

# Coding Challenge Generator

Fabian Torres Alvarez  
Nour Ardo

December 12, 2022

Word count: 1989.

## 1 Introduction

Coding challenges have become a de facto standard for technical job interviews in IT, leading to the rise of online platforms like LeetCode (<https://leetcode.com/>) offering vetted training resources, under a freemium business model. On the other hand, recent advances in large language models (LLMs) are creating a paradigm shift in automated text generation, where it is now possible for computers to produce paragraphs-long texts that are coherent, logical, and feel natural to humans. [1] This work proposes a system that generates coding challenges on-demand, for use as a training tool for students, job-seekers and competitive coders.

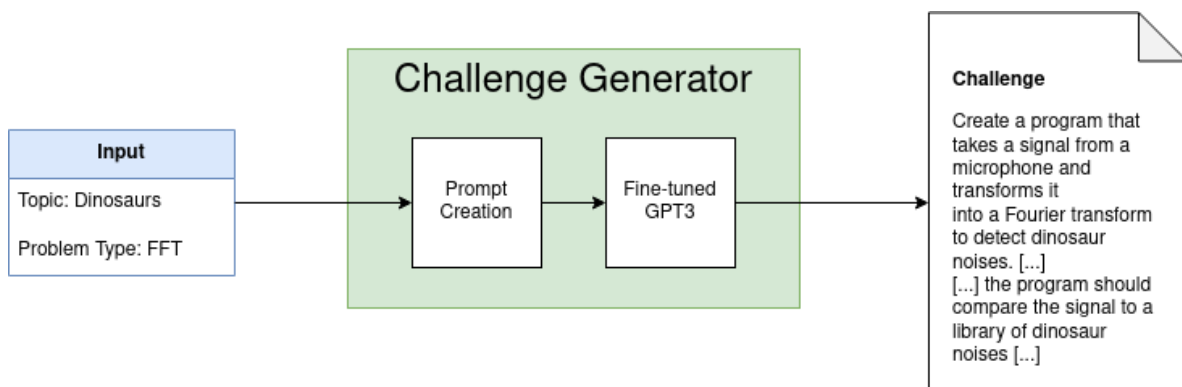


Figure 1: Block diagram of the project.

## 2 Background and Related Work

Previous works have focused on generating prompts and questions in the academic field to improve the students' experience and enhance their academic performance. They implemented models that generated exam questions and quizzes as well as practice questions for topics covered in class. Tsai et al. [2]. implemented in their work a short answer question generation model that generated questions for text topics using the BERT model to extract keywords and T5 for transfer learning, a model that generates questions from declarative sentences. This model was evaluated using BLEU and ROUGE evaluation methods, and results of 0.567 and 0.613 were reached respectively. However, implementing similar models for coding challenges has not been studied yet.

GPT-3 [3] has already achieved novel results for Natural Language Processing (NLP) applications. It exceeds previous versions of GPT and T5 models with the number of parameters and learning results. It was proposed by OpenAI as an improvement for GPT2 with 499B tokens, 175B parameters and 96 layers. Although it faced some criticism in lacking meaning and coherence in some outputs, it succeeded in different fields and applications like generating texts, poems, codes and also playing chess [4]. In this project, we will use GPT-3 based solutions to implement a model that generates new

coding challenges that could be used by software companies for recruiting, or by learners themselves to create new challenges and try to solve them.

Github and OpenAI introduced the Github Copilot project [5] to help developers using different IDEs like Visual Studio Code, Neovim to write their code from code prompts with Codex, a fine-tuned version of GPT-3 [6]. This might also fit in this field as extending this idea to generating the code solution of the coding challenge could be a future work for this project.

### 3 Data and Data Processing

We scraped all available 8156 problems on Codeforces [7] including title, description, metadata, and tags in plain text with a custom web scraper included in the scraping directory in our GitHub repository. An screenshot of Codeforces’ website and the corresponding scraped data are shown on Figures 2, and 3 respectively.

