GPT-3 vs Object Oriented Programming Assignments: An Experience Report

Bruno Pereira Cipriano
Pedro Alves
bcipriano@ulusofona.pt
pedro.alves@ulusofona.pt
Lusófona University, COPELABS
Lisbon, Portugal

ABSTRACT

Recent studies show that AI-driven code generation tools, such as Large Language Models, are able to solve most of the problems usually presented in introductory programming classes. However, it is still unknown how they cope with Object Oriented Programming assignments, where the students are asked to design and implement several interrelated classes (either by composition or inheritance) that follow a set of best-practices. Since the majority of the exercises in these tools' training dataset are written in English, it is also unclear how well they function with exercises published in other languages.

In this paper, we report our experience using GPT-3 to solve 6 real-world tasks used in an Object Oriented Programming course at a Portuguese University and written in Portuguese. Our observations, based on an objective evaluation of the code, performed by an open-source Automatic Assessment Tool, show that GPT-3 is able to interpret and handle direct functional requirements, however it tends not to give the best solution in terms of object oriented design. We perform a qualitative analysis of GPT-3's output, and gather a set of recommendations for computer science educators, since we expect students to use and abuse this tool in their academic work.

CCS CONCEPTS

• **Applied computing** → **Computer-assisted instruction**.

KEYWORDS

programming assignments, teaching, object oriented programming, large language models, gpt-3

ACM Reference Format:
Bruno Pereira Cipriano and Pedro Alves. 2023. GPT-3 vs Object Oriented Programming Assignments: An Experience Report. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2023), July 8–12, 2023, Turku, Finland.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3587102.3588814

ITiCSE 2023, July 8–12, 2023, Turku, Finland

ACM ISBN 979-8-4007-0138-2/23/07.
https://doi.org/10.1145/3587102.3588814

1 INTRODUCTION

Large Language Models (LLM) are machine learning models that are trained to predict the next word in a continuous vector representation of words instead of just measuring probabilities over strings drawn from a vocabulary [4]. These models are typically trained on very large datasets, and use a neural network architecture to learn patterns in the data and make predictions.

When these models are trained with data that includes software code, they have shown impressive performance in generating functioning computer code from natural language specifications [17].

An example of a tool with such capacity is GPT-3[1], which is currently open to the general public in multiple formats, such as a chat bot[2], an interactive "playground" which allows some parameter tuning[3], and as an API which can be called upon to perform text and code generation tasks[4].

Prior work has shown that GPT-3 is able to generate code at the level of an introductory programming class [5, 10]. However, as far as we know, no previous study focusing on the interaction between GPT-3 and Object Oriented Programming (OOP) assignments exists. As teachers of an advanced class, where students learn and apply OOP, we decided to investigate its ability to generate code for more complex problems: programs where the students have to create code across multiple classes, and where the evaluation depends not only on having working code, but also on semi-subjective evaluation of the code quality.

We were also curious about GPT's capacity to reply to prompts written in a language other than English, since it gathers information from the Common Crawl dataset which is primarily comprised of English content (46% as of 2022) [9]. All of our assignments are written in Portuguese, the language of the university and its student body, the vast majority of whom are either native speakers or fluent in that language. The Common Crawl dataset includes only 2% of Portuguese content [9].

This paper is organized as follows: first, we start by reviewing related work. Then, we describe our course, as well the assignments that were used to evaluate GPT-3's competence level. Afterwards, we present a detailed log of one of our interactions with GPT-3, after which we present the results of all the analyzed assignments, leading to a discussion of opportunities raised by GPT-3, as well as some recommendations for other teachers. Finally, we discuss the

[1] Generative Pre-trained Transformer
[2] https://openai.com/blog/chatgpt/
[3] https://beta.openai.com/playground
[4] https://beta.openai.com/docs/api-reference/introduction

limitations of our study, draw some conclusions and present future work.

