Bilkent University

Department of Computer Engineering

CS 491

Senior Design Project

CODED.

2023-24 Fall Semester

Project Specification Document

Project ID: T2322

Project Team Members:

Zeynep Hanife Akgül	(22003356)
Beyza Çağlar	(22003394)
Selin Bahar Gündoğar	(22001514)
Emre Karataş	(22001641)
Zeynep Selcen Öztunç	(21902941)

Project Advisor: Halil Altay Güvenir

Instructors: Atakan Erdem, Mert Bıçakçı

Innovation Expert: Bora Güngören

17 November 2023

Table of Contents

1. Introduction **3**
1.1. Description 3
1.2. High-Level System Architecture & Components of Proposed Solution 4
1.3. Constraints 6
1.3.1. Implementation Constraints 6
1.3.2. Economic Constraints 7
1.3.3. Coding Language Constraints 7
1.3.4. Ethical Constraints 8
1.4. Professional and Ethical Issues 8
2. Design Requirements **9**
2.1. Functional Requirements 9
2.2 Non-Functional Requirements 10
2.1.1. Usability 10
2.1.2. Reliability 11
2.1.3. Performance 11
2.1.4. Supportability 11
2.1.5. Scalability 12
2.1.6. Security 12
3. Feasibility Discussions **13**
3.1. Market & Competitive Analysis 13
3.1.1. Similarity Check 13
3.1.2. Clean Code Evaluation 14
3.1.3. Chatbot Tutor 14
3.1.4. Test Case Runner 15
3.2. Academic Analysis 16
3.2.1. Large Language Models for Coding Environment 16
4. Glossary **18**
5. References **21**

1. Introduction

1.1. Description

Bilkent University is currently facing a problem of finding lab tutors for the CS (Computer Engineering) and CTIS (Information Systems and Technologies) labs. Most students have questions about the lab assignments or have trouble debugging a specific part of their code during their lab periods. Thus, these students ask their questions to tutors and teacher assistants (TAs). Unfortunately, only 2-3 TAs and 1-2 tutors can be present during each lab session. The CS courses do not have trouble finding TAs for lab sessions as there is a graduate program in the CS department; though, the CTIS department has to rely on successful students to volunteer to fill the tutor vacancies for TA roles as the CTIS department does not have a graduate program. Furthermore, tutors for both CS and CTIS courses are selected among volunteering students who display proficiency in the course's coding language, and most tutors cannot stay throughout all of the lab hours due to their schedules. Even when they are able to do so, the student-to-tutor ratio is around 1/30 [1]. Henceforth, tutors do not have the time to pay attention to each student. Furthermore, hundreds of non-STEM (Science, Technology, Engineering, and Mathematics) students take mandatory coding courses offered by the CS department every semester, and non-STEM students are far more likely to ask trivial questions about the lab compared to STEM students. Therefore, there is an urgent lab tutor needed in the CS and CTIS departments. Nevertheless, this crisis can be solved with our application called "*CODED.*".

CODED. is a website application that assists students, tutors, TAs, and instructors in CS and CTIS labs throughout the lab process. The application answers student questions about the lab assignments through its chatbot property. It leads students in the coding process by guiding them to write clean code and checking for any code smells, which is "any characteristic in the source code of a program that possibly indicates a deeper problem."[2]. Followingly, the completed lab assignments are then uploaded to the application, where these assignments are tested for plagiarism based on the criteria that the instructor selects. The instructor can compare the code to a specific section, the whole class, code on Github, past codes, or check for AI (Artificial Intelligence) generated code similarity. Furthermore, the code will be passed by test cases provided by the instructor or created by the chatbot. The tool will grade the code based on the given criteria in a fair manner by looking at the main logic of the code and running every method separately if needed to check for correctness. Furthermore, the code will be statically checked for partial point calculations. The instructors will be given an insight into how the grading was tested and which sections of the code led to errors to take off points. The students will be able to view their grading report if the instructors allow it on the application, and students will be able to refute their generated grades. Therefore, the application will tremendously shorten the time it takes to grade the code and lighten the workload of tutors, TAs, and teaching assistants.