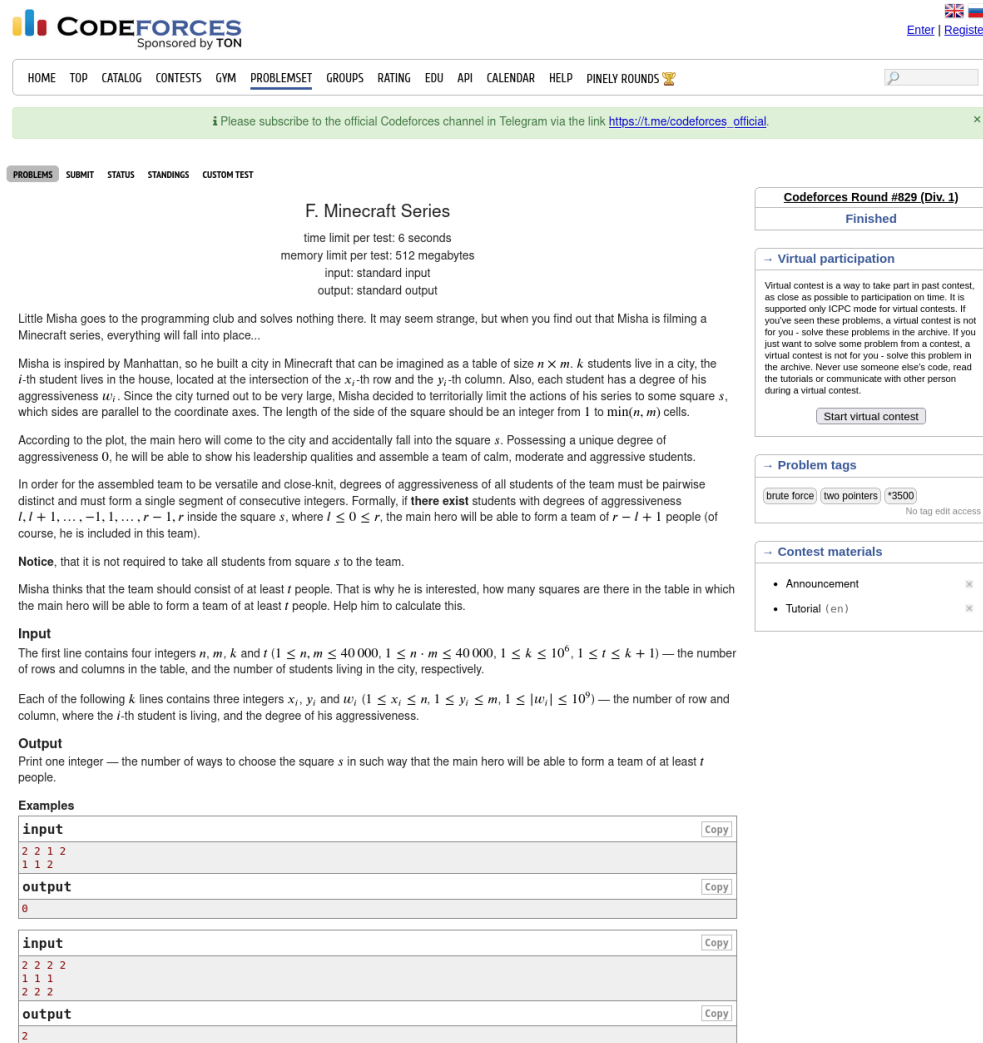


Figure 2: Codeforces example problem.

The scraped data is in already in plain text format and ready to be used. Before starting the project, we were concerned that GPT-3 may not perform well with LaTeX formatted equations between dollar signs (\$), but our first experiments showed it can understand them.

