# BTP-1 Report
# Subjective Autograding of Programming Assignments

Margav Savsani
200050072

December 2, 2023

## Contents

# 1 Introduction

Automated assessment of programming assignments traditionally relies on test cases to ensure functional correctness. However, these methods often lack the qualitative feedback necessary for enhancing students' learning experiences. In programming education, automated assessment mechanisms play a pivotal role in grading student submissions. While traditional approaches prioritize functional correctness through test cases, they often miss providing essential qualitative feedback crucial for fostering a deeper understanding of programming concepts. This research addresses the need for comprehensive evaluation by proposing an AI-assisted feedback and evaluation system for programming assignments.

The project is motivated by observed limitations in existing models like CodeBERT[2] and GraphCodeBERT[3], which struggled with long inputs and necessitated separate models for distinct tasks. To overcome these challenges, our focus turns to Large Language Models (LLMs). Specifically, we explore the capabilities of Code Llama - Instruct[4], a state-of-the-art LLM designed for code-related tasks. This shift towards LLMs offers the advantage of handling extensive context lengths and potentially grading various types of problem statements with a unified model.

In summary, this research aims to push the boundaries of automated programming assignment grading, emphasizing qualitative feedback, and fostering a more educational and comprehensive evaluation process.

# 2 Background and Motivation

The landscape of automated programming assignment assessment has undergone transformative changes, primarily relying on traditional test cases to validate functional correctness. While effective in certifying code functionality, this method falls short in providing essential qualitative feedback necessary for a deeper comprehension of programming concepts. This inadequacy becomes pronounced in the realm of Massive Open Online Courses (MOOCs), where scalable and informative autograding systems are imperative.

The surge in online computer science education, exacerbated by the COVID-19 pandemic, has underscored the need for autograding systems capable of handling large student populations. Conventional autograders, often anchored in simplistic test case evaluations, encounter limitations that contemporary deep learning-based autograders aim to overcome. Notably,in a previous study[1], the GraphCodeBert model, demonstrated the potential to consider the actual content within source code, expanding beyond the constraints of test case-centric assessments.

However, a persistent bottleneck in the development of sophisticated models is the scarcity of well-labeled training data. Motivated by observed limitations in existing autograding models, such as CodeBERT and GraphCodeBert, this research heralds a paradigm shift towards LLMs. The exploration of Code Llama - Instruct, designed explicitly for code-related tasks, represents a pivotal shift.

In addition to the shift towards LLMs, this research delves into the symbiotic relationship between code generation and grading tasks. Innovative approaches, such as extracting pseudocodes from research papers to generate corresponding code implementations, are explored. This multifaceted strategy aims to augment the LLM's proficiency in assessing code quality beyond functional correctness.

# 3 Key Components

Our approach to subjective autograding of programming assignments involves several key components, each playing a crucial role in the development and deployment of an effective AI-assisted feedback and evaluation system.

## 3.1 Grading Submission

The **Grading Submission** component is designed to retrieve a 'suitable' set of (Problem Statement, Rubric Criterion, Submission, Grade) from an existing database. A 'suitable set' is defined as having the same rubric criterion and offering good diversity in grades. This set serves as few-shot examples for the Large Language Model (LLM) during the grading process. In situations where entirely new programming assignments or criteria emerge, instructors can manually grade a few submissions to provide this essential set.

### 3.2 Rubric Criteria Suggestion

The **Rubric Criteria Suggestion** component aims to assist instructors by providing suggestions for rubric criteria based on a given Problem Statement. Instructors input the Problem Statement into the model, and the system generates recommendations for relevant rubric criteria, such as well-commented code, code complexity, and adherence to best practices. This feature facilitates the efficient creation of grading rubrics, and also helps to exploit the knowledge of pretrained models for those rubrics.