1.2. High-Level System Architecture & Components of Proposed Solution

In our system, there are 4 main components: Test Case Runners component, Chatbot component, Code Quality Checker component, and Plagiarism Checker component. When users (instructors, students, and tutors) use the system, the components make calls to servers accordingly. There is also the database component, which uses PostgreSQL to save each information related to the system. The users, the instructor, students, and tutors use their devices to connect to the system and to be able to use its features. Their browsers make requests to the servers of the system's main components to be able to use their features. To see these relationships clearly, a high-level system architecture diagram is given below:

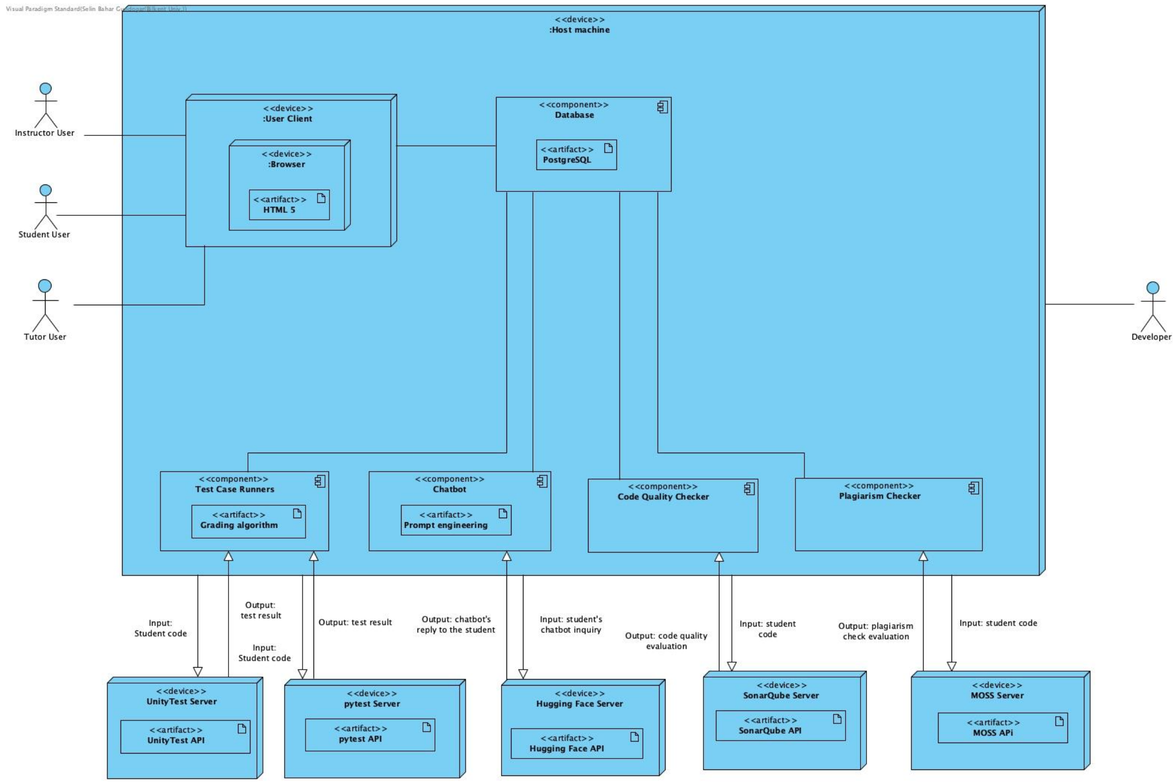

Figure 1: High Level System Architecture Diagram

The Test Case Runners component is called when students upload their code to the system. The component has its own grading algorithm since partial points are given in labs, and the grading is not binary. In order to grade the labs, the component makes calls to different APIs (Application Programming Interface), UnityTest for C labs, and pytest for Python labs. It sends the students' codes to these APIs and obtains the test results, such as how many tests the code passed. Then, according to the specialized grading algorithm that the component has, it finalizes the grades for the students. These grades are saved to the database and then returned to the user's browser for viewing.

