LLM-itation is the Sincerest Form of Data: Generating Synthetic Buggy Code Submissions for Computing Education

Juho Leinonen
Aalto University
Espoo, Finland
juho.2.leinonen@aalto.fi

Paul Denny
The University of Auckland
Auckland, New Zealand
paul@cs.auckland.ac.nz

Olli Kiljunen
Aalto University
Espoo, Finland
olli.kiljunen@aalto.fi

Stephen MacNeil
Temple University
Philadelphia, USA
stephen.macneil@temple.edu

Sami Sarsa
University of Jyväskylä
Jyväskylä, Finland
sami.j.sarsa@jyu.fi

Arto Hellas
Aalto University
Espoo, Finland
arto.hellas@aalto.fi

Abstract

There is a great need for data in computing education research. Data is needed to understand how students behave, to train models of student behavior to optimally support students, and to develop and validate new assessment tools and learning analytics techniques. However, relatively few computing education datasets are shared openly, often due to privacy regulations and issues in making sure the data is anonymous. Large language models (LLMs) offer a promising approach to create large-scale, privacy-preserving synthetic data, which can be used to explore various aspects of student learning, develop and test educational technologies, and support research in areas where collecting real student data may be challenging or impractical. This work explores generating synthetic buggy code submissions for introductory programming exercises using GPT-4o. We compare the distribution of test case failures between synthetic and real student data from two courses to analyze the accuracy of the synthetic data in mimicking real student data. Our findings suggest that LLMs can be used to generate synthetic incorrect submissions that are not significantly different from real student data with regard to test case failure distributions. Our research contributes to the development of reliable synthetic datasets for computing education research and teaching, potentially accelerating progress in the field while preserving student privacy.

CCS Concepts

• **Social and professional topics → Computing education**; • **Computing methodologies → Artificial intelligence**; • **Software and its engineering → Software testing and debugging**.

Keywords

generative AI, genAI, large language models, LLMs, GPT-4o, prompt engineering, synthetic data, bugs, submissions, data generation

ACE 2025, Brisbane, QLD, Australia

ACM ISBN 979-8-4007-1425-2/25/02
https://doi.org/10.1145/3716640.3716647

ACM Reference Format:
Juho Leinonen, Paul Denny, Olli Kiljunen, Stephen MacNeil, Sami Sarsa, and Arto Hellas. 2025. LLM-itation is the Sincerest Form of Data: Generating Synthetic Buggy Code Submissions for Computing Education. In *27th Australasian Computing Education Conference (ACE 2025), February 12–13, 2025, Brisbane, QLD, Australia.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3716640.3716647

1 Introduction

Computing education has witnessed a significant transformation with the rise of large language models (LLMs) [13]. LLMs have demonstrated remarkable capabilities in tasks relevant to computing educators and researchers, with a particular focus on their ability to solve introductory programming exercises [18]. More recently, these capabilities have been demonstrated for more complex programming tasks [20], with current state-of-the-art models appearing able to solve almost all typical introductory programming exercises [47]. This ability to generate correct code solutions is well-established, however less well explored is the potential of LLMs to deliberately create incorrect code.

Generating incorrect solutions is a relatively unexplored area but has many potential applications. Incorrect code solutions can be used to create debugging exercises, which are known to be beneficial for learning [54]. Additionally, they can help in generating synthetic datasets of student submissions, which include a mixture of correct and incorrect code. This is particularly valuable given the scarcity of openly shared programming education datasets, which are often constrained by strict privacy regulations and the challenges of de-identifying and anonymizing data [16, 30]. Leveraging LLMs to create synthetic data can overcome these barriers, providing a new avenue for developing and validating educational tools and techniques without compromising student privacy.

Building on the extensive body of literature that has established LLMs' capability to generate correct solutions, in this work we explore the possibility of using these models to generate incorrect code submissions for introductory programming problems. We investigate various prompting strategies to determine which approaches produce submissions that most closely resemble real student data. In our prior work, we established that LLMs can generate buggy solutions that follow bug distributions reported in literature [37]. However, this prior work did not utilize real student data, relying only on the frequency of specific bugs in the generated code and not comparing the synthetic data to real student data.

In this follow-up work, we compare the LLM-generated synthetic data to real student data. Our evaluation focuses on the distribution of test case failures as a measure of similarity between synthetic and real data. This study encompasses two programming languages, C and Dart, and includes data from institutions across different countries. The primary research question guiding this study is:

> To what extent can generative AI models be used to generate synthetic incorrect code submissions for introductory programming exercises?

We make two main contributions in this work. First, we provide an analysis of the capability of LLMs to generate synthetic data for computing education research. Second, we explore the effectiveness of prompting strategies in generating incorrect code submissions that mirror typical student errors. This study contributes to the development of reliable synthetic datasets, which can facilitate research and teaching in computing education while preserving student privacy.

2 Related Work

2.1 Bugs

Programming errors, or bugs, constitute a well studied topic in the computing education research literature. Many studies have identified common mistakes made by novice programmers, including, syntax errors, logic errors [17], or both [1]. Teaching students to debug has also gained a great deal of researchers' attention [32, 41]. Griffin's study [23] exemplifies that the use of intentionally erroneous code in instruction is not, however, limited to teaching debugging but is suitable for programming education more generally.

However, the research literature has, so far, only narrowly covered how to simulate programming errors made by students. Until the emergence of LLMs, perhaps the most promising approach was automatic program mutation, provided by mutation analysis tools that software engineers use in software testing. Clegg et al. [10] found that mutant code has similar faults as code written by students and is, thus, a good aid when designing automated grading systems. Perretta et al. [45], on the other hand, used code mutation for evaluating test suites written by students. They also conclude that mutant code can simulate student code to a reasonable extent.