### 3.3 Model Survey

To identify the most suitable starting point for our research, we conducted a comprehensive model survey of available open-source Large Language Models (LLMs). Initially, Llama2[5] was chosen, but a shift to Code Llama - Instruct occurred during the course. This component involves the evaluation, selection, and adaptation of LLMs to meet the specific requirements of our subjective autograding system. Below is a brief overview of the comparison of the LLMs

| Model | Size | HumanEval | | | MBPP | | |
|---|---|---|---|---|---|---|---|
| | | pass@1 | pass@10 | pass@100 | pass@1 | pass@10 | pass@100 |
| code-cushman-001 | 12B | 33.5% | - | - | 45.9% | - | - |
| GPT-3.5 (ChatGPT) | - | 48.1% | - | - | 52.2% | - | - |
| GPT-4 | - | **67.0%** | - | - | - | - | - |
| PaLM | 540B | 26.2% | - | - | 36.8% | - | - |
| PaLM-Coder | 540B | 35.9% | - | 88.4% | 47.0% | - | - |
| PaLM 2-S | - | 37.6% | - | 88.4% | 50.0% | - | - |
| StarCoder Base | 15.5B | 30.4% | - | - | 49.0% | - | - |
| StarCoder Python | 15.5B | 33.6% | - | - | 52.7% | - | - |
| StarCoder Prompted | 15.5B | 40.8% | - | - | 49.5% | - | - |
| Llama 2 | 7B | 12.2% | 25.2% | 44.4% | 20.8% | 41.8% | 65.5% |
| | 13B | 20.1% | 34.8% | 61.2% | 27.6% | 48.1% | 69.5% |
| | 34B | 22.6% | 47.0% | 79.5% | 33.8% | 56.9% | 77.6% |
| | 70B | 30.5% | 59.4% | 87.0% | 45.4% | 66.2% | 83.1% |
| Code Llama | 7B | 33.5% | 59.6% | 85.9% | 41.4% | 66.7% | 82.5% |
| | 13B | 36.0% | 69.4% | 89.8% | 47.0% | 71.7% | 87.1% |
| | 34B | 48.8% | 76.8% | 93.0% | 55.0% | 76.2% | 86.6% |
| Code Llama - Instruct | 7B | 34.8% | 64.3% | 88.1% | 44.4% | 65.4% | 76.8% |
| | 13B | 42.7% | 71.6% | 91.6% | 49.4% | 71.2% | 84.1% |
| | 34B | 41.5% | 77.2% | 93.5% | 57.0% | 74.6% | 85.4% |
| Unnatural Code Llama | 34B | **62.2%** | **85.2%** | **95.4%** | **61.2%** | **76.6%** | 86.7% |
| Code Llama - Python | 7B | 38.4% | 70.3% | 90.6% | 47.6% | 70.3% | 84.8% |
| | 13B | 43.3% | 77.4% | 94.1% | 49.0% | 74.0% | 87.6% |
| | 34B | 53.7% | 82.8% | 94.7% | 56.2% | 76.4% | **88.2%** |

Figure 1: LLM Comparison

# 4 CS 293 Dataset

The CS 293 Dataset is a curated dataset derived from the Lab Assignments of the CS293(Data Structures Lab) course offering for the batch of 2025 of IIT Bombay. This dataset is a crucial resource for training and evaluating the subjective autograding system.

### 4.1 Dataset Overview

The dataset encompasses the following components for a total of 12 assignments, each with an average of around 180 submissions:

- **Problem Statement**: The statement describing the programming task assigned to students.
- **Instructor's Model Solution**: A reference solution provided by the instructor, serving as a benchmark for grading.
- **Student Submissions**: Code submissions from the students.

- **Corresponding Grades**: The grades assigned to each student submission.

### 4.2 Implementation Details

To ensure the dataset's utility and maintain its integrity, the following implementation details were considered:

- **Anonymization of Roll Numbers**: SHA-256 based HMAC was employed to anonymize the roll numbers, preserving student privacy.
- **Hash Assignment for Anonymity**: A comprehensive list of hashes derived from the data of all assignments was generated. Each hash was then randomly assigned a student ID from 1 to 196.