The Chatbot component is called when the students want to ask questions about the lab. Inside the component, prompt engineering will be used so that the chatbot only guides the student in their code process and does not actually give the solutions to the lab questions. When the chatbot component inputs students' chatbot inquiries, it sends this input to the

Hugging Face API. Then, it gets the chatbot's reply to the student as an output from the API, and the reply is then immediately returned to the student's browser for viewing.

The Code Quality Checker component is called when the code the student uploads is to be checked in terms of quality. After it receives the code of the student, it calls on the SonarQube API in order for that code to be evaluated. Then, it receives the code evaluation from the API, and the resulting quality assessment is saved to the database returned to the user's browser.

The Plagiarism Checker component compares the student codes with each other in order to understand whether an act of plagiarism has occurred. In order to do that, it sends the students' code to the MOSS API. From there, it obtains information on the similarity rates of each student code. The similarity results are saved to the database and then returned to the user's browser.

1.3. Constraints

CODED. aims to help students, TAs, tutors, and instructors as effectively as possible while minimizing implementation, economic, and ethical constraints.

1.3.1. Implementation Constraints

- GitHub and Jira will be used to track the deadlines, issues, and code.
- React.JS framework will be used in the frontend.
- Spring Boot framework and Python (for chatbot development) will be used for the backend.
- PostgreSQL hosted by Amazon RDS server will be used for the database system.
- GitHub Pages will be used to host the application.
- SonarQube will be used to make clean code and code convention checks.

- CODE-LLAMA-7B model with a fine-tuned version developed by the *CODED.* team will be used for the chatbot.
- MOSS will be used in checking the codes uploaded by students for plagiarism.
- pytest will be used to run test cases on Python codes.
- UnityTest or CUnit will be used to run test cases on C codes. However, as there are many options in the market, other APIs or frameworks may be used due to compatibility issues.

1.3.2. Economic Constraints

Our project requires several external libraries, frameworks, and models. Therefore, our group has opted to use as many open-source frameworks as possible. Plenty of viable models for chatbot development exist, but due to economic restrictions, we decided to use open-source models, specifically the CODE-LLAMA model family, for this project. Since these models are complex and computationally expensive, we decided to use the simplest model, CODE-LLAMA-7B. It is worth noting that CODE-LLAMA-7B runs on a single CPU, which gives us the flexibility to run this model on free servers (Amazon Sagemaker and Google Colab) or even on our personal computers [3]. SonarQube's community edition will be used as it is free and supports C and Python languages. Furthermore, all other tools used in the project will be open source or free tools.

1.3.3. Coding Language Constraints

In accordance with Bilkent University CS and CTIS department chairs, our application will support C and Python coding languages to support CTIS 150, CTIS 151, CS115, and CS125 courses. CTIS 150 and CTIS 151 courses are the introductory coding courses for first-year CTIS students taught in the C language. CS 115 and CS125 courses are Python courses that teach non-CS students coding basics.

1.3.4. Ethical Constraints

All interactions and data collected from the users should be handled within data protection laws such as the General Data Protection Regulation (GDPR). To enforce the law, the users must be well informed about the application's limitations and scope before registration. The program will be utilizing personal data and storing said data in a database. Therefore, the users should be informed about how and why this data will be used and kept. Furthermore, the chatbot feature will be an AI model talking to the users, so the users should be warned about the AI nature and know that they will be talking with a machine rather than another human. Additionally, the users should be aware of the chatbot's limitations and capabilities. Even though the mechanisms to prevent harmful, disrespectful, and biased answers are handled within the AI models we will be using, the biases in the chatbot's responses must be minimized and tested before presenting the product to the users.

1.4. Professional and Ethical Issues

Our application is designed to lead students in their lab work to improve their learning curve and ease the workload of tutors, TAs, and instructors; though, this goal will be carefully handled to avoid inconveniences both from the students and the instructors' side. Firstly, all the questions asked to the chatbot will be confidential, and no data will be linked to an individual. Moreover, inappropriate questions asked to the chatbot will be answered appropriately, prompting the student to enter expected questions about the lab/their code. Furthermore, all the code uploaded to the application will be graded based on the test cases provided by the instructor and/or the chatbot. These test cases will be run, and the results will be compared to the expected results. Though the grading of these codes will not be just based on the result of the outcome, the application will also analyze each method provided to test the logic of the written code to consider partial points. While the application will be grading the students with an estimated grade, the instructors will be responsible for the actual grading of the students. The application will let the students view how their code was graded and which test cases the code failed/passed, along with the explanation of partial points.

