Solving Coding Problem With Machine Learning
CS496 Capstone Project

Joshua Brooks Jr, Sajiv Gnanasekaran
Advisor: Dr. Nguyen Ho

2024-05-08

Contents

Revision History

The following revisions have been made

- 4/19/24, Methodology and Experiment Design, instructor recommended items were moved from Experiment Design to Methodology
- 4/19/24, feature extraction, example of extracting features added
- 4/19/24, data collection, subsection rewritten to be more concise and scientific
- 4/20/24, Introduction, first paragraph rewritten according to recommended changes, last paragraph changed to more factual writing.
- 4/20/24, Motivation, subsections removed and citations provided

- 5/6/24, Methodology, Added hyper-parameters for each model
- 5/6/24, Analysis, Added interpretations of the Confusion matrices
- 5/6/24, Conclusion, Expanded on our conclusion
- 5/6/24, Methodology, Experiment Design, Conclusion, future work, fixed grammar

1 Introduction

AI or Artificial Intelligence is a technology that tries to simulate human thinking in machines. AI has been around since the 1950's, however thanks to recent advancements, AI has become more powerful than ever before. Many AI tools have been developed with the intention of making certain task easier for those who use them. These tools are now capable of performing complex activities, which have led to a boost in popularity since many industries can find a use for them. An example of one of these industries is heath care and medicine, with AI tools being able to take the role of a health care assistant.

ChatGPT, a popular AI tool, is a large language model or LLM [5]. LLM's are a sub-category of AI that uses deep learning techniques and exceptionally large data sets so that they can understand, generate, and predict new content [5]. Just like how AI has broken into many industries, LLM's also have a variety of uses. ChatGPT is a more general use tool, that will craft a response for almost any request but there are more specific use tools like Quillbot which specifically help you with grammar and paper checking. Another tool that functions similarly to ChatGPT is Google's LLM, Gemini.

An area where these tools are used is schooling. One reason for the popularity of ChatGPT is its communication ability when responding to questions posed to it [4]. ChatGPT is a helpful tool in this way as it provides quick feedback on the work you provide it, offering many options to help improve your work [5]. However, the tool is not always correct, and so can hinder you if you are not attentive to what it gives you [5]. Alongside students, educators can find ChatGPT helpful as it can assist them with how they run their classrooms and even how they grade students work. They also have concerns due to how easily the tools like ChatGPT can be used to cheat [5].

These benefits and concerns about the tools translate directly into the computer science field. ChatGPT can be helpful in many areas, such as learning how to code, writing new code, debugging and translating code from one language to another [5]. Alongside those areas, ChatGPT can be used as an instructor, with its ability to answer questions related to

programming languages and best practices in programming [5]. It can also provide example code with in-line comments explaining what each line does and a summary of what the code does overall [5]. The tool is also helpful to educators, as they can use it to generate code tutorials and questions that pertain to specific topics [5].

2 Motivation

Use of Large Language Models (LLMs) has exploded since the release of GPT 3 in November 2022. From corporations, to educators, to governments, it seems like every industry has found a use for this all-purpose software. However, because the software is still so new, the actual limitations and capabilities of LLMs are still widely unknown. We want to quantify those attributes with our research, allowing for actual metrics to define the performance of these models. Our specific look into LLM's completion of programming problems will provide metrics to be used by the labor market and by educators.

Since late 2022, there has been a dramatic increase in layoffs from software companies around the United States [5]. According to Statistica, in just 2023, more than 262,000 employees at tech companies worldwide have been laid off during the year across more than 1,180 firms, with most occurring in the United States. Companies including Google, Dropbox, Chegg, and Microsoft have all cited AI in memos to employees regarding these layoffs, as they attempt to cut costs by using LLMs to automate jobs previously carried out by humans. Google even claimed ChatGPT could have passed its interview process for software engineers [2].

The industry believes LLMs have reached a point to where they can replace human software developers on tasks such as debugging and testing, but there have been few research studies done to actually support this [2]. New graduates entering this labor market and existing developers within are all anxious about their future careers being replaced by LLMs, despite the fact that their superiority has not even been proven. When a new LLM is released,

benchmarks regarding code generation abilities are usually unspecific. Even Google only shows two metrics, it's 0-shot score, regarding coding abilities for its Gemini LLM, each only on python code generation. Our research goal is to provide metrics to ease these anxieties and reduce assumptions from employers.

We also want our research to define a threshold for using LLMs in computer science educational spaces. Tools like Google already decreased the time and effort it takes to find information. With how easy it is to look information, it almost became redundant to memorize it. We want to investigate if LLM have this effect on coding problems and concepts that are easy for it to solve.

Should students still learn how to solve introductory coding problems if tools like ChatGPT will make this redundant? Or should they redirect their efforts into utilizing these tools to enhance the creative process? The answer comes from determining if LLMs are actually accurate enough. Our research will provide metrics on basic programming usage that may prove the computer science educational space is ready for this change.