### 4.3 Dataset Preview

A preview of the directory structure of the datastet can be seen in 2. The complete dataset can be downloaded from **here**

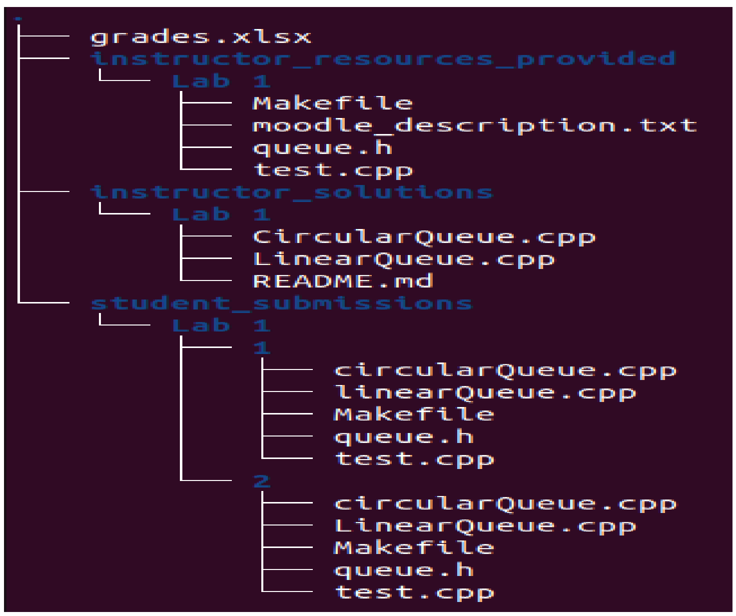


Figure 2: CS 293 Dataset Preview

## 5 Grading Submission

The **Grading Submission** component is a pivotal part of our subjective autograding system, focusing on the retrieval of a 'suitable' set of data and the subsequent fine-tuning process for the Code Llama - Instruct model.

### 5.1 Approach

The Code Llama - Instruct model is fine-tuned for the specific task of subjective autograding. The fine-tuning process explores Meta-Learning and approaches such as Lora and Qlora. This step is crucial for adapting the model to the nuances of grading programming assignments and achieving optimal performance.

### 5.2 Results

In our pursuit of autograding with no manual intervention, it is imperative for LLMs to precisely adhere to the provided output format instructions. The model's ability to support and process extensive contexts or prompts, encompassing various elements such as Problem Statements, Instructor Model Solutions, Rubric Criteria, Student Submission code files, and Few Shot Examples, is crucial for effective grading.

### 5.2.1 Illustrations

| User | ChatGPT |
|---|---|
| You have the task of grading a programming assignment submission. The problem statement and resources provided by the instructor will be given between [INSTRUCTOR] and [/INSTRUCTOR] tags. The rubric that is supposed to be graded will be given between [RUBRIC] and [/RUBRIC] tags. The student submission will be between [SUBMISSION] and [/SUBMISSION] tags. You are supposed to come up with a grade for the submission for the given rubric. I will give the instructor resources in the second prompt, then the rubric in the third prompt, and the student submission in the fourth prompt. So in response to this first, second, third prompts just note the contents and output just the word NOTED and after the fourth prompt give only the score in between the tags [GRADE] and [/GRADE]. | Understood. Please provide the instructor resources in the second prompt. |
| [INSTRUCTOR]<br>[Files]<br>{files_contents}<br>[/Files]<br>[/INSTRUCTOR] | NOTED |
| [RUBRIC]<br>{description}<br>[POSSIBLESCORES]<br>{list of possible scores}<br>[/POSSIBLESCORES]<br>[/RUBRIC] | NOTED |
| [SUBMISSION]<br>[Files]<br>{files}<br>[/Files]<br>[/SUBMISSION] | NOTED |
| Give me the grade for the submission in required format | [GRADE]1[/GRADE] |

(a) Prompt Labelled

| User | ChatGPT |
|---|---|
| First Prompt: Note that this prompt contains the top level instructions and you are supposed to output just the word NOTED in response. You have the task of grading a programming assignment submission. In the Second Prompt I will provide the problem statement and resources provided by the instructor between [INSTRUCTOR] and [/INSTRUCTOR] tags. In the Third Prompt I will provide the rubric that is supposed to be graded between [RUBRIC] and [/RUBRIC] tags. In the Fourth Prompt I will provide the student submission between [SUBMISSION] and [/SUBMISSION] tags. You are supposed to come up with a grade for the student submission for the given rubric. In response to the First , Second , Third prompts just note the contents and output just the word NOTED and after the Fourth prompt give only the score in between the tags [GRADE] and [/GRADE]. | NOTED |
| Second Prompt: Note that this contains the resources provided by instructor and you are supposed to output just the word NOTED in response.<br>[INSTRUCTOR]<br>[Files]<br>{files_contents}<br>[/Files]<br>[/INSTRUCTOR] | NOTED |
| Third Prompt: Note that this contains the rubric and you are supposed to output just the word NOTED in response.<br>[RUBRIC]<br>{description}<br>[POSSIBLESCORES]<br>{list of possible scores}<br>[/POSSIBLESCORES]<br>[/RUBRIC] | NOTED |
| Fourth Prompt: Note that this contains the student submission and you are supposed to output only the grade in required format in response.<br>[SUBMISSION]<br>[Files]<br>{files}<br>[/Files]<br>[/SUBMISSION] | [GRADE]2[/GRADE] |