Moreover, the students will be able to access the application through a personalized entrance code of the course. This will ensure that students are enrolled in the correct class.

2. Design Requirements

2.1. Functional Requirements

- The user must register/login via their email address and password.
- The user must specify their user type as an instructor, teaching assistant (TA), or student during registration.
- The system will utilize 2FA authentication during registration, where an email will be sent to the email address provided by the user.
- The user can log out of the system.
- The students must enroll in a class via the class code or link provided by the instructor or TA to use all features except the chatbot.
- The students can ask questions to the chatbot as long as they are logged in.
- The students can test their codes in the lab with a restricted reach to limited test cases. However, they cannot see the contents of the test cases, only the results.
- The students must submit their codes during the hours specified by the instructor or TA for evaluation.
- The students can use the test case generator and chatbot outside of lab hours for practice. (Optional)
- The students can view their past submissions and grades.
- The instructors can upload new lab assignments and specify lab hours.

- The instructors can create new classes and sections for classes.
- The instructors and TAs can assign test cases for the final evaluation of the lab or for students to test their codes.
- The instructors and TAs can adjust the test cases generated by the system.
- The chatbot can read files, meaning that the students can send their code as a file to the chatbot to ask questions to the chatbot.
- The code analyzer can accept files as well as copy-pasted text.
- The system can generate test cases on demand.
- The chatbot cannot directly give the solution to the problem asked by the student.
- The system can grade the student’s code based on customized test cases.
- The system gives partial grading to student’s code using unit tests.
- The system must check the codes uploaded by students for similarities between other students or codes in the database.
- The system must not allow students to use the test case generator during lab hours.
- The system will give feedback after running test cases for the instructors and TAs.

2.2 Non-Functional Requirements

2.1.1. Usability

The website will have a consistent user interface to ensure straightforward navigation and to make it effortless for users to have a plain understanding of the fundamentals of the program. One of the target audiences of *CODED.* is beginner students, so an intuitive and user-friendly interface is highly significant. Every page has a navigation bar, so reaching any main feature will require only two operations. Other than reaching the main tools, any system feature will be accessed within at most four clicks or operations. Chatbot, one of the most significant features, will be accessible and easily seen on any page as it will be on a slider view. This focus on user-centric design aims to make the website user-friendly and engaging.

2.1.2. Reliability

The system will be available to all users at all times, including lab hours and practicing hours, with minimal downtime. More specifically, the website will maintain an availability of at least 99.8% days over a year, with a total downtime of no more than 9 hours annually. High-security databases will prevent data loss and ensure the safe storage of personal data. The website will provide clear error messages to inform users in cases of problems. Errors will be logged for analysis and debugging purposes.

2.1.3. Performance

The system will be responsive even during most busy periods like lab hours. It will be able to handle the usage of multiple users without a significant performance slowdown. The system's response time will be under 0.5 seconds for any standard user action and transactions and under 2 seconds for more complex tasks, even when the website experiences a peak usage time. Regarding the fact that a standard lab has at most 50 students, the system will be able to maintain its performance with at least 200 users using the application simultaneously. Several APIs will be used during the development process, so the system’s goal is to ensure that the third-party API calls have a latency of less than 1 second. Furthermore, it is planned that the chatbot will respond to any input in less than 5 seconds.

2.1.4. Supportability

The website will be a modular system to easily integrate new features. This design will allow changes to be made without disrupting the entire system. More specifically, the integration of at least ten updates, not including bug fixes and new features per year, will be supported without causing any significant corruption to the existing system. The system will be maintained as long as it is being used, and the maximum response time for user support inquiries related to new and existing features will be 18 hours. Furthermore, the website will support at least five different web browsers and any device that is able to use web browsers to surf through websites.