2 RELATED WORK

Several studies have been conducted on the effectiveness of LLM-based code generating techniques for solving short programming exercises such as those usually found on introductory programming courses. In [6], researchers were able to solve 70% of 164 Python programming exercises using code generated by Codex, a GPT language model trained on code publicly available in GitHub[5]. The authors of [10] experimented with Codex as well on several introductory programming assignments and compared its results with the students' results. Codex managed to make the 17th position when ranked alongside 71 students. Finally, in [5], researchers have shown that GPT-J (an open-source alternative to the GPT-3 used in this article) generated correct solutions to 6 out of 7 programming exercises.

All these studies resort to short direct programming exercises that are usually solved with a single pure function, which simply transforms a set of inputs into an expected output without side-effects. To the best of our knowledge, no studies have experimented with OOP exercises which involve creating multiple classes (with mutable state and encapsulated behavior) that interact between each other to reach a certain goal.

The use of Automated Assessment Tools (AATs) to validate programming exercises is a well researched topic, with demonstrated benefits for students and teachers alike [11]. Not only these tools can assess the functional correctness of the submitted solutions, but their quality, completeness and efficiency as well, among other dimensions [13]. For OOP assignments in particular, the code quality dimension assumes a prominent place since reusability is one of the core tenets of this paradigm [16]. Although several studies refer having used AATs to validate the functional correctness [10], we didn't find any reference to automatic validation of code quality indicators regarding code generated by GPT-3.

3 OUR COURSE

This research was based on assignments from a course belonging to a Computer Engineering degree, where both authors of this paper have been involved as teachers since 2015. The course occurs on the 2nd curricular year and is mostly focused on teaching Object Oriented Design and Programming (OOP), using the Java programming language.

In this course, we use Drop Project, an open-source Automatic Assessment Tool (AAT) in order to guide students in their work [8]. The AAT is used for in-class exercises, homeworks, and evaluations such as projects and tests. This tool evaluates the student's code using teacher-defined Unit Tests. Besides validating the code's correctness, it also performs some code quality validations, using rules defined by the teachers. For example, we setup a rule that forbids the usage of keywords like "instanceof" and "getClass()" when we want to validate if the student's code follows the object oriented best practice of preferring method dispatching over explicit type testing. However, due to limitations of the technology used for the code quality validations (the Checkstyle plugin [12]), some validations have to be implemented using Unit Tests. An example of this is checking if a certain class was declared as abstract.

4 THE ASSIGNMENTS

In this study we analyzed 6 assignments that have been used in our course as tests focused on inheritance and polymorphism. In each school year, and depending on the number of enrolled students, we usually have 5 different versions of this test. Each version has a different business domain, as well as different requirements. All versions require that the students implement an inheritance relationship, a composition relationship, some getters and setters, a non-trivial "toString()" function and some functions that have to create, query and/or manipulate objects of several classes. Furthermore, all versions have certain behaviours that depend on the object type, in order to evaluate if the students are able to come with a solution which does not require explicit type testing such as the ones permitted by the "instanceof" keyword and the "getClass()" function. However, the behaviours that depend on the type are not always located in same place. In some cases, it's an auxiliary function that should only consider certain objects, in other cases the "toString()" function must be implemented only in the super-class while returning different values for each sub-class, and so on. Finally, in some cases, we ask for student information to be included as the value of one or more objects, to guarantee unique solutions among students (for example, set the 'name' attribute of an object to be equal to the student's student ID number).

We used 6 assignments:

- 3 assignments from the current school year (i.e. 2022/2023)
- 2 assignments from the previous school year (i.e. 2021/2022)
- 1 assignment from 2018/2019.

The first five assignments were chosen randomly from a total of ten that were used in the respective school years. The last assignment was added to the collection because it is of a higher difficulty level than the most recent assignments. The purpose of this cherry picked assignment was to better understand the tool's capacities and limitations.

5 METHODOLOGY

To interface with GPT-3, we used the "OpenAI Playground". The used model was "text-davinci-003". This model is described in OpenAI's Playground's web page[6] as the "Most capable model in the GPT-3 series. Can perform any task the other GPT-3 models can, often with higher quality, longer output and better instruction following." The temperature parameter (which controls randomness) was left at the default value of 0.7 and the maximum length parameter was left at the default of 256.