This research could be built on with writing and mathematics in future studies. If research shows it is more efficient and productive for workers to use these tools, rather than write or perform calculations themselves, it could cause a dramatic shift in their educational spaces as well. After all, advanced math courses use calculators for simple arithmetic, so why wouldn't an advanced English course use ChatGPT for simple outlining?

We want our research to provide a comprehensive analysis on the current state of these LLMs to ease the anxieties of developers, but also to provide a message to employers on the reality of implementing these models rather than an actual human. We also want the education industry to know how accurate these tools on to prepare for a possible shift in how they educate.

3 Overall Plan

Step 1: Investigate possible LLM tools, ChatGPT has already been done by our client Dr. Ho, so we are looking into other options such as GitHub co-pilot, Amazon Code Whisperer, edge co-pilot, and others. We could also work on all of the aforementioned tools, researching all of their capabilities. Another possibility would be using both ChatGPT and another tool and directly comparing them. The decision will be made after we analyze each tool and determine if focusing on all of them would be too much for the scope of our project.

Step 2: Investigate programming tools for data collection, the main tool for this project that we could use is Kattis. Kattis is an online judge system for programming problems. Another tool for this is Leetcode, however the problems on this site are well known due to its popularity so it might not be the best choice for our purposes, the problems on these sites have difficulties attached to them, so we have to identify a boundary of which to select problems.

Step 3: Data collection, we would scrape the site for problems that fall within the boundary and plug them into the LLM(s) and extract the solutions. The solutions would then be entered into the evaluator on the site they were gathered, and we would evaluate the correctness and efficiency of the solution. The solutions will fall into two categories at the end of evaluation passes and failures.

Step 4: Data Analysis and Exploration, we would then analyze the problems and their solutions to see if we can find any trends that led to it being a correct or incorrect solution. Here we would also search for any other reasons as to why the solutions came to the result given, such as if the answer were actually correct but some syntax caused it to fail.

Step 5: Model Selection and Training, we would determine an appropriate machine learning model to use with our collected data, for predicting how well the LLM(s) would answer a programming question. We have to split our collected data into two sets, training, and testing. The training set will be used to improve our models prediction accuracy.

Step 6: Model Evaluation, we have to evaluate our model by introducing the test set;

this will give us a measure of how our model performs and the speed it does so. Also, it will help us identify areas where the model may need to be improved.

Step 7: Model Optimization, to assess if the accuracy of the model could be improved, we would have to tune the parameters and introduce unseen data to evaluate the performance. This will be repeated until we feel that accuracy can no longer be improved.

Step 8: Prediction and Analysis, we would use the optimized model to predict the outcomes on unseen data and analyze the results of the predictions and compare them with any expectations we have. Finally, we would compile our findings from the models performance.

These steps will not require any special software, nor the use of external hardware.

4 Related Work

Lewkowycz et al. worked on enhancing the quantitative reasoning capabilities of large language models in the fields of mathematics, science, and engineering. Large language models were shown to have struggled in these areas while showing success in natural language tasks [3].The team created a language model called Minerva. Minerva's aim is to address the limitations that large language models face by being a tool that can process scientific and mathematical questions and generate a response with correct LATEX notation [3]. The team uses a large training dataset collected from sources like the arXiv preprint server and web pages and trains Minerva with that data [3]. This method resulted in Minerva having great performance on various quantitative reasoning benchmarks. Minerva was based on a general language model called PaLM since it showed promise when solving mathematical questions. Minerva differs in that it does not use external tools where PaLM relied on them [3]. After its training with the initial training dataset, Minerva was further trained using undergraduate-level questions in science and mathematics; this was to further evaluate its effectiveness [3].

The main aspect in which their research is related to ours is the assessment of a large

language model. As previously stated, their team created Minerva for use on quantitative reasoning problems; with this, they needed to recognize the mistakes and false positives that their model would make in order to make it the best it could be [3]. Where we differ is that while we are evaluating a large language model, it will be one that is readily available. Despite the difference, we will also need to recognize mistakes or false flags in the solutions given to us by the LLM. Also, the domains in which we are working are different, Minerva is focused on mathematics and science-based questions, while we are focused on programming problems [3]. Both of these domains can be complex, given that they can require specific knowledge and reasoning to solve them.

One idea from their research that could be helpful to ours is that they modified their collected questions before introducing them to the model. The subtle changes that they made were a good way of determining whether their model relied on memorization rather than actually solving the problem. While this could be helpful, it would also introduce a limitation to our research in that there wouldn't be an easy way to determine the correctness of the solution [3].

Chen et al. are a research team that worked on creating a model that could perform the task of generating code from a natural language description [1]. The model they introduced is called Codex, a specialized version of the GPT model whose purpose is to excel at solving programming problems [1]. The process the team followed was to use the model to generate Python functions from provided doc-strings and test their correctness by using unit tests. Rather than collecting problems from public sources, the team hand-wrote problems and unit tests for use in evaluating the model [1]. This was necessary as they had trained their model using GitHub, meaning that the model had already seen a large number of solutions from a variety of public sources [1].