2.1.5. Scalability

We target a large audience using the system simultaneously since students participate in the lab during the same hours. Thus, the significance of the scalability increases so that the application will easily adapt and will be able to expand in response to growing demands. As the user volume increases, the performance and responsiveness of the system will remain consistent. For this purpose, we set a threshold that the website will accommodate at least 50% of the growing demand regarding the number of users and content volume. The importance of scalability will increase in the future as the system will be developed regarding the possibility of usage from different schools or even companies, growing the volume content of the application.

2.1.6. Security

The system will store limited personal information: names, Bilkent school IDs, and email addresses. While it is crucial to keep this information safe, the system should also be secure to keep the students' work and grades. To ensure the security of this data, we will employ encrypted databases, utilizing industry-standard encryption algorithms to safeguard sensitive information. The system will also implement several actions to strengthen security. User access to the system will be strictly controlled, limiting access to authorized personnel only: enrolled students, instructors, teaching assistants, and tutors. Furthermore, in the cases of security breaches, the system will have an incident response time of up to 30 minutes.

3. Feasibility Discussions

3.1. Market & Competitive Analysis

CODED. has four main features: a similarity check tool for students' codes, a clean code evaluator, a chatbot tutor, and a test case runner. Upon extensive research, we found some products that individually accomplish what *CODED.* offers as a complete package. *CODED.* is set out to be a teacher for code learners worldwide with broad support for various aspects of code. There is not any software that offers monitoring throughout the coding process that will also expedite learning. Moreover, *CODED.* aims to provide customizable solutions to its users. For instance, an instructor can choose the conventions used in that course and adjust the clean code evaluator accordingly. Thus, although existing products in the market will be utilized, we will customize them to align with our specific objectives.

3.1.1. Similarity Check

Plagiarism is a sensitive issue at Bilkent University and in any academic work. Therefore, it is crucial that *CODED.* has a feature supporting plagiarism detection among students and sources on the Internet. Since creating such an extensive database is complicated and costly, *CODED.* will implement existing tools for the similarity check feature. Three possible tools may be used in the program: Codequiry, Copyleaks, and Moss. Codequiry specializes in code plagiarism and supports peer and Web checks while detecting code generated by ChatGPT [4]. Copyleaks can recognize writing and source code generated by AI, such as ChatGPT and Bard, even if a human has modified it [5]. MOSS is a tool the Bilkent University CS department already uses for code comparison. It highlights similar parts in two texts and supports over 20 programming languages [6]. *CODED.* will use one or more of these tools to ensure reliable plagiarism detection.

3.1.2. Clean Code Evaluation

One of the target audiences of *CODED.* is students who are taking introductory courses in programming. Given their beginner status, it's understandable if their solutions are functional but not necessarily the most efficient. A problem may have infinite coding solutions; however, only several of them are the most efficient ones, which may not be so obvious to beginner-level students. This problem highlights the importance of clean code evaluation. Several tools exist on the market to check code quality and security. For example, SonarQube supports 27 languages while doing static code analysis, code security checks, and finding typos, dead code, and potential vulnerabilities [7]. A very similar tool is PVS-Studio, but it supports fewer programming languages than SonarQube [8]. *CODED.* aims to incorporate these tools to do clean code evaluation on pieces of code written by beginners.

3.1.3. Chatbot Tutor

During lab hours, the students can ask tutors questions, and the tutors are not allowed to give the solution of the problem to the students directly. However, since there are only 1 or 2 tutors per 60 students, the students may wait a long time to ask a simple question [1]. The chatbot tutor aims to quickly and effectively answer the students' questions as a human tutor would. So, it is forbidden to reveal the correct answer to the students, unlike current popular generative artificial intelligence (GenAI) models such as ChatGPT. The market analysis for such a tool led us to Khanmigo, a chatbot developed by Khan Academy to assist students in the courses they offer [9]. Khanmigo is powered by GPT-4 and fine-tuned with Khan Academy's courses [10]. For instance, the students can ask questions about a mathematical problem or equation and get hints about the correct solution based on their solution steps [9]. Khanmigo is not a tool created explicitly for coding as opposed to the idea behind *CODED.,* even though it can assist with simple code. The chatbot tutor feature in *CODED.* aims to alleviate the workload of tutors, assistants, and instructors while also encouraging students to learn. The bot should not reveal the correct answer, especially generate code, even if it is asked to by the students. Instead, it may give instructive hints for the next correct step or lead students to find their mistakes. It may ask the student questions to learn more about their logic for better guidance. In the end, the students should have completed the task on their own by discussing with the chatbot as if they were asking for help from a tutor.