(b) Prompt Labelled with Reminder

Figure 3: Zero Shot Prompting on Chatgpt

It is evident from Figure 3a that ChatGPT can forget important instructions in the case of larger contexts. Providing reminders at required locations helps it strictly adhere to the instructions as evident from 3b.

| User | ChatGPT |
|---|---|
| First Prompt: You have the task of grading a programming assignment submission. The problem statement and resources provided by the instructor will be given between [INSTRUCTOR] and [/INSTRUCTOR] tags. The rubric that is supposed to be graded will be given between [RUBRIC] and [/RUBRIC] tags. The student submission will be between [SUBMISSION] and [/SUBMISSION] tags. You are supposed to come up with a grade for the submission for the given rubric. I will give the instructor resources in second prompt then the rubric in the third prompt. In the fourth prompt I will give a example between [EXAMPLE] and [/EXAMPLE] tags which will show the format of student submission and the expected format for the grade. In the fifth prompt I will give the student submission and in response to that you are supposed to provide the grade in the same format as shown in example and I have also explained the format next . In response to this First , Second , Third, Fourth prompts just note the contents and output just the word NOTED and after the Fifth prompt give only the grade in between the tags [GRADE] and [/GRADE]. | NOTED |
| Second Prompt: [INSTRUCTOR]<br>[Files]<br>{files_contents}<br>[/Files]<br>[/INSTRUCTOR] | NOTED |
| Third Prompt: [RUBRIC]<br>{description}<br>[POSSIBLESCORES]<br>{list of possible scores}<br>[/POSSIBLESCORES]<br>[/RUBRIC] | NOTED |
| Fourth Prompt: [EXAMPLE]<br>[SUBMISSION]<br>[FILES]<br>{files}<br>[/FILES]<br>[/SUBMISSION]<br><br>Response:<br>[GRADE]1[/GRADE]<br><br>[/EXAMPLE] | NOTED |
| Fifth Prompt:[SUBMISSION]<br>[Files]<br>{files}<br>[/Files]<br>[/SUBMISSION] | NOTED |
| Give me the grade for the submission in Fifth prompt in required format | [GRADE]2[/GRADE] |
| The example submission had a score of 1 but you provided a score of 2 to this submission. What is the difference in between this submission and theexample submission that you provided this submission a better grade. Can you provide your reasoning | Apologies for the oversight. The main difference between this submission and the example submission is the inclusion of comments in the code. This submission provides more detailed comments, explaining the logic and functionality of various parts of the code. The comments contribute to better code understanding and enhance the overall readability of the codebase. As a result, I have assigned a higher score of 2 to this submission, reflecting the better documentation and clarity in the code. |