This research is very related to ours, with the main relation being the evaluation of how well a large language model can generate a solution to a complex problem. Chen et al.'s work included the creation of a new model that could perform this task, and after the creation

of the model, they did what we intended to do, which was introduce complex problems to a LLM and evaluate the solution. Also, their created model is an extension of GPT-3 and was the basis for the GitHub co-pilot, which are two models that we are considering evaluating [1]. Due to the relationship, there is a very good tool that we can find helpful from this research. That tool being the HumanEval dataset, this is the collection of handwritten problems that the team used in the evaluation of Codex. This would be a useful collection of questions that we could use in order to evaluate the models that we intend to test [1].

Wermelinger et al. is researcher who evaluated the performance of GitHub Copilot to see how its abilities differed from Codex. He motivation for this research was similar to ours, wanting to evaluate the limitations and abilities of Copilot because of educators. "Educators need to know what Copilot is capable of, in order to adapt their teaching to AI-powered programming assistants," he writes [8]. Tools like Codex were previously only paid and needed to be used in a special program that called OpenAI's API, but now that Copilot is available for free to students in IDE's like Visual Studio Code, Wermelinger et al. claims it will inevitably be used by them.

Wermelinger et al. evaluated Copilot on three metrics, how it wrote code, how it tested code, and how it could explain code, all things a student in a programming course would have to do. Because Copilot runs in an IDE, its testing was unique in that its language prompts come from comments in the code. Wermelinger made sure to use Copilot like a student, generating code from programming exercises on Moodle, seeing if the tests written considered edge cases like students are asked to, and seeing if code explanations demonstrated an ability to translate code into natural language that an educator could understand. [8]

We will need to make similar conclusions on the implications for the education space Wermelinger et al.'s conclusions on the implications for the education space give us a baseline on what we should look for. For example, he wrote it would be impossible and futile to detect or punish the use of Copilot because it can write code indistinguishable from human code. This is something we would like to prove or disprove based on the solutions we generate.

Wermelinger also found that the limitations of Copilot define what students still need to know, something we want our research to determine as well. Copilot can only describe low level detail for code explanations without including important aspects, meaning today's students need to still write high-level, clear documentation and know why code doesn't work in order to fix it [8]. If we find GPT 4 and Gemini can do this already, students should focus their efforts on utilizing the tool to create more complex projects, like using a calculator to do basic arithmetic for a long problem.

It was also helpful for us to analyze a study by Sobania et al, evaluating LLM performance in solving programming bugs. This research team sought to compare GPT 3's automatic bug fixing abilities to industry standard methods. The first methods are APR (automated program repair) systems. These automatically suggest patches to software, tracking bugs during a workday and suggesting fixes the next day. However, these can take hours to run, so deep learning program repair tools have started to be used, like Codex and CoCoNut. Chat GPT 3 extends on those tools as a general use LLM. This study used a dataset of programming questions called QuixBugs as a standard benchmark to compare the different methods of fixing bugs.

Solving a bug within code is similar to the programming problems we plan to test on. We can learn from the Sobania et al's use of a benchmark test set and how they evaluated Chat GPT 3 specifically. GPT 3 was unique among the tools compared because of its conversational context abilities, so this research team adapted how it was prompted, using a discussion style, something we plan to implement in our study as well. [6].

All of these works will help us complete a comprehensive analysis of our models, along with the research our client, Tran et al. completed [7]. This was directly pulling programming problems from Kattis and evaluating just GPT 3's ability to solve them, and will act as a baseline for our evaluation of GPT 4 and Google's Gemini. These other related works give us an idea of what to add to her research process in order to make sure we properly evaluate these models and provide accurate predictions for the future of AI-enabled education.

5 Methodology

We hypothesize that a machine learning model can be created to predict whether a LLM's solution will be correct, wrong, or produce an error by being provided a text prompt of a programming problem. Investigating our hypothesis requires data to be collected and analyzed so that we can use it to train and evaluate prediction models. The data will be collected from the LLM's by providing a programming question from Kattis. The solution that the LLM's provide will be given back to Kattis, and from there we will be given the result and how many tests have been passed; alongside those, we will also log the difficulty of the Kattis question. Our analysis of that data will be handled with the Python library Pandas. Pandas allows for easy manipulation and analysis of data, for example, checking for imbalances. We will also be analyzing the text prompts with another library in order to engineer features to be used in our models. The models that will be used for our research can be any of four options: classifier, regression, linear SVM, and neural network. These four models can each be used to provide predictions based on our data.

5.1 Setup

For this research project, we are working in the programming language Python. Python is commonly used in the machine learning world for its simplicity, readability, and quick development time. It has a wide variety of libraries and frameworks for machine learning and data analysis, such as Pandas and Sci-kit Learn, which we will be leveraging in our research project. The development environment will be set up with Jupyter Notebooks. Jupyter Notebooks are a type of JSON document where code is written in cells, which can be individually run with the output shown directly underneath. They are commonly used in data science, scientific computing, and machine learning and will be essential to our work. These notebooks can also be accessed via a web browser, executing with a shared external Python kernel.