2.2 LLMs in Computing Education

With the introduction of generative AI, educators can now produce high-quality, personalized learning materials at scale for their students. These models can produce diverse explanations [39] that students find engaging [38] and that are often rated higher [29] or equal [3] in quality compared to explanations generated by peers. While LLM-generated code explanations have been found to be similar semantically and lexically to those created by experts, their readability is lower compared to explanations created by experts or students [31]. Students and instructors can also use the models to generate personally relevant analogies [6, 7], programming assignments [49], and to create multiple choice questions [15, 52]. As such, generative AI has become a legitimate source of help for many computing students [25, 47] with over 26% of students using it on a daily basis [25].

To further support students, researchers have also developed interactive systems to scaffold student's use of generative AI in classroom settings. This scaffolding is essential to prevent misuse [4, 28, 55] and to address challenges that some students face in effectively prompting and interpreting responses from generative AI [25, 48]. Such systems include, for example, CodeAid [27], CodeHelp [34], and Promptly [12]. Despite the emergence of tools that scaffold the use of generative AI, less research has been dedicated to investigating whether generative AI can be used to simulate student behaviors or generate synthetic student data. Markel et al. simulated student questions to train teaching assistants [40], highlighting a potential area for further investigation.

2.3 Generating Synthetic Data

Synthetic data is highly useful for multiple data science related purposes, including releasing privacy-preserving data in sensitive domains, construction of datasets without unwanted biases present in real world data, and augmenting scarce real world data [26]. Synthetic data generators have been studied in various fields such as finance [2], medicine [22], and computer science [8, 42]. Likewise, the usefulness of synthetic data generation has been noted in the fields of education and learning analytics [5], e.g., to evaluate knowledge tracing models [46, 50], to train performance prediction models [14, 21], and to simulate student behaviour for further research [43, 56]. However, the focus in generating datasets has been on numerical or categorical data, and not textual data such as student submissions for open text or programming exercises.

Enter LLMs which excel in generating text resembling that of humans and can be easily prompted to produce specific kinds of texts. On their own, they already are highly versatile and capable synthetic data generators for textual data given the correct prompts [33, 53]. As a prime example, in a recent study by Møller et al. [42], augmenting training data of classification models with synthetic data (using GPT-4 and LLama2) was found to outperform augmenting it with crowdsourced data on some NLP classification tasks, particularly on multi-class tasks or tasks with rare classes, and to be beneficial on others although not as much as crowdsourcing. They note that using LLMs directly for various tasks is mostly inferior to using an LLM that is fine-tuned (i.e., trained further) using synthesized data, a result echoed by Tang et al. [51] who investigated the capabilities of LLMs for healthcare related tasks.

While it is straightforward to generate synthetic data with LLMs, generating data of high quality and variability can require specific prompting strategies or knowledge enhancement [36, 44]. Evaluating the quality of generated open-ended textual data directly as opposed to, e.g., evaluating it through classification models, can be laborious, requiring manual evaluations or auxiliary models [9, 36].

The main limitation of synthetic data is that it might not resemble real data – the closer synthetic data is to real data, the more valuable it is. The goal of our study is to evaluate to what extent LLM-generated synthetic incorrect code submissions resemble real incorrect code submissions by students, shedding light on whether they could be used to generate synthetic student submissions.

Table 1: Summary of the C programming project functions from 2016 and 2017. Every function had 20 tests.

Year	Prototype Declaration	Description
2016	`int PrimeBelow(int upper);`	Returns the largest prime number less than the given upper limit.
2016	`void Strikeout(char *hide, char *phrase);`	Modifies a phrase by striking out occurrences of the word specified in hide.
2016	`int KthLargest(int k, int *values, int numValues);`	Returns the k-th largest element from an array of integers.
2016	`Rectangle BoundingRectangle(Rectangle r1, Rectangle r2);`	Computes the smallest rectangle that can enclose two given rectangles.
2016	`int TallestVine(int seedA, int seedB, int days);`	Simulates the growth of vines over a specified number of days based on seed values.
2017	`double AverageSheep(int *counts);`	Calculates the average count of sheep over a period (similar to a "rainfall" computation).
2017	`int PrimeFactors(int n, int *factors);`	Determines the prime factors of a given integer, n.
2017	`void ConnectTwo(int maze[10][10]);`	Modifies a 10x10 2D array to show the shortest connection between two specified cells.
2017	`void DayTrader(int *prices, int len, int *run, int *runIndex);`	Identifies the best run of consecutive days for maximizing return on stock prices.
2017	`void AddOne(char *input, char *output);`	Increments an arbitrarily large numeric string storing the result in an output string.

3 Methods

Our data on student test case failures is taken from two distinct contexts, at institutions in different countries teaching different programming languages. This diverse data allows us to better understand how well our proposed synthetic data generation approach might generalize to new contexts.

3.1 Context and Data

3.1.1 C Context. Our first set of data is obtained from a six-week C programming module which is part of an introductory programming course taught at the University of Auckland, a research university located in New Zealand. The content of this six-week module focuses on fundamental topics including basic syntax, data types, operators, standard I/O, control structures, functions, arrays, strings and file I/O. Students take part in weekly lab sessions, where they complete short programming exercises and receive immediate feedback from an auto-grader. The module concludes with a programming project, contributing 12% towards their final grade for the course, for which students do not receive feedback until after the submission deadline. Our data for this research includes student submissions to this final project taken from two consecutive deliveries of the course in 2016 and 2017. The data used in this study comprises a total of 8598 submissions of which 2405 are incorrect (i.e., do not pass all the tests) from 1751 students.

The programming project includes the requirement to implement five distinct functions of varying difficulty. Students are encouraged to thoroughly test their code prior to submitting it for grading as code that does not compile is not graded, and credit is only awarded for functions which pass all 20 of the tests in the corresponding test suite. Table 1 lists the prototype declarations, along with brief descriptors (students and the LLM were provided more detailed specifications), for the ten functions (five from each year) for which we analyze student submissions and test case failures.