For each assignment, we followed the procedure of pasting the corresponding text into OpenAI's Playground and submitting it as the initial prompt. After examining the output, we determined if it met the necessary criteria (i.e. all mandatory classes and functions are present). When GPT-3 reply was lacking or incomplete, extra prompts were provided to direct it towards the correct answer. This process was repeated until GPT-3 provided all the necessary classes

[5] https://github.com/

[6] https://beta.openai.com/playground

and functions. Subsequently, a manual inspection was conducted to identify any syntactic, logical, and output format errors. The identified syntactical errors were manually fixed. The adjusted solution was then submitted to the AAT, with the goal of verifying its correctness, completeness, and quality of the code. After cross-referencing the results of the AAT with our manual inspection, minor logical and output format mistakes were corrected and the updated solution was re-evaluated. Finally, the updated validation results were analyzed to confirm the accuracy of our initial analysis.

6 OUR EXPERIMENT WITH THE 'IT COMPANY' ASSIGNMENT

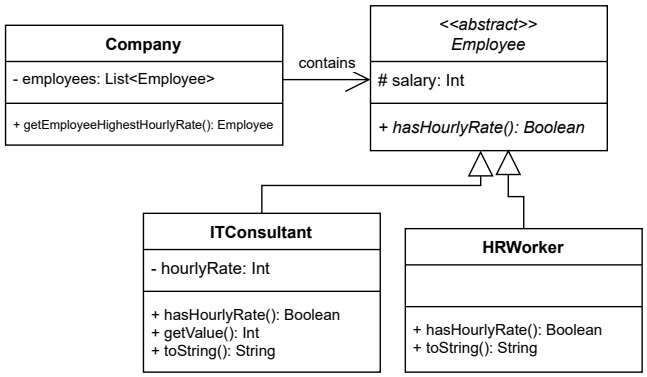

Figure 1: Simplified UML class diagram of the "IT Company" assignment. Constructors and less relevant attributes and methods were omitted. This diagram represents part of the expected solution and is not supplied to the students, who only receive textual instructions.

In this section, we describe our interactions with GPT-3 while trying to solve the 'IT Company' assignment, given to students on 2022/23. This assignment is identified as "MT 2 2022/2023 v4".

The business domain of the assignment is an IT company with the following concepts: "Company", "Employee", "ITConsultant", and "HRWorker". The concepts and their attributes are explained to the students by text. In this assignment, we expect the students to understand that there is an inheritance relationship in the concepts, with the class "Employee" being the super-class of the classes that represent an "ITConsultant" and an "HRWorker". The students should also understand that the "Employee" class is abstract, since the text mentions that it must not be possible to instantiate that class. Finally, the students must implement a number of mandatory functions, identifying the class of the hierarchy where each function belongs. For example, there is an attribute which only makes sense for the "ITConsultant" class: the hourly rate. As such, only that class should have that attribute and the respective getter, which must be called "getValue()". Also mandatory is the creation of a function called "getEmployeeHighestHourlyRate()" in the "Company" class. This function must return the "Employee" with the highest hourly rate. However, since the hourly rate concept only applies to the "ITConsultant" class, the student must find the proper Object-Oriented design to implement this challenge. As such, if the student uses "instanceof" or "getClass()", a penalty will be applied to the respective grade. Figure 1 presents an overview of this assignment's required classes, as well as the most relevant design expectations. An English version of this assignment is available online [7].

Following is an overview of our interactions with GPT-3 with the goal of obtaining a working solution for the above mentioned assignment. Please note that our interactions with GPT-3 were done in Portuguese. As such, when we quote a certain prompt, we are using Google Translate's[8] translation of the Portuguese text we actually used.