(a) Prompt Labelled With Example

| User | ChatGPT |
|---|---|
| First Prompt: Note that this prompt contains the top level instructions and you are supposed to output just the word NOTED in response. You have the task of grading a programming assignment submission. In the Second Prompt I will provide the problem statement and resources provided by the instructor between [INSTRUCTOR] and [/INSTRUCTOR] tags. In the Third Prompt I will provide the rubric that is supposed to be graded between [RUBRIC] and [/RUBRIC] tags. In the Fourth Prompt I will provide an example between [EXAMPLE] and [/EXAMPLE] tags which will make the format for submission in Fifth prompt and the expected output to that more clear. In the Fifth Prompt I will provide the student submission between [SUBMISSION] and [/SUBMISSION] tags. You are supposed to come up with a grade for the student submission for the given rubric. In response to the First , Second , Third and Fourth prompts just note the contents and output just the word NOTED and after the Fifth prompt give only the grade in between the tags [GRADE] and [/GRADE]. | NOTED |
| Second Prompt: Note that this contains the resources provided by instructor and you are supposed to output just the word NOTED in response. [INSTRUCTOR]<br>[Files]<br>{files_contents}<br>[/Files]<br>[/INSTRUCTOR] | NOTED |
| Third Prompt: Note that this contains the rubric and you are supposed to output just the word NOTED in response. [RUBRIC]<br>{description}<br>[POSSIBLESCORES]<br>{list of possible scores}<br>[/POSSIBLESCORES]<br>[/RUBRIC] | NOTED |
| Fourth Prompt: Note that this contains the example to make the format for submission and expected grade format clear and you are supposed to output just the word NOTED in response. [EXAMPLE]<br>[SUBMISSION]<br>[FILES]<br>{files}<br>[/FILES]<br>[/SUBMISSION]<br><br>Response:<br>[GRADE]1[/GRADE]<br><br>[/EXAMPLE] | NOTED |
| Fifth Prompt: Note that this contains the student submission and you are supposed to output only the grade in required format in response.<br>[SUBMISSION]<br>[Files]<br>{files}<br>[/Files]<br>[/SUBMISSION] | [GRADE]2[/GRADE] |
| The example submission had a score of 1 but you provided a score of 2 to this submission. What is the difference in between this submission and the example submission that you provided this submission a better grade. Can you provide your reasoning | I apologize for the oversight. Upon closer inspection, the student submission demonstrates better adherence to the rubric criteria compared to the example submission.The key improvements include:<br>{some improvements}<br>These factors collectively contribute to a more well-structured and commented code, aligning better with the rubric criteria. Therefore, I have revised the grade to better reflect the student's performance in meeting the specified criteria. |

(b) Prompt Labelled with Reminder And Example

Figure 4: Prompting with Example on Chatgpt

Similar observations as seen in 3 are also visible in 4. Furthermore, we can see that ChatGPT is capable of providing logical explanations for the grade it provided, thereby ensuring it performs the grading task accurately.

| User | CodeLlama Instruct - 34B |
|---|---|
| First Prompt: Note that this prompt contains the top level instructions and you are supposed to output just the word NOTED in response. You have the task of grading a programming assignment submission. In the Second Prompt I will provide the problem statement and resources provided by the instructor between [INSTRUCTOR] and [/INSTRUCTOR] tags. In the Third Prompt I will provide the rubric that is supposed to be graded between [RUBRIC] and [/RUBRIC] tags. In the Fourth Prompt I will provide the student submission between [SUBMISSION] and [/SUBMISSION] tags. You are supposed to come up with a grade for the student submission for the given rubric. In response to the First , Second , Third prompts just note the contents and output just the word NOTED and after the Fourth prompt give only the score in between the tags [GRADE] and [/GRADE]. | NOTED |
| Second Prompt: Note that this contains the resources provided by instructor and you are supposed to output just the word NOTED in response.<br>[INSTRUCTOR]<br>[Files]<br>{files_contents}<br>[/Files]<br>[/INSTRUCTOR] | NOTED |
| Third Prompt: Note that this contains the rubric and you are supposed to output just the word NOTED in response.<br>[RUBRIC]<br>{description}<br>[POSSIBLESCORES]<br>{list of possible scores}<br>[/POSSIBLESCORES]<br>[/RUBRIC] | NOTED |
| Fourth Prompt: Note that this contains the student submission and you are supposed to output only the grade in required format in response.<br>[SUBMISSION]<br>[Files]<br>{files}<br>[/Files]<br>[/SUBMISSION] | [GRADE] 5 [/GRADE] This student has submitted their work on time, and it appears to be properly formatted. However, upon reviewing the code, there are several issues that need to be addressed.{issues}. Therefore, I am awarding a grade of 5 out of 10. |