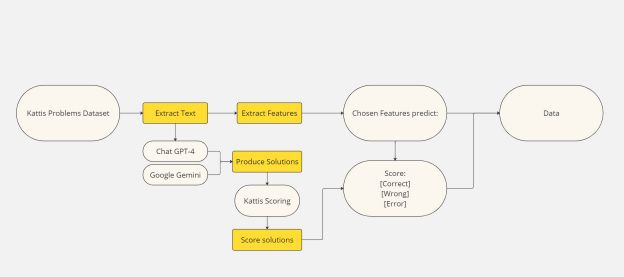


Figure 1: Data Collection

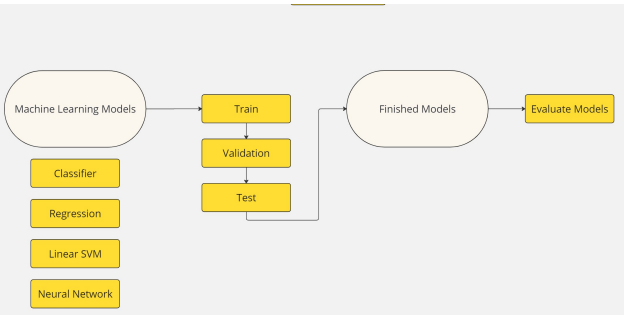


Figure 2: Machine Learning

Once notebook has been setup, we then will create the .ipynb files needed to do our research. These files will be stored on a GitHub Repository. Our research will not require the use of any external hardware nor a computer cluster, anyone could clone the repository and run the code on their device

5.2 Data Collection: Programming Problems

We will source our programming problems from Kattis, an online judging system for programming problems. 164 Kattis prompts have already been collected during our clients previous research, so for these, we just need to parse through the text files and feed them into the LLMs. This will be done using internal Python libraries. An additional 3,705 programming questions were collected from Kattis using AutoKattis, a Kattis API wrapper. They will be accessed as follows:

```
from autokattis import Kattis
import pandas as pd
```

```
import os
import json


user = os.getenv('KATTIS_USER')
pwd = os.getenv('KATTIS_PASS')
kt = Kattis(user, pwd)
all_probs = kt.problems(*[True]*4)
all_ids = [item['id'] for item in all_probs]


filename = 'kattis_problems.json'
with open(filename, 'w') as json_file:
    json.dump(problems, json_file, indent=4)
print(f"Dictionary has been exported as JSON to '{filename}'")
```

5.3 LLM Access

First we will access GPT-4, a paid service from OpenAI. It is normally accessed through a web interface with a text box for input. It would be time consuming to individually input our problems, so we will use the OpenAI API to automate feeding our prompts into the model. API Credits for OpenAI will need to be purchased so the API can be used, providing an API key, GPT-4 will be accessed as follows:

```
import os
from openai import OpenAI
client = OpenAI(
    api_key=os.getenv("OPENAI_API_KEY"),
)
prompt = "Solve the following using Python"
chat_completion = client.chat.completions.create(
```

```
    messages=[ {"role": "user",
                "content": prompt,  } ],
    model="gpt-4",
)
```

The prompt variable will be replaced as we iterate through problems, and the returned chat_completion being indexed for the returned Python solution. This returned Python solution will be saved in a gpt.py file within the problem sub-folder.

Accessing Gemini will be much different. We will also use the Gemini API provided by Google, which is free to use but requires a Google Developer account to be setup with the proper authorizations. Once an API key is obtained, Gemini will be accessed as follows:

```
import google.generativeai as genai
genai.configure(api_key=os.getenv('GOOGLE_API_KEY'))

model = genai.GenerativeModel('gemini-pro')
prompt = "Solve the following using Python"
response = model.generate_content(prompt)
response.text
```

The prompt variable will be replaced as we iterate through the problems and the Python solution can be obtained from response.text. This returned Python solution will be saved in a gemini.py file within the problem's subfolder.

5.4 Scoring LLM Solutions

Now that we have the prompt and two solutions we need to input the solutions into Kattis to determine their accuracy. The returning result from Kattis will be stored as the following

vector:

$$\begin{bmatrix} Difficulty \\ Result \\ Tests\ passed \end{bmatrix} \tag{1}$$

- Difficulty: double value representing Kattis' ranking of how difficult a problem is
- Result: the judgement from Kattis can be accepted, wrong answer, or any form of error such as time-limit exception
- Tests passed: fraction holding how many tests were passed by the inputted code

We will need to iterate through all the solution files (gemini.py or gpt.py) and use the Kattis CLI (Command Line Interface) to obtain the results. They will then be stored as comments at the top of each .py file, for example:

```
# 2.3 accepted 17/17
# code below
```