There are 65 total unique tags, of which 34 are filtered out as not useful. Examples of useful tags: “greedy”, “data structures”, and “brute force”. Examples of not useful tags are: “implementation”, “\*800”, “\*1900”. Figure 4 shows an histogram of useful category tags.

```

JSON Raw Data Headers
Save Copy Collapse All Expand All Filter JSON
▼ header:
  title: "F. Minecraft Series"
  time_limit: "6 seconds"
  memory_limit: "512 megabytes"
  input_file: "standard input"
  output_file: "standard output"
▼ txt:
  ▼ 0: "Little Misha goes to the programming club and solves nothing there. It may seem strange, but when you find out that Misha is filming a Minecraft series, everything will fall into place..."
  ▼ 1: "Misha is inspired by Manhattan, so he built a city in Minecraft that can be imagined as a table of size $$$n \times m$$$. $$$k$$$ students live in a city, the $$$i$$$-th student lives in the house, located at the intersection of the $$$x_i$$$-th row and the $$$y_i$$$-th column. Also, each student has a degree of his aggressiveness $$$w_i$$$ . Since the city turned out to be very large, Misha decided to territorially limit the actions of his series to some square $$$s \times s$$$ , which sides are parallel to the coordinate axes. The length of the side of the square should be an integer from $$$1$$$ to $$$\min(n, m)$$$ cells."
  ▼ 2: "According to the plot, the main hero will come to the city and accidentally fall into the square $$$s \times s$$$ . Possessing a unique degree of aggressiveness $$$s$$$ , he will be able to show his leadership qualities and assemble a team of calm, moderate and aggressive students."
  ▼ 3: "In order for the assembled team to be versatile and close-knit, degrees of aggressiveness of all students of the team must be pairwise distinct and must form a single segment of consecutive integers. Formally, if "
  ▼ 4: " students with degrees of aggressiveness $$$l, l+1, \dots, -1, 1, \dots, r-1, r$$$ inside the square $$$s \times s$$$ , where $$$l \le 0 \le r$$$ , the main hero will be able to form a team of $$$r-l+1$$$ people (of course, he is included in this team)."
  ▼ 5: ", that it is not required to take all students from square $$$s \times s$$$ to the team."
  ▼ 6: "Misha thinks that the team should consist of at least $$$t$$$ people. That is why he is interested, how many squares are there in the table in which the main hero will be able to form a team of at least $$$t$$$ people. Help him to calculate this."
  ▼ 7: "The first line contains four integers $$$n, m, k$$$ and $$$t$$$ ($$$1 \le n, m \le 400000$$$ , $$$1 \le n \cdot m \le 400000$$$ , $$$1 \le k \le 10^6$$$ , $$$1 \le t \le k + 1$$$ ) – the number of rows and columns in the table, and the number of students living in the city, respectively."
  ▼ 8: "Each of the following $$$k$$$ lines contains three integers $$$x_i, y_i, w_i$$$ ($$$1 \le x_i \le n$$$ , $$$1 \le y_i \le m$$$ , $$$1 \le w_i \le 10^9$$$ ) – the number of row and column, where the $$$i$$$-th student is living, and the degree of his aggressiveness."
  ▼ 9: "Print one integer – the number of ways to choose the square $$$s \times s$$$ in such way that the main hero will be able to form a team of at least $$$t$$$ people."
▼ input_specification:
  ▼ 0: "The first line contains four integers $$$n, m, k$$$ and $$$t$$$ ($$$1 \le n, m \le 400000$$$ , $$$1 \le n \cdot m \le 400000$$$ , $$$1 \le k \le 10^6$$$ , $$$1 \le t \le k + 1$$$ ) – the number of rows and columns in the table, and the number of students living in the city, respectively."
  ▼ 1: "Each of the following $$$k$$$ lines contains three integers $$$x_i, y_i, w_i$$$ ($$$1 \le x_i \le n$$$ , $$$1 \le y_i \le m$$$ , $$$1 \le w_i \le 10^9$$$ ) – the number of row and column, where the $$$i$$$-th student is living, and the degree of his aggressiveness."
▼ output_specification:
  ▼ 0: "Print one integer – the number of ways to choose the square $$$s \times s$$$ in such way that the main hero will be able to form a team of at least $$$t$$$ people."
▼ sample_tests:
  ▼ input_tests:
    0: "\n2 2 1 2\n1 2\n"
    1: "\n2 2 2 2\n1 1\n2 2\n"
    2: "\n2 4 2 1 1\n1 1 -1\n1 2 1\n2 2 1\n"
  ▼ output_tests:
    0: "\n0\n"
    1: "\n2\n"
    2: "\n4\n"
  note: []
▼ tags:
  0: "brute force"
  1: "two pointers"
  2: "+3500"