3.1.2 Dart Context. The second dataset comes from an online course platform that hosts a variety of courses offered by Aalto University, a research university located in Finland. For this study, ten exercises from two different courses were chosen. Both courses use the Dart programming language. The first course is an open online introductory programming course where participants are typically novices without any prior programming experience. The course teaches students basic programming concepts such as variables, printing output and reading input, conditional statements, iteration, lists, and functions. The course is worth 2 ECTS credits which corresponds to about 50-60 hours of workload.

The second course is an advanced programming course where students are expected to know basic programming in some programming language. The goal of the second course is to teach students about developing software that supports a wide variety of devices using Dart and Flutter. The course is worth 5 ECTS credits which corresponds to about 135 hours of workload. As the participants are not expected to know Dart or Flutter, the course has a short introduction to key parts of the Dart programming language.

The introductory course is taught in Finnish while the advanced course is taught in English. For this article, we have translated the problem names and descriptions into English; however, the original language was used when prompting the LLM.

Both courses can be completed fully online and contain multiple small programming exercises embedded within the online materials. The data used for this study comes from a sample of ten of these small exercises. Table 2 shows brief descriptions of the exercises used in this study. Students and the LLM were provided the actual, more comprehensive problem descriptions. The data comprises a total of 44742 submissions of which 24719 are incorrect (i.e., do not pass all the tests) from 5322 students (5063 from the introductory course and 259 from the advanced course).

3.2 Prompting the LLM

For this study, we chose to use the GPT-4o large language model[1], which at the time of writing, was the state-of-the-art model according to online leaderboards[2]. As the state-of-the-art model at the time of writing, we believe that it is the best choice to evaluate, as its performance should be closest to future models with increased capabilities.

We evaluate three different prompts to explore how prompt engineering affects the generated incorrect solutions. These prompts are based on our prior work [37]. The prompts used in this study are the following (for the exact phrasing, see Figure 1).

[1]More specifically, the GPT-4o-2024-05-13 version.
[2]According to the LMSYS Chatbot Arena Leaderboard, accessed July 20th, 2024: https://chat.lmsys.org/?leaderboard

Table 2: Summary of Dart Programming Exercises.

Course	Exercise	Description	# of tests
Introductory	Grade as text	Ask the user for a numerical grade and print the corresponding textual description of the grade.	6
Introductory	Sum of three numbers	Ask the user for three numbers and then print the sum of the numbers.	2
Introductory	Ask for password	Write a function that takes a correct password as a parameter and then asks the user for the password until they input the correct password.	3
Introductory	Sum of positive numbers	Write a function that takes in a list and returns the sum of the positive numbers in the list.	2
Introductory	Authentication	Ask the user for specified username and password, and print different messages to the user depending on whether they input the correct username ("admin") and/or password ("radish").	3
Advanced	Average of positives	Return the average value of positive numbers in a given list (similar to the Rainfall problem).	4
Advanced	Budget check	Given two doubles, budget and spending, print whether the budget is okay or not.	3
Advanced	Mystery function	Write a function that returns a string depending on which number is passed to the function and whether that number is divisible by 5 or 7 or both (similar to the FizzBuzz problem).	5
Advanced	Sum with formula	Write a function that takes in two numbers and returns the written sum formula of those two numbers (e.g., for input 1 and 2, returns "1+2=3").	2
Advanced	Video and playlist	Implement two classes, Video and Playlist. A video has a name (String) and duration in seconds (int) and a toString method. A playlist contains a list of videos and has methods for adding videos, checking if a video is on the playlist, and for returning the total duration of the playlist.	3

Problem description:
<Problem description>

Test cases:
<List of test cases>

Test case failure frequencies:
<List of failure frequencies for each test case>

Your task:
Please generate five incorrect solutions to this programming problem that include one or more semantic bugs. Place the delimiter CODE_START before every solution example you'll generate and CODE_END at the end of the solution code to help me extract just the generated code. Importantly, it should be possible to compile the incorrect solutions and it should be possible to run unit tests for the code. When generating the solutions, please try to follow the distribution of failing tests given above under "Test case failure frequencies". Use the **<Dart/C>** programming language.

Figure 1: The prompts used in the study. The baseline prompt did not include the blue and yellow highlighted parts. The test-case-informed prompt included the blue highlighted part on top of the baseline prompt. The frequency-informed prompt included both the blue and yellow highlighted parts on top of the baseline prompt. The bolded parts indicate variables for which content depended on the exercise.

- A *baseline* prompt that just asks the model to generate semantically incorrect solutions to the problem.
- A *test-case-informed* prompt where the model is also provided the test cases for the exercise.
- A *frequency-informed* prompt where the model is provided both the test cases and the frequencies of how often incorrect submissions fail each specific test case. In addition, the model is instructed to try follow the distribution of the failure frequencies.

In our study, we only focus on semantic bugs. There are a few reasons for this. Firstly, we believe that semantic bugs are more interesting for potential debugging exercises. Secondly, if the code is not syntactically correct, then it would not be possible to run the test suite against the generated code, making it infeasible to evaluate the distribution of test case failures for the generated synthetic data. This is also why for all prompts, we explicitly tell the model that the generated solutions should compile and that it should be possible to run unit tests for the generated code.

When generating the submissions, for each combination of prompt, exercise, and context, we generate five batches of five incorrect solutions (i.e., 25 total for each unique combination of prompt, exercise and context). This results in a total of 3 prompts × 5 batches × 5 solutions × 10 exercises × 2 contexts = 1500 generated solutions.