In this way, a prompt, solutions, and scores are stored as shown in Figure 3

5.5 Data Analysis

The analysis of that data is handled by many libraries in Python. In this context, libraries are a collection of prewritten code that you use to perform certain tasks. The libraries that we are implementing are Pandas and Matplotlib. Pandas will be used to reformat our data into a DataFrame, which will allow us to clean and manipulate it for future graphing. The data that we are working with is in the form of comments at the top of Python files called gpt.py, and in order to obtain the data, we needed to develop code that could parse through many folders and obtain the comments at the top of each gpt.py file. We used os to walk through the directory where the files are located and used re, or regular expression,

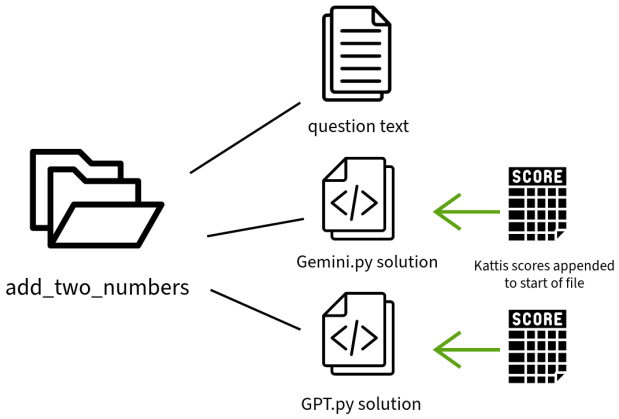


Figure 3: How our data is stored

to match the pattern of the comments so that we could transfer the data into a csv file with appropriate columns.

The graphing of our data will be handled by Matplotlib, with it we can create meaningful graphs such as a stacked bar chart showing the outcomes at each difficulty of problem.

5.6 Feature Analysis

Spacy is another library that will be used in order to tokenize the text of the programming problems. Tokenization is the process of breaking text into smaller parts so that machine analysis will be easier. With the tokenized problem, we can obtain word frequency as a feature. We will then use the textstat library to obtain features of readability, complexity, and grade level of the problems using many of the functions that it provides such as the Dale-Chall readability score, a score which describes the grade level that can understand the text, and text standard. With all the features collected, we will need to perform unsupervised learning in order to find the features that will best allow for a model to predict results when given a problem.

5.7 Feature Extraction - What makes a problem difficult?

One question that our research must answer is, how do we represent determine what makes a programming problem difficult for LLMs to answer? To answer this question we need to analyze the problems gathered from kattis and convert them into numeric features to be used as input in our model. This process is known as feature engineering, we can take a problem such as 2048:

```
2048 is a single-player puzzle game created by Gabriele Cirulli1.
It is played on a 4x4 grid that contains integers >=2 that are powers of 2.
The player can use a keyboard arrow key (left/up/right/down)
to move all the tiles simultaneously.
Tiles slide as far as possible in the chosen direction
until they are stopped by either another tile or the edge of the grid.
If two tiles of the same number collide while moving,
they will merge into a tile with the total value of the two tiles that collided.
The resulting tile cannot merge with another tile again in the same move.
Please observe this merging behavior carefully in all Sample Inputs and Outputs.
```

The problem could be saved in a variable and used in various textstat functions:

```
ARI = textstat.automated_readability_index(problem)
DCR = textstat.dale_chall_readability_score(problem)
DCR_v2 = textstat.dale_chall_readability_score_v2(problem)
FRE = textstat.flesch_reading_ease(problem)
FKG = textstat.flesch_kincaid_grade(problem)
SMOG = textstat.smog_index(problem)
CLI = textstat.coleman_liau_index(problem)
LINSEAR = textstat.linsear_write_formula(problem)
GF = textstat.gunning_fog(problem)
txtstd = textstat.text_standard(problem, float_output=False)
```

```
lex_count = textstat.lexicon_count(problem, removepunct=True)
difficult_words = textstat.difficult_words(problem)
```

Which will result in the problem being represented numerical in terms of readability:

```
ARI,DCR,DCR_V2,FRE,FKG,SMOG,CLI,LINSEAR,GF,Textstd,LexCount,Difficult Words
7.3,7.44,6.12,70.94,7.6,10.3,6.2,10.333333333333334,8.69,2.0,310,32
```

6 Experimental Design

After collecting our data, our experiment itself consists of Model Training and Evaluation/Implementation

6.1 Model Training

Once we have settled on features, we can begin training our models. We will be using four machine learning models: a classifier, a regression model, a linear SVM (support vector machine), and a neural network. We will use XGBoost or Random Forest for the text classifier and Sci-kit Learn for the remaining three models. Our data will now be split into our X variables, which predict our Y variables. The X variable will be the text prompt for the problem.

$$x = \left[textprompt \right] \tag{2}$$

The Y variable will be the score vector. This vector will now be one-hot encoded.

$$y = \begin{bmatrix} correct \\ incorrect \\ errorr \end{bmatrix} \tag{3}$$

So, a correct solution would look like:

$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{4}$$

An incorrect solution would look like:

$$\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \tag{5}$$

A solution causing an error would look like:

$$\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{6}$$

Our data containing the question text, two solutions, and score appended at the top of our files will be split into a training and test data set, with 80% allocated for training and 20% allocated for testing. We will use grid-search to select the best combination of parameters. The parameters to be used in the grid search will depend on the model used. For a logistic regression model, the parameters are the C value, penalty, solver, and max iteration. For an artificial neural network, the parameters are the hidden layer sizes, activation function, solver, alpha value, learning rate, and maximum iterations. For a linearSVM, the parameters will be the C value, kernel, decision function shape, and gamma value. We will also perform cross-validation on the data, and select the best criteria for later evaluation, this being accuracy, precision, or recall.

6.2 Model Evaluation

Once our model has been trained and validated, we will need to evaluate its performance. This will be done by making a prediction on the test set and using Sci-kit Learns classification report to see the accuracy, recall, precision, and f1-score. All of which are important metrics that will help us determine how the model is performing and if we need to go back and tune the parameters again.

7 Results

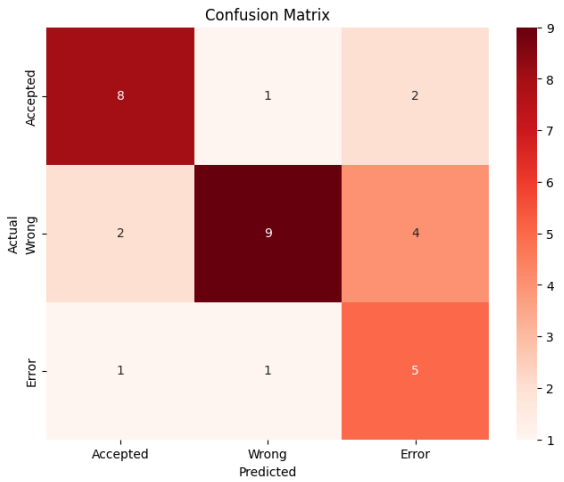
Our current results consists of three completed models: logistic regression, LinearSVM and an artificial neural network. These models were trained, evaluated, and tested on the 164 Kattis problems and data that was provided to us by our client. To analyze the results of our models, we used Scikit's classification report, as it prints several metrics, precision, recall, f1-score, and accuracy. For our research, the f1-score a reliable metric to take into account because of our data imbalance seen in figure 5. We also have confusion matrices for the models which are another way to see the prediction results of our models. The matrices display the number of instances produced by the model on the test data, the instances being true positives(TP), true negatives(TN), false positives(FP) and false negatives(FN).

As shown by the classification report in figure 4, our logistic regression can predict how well chat GPT-3.5 can answer a programming problem with 67% accuracy. By looking at the f1-scores, we can determine the classes that the model was best able to predict. In this case, the model performs well in predicting whether a solution will be accepted or wrong, and performs the worst on predicting if the solution will produce an error. Looking the the confusion matrix we can calculate these values:

```
Accepted Class:
    True Positives: 8
    False Positives: 3
```

	precision	recall	f1-score	support
accepted	0.73	0.73	0.73	11
wrong	0.82	0.60	0.69	15
error	0.45	0.71	0.56	7
accuracy			0.67	33
macro avg	0.67	0.68	0.66	33
weighted avg	0.71	0.67	0.67	33

(a) Logistic Reg. Classification Report



(b) Logistic Reg. Confusion Matrix

Figure 4: Logistic regression model's results with features selected using Principal Component Analysis

```
        False Negatives: 3
        True Negatives: 19
    Wrong Class:
        True Positives: 9
        False Positives: 2
        False Negatives: 6
        True Negatives: 16
    Error Class:
        True Positives: 5
        False Positives: 6
        False Negatives: 2
        True Negatives: 21
```

These values correspond with the results in the classification report showing us that for both the accepted and wrong class, the model performs well. The error class has more false positives than true positives showing that the model doesn't perform well on that class. This could be due to the low instances of that class.

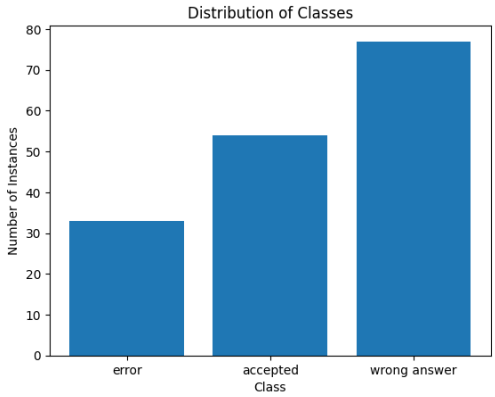
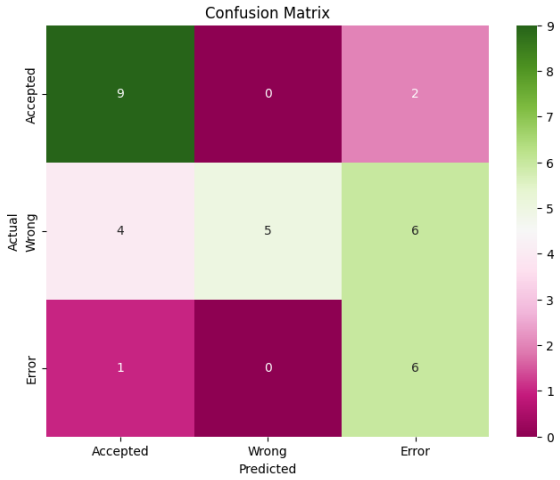