(a) Prompt Labelled With Example

| User | CodeLlama Instruct - 34B |
|---|---|
| First Prompt: Note that this prompt contains the top level instructions and you are supposed to output just the word NOTED in response. You have the task of grading a programming assignment submission. In the Second Prompt I will provide the problem statement and resources provided by the instructor between [INSTRUCTOR] and [/INSTRUCTOR] tags. In the Third Prompt I will provide the rubric that is supposed to be graded between [RUBRIC] and [/RUBRIC] tags. In the Fourth Prompt I will provide an example between [EXAMPLE] and [/EXAMPLE] tags which will make the format for submission in Fifth prompt and the expected output to that more clear. In the Fifth Prompt I will provide the student submission between [SUBMISSION] and [/SUBMISSION] tags. You are supposed to come up with a grade for the student submission for the given rubric. In response to the First , Second , Third and Fourth prompts just note the contents and output just the word NOTED and after the Fifth prompt give only the grade in between the tags [GRADE] and [/GRADE]. | NOTED |
| Second Prompt: Note that this contains the resources provided by instructor and you are supposed to output just the word NOTED in response. [INSTRUCTOR]<br>[Files]<br>{files_contents}<br>[/Files]<br>[/INSTRUCTOR] | NOTED |
| Third Prompt: Note that this contains the rubric and you are supposed to output just the word NOTED in response. [RUBRIC]<br>{description}<br>[POSSIBLESCORES]<br>{list of possible scores}<br>[/POSSIBLESCORES]<br>[/RUBRIC] | NOTED |
| Fourth Prompt: Note that this contains the example to make the format for submission and expected grade format clear and you are supposed to output just the word NOTED in response. [EXAMPLE]<br>[SUBMISSION]<br>[FILES]<br>{files}<br>[/FILES]<br>[/SUBMISSION]<br><br>Response:<br>[GRADE]1[/GRADE]<br><br>[/EXAMPLE] | NOTED |
| Fifth Prompt: Note that this contains the student submission and you are supposed to output only the grade in required format in response.<br>[SUBMISSION]<br>[Files]<br>{files}<br>[/Files]<br>[/SUBMISSION] | Response:<br>[GRADE]85[/GRADE] |

(b) Prompt Labelled with Reminder And Example

Figure 5: Prompting with Remainder on CodeLlama Instruct - 34B

It is evident from Figure 5 that CodeLlama forgets the possible scores it is allowed to give even after providing some sort of reminder. This also raises a question on how accurately it is providing the grade as this phenomenon of forgetfulness might also be occuring with other sections of the prompt like the student code files and thus the grade provided by the model might not be reliable at all. We will study this in detail to find out whether CodeLlama is able to provide coherent logical explanations for the grade it is providing.

### 5.2.2 Performance Insights

While LLMs have the capacity to handle very large prompts (context length), they tend to forget the context in the case of larger prompts. Consequently, they struggle to strictly adhere to output format instructions. Fragmenting information into separate parts or components and presenting them conversationally proves more effective than providing the complete information in a single prompt. The structure of the prompt significantly influences the model's ability to follow instructions.

### 5.2.3 Comparison with Baselines

Notably, the Code Llama - Instruct model shows a relatively lower proficiency in adhering to instructions compared to ChatGPT. Comparative insights are drawn from the performance of various baselines, including ChatGPT and Code Llama - Instruct before fine-tuning, as well as other open-source LLMs.

## 5.3 Future Work

Our next objective involves refining the prompt to enhance the Code Llama - Instruct's adherence to the output format requirements as seen from Illustrations. We will also compare the differences in instruction following and grading capabilities between Codellama Instruct - 34B and baselines like ChatGPT. Following this optimization, we will proceed with the subsequent steps outlined in the approach, including fine-tuning and performance evaluation.

# References


[1] Pranshu Chourasia et al. "Algorithm Identification in Programming Assignments". In: *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension*. ICPC '22. Virtual Event: Association for Computing Machinery, 2022, pp. 471–481. ISBN: 9781450392983. DOI: `10.1145/3524610.3527914`. URL: `https://doi.org/10.1145/3524610.3527914`.

[2] Zhangyin Feng et al. *CodeBERT: A Pre-Trained Model for Programming and Natural Languages*. 2020. arXiv: `2002.08155 [cs.CL]`.

[3] Daya Guo et al. *GraphCodeBERT: Pre-training Code Representations with Data Flow*. 2021. arXiv: `2009.08366 [cs.SE]`.

[4] Baptiste Rozière et al. *Code Llama: Open Foundation Models for Code*. 2023. arXiv: `2308.12950 [cs.CL]`.

[5] Hugo Touvron et al. *Llama 2: Open Foundation and Fine-Tuned Chat Models*. 2023. arXiv: `2307.09288 [cs.CL]`.