3.3 Analysis

We only include incorrect submissions in our analysis. This is done as our focus in this work is to generate incorrect synthetic solutions – prior work has established that LLMs can generate correct solutions [13, 19, 47]. For all the generated synthetic submissions, as well as for the real data used as a comparison point, we ran the unit test suites that had been used for those exercises when they were part of the course. For each individual unit test, we then calculated the percentage of cases where the solutions pass and fail the unit test, i.e., a unit test "pass rate". This gives us one percentage, or pass rate, for each unit test. As there are hundreds of unit tests altogether for the 20 exercises analyzed in this work, we calculate the minimum, maximum, mean, and standard deviation of these unit test pass rates for each exercise (which each comprise multiple unit tests). This way, we can compare if the unit test pass rates between the real and the synthetic data are different with regard

Table 3: Results of the analysis. The "Real" column shows statistics for the real student data (only incorrect solutions). The other three columns show the statistics for each of the three prompts we used. For each exercise, there were multiple unit tests. The range, mean, and standard deviation of the unit test pass rates are shown for each condition (real, baseline, test-case informed, frequency-informed). In addition, the average differences (deltas) between the real data and the synthetic data are shown for the means and standard deviations.

		Real			Baseline			Test-case-informed			Frequency-informed		
Language	Exercise	Range	μ	σ	Range	μ	σ	Range	μ	σ	Range	μ	σ
C	Prime Below	[50, 92]	81.1	12.6	[58, 100]	80.5	9.9	[50, 100]	68.8	14.7	[61, 100]	83.7	9.9
C	Strikeout	[29, 75]	57.5	13.1	[0, 96]	10.2	29.4	[0, 83]	9.5	25.2	[0, 96]	10.0	29.4
C	Kth Largest	[43, 73]	61.2	8.7	[40, 96]	66.2	19.0	[48, 83]	62.3	12.5	[56, 96]	70.0	14.8
C	Bounding Rectangle	[22, 74]	52.7	12.2	[0, 28]	8.4	7.6	[4, 32]	12.6	7.5	[12, 44]	22.4	9.7
C	Tallest Vine	[31, 49]	38.3	5.7	[4, 62]	40.2	24.0	[0, 64]	40.4	26.1	[4, 76]	50.2	29.7
C	Average Sheep	[15, 84]	70.8	17.4	[32, 77]	49.7	15.7	[35, 83]	53.5	18.0	[42, 88]	61.4	18.4
C	Prime Factors	[24, 79]	61.7	18.3	[61, 74]	69.4	4.7	[39, 87]	61.2	12.6	[52, 81]	66.7	9.4
C	Connect Two	[23, 59]	35.7	10.2	[20, 30]	23.5	3.3	[23, 41]	27.3	5.7	[6, 24]	8.7	5.0
C	Day Trader	[30, 74]	54.9	17.9	[8, 28]	11.8	5.3	[0, 12]	2.2	4.0	[16, 28]	19.8	4.2
C	Add One	[23, 49]	40.0	8.9	[24, 76]	50.0	19.9	[76, 80]	77.6	2.0	[44, 96]	73.0	22.4
Average deltas to real data for mean and standard deviation.						19.3	9.8		22.0	7.5		21.1	9.4
Dart	Grade as text	[20, 48]	38.7	9.2	[44, 72]	62.0	9.2	[44, 76]	58.0	11.3	[44, 72]	64.7	9.9
Dart	Average of positives	[31, 46]	38.0	6.6	[24, 64]	39.0	15.1	[24, 48]	33.0	9.9	[8, 64]	21.0	16.3
Dart	Budget check	[24, 38]	30.0	5.9	[12, 40]	21.3	13.2	[12, 44]	30.7	13.6	[32, 64]	49.3	13.2
Dart	Sum of three numbers	[1, 2]	1.5	0.5	[0, 0]	0.0	0.0	[0, 0]	0.0	0.0	[4, 4]	4.0	0.0
Dart	Ask for password	[2, 31]	17.3	11.9	[0, 0]	0.0	0.0	[0, 0]	0.0	0.0	[0, 0]	0.0	0.0
Dart	Mystery function	[0, 82]	64.2	32.1	[4, 80]	48.0	27.8	[8, 68]	44.0	20.2	[8, 52]	41.6	16.9
Dart	Sum of positive numbers	[1, 6]	3.5	2.5	[24, 24]	24.0	0.0	[28, 32]	30.0	2.0	[20, 28]	24.0	4.0
Dart	Sum with formula	[0, 0]	0.0	0.0	[0, 4]	2.0	2.0	[0, 0]	0.0	0.0	[0, 0]	0.0	0.0
Dart	Authentication	[24, 58]	38.0	14.5	[24, 44]	33.3	8.2	[28, 44]	34.7	6.8	[32, 72]	49.3	16.8
Dart	Video and playlist	[33, 71]	50.7	15.6	[16, 40]	24.0	11.3	[24, 44]	34.7	8.2	[40, 52]	45.3	5.0
Average deltas to real data for mean and standard deviation.						12.2	4.8		11.0	5.3		14.2	6.0

to the pass rate range (i.e., minimum and maximum), mean, and standard deviation.

Even though we asked the model to produce code that unit tests could be run against, the model would sometimes produce code that crashed. For C, there were 48 (out of 750) generated programs that crashed when trying to run the test suite (typically due to a segmentation fault). These were ignored in calculating the statistics.

To analyze the difference between the generated synthetic data and real data statistically, we conduct a Kruskal-Wallis H test between the test pass rates between all four distributions (real, baseline, test-case-informed, and frequency-informed) for both programming languages separately. In case either of the Kruskal-Wallis H test results suggests that the distributions are statistically significantly different using an alpha threshold of 0.05, we conduct pairwise Mann-Whitney U tests between the real data and each synthetic dataset separately to analyze which of the synthetic datasets are significantly different from the real data. As we do multiple statistical comparisons, we employ the Bonferroni correction to avoid finding spurious statistically significant differences.