3.1.4. Test Case Runner

Any lab work, quiz, or assignment has an expected output for given inputs. These inputs and regarded outputs are referred to as test cases, and a work of code is often graded by running test cases. There are several tools to run test cases on codes as long as inputs and expected outputs are provided. These tools compile and run the code with given inputs and check whether the code output matches the one provided by the user. For different languages, different tools are available, as each programming language has its unique way of compilation and running. For example, pytest is a common test case runner API for pieces of code written in Python, whereas UnityTest or CUnit is often used for C [11, 12, 13]. In addition to the ability of these tools to run test cases, unit tests are also supported. More specifically, if the whole code gives compiler error, but some part of it still works, the developer gets credit for it. Labs at Bilkent University are not graded binary such that student gets total points if the code runs correctly and zero otherwise. Instead, partial points are given for any part done and working. To accomplish this, unit test features of existing test case runner tools will be used but with several customizations such that desired units will cause higher points to be lost or gained and vice-versa.

3.2. Academic Analysis

3.2.1. Large Language Models for Coding Environment

Large Language Models (LLMs) are increasingly used in many applications in code evaluation and generation areas since their proficiency in natural language allows understanding prompts and acting on various tasks according to these prompts [3]. A distinguished demonstration of LLMs is the interaction with computer systems, such as program synthesis from natural language, debugging, and code completion [14].

For the *CODED.* project, it is decided to use the ***CODE LLAMA*** models, a family of LLMs for code generation and infilling derived from *LLAMA 2* [15]. Unlike the majority of Language Models for code generation like *AlphaCode* [16], *InCoder* [17], and *StarCoder* [18], which are solely trained on code, *Codex* [19] underwent fine-tuning from a general language model. Similarly, the baseline occurs from the foundation models listed above, *LLAMA 2* [15], pre-trained on the general-purpose text and code data.

Prompt: In Bash, how do I list all text files in the current directory (excluding subdirectories) that have been modified in the last month

Response: You can use the 'find' command in Bash to list all text files in the current directory (excluding subdirectories) that have been modified in the last month. The command would be:

```
find . -maxdepth 1 -type f -mtime -30 -name "*.txt"
```

Here's a breakdown of the command:
* '.' specifies the current directory.
* '-maxdepth 1' tells 'find' to only search the current directory and not any subdirectories.
* '-type f' tells 'find' to only search for files (not directories or other types of files).
* '-mtime -30' tells 'find' to only search for files that have been modified in the last 30 days.
* '-name "*.txt"' tells 'find' to only search for files with the '.txt' extension.
Note that the '-mtime' option uses the number of days since the file was last modified, so '-mtime -30' means "modified in the last 30 days". If you want to search for files modified in the last month, you can use the '-mtime -30' option.

Figure 2: Example of response of CODE LLAMA-INSTRUCT (34B) [15].

A family of code-specialized *LLAMA 2* models with three main variants that have been released in three sizes (**7B**, **13B**, and **34B** parameters):

- ***CODE-LLAMA:*** a foundational model for code generation tasks,

- ***CODE-LLAMA-PYTHON:*** a version specialized for Python,
- ***CODE-LLAMA-INSTRUCT:*** a version fine-tuned with human instructions and self-instruct code synthesis data.

It is worth noting that all models were released on 13 September 2023 [15].