```

Figure 3: Scraped data from the example problem from Figure 2

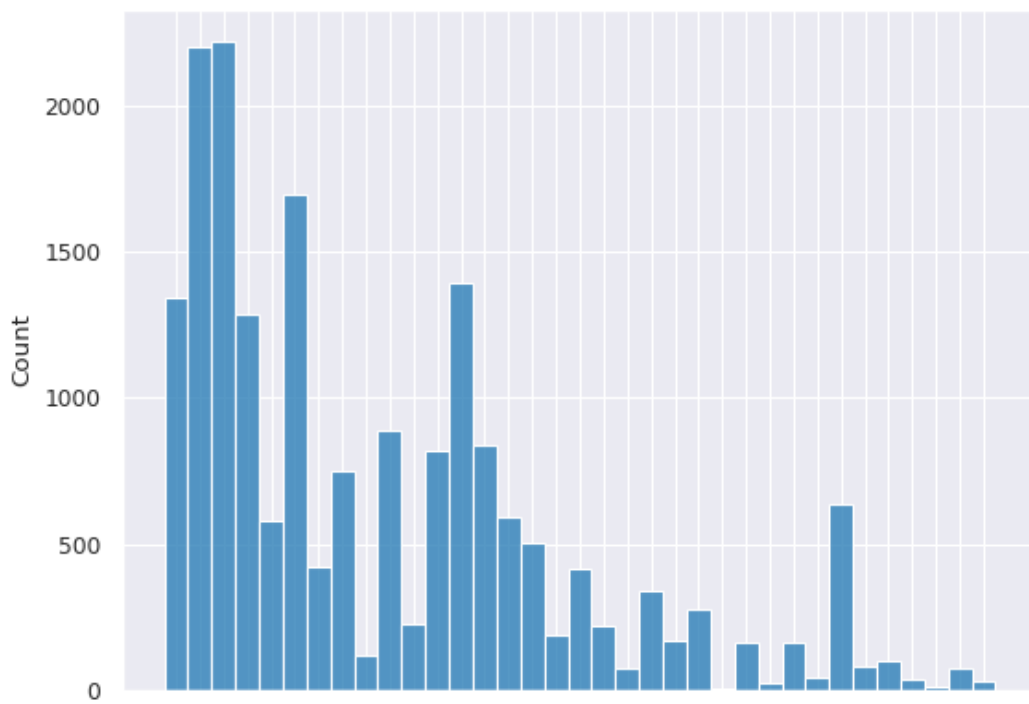


Figure 4: Histogram of problem category tags.

## 4 Architecture and Software

Our model is based on fine-tuning OpenAI GPT-3 models on the coding challenge prompts of Codeforces. The model receives an input formed of the title of the challenge and one of the tags (sampled at random) assigned to it: title, tag. The problem text constitutes the completion target for the GPT models.

A subset of the Codeforces problems were randomly sampled, then passed to one of two GPT-3 models: Curie and Davinci. A digram of the fine-tuning task is shown in Figure 5.

Due to the monetary cost of training with OpenAI’s API, we used 2500 prompts for the curie model and 140 prompts for the davinci model.