4 Results and Discussion

The results of the analysis are shown in Table 3. Many interesting observations can be made based on the table. First, there seem to be differences between exercises in how well the model can generate incorrect solutions to the exercise. Some exercises seem hard for the model to solve "partially incorrectly", i.e., to generate a bug that allows some tests to pass. This is the case, for example, for the "Ask for password" Dart exercise and the "Day Trader" C exercise. For the former, the mean pass rate in the real data is 17.3%, but all the LLM-generated incorrect solutions always fail all the tests. For the latter, the mean pass rate in the real data is 54.9%, which is considerably higher than the mean pass rate for all three prompts: 11.8%, 2.2%, and 19.8% for the baseline, test-case informed, and frequency-informed prompts respectively. This suggests that for these two exercises, the bugs generated by the model tend to cause most of the tests to fail, while in the real data, student bugs are more subtle and only cause part of the tests to fail. This finding is similar to synthesized code that is aimed to be correct where it has also been found that LLM performance is problem dependent [35].

When considering the results, the number of test cases should be taken into account for the Dart data (all the C exercises had exactly 20 test cases each). For example, the "Sum of three numbers" and the "Sum with formula" exercises both only had two test cases. For both of these exercises, the tests mainly check that the student has not hard coded the response, and thus most bugs (other than hard coding) will cause both tests to fail concurrently. For these two exercises, the very low ranges and means of unit test pass rates suggest that real buggy submissions almost exclusively fail both tests, i.e., it is very rare that one test would pass and the other not.

Somewhat surprisingly, there do not seem to be large differences between the different prompts in how well they work for generating synthetic incorrect submissions. This is most visible by looking at the average deltas between the real data and the synthetic data. This is confirmed for the Dart data by a Kruskal-Wallis H test ($H = 0.87$, $p = 0.83$), which suggests that all four distributions are statistically equivalent. However, for the C data, the results of the Kruskal-Wallis H test ($H = 28.4$, $p < 0.0001$) suggest that at least one distribution is significantly different from the others. Pairwise Mann-Whitney U tests between the real data and each synthetic dataset separately reveal that both the baseline ($U = 25739.0$, $p < 0.0001$) and the test-case-informed ($U = 25036.5$, $p < 0.0001$) synthetic datasets are significantly different from the real data. However, the difference is not significant for the frequency-informed synthetic dataset with our threshold for significance alpha = 0.05 ($U = 22816.0$, $p = 0.07$). For our study, these results imply that all three prompts led to "good" synthetic data for Dart (as it was not significantly different from real data), while only the frequency-informed prompt led to "good" synthetic data for C.

As providing test case information and failure distribution to GPT-4o does not always appear to help it generate submissions with more similar distributions to real data, more sophisticated prompt engineering approaches could be a useful area to explore, at least for the current generation of state-of-the-art models. Previous work has found differences between student and LLM-generated code, for example, in what constructs and keywords are used [11, 24]. In general, generating synthetic content using LLMs risks monotonicity, especially if content is generated without controls aimed at increasing the diversity of the generated content [36].

The finding that the synthetic Dart data seems to be more similar to the real student data with regard to test case failure distributions is corroborated by looking at the average deltas. For the Dart data, the average deltas between the mean pass rates for the real data and the synthetic data are considerably lower (12.2% for baseline, 11.0% for test-case-informed, and 14.2% for frequency-informed) than for the C data (19.3% for baseline, 22.0% for test-case-informed, and 21.1% for frequency-informed). This finding is surprising as the model has likely been trained with more C code than Dart code since C is a vastly more common programming language compared to Dart. This suggests that it might be harder for the model to generate semantic bugs that are similar to bugs found in student programs for C than for Dart. On the other hand, the Dart exercises are less complex than the C exercises, which could also contribute to the observation.

Two of the C exercises had diagrams in their problem descriptions that were not shown to the model during prompting. For the "Tallest Vine" exercise, this does not seem to have been a problem for the model as the mean pass rates for the synthetic data are higher than for the real data (40.2% for baseline, 40.4% for test-case-informed, and 50.2% for frequency-informed versus 38.3% for real data), suggesting the model was able to generate solutions that pass some of the tests. However, for the "Bounding Rectangle" exercise, this might explain why the mean pass rates for the synthetic data are considerably lower than for the real data (8.4% for baseline, 12.6% for test-case-informed, and 22.4% for frequency-informed versus 52.7% for real data).

5 Limitations

There are some limitations to this work. We only ask the model to generate incorrect solutions. In both contexts, a large portion of submissions pass all the tests (45% of submissions for Dart and 72% of submissions for C). Our analysis does not look at whether the model can generate realistic correct solutions, which is left for future research. Prior work suggests that LLMs can solve most introductory programming exercises correctly [47], although LLM-generated solutions have distinct patterns that make it possible to distinguish them from student-generated solutions [24]. Thus, future work should study whether LLMs can be used to generate realistic synthetic correct solutions.

We only used a single LLM and two datasets in our study. Thus, it is not certain our results would generalize to other LLMs or other contexts, e.g., ones using different programming languages or ones that have different student populations.

We only evaluate the similarity of the synthetic data to the real data with regard to test case failure distributions. For example, we do not look at constructs used in the code, what strategies are employed in the program to solve the problem, or the actual bugs in the code. However, our prior work showed that with proper prompting, the bug distributions in generated synthetic submissions are similar to bug distributions reported in the literature [37].

Some of the test suites of the Dart exercises were not very comprehensive, only including a couple of tests. This means that some bugs that the LLMs generate might not be captured by the test suite. For the C exercises, two of them had diagrams in the problem descriptions that were not shown to the LLM. Thus, the LLM was not provided the same information as the students, which could have made it more difficult for the LLM to generate the incorrect solutions, potentially affecting the results.