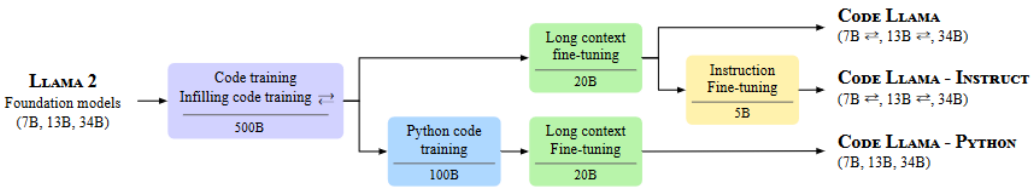

Figure 3: CODE-LLAMA Specialization Pipeline [15].

During the initial training phase, *CODE-LLAMA* is trained on 500 billion tokens, starting with the 7 billion, 13 billion, and 34 billion token versions of *LLAMA 2*. As depicted below in *Table 1*, the training primarily involves a near-deduplicated dataset of publicly accessible code. Additionally, 8% of the sample data is sourced from natural language datasets associated with the code, encompassing numerous discussions about code and code snippets found within natural language questions or responses. A minor portion of the batches are sampled from a natural language dataset to maintain the model's proficiency in understanding natural language. The data is tokenized using byte pair encoding [20], utilizing the same tokenizer employed in *LLAMA* and *LLAMA 2* [15].

Dataset	Sampling prop.	Epochs	Disk size
Code Llama (500B tokens)			
Code	85%	2.03	859 GB
Natural language related to code	8%	1.39	78 GB
Natural language	7%	0.01	3.5 TB
Code Llama - Python (additional 100B tokens)			
Python	75%	3.69	79 GB
Code	10%	0.05	859 GB
Natural language related to code	10%	0.35	78 GB
Natural language	5%	0.00	3.5 TB

Table 1: Training dataset of CODE-LLAMA and CODE-LLAMA-PYTHON [15].

4. Glossary

- **Artificial Intelligence (AI)**: An acronym that refers to the simulation of human intelligence in machines that are programmed to think and act like humans. This includes capabilities such as learning, reasoning, self-correction, and problem-solving.[21].
  - **Acronym**: AI
- **Application Programming Interface (API)**: A set of protocols and tools that allow different software applications to communicate with each other, enabling the exchange of data and functionalities [21].
  - **Acronym**: API
  - **Example**: The Hugging Face API allows developers to access and utilize a variety of machine learning models for natural language processing tasks, such as text generation, translation, and sentiment analysis.
- **Computer Science (CS)**: The study of computers and computational systems, focusing on algorithms, software design, and the theoretical underpinnings of computing [21].
  - **Acronym**: CS
- **General Data Protection Regulation(GDPR)**: It is a regulation in EU law on data protection and privacy in the European Union and the European Economic Area [21].
  - **Acronym**: GDPR
- **Information Systems and Technologies (CTIS)**: An interdisciplinary field that combines computing technology with business practices to facilitate data storage, analysis, and communication within organizations [21].
  - **Acronym**: CTIS

- **Large Language Models (LLM)**: Machine learning models trained on extensive datasets to understand and generate human-like text based on the input they receive [21].
    - **Acronym**: LLM
    - **Example**: GPT-4 is an example of a large language model used for various natural language processing tasks.
- **Measure of Software Similarity (MOSS)**: An automated system for detecting plagiarism in software assignments. It is commonly used in academic settings to compare student submissions and identify similarities that may indicate copying [21].
    - **Acronym**: MOSS
    - **Example**: Many universities use MOSS to maintain academic integrity in computer science courses by checking if students have plagiarized code for their assignments.
- **Science, Technology, Engineering, and Mathematics (STEM)**: An interdisciplinary approach to learning that integrates these four disciplines into a cohesive curriculum focused on real-world applications [21].
    - **Acronym**: STEM
    - **Example**: Many educational initiatives aim to boost student interest and competence in STEM fields due to their importance in the modern workforce.
- **Teaching Assistant (TA)**: An individual, often a graduate or undergraduate student, assists a professor in instructional responsibilities, including grading, answering student queries, and occasionally teaching [21].
    - **Acronym**: TA

- **Two Factor Authentication(2FA)**: A security process that requires users to provide two different forms of identification before gaining access to an account or system. This adds an extra layer of security compared to single-factor authentication [21].
    - **Acronym**: 2FA

5. References

[1] K. Emre, "Discussion on CS labs in Bilkent University," Academic interview, Oct. 28, 2023.