Figure 5: Bar chart showing the number of results in our data

	Precision	Recall	f1-score	support
accepted	0.64	0.82	0.72	11
wrong	1.00	0.33	0.50	15
error	0.43	0.86	0.57	7
accuracy			0.61	33
macro avg	0.69	0.67	0.60	33
weighted avg	0.76	0.61	0.59	33

(a) SVM Classification Report



(b) SVM Confusion Matrix

Figure 6: LinearSVM results with features selected from SelectKBest

As shown in figure 6, the linearSVM model is our worst performing model with it having an accuracy of 61%. The model predicts accepted solutions to best with an f1-score of 72%, with the other classes score being less than 60%. Looking the the confusion matrix we can calculate these values:

```
Accepted Class:
    True Positives: 9
    False Positives: 5
    False Negatives: 2
    True Negatives: 17
```

```
Wrong Class:
    True Positives: 5
    False Positives: 0
    False Negatives: 10
    True Negatives: 18
Error Class:
    True Positives: 6
    False Positives: 8
    False Negatives: 1
    True Negatives: 18
```

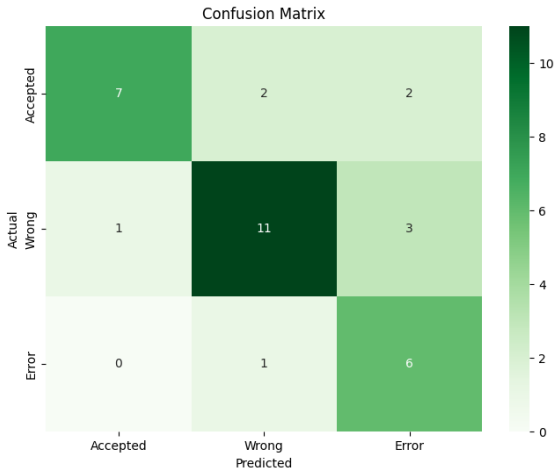
These values show us the model incorrectly predicts a majority of the wrong class as the other two classes and that for the error class the model incorrectly predicts that both of the other classes are errors. This corresponds with the report as the only class that performs well is the accepted class.

Our best performing model is the artificial neural network, with an accuracy of 73%, as shown in 7. The model performs well on predicting all classes, as they all have an f1-score above 60%. Looking the the confusion matrix we can calculate these values:

```
Accepted Class:
    True Positives: 7
    False Positives: 1
    False Negatives: 4
    True Negatives: 21
Wrong Class:
    True Positives: 11
    False Positives: 3
    False Negatives: 4
    True Negatives: 15
```

	Precision	Recall	f1-score	support
accepted	0.88	0.64	0.74	11
wrong	0.79	0.73	0.76	15
error	0.55	0.86	0.67	7
accuracy			0.73	33
macro avg	0.74	0.74	0.72	33
weighted avg	0.76	0.73	0.73	33

(a) ANN Classification Report



(b) ANN Confusion Matrix

Figure 7: Neural networks results with features selected from SelectKBest

```
Error Class:
    True Positives: 6
    False Positives: 5
    False Negatives: 1
    True Negatives: 21
```

These values show that the model does a good job at predicting the actual results across all the classes which corresponds to the classification report.

8 Conclusions

With our research, we wanted to find out if a machine learning model could be built to predict whether a LLM would be able to answer a programming question when provided with a prompt. We wanted to find the answer to this question by creating models on the data of two LLMs, GPT 4 and Gemini; however, due to our data collection not being completed when in a timely manner, we created models on the data of GPT 3.5. Overall, we succeeded, as we have three prediction models, with one of those models having an accuracy over 70%. However, since the model will still be wrong 30% of the time, it is unlikely our

model could be actually used in a professional setting like we intended. One threat to the validity of our research is the small dataset on which our models were trained. Our models were only trained using 164 Kattis questions that were given to us, but we have collected 3,705 more questions. We were unable to collect the results of those problems in a timely manner, as we ran into issues when grading them through the Kattis CLI. The solutions seemed to be always incorrect, and we could not verify if this was an issue with our prompt engineering. If we had more data, we could ensure that the models performed better as there would be a greater quantity of questions to train our models with. The implications of our findings are that software companies should be aware that LLMs will continue to struggle with problems with difficult words, low readability, and long prompts. It will likely take several years before they catch up to human developers. Meanwhile, computer science educators can use this tool (when it becomes more accurate) to confirm if their material is challenging. Other education disciplines can also consult our feature analysis to determine if their coursework can be easily completed by a LLM.