6 Conclusions

We investigated the capability of generative AI models in generating synthetic incorrect code submissions to programming exercises. This could be useful for creating debugging exercises for students and for generating synthetic datasets for research purposes. Our findings are a first step towards investigating whether LLMs can be used to generate synthetic incorrect submissions. Our results are promising in this regard, suggesting that LLM-generated synthetic submissions are sometimes not significantly different from real student data with regard to test case failure distributions. This shows promise that LLMs could potentially be used for generating satisfiably diverse synthetic code submission data, lowering barriers to conducting research with such data, and making it easier to provide students with debugging practice.

However, more research is necessary to explore the closeness of LLM-generated synthetic code submissions to that of real student data in more detail, such as what in the code makes the test cases fail and can the patterns in synthetic code submissions be made to more closely resemble that of real student code submissions.

Acknowledgments

This research was supported by the Research Council of Finland (Academy Research Fellow grant number 356114).

References

[1] Amjad Altadmri and Neil CC Brown. 2015. 37 million compilations: Investigating novice programming mistakes in large-scale student data. In *Proc. of the 46th ACM Technical Symp. on Computer Science Education.* 522–527.

[2] Samuel A Assefa, Danial Dervovic, Mahmoud Mahfouz, Robert E Tillman, Prashant Reddy, and Manuela Veloso. 2020. Generating synthetic data in finance: opportunities, challenges and pitfalls. In *Proceedings of the First ACM International Conference on AI in Finance.* 1–8.

[3] Rishabh Balse, Viraj Kumar, Prajish Prasad, and Jayakrishnan Madathil Warriem. 2023. Evaluating the Quality of LLM-Generated Explanations for Logical Errors in CS1 Student Programs. In *Proceedings of the 16th Annual ACM India Compute Conference* (Hyderabad, India) *(COMPUTE '23)*. Association for Computing Machinery, New York, NY, USA, 49–54. https://doi.org/10.1145/3627217.3627233

[4] Brett A Becker, Paul Denny, James Finnie-Ansley, Andrew Luxton-Reilly, James Prather, and Eddie Antonio Santos. 2023. Programming Is Hard – Or at Least It Used to Be: Educational Opportunities And Challenges of AI Code Generation. In *Proc. of the 54th ACM Technical Symp. on Computer Science Education V. 1.*

[5] Alan Mark Berg, Stefan T Mol, Gábor Kismihók, and Niall Sclater. 2016. The role of a reference synthetic data generator within the field of learning analytics. *Journal of Learning Analytics* 3, 1 (2016), 107–128.

[6] Seth Bernstein, Paul Denny, Juho Leinonen, Lauren Kan, Arto Hellas, Matt Littlefield, Sami Sarsa, and Stephen MacNeil. 2024. "Like a Nesting Doll": Analyzing Recursion Analogies Generated by CS Students Using Large Language Models. In *Proc. of the 2024 on Innovation and Technology in CS Education V. 1.* 122–128.

[7] Seth Bernstein, Paul Denny, Juho Leinonen, Matt Littlefield, Arto Hellas, and Stephen MacNeil. 2024. Analyzing Students' Preferences for LLM-Generated Analogies. In *Proc. of the 2024 on Innovation and Technology in Computer Science Education V. 2.*

[8] Alessio Botta, Alberto Dainotti, and Antonio Pescapé. 2012. A tool for the generation of realistic network workload for emerging networking scenarios. *Computer Networks* 56, 15 (2012), 3531–3547.

[9] John Chung, Ece Kamar, and Saleema Amershi. 2023. Increasing Diversity While Maintaining Accuracy: Text Data Generation with Large Language Models and Human Interventions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers).* 575–593.

[10] Benjamin Simon Clegg, Phil McMinn, and Gordon Fraser. 2021. An Empirical Study to Determine if Mutants Can Effectively Simulate Students' Programming Mistakes to Increase Tutors' Confidence in Autograding. In *Proc. of the 52nd ACM Technical Symp. on Computer Science Education.* 1055–1061.

[11] Paul Denny, Hassan Khosravi, Arto Hellas, Juho Leinonen, and Sami Sarsa. 2023. Can we trust AI-generated educational content? comparative analysis of human and AI-generated learning resources. *arXiv preprint arXiv:2306.10509* (2023).

[12] Paul Denny, Juho Leinonen, James Prather, Andrew Luxton-Reilly, Thezyrie Amarouche, Brett A Becker, and Brent N Reeves. 2024. Prompt Problems: A new programming exercise for the generative AI era. In *Proc. of the 55th ACM Technical Symp. on Computer Science Education V. 1.*

[13] Paul Denny, James Prather, Brett A. Becker, James Finnie-Ansley, Arto Hellas, Juho Leinonen, Andrew Luxton-Reilly, Brent N. Reeves, Eddie Antonio Santos, and Sami Sarsa. 2024. Computing Education in the Era of Generative AI. *Commun. ACM* 67, 2 (2024), 56–67.

[14] Mohsen Dorodchi, Erfan Al-Hossami, Aileen Benedict, and Elise Demeter. 2019. Using synthetic data generators to promote open science in higher education learning analytics. In *2019 IEEE Int. Conf. on Big Data.* IEEE, 4672–4675.

[15] Jacob Doughty, Zipiao Wan, Anishka Bompelli, Jubahed Qayum, Taozhi Wang, Juran Zhang, Yujia Zheng, Aidan Doyle, Pragnya Sridhar, et al. 2024. A comparative study of AI-generated (GPT-4) and human-crafted MCQs in programming education. In *Proc. of the 26th Australasian Computing Education Conf.* 114–123.

