Project Report: DejaCode – A Code Plagiarism Checker

PRAGYAN DAS (B24PH1019)

OMPRAKASH (B24CI1038)

HRUJUL MENDHE (B24EE1026)

PRIYAM PATEL (B24CS1058)

SARAIYA VED GAURAV (B24EE1069)

1. Introduction:

Plagiarism in code isn’t always obvious — a few variable name changes or tweaks in formatting can often hide it well. That’s where DejaCode comes in. It’s a tool we built to detect similarities in code files, especially helpful in academic settings to catch copied assignments. The idea was to create something simple, effective, and interactive — with a user-friendly interface and a solid backend doing the heavy lifting.

2. Interactive UI with Raylib:

We used Raylib to build the graphical interface. It's a lightweight C library that made it easy to create menus, buttons, and file selection dialogs. Through this interface, users can:

Select the files they want to compare.

Get a clean visual output showing the similarity percentage.

Interact with the tool without needing to use the command line.

Raylib helped us keep things visually appealing and easy to use — even for someone who’s not super tech-savvy.

---

3. How the Backend Works:

Behind the scenes, DejaCode goes through a few key steps to analyze the code:

a. Reading and Cleaning the Code:

First, we read the code files using C's file handling functions. But we don't just blindly compare everything — we clean the code up first. That means:

Ignoring comments (both single-line and multi-line).

Skipping blank lines and extra spaces.

Focusing only on the actual logic of the code — things like function calls, keywords, and operators.

This helps us avoid false positives caused by formatting or comment differences.

b. Storing Code Efficiently in a Linked List:

Once the code is cleaned, we break it down into tokens and insert them into a linked list. The reason we used a linked list instead of a large array because a linked list can scale to large sizes very well compared to array. In case of an array, we would have to resort to using realloc each time the array fills up.

c. Storing Unique elements in a Hash Table:

Since many words repeat throughout a code or an essay and for calculating union and intersection, we need unique elements so we use a hash table for it. Basically, we iterate over the linked list and whenever we find an element not in hash table we would add the element to the hash table. After getting unique elements of both sets (linked lists) we get unique and intersection by comparing the elements of two hash tables.

---

4. Measuring Similarity with Jaccard Index:

Once both files are processed and stored, we compare them using the Jaccard Similarity formula:

Jaccard Similarity = (Common tokens)/(Total unique tokens)

Basically, we check how many tokens the two files have in common versus how many unique ones they contain altogether. This gives us a percentage — the higher the percentage, the more similar the code is.

---

5. Wrapping It Up:

With DejaCode, we set out to build a simple, smart plagiarism checker — and we're happy with how it turned out. The UI is clean and easy to use, the backend is efficient and accurate, and the results are reliable. It's a project that blends practical programming with real-world usefulness, especially for educators and students alike.

CONTRIBUTIONS:

SARAIYA VED GAURAV	:	Frontend designing of “About us” page. Coding and implementation of introduction audio.
Omprakash	:	Frontend designing of “Home page” and code cleaning.
Priyam Patel	:	Frontend designing of “ Plagiarism Checker” page and file handling system.
Hrujul Mendhe	:	Ideation and backend of project.
Pragyan Das	:	Ideation and backend of project and code cleaning.