[2] "Code smell," *Wikipedia*. [Online]. Available: https://en.wikipedia.org/wiki/Code_smell. [Accessed: Oct. 28, 2023].

[3] OpenAI, "GPT-4 technical report," *arXiv.org*. [Online]. Available: https://arxiv.org/abs/2303.08774. [Accessed: Oct. 21, 2023].

[4] "Codequiry," *Codequiry*. [Online]. Available: https://codequiry.com/. [Accessed: Sep. 10, 2023].

[5] "Copyleaks," *Copyleaks*. [Online]. Available: https://copyleaks.com/. [Accessed: Sep. 10, 2023].

[6] "MOSS: A System for Detecting Software Similarity," *Stanford University*. [Online]. Available: https://theory.stanford.edu/~aiken/moss/. [Accessed: Sep. 10, 2023].

[7] "Code quality tool & secure analysis with SonarQube," *Sonar*. [Online]. Available: https://www.sonarsource.com/products/sonarqube/. [Accessed: Oct. 28, 2023].

[8] "Studio is a static analyzer on guard of Code Quality, security (SAST), and code safety," *PVS*. [Online]. Available: https://pvs-studio.com/en/pvs-studio/. [Accessed: Oct. 28, 2023].

[9] "World-class AI for education," *Khan Academy*. [Online]. Available: https://www.khanacademy.org/khan-labs#khanmigo. [Accessed: Sep. 11, 2023].

[10] E. Ofgang, "What Is Khanmigo? The GPT-4 Learning Tool Explained by Sal Khan," *Tech & Learning*, Jul. 7, 2023. [Online]. Available: https://www.techlearning.com/news/what-is-khanmigo-the-gpt-4-learning-tool-explained-by-sal-khan. [Accessed: Sep. 11, 2023].

[11] "Helps you write better programs" *pytest*. [Online]. Available: https://docs.pytest.org/en/7.4.x/. [Accessed: Oct. 28, 2023].

[12] "Unit testing tutorials," *EmbeTronicX*. [Online]. Available: https://embetronicx.com/unit-testing-tutorials/. [Accessed: Oct. 28, 2023].

[13] "Cunit," *CUnit Home*. [Online]. Available: https://cunit.sourceforge.net/index.html. [Accessed: Oct. 28, 2023].

[14] Y. Sun et al., “A length-extrapolatable transformer,” *arXiv.org*. [Online]. Available: https://arxiv.org/abs/2212.10554. [Accessed: Oct. 21, 2023].

[15] B. Rozière et al., “Code llama: Open foundation models for code,” *arXiv.org*. [Online]. Available: https://arxiv.org/abs/2308.12950. [Accessed: Oct. 21, 2023].

[16] Y. Li et al., “Competition-level code generation with AlphaCode,” *arXiv.org*. [Online]. Available: https://arxiv.org/abs/2203.07814. [Accessed: Oct. 21, 2023].

[17] D. Fried et al., “Incoder: A generative model for code infilling and synthesis,” *arXiv.org*. [Online]. Available: https://arxiv.org/abs/2204.05999. [Accessed: Oct. 21, 2023].

[18] R. Li et al., “Starcoder: May the source be with you!,” *arXiv.org*. [Online]. Available: https://arxiv.org/abs/2305.06161. [Accessed: Oct. 21, 2023].

[19] M. Chen et al., “Evaluating large language models trained on code,” *arXiv.org*. [Online]. Available: https://arxiv.org/abs/2107.03374. [Accessed: Oct. 21, 2023].

[20] R. Sennrich, B. Haddow, and A. Birch, “Neural machine translation of rare words with Subword units,” *ACL Anthology*. [Online]. Available: https://aclanthology.org/P16-1162/. [Accessed: Oct. 21, 2023].

[21] OpenAI, "ChatGPT [Large language model]," 2023. [Online]. Available: https://chat.openai.com.u