[16] John Edwards, Kaden Hart, Raj Shrestha, et al. 2023. Review of CSEDM Data and Introduction of Two Public CS1 Keystroke Datasets. *J. of Educational Data Mining* 15, 1 (2023), 1–31.

[17] Andrew Ettles, Andrew Luxton-Reilly, and Paul Denny. 2018. Common Logic Errors Made by Novice Programmers. In *Proc. of the 20th Australasian Computing Education Conf.* ACM, New York, NY, USA, 83–89.

[18] 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 Conf.* ACM, New York, NY, USA, 10–19.

[19] 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 Conf.* 10–19.

[20] James Finnie-Ansley, Paul Denny, Andrew Luxton-Reilly, Eddie Antonio Santos, James Prather, and Brett A. Becker. 2023. My AI Wants to Know if This Will Be on the Exam: Testing OpenAI's Codex on CS2 Programming Exercises. In *Proc. of the 25th Australasian Computing Education Conf.* ACM, 97–104.

[21] Brendan Flanagan, Rwitajit Majumdar, and Hiroaki Ogata. 2022. Fine Grain Synthetic Educational Data: Challenges and Limitations of Collaborative Learning Analytics. *IEEE Access* 10 (2022), 26230–26241.

[22] Akshay Goel, Almog Gueta, Omry Gilon, Chang Liu, Sofia Erell, Lan Huong Nguyen, Xiaohong Hao, Bolous Jaber, Shashir Reddy, Rupesh Kartha, et al. 2023. Llms accelerate annotation for medical information extraction. In *Machine Learning for Health (ML4H).* PMLR, 82–100.

[23] Jean M. Griffin. 2019. Designing Intentional Bugs for Learning. In *Proc. of the 2019 Conf. on United Kingdom & Ireland Computing Education Research.*

[24] Muntasir Hoq, Yang Shi, Juho Leinonen, Damilola Babalola, Collin Lynch, Thomas Price, and Bita Akram. 2024. Detecting ChatGPT-generated code submissions in a CS1 course using machine learning models. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1.* 526–532.

[25] Irene Hou, Sophia Mettille, Owen Man, Zhuo Li, Cynthia Zastudil, and Stephen MacNeil. 2024. The Effects of Generative AI on Introductory Students' Help-Seeking Preferences. In *Australasian Computing Education Conference.*

[26] James Jordon, Lukasz Szpruch, Florimond Houssiau, Mirko Bottarelli, Giovanni Cherubin, Carsten Maple, Samuel N Cohen, and Adrian Weller. 2022. Synthetic Data–what, why and how? *arXiv preprint arXiv:2205.03257* (2022).

[27] Majeed Kazemitabaar, Runlong Ye, Xiaoning Wang, Austin Zachary Henley, Paul Denny, Michelle Craig, and Tovi Grossman. 2024. Codeaid: Evaluating a classroom deployment of an llm-based programming assistant that balances student and educator needs. In *Proceedings of the CHI Conference on Human Factors in Computing Systems.* 1–20.

[28] Sam Lau and Philip Guo. 2023. From "Ban It Till We Understand It" to "Resistance is Futile": How University Programming Instructors Plan to Adapt as More Students Use AI Code Generation and Explanation Tools such as ChatGPT and GitHub Copilot. In *Proc. of the 2023 ACM Conf. on Int. Computing Education Research - Vol. 1.* ACM, 106–121.

[29] Juho Leinonen, Paul Denny, Stephen MacNeil, Sami Sarsa, Seth Bernstein, Joanne Kim, Andrew Tran, and Arto Hellas. 2023. Comparing Code Explanations Created by Students and Large Language Models. In *Proc. of the 2023 Conf. on Innovation and Technology in Computer Science Education V. 1.* 124–130.

[30] Juho Leinonen, Petri Ihantola, and Arto Hellas. 2017. Preventing keystroke based identification in open data sets. In *Proc. of the Fourth (2017) ACM Conference on Learning @ Scale.* 101–109.

[31] Arun-Balajiee Lekshmi-Narayanan, Priti Oli, Jeevan Chapagain, Mohammad Hassany, Rabin Banjade, Peter Brusilovsky, and Vasile Rus. 2024. Explaining Code Examples in Introductory Programming Courses: LLM vs Humans. arXiv:2403.05538 [cs.CY] https://arxiv.org/abs/2403.05538

[32] Chen Li, Emily Chan, Paul Denny, Andrew Luxton-Reilly, and Ewan Tempero. 2019. Towards a Framework for Teaching Debugging. In *Proc. of the Twenty-First Australasian Computing Education Conf.* 79–86.

[33] Zhuoyan Li, Hangxiao Zhu, Zhuoran Lu, and Ming Yin. 2023. Synthetic data generation with large language models for text classification: Potential and limitations. *arXiv preprint arXiv:2310.07849* (2023).

[34] Mark Liffiton, Brad E Sheese, Jaromir Savelka, and Paul Denny. 2023. Codehelp: Using large language models with guardrails for scalable support in programming classes. In *Proc. of the 23rd Koli Calling Int. Conf. on Computing Education Research.*

[35] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2024. Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation. *Advances in Neural Information Processing Systems* 36 (2024).

[36] Lin Long, Rui Wang, Ruixuan Xiao, Junbo Zhao, Xiao Ding, Gang Chen, and Haobo Wang. 2024. On LLMs-Driven Synthetic Data Generation, Curation, and Evaluation: A Survey. *arXiv preprint arXiv:2406.15126* (2024).