To perform the experiment, the assignment's description text was provided to GPT-3 as an initial prompt. For the first 5 submissions, GPT-3's output didn't include any code. We added a new prompt: "Implement the code for this system". This was not effective, since we performed 2 submissions without receiving any output. We changed the prompt to specifically ask for Java code, which resulted in 2 submissions without improvements. Then, we deleted GPT-3's original output and kept only the previous prompt. The 10th submission resulted in code output. Since GPT-3's output was only part of a class, we pressed the Submit button to get more data. Five more submissions were needed to get the code for most classes, with the exception being the "Main" class. Then, we performed extra prompts to guide GPT-3 towards creating the "Main" class. The first of those prompts was "Also implement the Main class as indicated". GPT-3 replied with a "Main" class containing a "main(...)" function with code to perform the creation of a "Company" object, as well as a call to "getEmployeeHighestHourlyRate()" (this was the 16th submission). Next, we asked GPT-3 to generate the missing function: "Add the myCompany() function to the "Main" class". GPT-3 returned a function with an incompatible signature: returning void instead of "Company" and receiving an argument of type "Company", instead of not receiving anything. Finally, we repeated the assignment's original text regarding class "Main", which resulted in GPT-3 returning an implementation of the "Main" class containing a "myCompany()" function with the required protocol.

At this point, after 18 submissions, we had a solution that contained all mandatory components (classes and functions). Table 1 presents an overview of all the required classes and functions, as well as GPT-3's performance over each item. In a preliminary analysis, the following problems were found:

- The ArrayList class, which was used to implement class "Company", was not imported. Item (a) in Table 1.
- The "getEmployeeHighestHourlyRate()" function was implemented using "instanceof". Item (b) in Table 1.
- In the "ITConsultant" class, the "toString()" function was incorrect: the assignment asked for "ITConsultant" to be returned, but GPT-3's implementation returns that value prefixed with the package name (i.e. "lp2.minitest2.ITConsultant"). Item (c) in Table 1.

We manually fixed the compilation error, since it would not compile when submitted to the AAT. Afterwards, we submitted the adjusted solution to the AAT. GPT-3's solution passed 9 out of

[7] https://github.com/drop-project-edu/itCompanyAssignment
[8] https://translate.google.com/

Table 1: Overview of the required classes and functions in the 'IT Company' assignment. GPT-3 created all required classes and functions, although some errors were found. The Main class and the myCompany() function were only created after multiple prompts.

Item	Done?
package	✓
class Employee w/ 3 attributes	✓
class HRWorker extending Employee	✓
class ITConsultant extending Employee	✓
class Company w/ 1 attrib. (name)	✓ With problems (a)
Constructor HRWorker w/ 3 args	✓
Constructor ITConsultant w/ 3+2 args	✓
Employee: three getters (id, name and salary)	✓
ITConsultant: 2 getters (client, value)	✓
Company.getName()	✓
Company.getEmployees()	✓
Company.addEmployee()	✓
Company.getEmployeeHighestHourlyRate()	✓ With problems (b)
ITConsultant.toString()	✓ With problems (c)
Main class w/ Main.myCompany()	✓

13 unit tests. It failed the code quality validation, because it uses "instanceof". The reasons for each failed test are:

- Test 3 fails because the "Employee" class is not abstract.
- Tests 10 and 12 fail due to problems in the ITConsultant's "toString()" function.
- Test 13 fails because the assignment asks that the "Company" object returned by "myCompany()" has as 'name' the last four digits of the student's ID. Since we did not input the student's ID to GPT-3, this problem was expected.

To ensure that tests 10 and 12 were failing strictly due to the problem identified in our pre-submission analysis, we removed the package name from the ITConsultant's "toString()" function. After this, we re-submitted, and the adjusted solution achieved a 11/13 result in the unit tests.

6.1 Attempts to improve the solution

We tried guiding GPT-3 towards a solution that did not use "instanceof" by entering the following prompt: "Change the code to not use instanceof". GPT-3's reply consisted on returning the Main class, this time adding the "getEmployeeHighestHourlyRate()" to it. Note that this is not correct, because that method belongs in the "Company" class. It also results in a compilation error, because the function uses a variable - the list of "Employees" - that is not declared in the "Main" class. That being said, GPT-3 did remove the usage of "instanceof" from the function. However, while doing so, it assumed that all instances of the "Employee" class implement the function that returns the hourly rate (i.e. "getValue()").