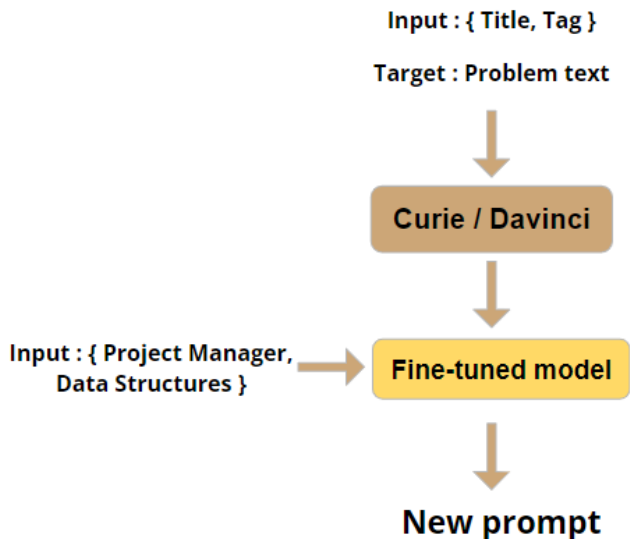


Figure 5: Diagram of the fine-tuning task

## 5 Baseline Model

OpenAI [playground](#) has been used recently to generate stories, questions or even have conversations with it. At the time of this project, the most powerful GPT-3 model is Davinci003, published on the OpenAI playground on November 28th. It improved the language models by using reinforcement learning with human feedback. We built our baseline model using zero shot propmting and Davinci-003 on OpenAI playground, because it offered significantly better performance than other models.

The prompt for the zero shot has the following format:

**“An elaborate programming challenge about {tag} and {topic}”.**

The “topic” represents the title assigned to the coding challenge problem in Codeforces, and the “tag” represents a single one of the tags assigned to the problem (chosen randomly).

The completions were evaluated manually and were used as inputs for our evaluation classifier that will be described shortly.

Smaller GPT-3 models like *ada*, *babbage* and *curie* were also explored, but Davinci performed best for this task.

An example of the zero shot completion having “vampire cricket players” as a title and “hashing” as a tag is presented in Figure 6.

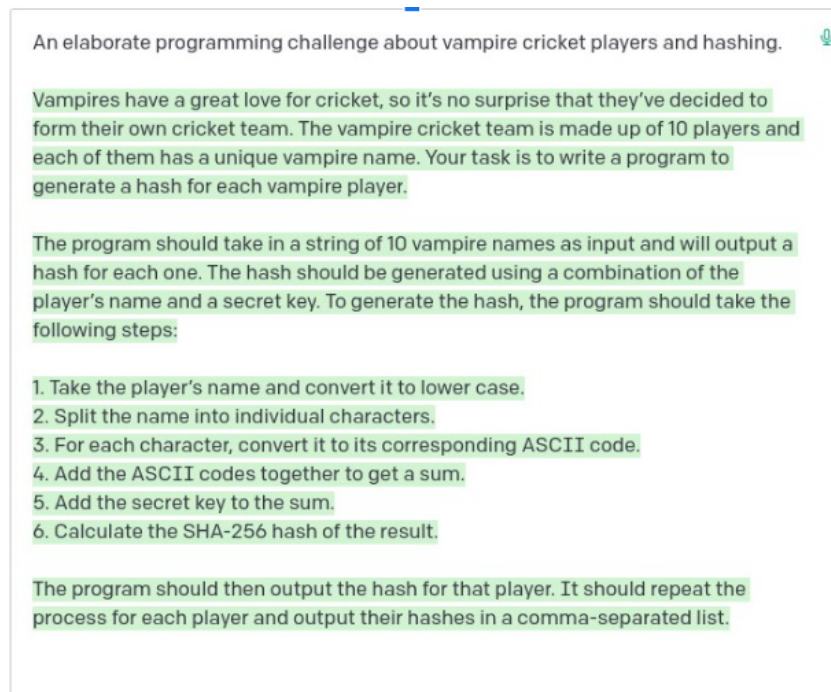


Figure 6: Zero shot completion with Davinci-003.

## 6 Evaluation

### 6.1 Evaluation Criteria

To evaluate the completions of our models, we set the following criteria for a good completion:

1. Is written like a challenge in the coding domain.
2. Has little or no ambiguity.
3. Is not trivial, nor just repeats a definition.
4. Has no spurious equations or symbols.
5. *Nice to have*: Is tractable.