[37] Stephen MacNeil, Magdalena Rogalska, Juho Leinonen, Paul Denny, Arto Hellas, and Xandria Crosland. 2024. Synthetic Students: A Comparative Study of Bug Distribution Between Large Language Models and Computing Students. In *Proceedings of the 2024 on ACM Virtual Global Computing Education Conference V. 1* (Virtual Event, NC, USA) *(SIGCSE Virtual 2024)*. Association for Computing Machinery, New York, NY, USA, 137–143. https://doi.org/10.1145/3649165.3690100

[38] Stephen MacNeil, Andrew Tran, Arto Hellas, Joanne Kim, Sami Sarsa, Paul Denny, Seth Bernstein, and Juho Leinonen. 2023. Experiences from Using Code Explanations Generated by Large Language Models in a Web Software Development E-Book. In *Proc. of the ACM Technical Symp. on Computing Science Education.* ACM, 6 pages.

[39] Stephen MacNeil, Andrew Tran, Dan Mogil, Seth Bernstein, Erin Ross, and Ziheng Huang. 2022. Generating Diverse Code Explanations Using the GPT-3 Large Language Model. In *Proc. of the 2022 ACM Conf. on Int. Computing Education Research - Volume 2.* ACM, 37–39.

[40] Julia M Markel, Steven G Opferman, James A Landay, and Chris Piech. 2023. GPTeach: Interactive TA training with GPT-based students. In *Proc. of the Tenth ACM Conf. on Learning @ Scale.* 226–236.

[41] Renée McCauley, Sue Fitzgerald, Gary Lewandowski, Laurie Murphy, Beth Simon, Lynda Thomas, and Carol Zander. 2008. Debugging: a review of the literature from an educational perspective. *Computer Science Education* 18, 2 (2008), 67–92.

[42] Anders Giovanni Møller, Arianna Pera, Jacob Dalsgaard, and Luca Aiello. 2024. The Parrot Dilemma: Human-Labeled vs. LLM-augmented Data in Classification Tasks. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 2: Short Papers)*. 179–192.

[43] Yaneth Moreno, Anthony Montero, Francisco Hidrobo, and Saba Infante. 2023. Synthetic Data Generator for an E-Learning Platform in a Big Data Environment. In *Int. Conf. in Information Technology and Education*. Springer, 431–440.

[44] Jeiyoon Park, Chanjun Park, and Heuiseok Lim. 2024. ChatLang-8: An LLM-Based Synthetic Data Generation Framework for Grammatical Error Correction. *arXiv preprint arXiv:2406.03202* (2024).

[45] James Perretta, Andrew DeOrio, Arjun Guha, and Jonathan Bell. 2022. On the use of mutation analysis for evaluating student test suite quality. In *Proc. of the 31st ACM SIGSOFT Int. Symp. on Software Testing and Analysis*. 263–275.

[46] Chris Piech, Jonathan Bassen, Jonathan Huang, Surya Ganguli, Mehran Sahami, Leonidas J Guibas, and Jascha Sohl-Dickstein. 2015. Deep knowledge tracing. *Advances in neural information processing systems* 28 (2015).

[47] James Prather, Paul Denny, Juho Leinonen, Brett A. Becker, Ibrahim Albluwi, Michelle Craig, Hieke Keuning, Natalie Kiesler, Tobias Kohn, Andrew Luxton-Reilly, Stephen MacNeil, Andrew Petersen, Raymond Pettit, Brent N. Reeves, and Jaromir Savelka. 2023. The Robots Are Here: Navigating the Generative AI Revolution in Computing Education. In *Proc. of the 2023 Working Group Reports on Innovation and Technology in Computer Science Education*. ACM, 108–159.

[48] James Prather, Brent Reeves, Juho Leinonen, Stephen MacNeil, Arisoa S Randrianasolo, Brett Becker, Bailey Kimmel, Jared Wright, and Ben Briggs. 2024. The Widening Gap: The Benefits and Harms of Generative AI for Novice Programmers. *arXiv preprint arXiv:2405.17739* (2024).

[49] Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. 2022. Automatic Generation of Programming Exercises and Code Explanations Using Large Language Models. In *Proc. of the 2022 ACM Conf. on Int. Computing Education Research - Volume 1*. ACM, 27–43.

[50] Sami Sarsa, Juho Leinonen, Arto Hellas, et al. 2022. Empirical Evaluation of Deep Learning Models for Knowledge Tracing: Of Hyperparameters and Metrics on Performance and Replicability. *Journal of Educational Data Mining* 14, 2 (2022).

[51] Ruixiang Tang, Xiaotian Han, Xiaoqian Jiang, and Xia Hu. 2023. Does synthetic data generation of llms help clinical text mining? *arXiv preprint arXiv:2303.04360* (2023).

[52] Andrew Tran, Kenneth Angelikas, Egi Rama, Chiku Okechukwu, David H Smith, and Stephen MacNeil. 2023. Generating multiple choice questions for computing courses using large language models. In *2023 IEEE Frontiers in Education Conference (FIE)*. IEEE, 1–8.

[53] Stefan Sylvius Wagner, Maike Behrendt, Marc Ziegele, and Stefan Harmeling. 2024. The Power of LLM-Generated Synthetic Data for Stance Detection in Online Political Discussions. *arXiv preprint arXiv:2406.12480* (2024).

[54] Jacqueline Whalley, Amber Settle, and Andrew Luxton-Reilly. 2021. Novice Reflections on Debugging. In *Proc. of the 52nd ACM Technical Symp. on Computer Science Education*. ACM, New York, NY, USA, 73–79.

[55] Cynthia Zastudil, Magdalena Rogalska, Christine Kapp, Jennifer Vaughn, and Stephen MacNeil. 2023. Generative ai in computing education: Perspectives of students and instructors. In *IEEE Frontiers in Education Conference*. IEEE, 1–9.

[56] Chen Zhan, Oscar Blessed Deho, Xuwei Zhang, Srecko Joksimovic, and Maarten de Laat. 2023. Synthetic data generator for student data serving learning analytics: A comparative study. *Learning Letters* (2023).