We performed another experiment by asking GPT to recreate the "Company" class without using "instanceof". Our prompt was: "Change the code of the Company class so as not to use instanceof". GPT's reply to this was to create the "Company" class and add to it the same version of the method that it had previously placed in the "Main" class. This version does not have the compilation error related with the list of "Employees", but it does have the error related with the "getValue()" function.

Afterwards, we described the problem to GPT-3 in order to evaluate its reaction. Prompt: "This code doesn't work, because the HRWorker class doesn't have the getValue() function." GPT's reply: "For this code to be valid, it will be necessary to add the getHourlyRate() function to the Employee class, and to the HRWorker class add the getHourlyRate() function, so that both return the value 0.". The suggested solution has two issues:

- It mentions a function called "getHourlyRate()", but the name of the hourly rate getter was explicitly called "getValue()".
- It suggests adding the "getHourlyRate()" function to both the super-class and the sub-class, having both of them return 0, which is not ideal, since it results in code duplication. It would be preferable to: 1) declare the function in the super-class to return 0, or 2) declare it in the super-class as abstract and implement a concrete version that returns 0 in the "HRWorker" class.

Our experience shows that, even with proper guidance, GPT-3 wasn't able to provide a working solution that did not use explicit type testing. A full log of our interaction is available in [7].

7 RESULTS

In this section we describe GPT-3's behaviour over the six assignments. Table 2 presents an overview of GPT-3's performance. The following subsections highlight the problems found in GPT-3's solution for each assignment.

MT 2 2022/23 v1 - Problems and notes

Theme: Real estate consultancy agency with the following concepts: Realtor, RealEstate, Apartment, and House.

Compilation errors: uses the ArrayList class, but does not import it. Missing end-of-class curly bracket (e.g. }) in class "Main". Classes "Apartment" and "House" were created as static inner classes of the "RealEstate" class, which causes a compilation error in the AAT.

The assignment had a mistake: it did not request the "toString()" for classes "Apartment" and "House". This causes two errors in tests, but the original problem was in the assignment. We only discovered this mistake after analysing GPT-3's solution.

In "Realtor.toString()", the package name was included in Strings that were used for output, causing two errors. For example: instead of "Apartment", "lp2.minitest2.Apartament" was used.

Incorrect student number and name in returned object (expected).

The code quality validation failed because "instanceof" was used twice, to implement the "Realtor.toString()" function.

MT 2 2022/2023 v4 - Problems and notes

Note: this assignment is the one described in detail in Section 6.

MT 2 2022/23 v5 - Problems and notes

Theme: Banking application, with multiple types of bank movements/operations. Concepts: Operation, Transfer, DebitTransfer, CreditTransfer, ServicePayment, GovernmentalPayment.

Compilation error: uses the ArrayList class without importing it.

Fails a test because the super-class was not declared as abstract.

Fails two tests because it tried to distinguish debits from credits using the object's value attribute (i.e. less than zero would be a

Table 2: Evaluation of GPT-3's solution for each assignment. The values are related to the first solution that contained all mandatory classes and functions. For example, it took 5 submissions to give all classes and functions for the first assignment, that solution had 4 compilation errors, the code quality failed and it passed 8/13 teacher tests.

ID	Nr. of Submits	Nr. of Compilation Errors	Code quality validation Ok?	Tests passed
MT 2 2022/23 v1	5	4	No	8/13
MT 2 2022/23 v4	18	1	No	9/13
MT 2 2022/23 v5	7	1	Yes	7/13
MT 2 2021/22 v3	5	12	Yes	11/13
MT 2 2021/22 v5	5	3	Yes	8/10
MT 2 2018/19 v5	11	3	No	11/16

debit, and vice versa). In this case, the expected solution was for the class itself to differentiate between a debit and a credit (i.e. "DebitTransfer" corresponds to a debit operation). Also, GPT-3's solution is not coherent with its own "toString()", which explicitly writes a "-" prefixing the value of the attribute, which means that a value of -10 would result in "- -10" being returned.

Fails three tests because the assignment indicated that the sub-classes should not have a specific "toString()" method (i.e. only the super-class should have a "toString()").

