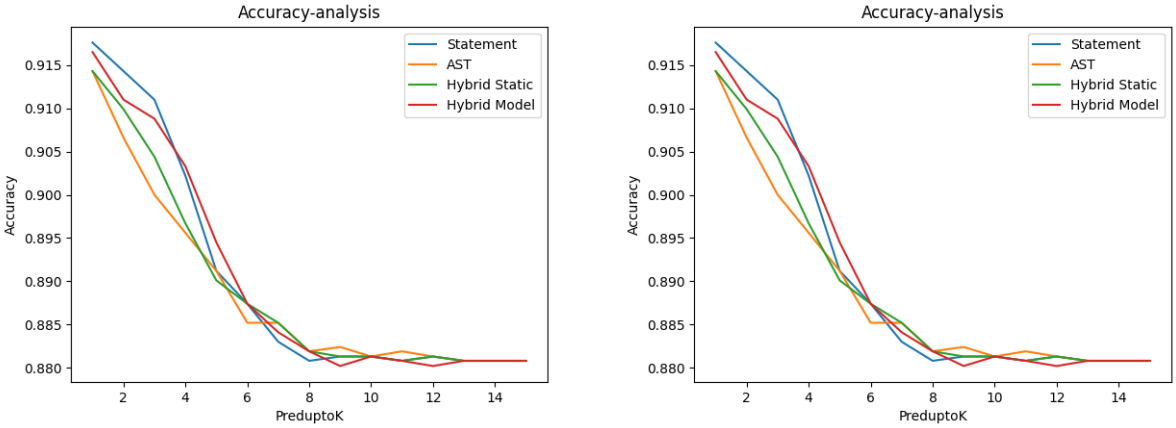
- **Model-wise analysis:**Among the various models examined, Random Forest classifier emerged as the most effective choice. It exhibited the highest overall F-score and accuracy, indicating

Label	Accuracy	Precision	Recall	F1_score
Arithmetic_Bit	0.9846	0.0	0.0	0.0
Algorithms_Greedy	0.9846	0.0	0.0	0.0
Arithmetic_Basic	0.8	0.6667	0.4706	0.5517
Loops_Basic	0.7692	0.5455	0.375	0.4444
TerminalIO_Advanced	0.8308	0.4	0.75	0.5217
Arrays_Basic	0.7692	0.375	1.0	0.5455
Pointers_Advanced	0.9692	0.0	0.0	0.0
Conditionals_Flags	0.9077	0.0	0.0	0.0
Structures_Basic	1.0	1.0	1.0	1.0
Char-String_Basic	0.8154	0.3125	0.8333	0.4545
Conditionals_Basic	0.7231	0.4762	0.5882	0.5263
Functions_Advanced	0.7846	0.5185	0.9333	0.6667
Arithmetic_Advanced	0.9538	0.0	0.0	0.0
Conditionals_Switch	1.0	1.0	1.0	1.0
Algorithms_Recursion	0.9385	0.7	0.875	0.7778
Structures_Advanced	0.9385	0.0	0.0	0.0
Algorithms_DP	1.0	1.0	1.0	1.0
Conditionals_Advanced	0.8923	0.2222	1.0	0.3636
Loops_Advanced	0.9077	0.8571	0.8571	0.8571
Arrays_Advanced	0.9538	0.7143	0.8333	0.7692
Char-String_Advanced	0.9077	0.0	0.0	0.0
TerminalIO_Basic	0.8308	0.0	0.0	0.0
Pointers_Basic	0.8923	0.3333	0.75	0.4615
Structures_DS	0.9846	0.6667	1.0	0.8
Arrays_Memory	0.9692	0.6667	1.0	0.8
Algorithms_DC	0.9846	0.0	0.0	0.0
Functions_Basic	0.7385	0.4643	0.8667	0.6047
Loops_Invariants	0.9385	0.0	0.0	0.0
Overall	0.8989	0.4772	0.6647	0.5556
Micro	0.8989	0.5082	0.6791	0.5501