8.1 Future Work

Our main goal for the future is to retrain our models on data from other large language models. We were unable to incorporate data from Google's Gemini and OpenAI's GPT-4, so collecting that data will be our next step. We also wanted to get more programming problems, exploring more unconventional sources for questions, preferably ones that aren't available on the internet. During the course of our research, two additional LLMs were released: Anthropic's Claude 3 Sonnet model and Meta's Llama 3. A future direction of our research would be investigating the effectiveness of those LLM's, as they both boast impressive scoring metrics on the HumanEval set of programming questions. Finally, another future direction of our research would be transforming our models into an actual product that could be distributed to educators, so they can have access to our findings and evaluate their own programming problems for the classroom.

9 Reflections on the Project

9.1 Joshua

The main thing I learned from this project was how to perform research. This was the first time that I've performed research in this way and the entire process from writing this report to training machine learning models was something new that I've learned. The most rewarding part of the research was working with the models, seeing the models produce results that I wanted to see after tuning parameters is the best part because it shows that what we're doing is working. However, I will say that it was also challenging working with the models due to it being my first time working with them.This is where Dr. Ho would come in and steer us on the right track with what functions were best to use, and what parameters weren't needed for our purposes. For other people who are doing a research project, I would suggest that you search for a public dataset that you can use in your research, and if there isn't one then make sure that you focus on collecting that data since the more data you have the better. Also, if you can, have a mentor who can help with the project. If you encounter an issue, they'll know what's going wrong and if they don't they can help you figure it out.

9.2 Sajiv

This project was the first time I was able to do college-level research with a formatted schedule. I am glad I finally got the chance to work at this high level of academia. Over my 4 years at Loyola, I have worked on many of my own personal projects, but never with the careful supervision and advice I received from Dr. Ho and Dr. Olsen. Before, I would just ask a question whenever I needed help and on my own timeline, but with this capstone project, my partner and I were always expected to perform at the highest level and given the weekly support to do so. I learned that research is painfully slow. Usually, with computer science projects we can expect immediate or short-term results, but this time I needed to be much more patient. It made the discoveries we made that much more rewarding in the end,

however. The most difficult part of this project was the documentation. Working with large amounts of data and training machine learning models is a workflow I was familiar with thanks to my internship in the same field, but writing a 20+ research paper as we went took a lot of extra time. It seemed like every time I made a small bit of progress with the code, it was time to drop all of it in order to document what we were doing. While burdensome in the moment, it is definitely satisfying to see the payoff from that incremental documentation, as we can see our full final product. We developed machine learning models as a part of our research, which meant we made programs to scrape our data and pipe it through the necessary APIs. We also made programs to train our models, but since we were just using text data, it did not require major computing. It still had a fair bit of complexity, but on a lower scale. For anyone else doing a research project, I would advise them to find conviction with the importance and relevance of their research goals in the real world. It can act as a great motivator when the work gets monotonous or you get stuck. There were several times where I wanted to just give up on the project, but then I explained what I was doing to my peers and family members, and they seemed really interested in the results, so I knew I had to keep working to find the answers for them.

References

[1] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, An-

drew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *CoRR*, abs/2107.03374, 2021.

[2] Emily Drebelbeis. Chatgpt passes google coding interview for level 3 engineer with $183k salary. 2023.

[3] Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Venkatesh Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. Solving quantitative reasoning problems with language models. *ArXiv*, abs/2206.14858, 2022.

[4] Spyros Makridakis, Fotios Petropoulos, and Yanfei Kang. Large language models: Their success and impact. *Forecasting*, 5(3):536–549, 2023.

[5] Jesse G. Meyer, Ryan J. Urbanowicz, Patrick C. N. Martin, Karen O'Connor, Ruowang Li, Pei-Chen Peng, Tiffani J. Bright, Nicholas Tatonetti, Kyoung Jae Won, Graciela Gonzalez-Hernandez, and Jason H. Moore. Chatgpt and large language models in academia: opportunities and challenges. *BioData Mining*, 16(1):20, Jul 2023.

[6] Dominik Sobania, Martin Briesch, Carol Hanna, and Justyna Petke. An analysis of the automatic bug fixing performance of chatgpt. In *2023 IEEE/ACM International Workshop on Automated Program Repair (APR)*. IEEE, May 2023.

[7] Nghia D. Tran, James J. May, Nguyen Ho, and Linh B. Ngo. Exploring chatgpt's ability to solve programming problems with complex context. *J. Comput. Sci. Coll.*, 39(3):195–209, oct 2023.

[8] Michel Wermelinger. Using github copilot to solve simple programming problems. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V.*

1, SIGCSE 2023, page 172–178, New York, NY, USA, 2023. Association for Computing Machinery.