MT 2 2021/22 v3 - Problems and notes

Theme: Condominium payments application, with the ability to calculate each unit's monthly payment. Concepts: RealEstate, Apartment, Store, Garage.

Compilation errors: Did not declare the package. Uses the List and ArrayList classes, but does not import them. Declared the 3 attributes of the super-class as private, but tries to use them in the sub-classes, resulting in 6 compilation errors (2 per sub-class). Constructors with extra arguments: did not realise that some default values only vary by object type and added extra arguments to initialize those values (e.g. the floor number of a "Garage" should always be -1). Although this case corresponds to valid Java code, failing to respect the required protocols results in the compilation failing in the AAT (3 errors, one per sub-class).

Fails two tests because the argument order was not respected in the constructor for the Apartment class. In this case, GPT-3 did not comply with the instruction that said that the arguments should be declared in the same order that they appear in the text. This problem went unnoticed in our initial analysis - we needed the AAT's feedback to detect it.

Another test failed due to an incorrect student number in object returned (expected).

MT 2 2021/22 v5 - Problems and notes

Theme: Home cinema system with multiple audio/video components. Concepts: Equipment, Television, AudioColumn (speakers), and Blu-Ray player.

Compilation errors: 3, all due to missing imports in the Main class: List, ArrayList and Arrays.

Two tests fail because the "toString()" are displaying the information in the wrong order: the object type's (e.g. "Television") was placed at the beginning when it should be at the end.

MT 2 2018/19 v5 - Problems and notes

Theme: Railway operator with multiple Rolling Stock Material (or vehicle) types. Concepts: RollingStock, Locomotive (has traction), Railcar (self-propelled car; has traction), Carriage, and Wagon.

Compilation errors: Did not declare the package. Failed to include 3 used library classes: List, ArrayList and HashMap.

Fails a test because the name of the package is included in the toString()'s return value.

Fails a test because it did not understand that "Railcars" have traction capacity, which resulted in an incorrect implementation of a function that must be able to check if a train consist (sequence of rolling stock material) is valid. For example, one of the rules for a consist to be considered valid is: the first vehicle must have traction capacity. Note that it correctly modelled "Locomotives" as having traction capacity. The assignment's text indicates that "Railcars" have such capacity in the same way that it does for "Locomotives".

Fails a test because it uses the name of the package in a function that was required to create an hash-table. The hash-table's key is supposed to be the name of object type, and prefixing that name with the package causes problems.

Fails another test because class "Wagon" inherits the method "getNrFirstClassSeats()" from the "RollingStock" class. Although this solution allows escaping the usage of "instanceof" in one of the required functions, it also results in placing a method in a class where it does not make a lot of sense, since not all "RollingStock" objects have the number of first/second class seats concepts.

Fails a test due to an incorrect student number (expected).

Fails the code quality validation due to 6 uses of "instanceof".

As we mentioned earlier, this assignment was the hardest in the collection. GPT-3's solution passed 11/16 tests (which is average, compared to the other assignments) but had worse results on code quality.

7.1 Summary

The following is a summary of all problems found in GPT-3's solutions for the 6 assignments:

- Failing to declare a required package.
- Incorrectly including the package name in Strings.
- Using type testing to decide behaviours (i.e. "instanceof" / "getClass()").
- Failing to include library imports.
- Accessing a super-class's private fields from its sub-classes.
- Failing to declare a class as abstract when the assignment text explicitly says that it should not be instantiated.

- Not respecting the required syntax of the "toString()" (e.g. wrong order of components in returned String).
- Declaring constructors with problems (e.g. adding arguments that should not be in the constructor, and/or not respecting the argument order indicated by the assignment).
- Failing to correctly identify and implement business rules (e.g. "DebitTransfer" in "MT 2 2022/23 v5").
- Failing to apply inheritance best practices.
- Suggesting solutions with code duplication.