Table 5.5: Late fusion using Random forest and Ranking-style prediction

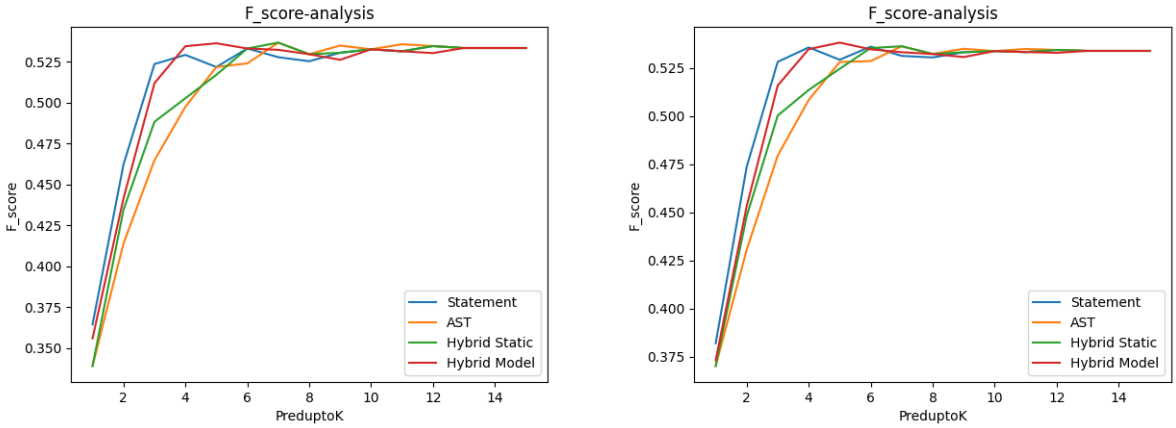
its proficiency in addressing the complexities of the problem domain. Moreover, Random Forest consistently outperformed other models, demonstrating superior micro F-scores and accuracy metrics.

- **Method-wise:** As per the methods we experimented including standard prediction, multilabel ranking and ranking style prediction. The Ranking-style prediction with all values of k shows the best result.This method consistently outperformed alternative approaches, highlighting its effectiveness in accurately predicting relevant labels.



(a) Overall labelwise (b) Micro

Figure 5.1: Late Fusion Accuracy Variation Overall Comparision for different T-value in Ranking style prediction



(a) Overall labelwise (b) Micro

Figure 5.2: Late Fusion F-score Variation Overall Comparsion for different T-value in Ranking style prediction

Bibliography

[1] doT.js - the fastest and concise javascript template engine for Node.js and browsers — olado.github.io. https://olado.github.io/doT/index.html, 2024.

[2] Express - Node.js web application framework — expressjs.com. https://expressjs.com/, 2024.

[3] Home — docs.docker.com. https://docs.docker.com/, 2024.

[4] Index | Node.js v22.1.0 Documentation — nodejs.org. https://nodejs.org/docs/latest/api/, 2024.

[5] MySQL :: MySQL Documentation — dev.mysql.com. https://dev.mysql.com/doc/, 2024.

[6] Debanjan Chatterjee. Intelligent program analysis and program indexing - i, 2022.

[7] Rajdeep Das, Umair Z. Ahmed, Amey Karkare, and Sumit Gulwani. Prutor: A system for tutoring cs1 and collecting student programs for analysis, 2016.

[8] Sharath H. Padmanabha, Fahad Shaikh, Mayank Bansal, Debanjan Chatterjee, Preeti Singh, Amey Karkare, and Purushottam Kar. Priority: An intelligent problem indicator repository. In *Proceedings of the 16th Innovations in Software Engineering Conference*, ISEC '23, New York, NY, USA, 2023. Association for Computing Machinery.

[9] Sharath Hp. Real world deployments of ai-assisted tools for compilation error repair and program retrieval, 2021.

[10] Ayush Sahni. Program repair and retrieval on the prutor platform - i, 2024. Unpublished MTech thesis.

[11] Jeet Sarangi. Program repair and retrieval on the prutor platform - ii, 2024. Unpublished MTech thesis.

[12] Preeti Singh. Intelligent program analysis and program indexing - ii, 2022.