### 6.2 Evaluation Classifier

Aside from the hand labeling of the new completions, we built a binary classifier that classifies good and bad challenges. This classifier is built using minGPT model introduced in Assignment 3. [8]

We collected the completions with the baseline model and labeled each by hand using the evaluation criteria mentioned before. We got 400 completions with 195 out of them labeled as bad and 205 labeled as good. Most of the bad completions were generated using a model different than davinci-003 in OpenAI playground.

The model was finetuned with a classifier head on 90% of the dataset, then validated on the remaining completions. It was trained with a learning rate of  $5e-5$ , 4200 iterations and a single batch size.

An accuracy of 75% was reached on the validation dataset.

A diagram of the classifier is shown in Figure 7.

This classifier is later used to evaluate the fine-tuning models that we studied for the coding challenge generation task.

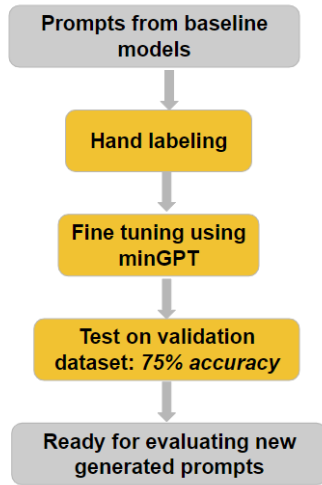


Figure 7: Diagram of the evaluation classifier

## 7 Quantitative Results

The completions of the baseline model were labeled by hand as mentioned earlier and 72% of them were classified as good.

For each fine-tuned models, we passed 100 completions to the classifier, then verified the results through hand labeling. Results are shown on Table 1.

Model	Performance (Classifier)	Performance (Hand labeling)
Baseline model (Zero shot)	72%	
Curie (Fine-tuning)	70%	66%
Davinci (Fine-Tuning)	71%	77%

Table 1: Quantitative Results

## 8 Qualitative Results

For the baseline model generation using zero shot prompting, GPT-3 generated better results when some details were given. Generating good completions only from a title was hard and most of the outputs did not include a creative challenge story like Codeforces problems. Instead, they were straightforward commands like “Given a string, find the length of the longest substring without repeating characters.” or failed to ask coding questions like “What are the pros and cons of a round table discussion?”. This motivated us to use a fine-tuned GPT-3 model.

The following is an example of a good completion with our fine-tuned GPT-3:

**Title:** Feed with candy.

**Tag:** Greedy algorithms.

*In this game, you play as a bear named Carl. Carl is really into candy, so he decided to spend his vacation in a candy factory. One day, he heard that a candy factory is being attacked by a swarm of wasps. The wasps are trying to steal the candies that the factory produces. [...]*

*Carl has a net that he can throw to capture wasps. The wasps are flying in a pre-defined path. [...]*

*The factory produces candies at a fixed rate. Every second, there is a chance that a candy will appear on the floor, and a wasp will try to steal it. The chance is  $1/(t/10)$  where  $t$  is the current number of seconds. If a new candy appears, Carl has a chance to catch the wasp that stole the candy.*

*The game ends when Carl has caught at least 1 wasp or when the wasp successfully steals all of the candies. [...]*

This completion worked well because it is a coding challenge with clear rules, internal logic, and content relevant to the original title.