8 IMPACT ON COMPUTER SCIENCE EDUCATION

Our research shows that GPT-3 has problems handling OOP assignments: although some functional requirements are met, it fails to adhere to the fundamental best practices of that paradigm, resulting in incorrect behaviors, as well as lower quality code (e.g. with repetitions). Still, it managed to pass several tests in our AAT. This creates some challenges for computer science educators. That being said, we should embrace the opportunities it creates [3]. In this section will will discuss some possibilities that this technology presents us, as well as some recommendations for the computer science education community.

8.1 Opportunities for CS Educators

GPT-3 can be used for assignment validation, both to check for completeness (as demonstrated by one of our examples, where GPT-3 did not create two "toString()" functions because they were missing from the assignment's description), as well as to calibrate the assignment's difficulty.

It can also be used to generate automatic tests for assignments. We performed such an experience, supplying it with a simple class and asking it to create some unit tests for that class. GPT-3 provided us with a valid response. Our findings are consistent with the work of [15]. However, care must be taken in order to ensure that the generated tests are good.

These two capabilities could be leveraged by teachers to revamp existing assignments and even to develop new ones every year.

8.2 Recommendations for the CSE community

We believe that GPT-3 should be used as a tool to support the students while they are doing the programming exercises, in the same way IDEs provide hints and auto-completions, following a trend of augmenting their capabilities through AI [1]. However, the exercises should be adapted to this new reality, with problems that can't be solved with simple direct prompts to GPT-3 and little student intervention. Here are our recommendations to decrease the likelihood of getting a good grade just by performing prompts.

- Require the usage of specific packages and consider even asking for multiple packages.
- Improve the validation and evaluation of code quality, such as proper OO design (e.g. disallow the usage of keywords such as "instanceof" and "getClass()").
- Shift to "project based" evaluation schemes, requiring that the students model more complex realities and implement more complex interactions.
- Design exercises with certain ambiguities in the text. If an AAT is being used to validate the assignment, move some information from the assignment's text to the unit tests that are used to validate the assignment.
- Use exercises that partially depend on student information, such as student's name or ID. For example, a function could have a certain behaviour when the student's ID is an even number and a another behaviour when it is an odd number. Although the student can guide GPT-3 to overcome this issue, in situations where time is limited, such as tests, it might create an extra layer of difficulty in obtaining a good grade (as Table 2 shows, GPT-3 typically requires multiple submissions before returning a solution).
- Embrace GPT-3 into your classroom, teaching students how to critically evaluate its solutions and to guide it towards higher quality code. This is particularly important, since previous research has shown that these data-driven code generation techniques sometimes suggest code with known vulnerabilities [2, 14].

9 LIMITATIONS

Following are some limitations of our study: GPT-3's behavior is not deterministic, making it hard to replicate our study. Also, since its behaviour changes between executions of the same prompts, it is not possible for us to affirm that it will never give a better (or worse) performance, given the same assignment text. Additionally, we did not perform cross validations using different values for the temperature and maximum length parameters. Finally, since we used a free account to interact with GPT-3, the amount of tokens processed at each time was limited. In some of our experiments, this led to the need to adjust prompts and perform extra submissions.

10 CONCLUSIONS AND FUTURE WORK

Our observations show that GPT-3 is currently able to implement solutions to OOP exercises with minor compilation errors and with some logic errors. These solutions are able to get decent to good scores in evaluations done by an Unit tests-based AAT.

We also observed that GPT-3 does not give the ideal solutions from the OOP perspective. While this might change in the future, it might be an opportunity for educators to improve their current assessment practices.

GPT-3 is also able to handle the Portuguese language without apparent difficulty, but we did not perform any cross validations using assignments in English. However, we expect the English version to perform equally well, if not better, since it has more training data in English than in Portuguese [9].

We will investigate GPT-3's (or newer versions') ability to handle larger OOP assignments (e.g. 10-15 classes), using a different methodology where we provide the AAT's feedback to GPT-3 in an interactive fashion.

ACKNOWLEDGMENTS

This research was funded by the Fundação para a Ciência e a Tecnologia under Grant No.: UIDB/04111/2020 (COPELABS).

REFERENCES

[1] Pedro Alves and Bruno Pereira Cipriano. 2023. The centaur programmer – How Kasparov's Advanced Chess spans over to the software development of the future. arXiv:2304.11172 [cs.HC]

[2] Owura Asare, Meiyappan Nagappan, and N. Asokan. 2022. Is GitHub's Copilot as Bad As Humans at Introducing Vulnerabilities in Code? https://doi.org/10.48550/arXiv.2204.04741 arXiv:2204.04741 [cs].

[3] Brett A. Becker, Paul Denny, James Finnie-Ansley, Andrew Luxton-Reilly, James Prather, and Eddie Antonio Santos. 2022. Programming Is Hard – Or at Least It Used to Be: Educational Opportunities And Challenges of AI Code Generation. http://arxiv.org/abs/2212.01020 arXiv:2212.01020 [cs].

[4] Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. 2000. A Neural Probabilistic Language Model. In *Advances in Neural Information Processing Systems*, Vol. 13. MIT Press. https://proceedings.neurips.cc/paper/2000/hash/728f206c2a01bf572b5940d7d9a8fa4c-Abstract.html

[5] Stella Biderman and Edward Raff. 2022. Fooling MOSS Detection with Pretrained Language Models. http://arxiv.org/abs/2201.07406 arXiv:2201.07406 [cs].

[6] 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, Andrew 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. 2021. Evaluating Large Language Models Trained on Code. http://arxiv.org/abs/2107.03374 arXiv:2107.03374 [cs].

[7] Bruno Pereira Cipriano and Pedro Alves. 2023. (2023). GPT-3 vs MT2 2022/2023 v4 - Full Interaction Log (1.0.0). https://doi.org/10.5281/zenodo.7851214

[8] Bruno Pereira Cipriano, Nuno Fachada, and Pedro Alves. 2022. Drop Project: An automatic assessment tool for programming assignments. *SoftwareX* 18 (June 2022). https://doi.org/10.1016/j.softx.2022.101079

[9] CommonCrawl. 2023. Statistics of Common Crawl Monthly Archives by commoncrawl. https://commoncrawl.github.io/cc-crawl-statistics/plots/languages

[10] James Finnie-Ansley, Paul Denny, Brett A. Becker, Andrew Luxton-Reilly, and James Prather. 2022. The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming. In *Australasian Computing Education Conference*. ACM, Virtual Event Australia, 10–19. https://doi.org/10.1145/3511861.3511863

[11] Petri Ihantola, Tuukka Ahoniemi, Ville Karavirta, and Otto Seppälä. 2010. Review of recent systems for automatic assessment of programming assignments. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research - Koli Calling '10*. ACM Press, Berlin, Germany, 86–93. https://doi.org/10.1145/1930464.1930480

[12] Roman Ivanov et al. 2023. Checkstyle. https://checkstyle.org/. [Online; last accessed 20-January-2023].

[13] José Carlos Paiva, José Paulo Leal, and Álvaro Figueira. 2022. Automated Assessment in Computer Science Education: A State-of-the-Art Review. *ACM Transactions on Computing Education* 22, 3 (Sept. 2022), 1–40. https://doi.org/10.1145/3513140

[14] Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, and Ramesh Karri. 2022. Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions. In *2022 IEEE Symposium on Security and Privacy (SP)*. 754–768. https://doi.org/10.1109/SP46214.2022.9833571 ISSN: 2375-1207.

[15] Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. 2022. Automatic Generation of Programming Exercises and Code Explanations Using Large Language Models. In *Proceedings of the 2022 ACM Conference on International Computing Education Research V.1*. ACM, Lugano and Virtual Event Switzerland, 27–43. https://doi.org/10.1145/3501385.3543957

[16] Peter Wegner. 1990. Concepts and paradigms of object-oriented programming. *ACM SIGPLAN OOPS Messenger* 1, 1 (Aug. 1990), 7–87. https://doi.org/10.1145/382192.383004

[17] Frank F. Xu, Uri Alon, Graham Neubig, and Vincent Josua Hellendoorn. 2022. A systematic evaluation of large language models of code. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*. ACM, San Diego CA USA, 1–10. https://doi.org/10.1145/3520312.3534862