A bad completion is the following:

**Title:** Easter Eggs.

**Tag:** Constructive Algorithms.

*Monocarp is an egg-loving monster. He wants to hide an integer  $x$  inside an egg. The egg is a box of size  $n$  with a door on its left. The egg has  $n-1$  eggs inside (counting from the door side). The  $i$ -th egg has size  $2i+1$ .*

*Monocarp can add or remove an egg from an arbitrary side of the box. Also, he can destroy an egg and leave a hole on the side of the box opposite to the destroyed egg.*

*Monocarp wants to add an egg  $x$  into the egg with index  $x$  as close to zero as possible. For example, if  $x=1$ , then he can put the egg into the second egg (index 1), if  $x=2$ , then he can put the egg into the first egg (index 0), if  $x=3$ , then he cannot place the egg into any of the eggs.*

*What is the minimum number of eggs Monocarp can destroy to satisfy his desire?*

This is a bad completion because the description does not have any real world logic in it, and the question asked cannot be answered with the information given. Overall, we observed that our model struggles to generate logical coding challenges when the title consists of ordinary real world objects, but it does well with more abstract titles.

## 9 Discussions and Learnings

GPT-3 performed well for generating arbitrary coding challenges. Even though GPT-3 models had been used to generate problems and work with code before, we were surprised by the quality of the completions, especially in the zero-shot scenario. The system is capable of generating non-trivial coding challenges that can be difficult even for a professional programmer.

Even though the system was able to generate logical coding challenges, we think it would be too ambitious to use its outputs without human supervision. One limitation is that it is very hard to adjust the difficulty of the generated problems, they can range from very easy to impossible with little human control. Most generated texts were good enough for regular usage, but some outputs are still nonsensical, even when syntactically and grammatically correct. Because of these limitations, we would not recommend to use this tool without human supervision for any application with real world consequences like academic evaluation or job interviews.

During the course of this project, we spent significant time and effort fine-tuning the smaller GPT-3 models before trying the largest model available (davinci). This was a good strategy for setting up our data pipeline, and identify potential issues without costly training loops. However, after our first successful iterations, it would have been good to spend more of the project focusing in improving the davinci model instead of attempting to extract better performance out of weaker models.

In our first project plan, the baseline model consisted of a recursive neural network (RNN) that we wrote the code for, but discarded after a class discussion about zero-shot model generators. We believe that with today's state of the art, zero shot learning is the easiest and the most relevant architecture for baseline models in this type of projects.

Lastly, we observed that many problems with the quality of our outputs stemmed from problems with the input data we were fine-tuning the model with, as LLMs are capable of learning both good and bad patterns in the training data. It is extremely important to review that the data fed to a model is correct, otherwise results will suffer. Labelling large corpora is tedious but vital work for a project in this space to succeed.



## 10 Individual Contributions

Fabian implemented the custom scraper, hand labelled 250+ data samples, worked on the zero-shot baseline software, implemented the gradio interface, and was responsible for fine-tuning the GPT-3 models.

Nour implemented the first baseline RNN, hand labelled 150+ data samples, worked on the zero-shot baseline software, implemented the evaluation classifier, and fine-tuned smaller GPT-3 models.

## References

- [1] A. Tamkin, M. Brundage, J. Clark, and D. Ganguli, “Understanding the capabilities, limitations, and societal impact of large language models,” 2021. [Online]. Available: <https://arxiv.org/abs/2102.02503>
- [2] D. C. Tsai, A. Y. Huang, O. H. Lu, and S. J. Yang, “Automatic question generation for repeated testing to improve student learning outcome,” pp. 339–341, 2021.
- [3] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, “Language models are few-shot learners,” 2020. [Online]. Available: <https://arxiv.org/abs/2005.14165>
- [4] R. Dale, “Gpt-3: What’s it good for?” p. 113–118, 2021.
- [5] [Online]. Available: <https://github.com/features/copilot/>
- [6] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba, “Evaluating large language models trained on code,” 2021. [Online]. Available: <https://arxiv.org/abs/2107.03374>
- [7] [Online]. Available: <https://codeforces.com/>
- [8] Andrej, “mingpt,” Dec 2022. [Online]. Available: <https://github.com/karpathy/minGPT>

## Permissions

Group Member	Permission to Post	Granted
Fabian	Video	Wait to see
Fabian	Final report	Yes
Fabian	Source code	Available upon request
Nour	Video	Wait to see
Nour	Final report	Yes
Nour	Source code	Available upon request