4-2024

Experience Report: Identifying common misconceptions and errors of novice programmers with ChatGPT

Hua Leong FWA
Singapore Management University, hlfwa@smu.edu.sg

Experience Report: Identifying common misconceptions and errors of novice programmers with ChatGPT

Author1
anon1@university.edu
Author1Institution
Dublin, Ohio, USA

ABSTRACT

Identifying the misconceptions of novice programmers is pertinent for informing instructors of the challenges faced by their students in learning computer programming. In the current literature, custom tools, test scripts were developed and, in most cases, manual effort to go through the individual codes were required to identify and categorize the errors latent within the students' code submissions. This entails investment of substantial effort and time from the instructors. In this study, we thus propose the use of ChatGPT in identifying and categorizing the errors. Using prompts that were seeded only with the student's code and the model code solution for questions from two lab tests, we were able to leverage on ChatGPT's natural language processing and knowledge representation capabilities to automatically collate frequencies of occurrence of the errors by error types. We then clustered the generated error descriptions for further insights into the misconceptions of the students. The results showed that although ChatGPT was not able to identify the errors perfectly, the achieved accuracy of 93.3% is sufficiently high for instructors to have an aggregated picture of the common errors of their students. To conclude, we have proposed a method for instructors to automatically collate the errors latent within the students' code submissions using ChatGPT. Notably, with the novel use of generated error descriptions, the instructors were able to have a more granular view of the misconceptions of their students, without the onerous effort of manually going through the students' codes.

CCS CONCEPTS

• **Applied computing** → **Interactive learning environments**; • **Computing methodologies** → *Information extraction.*

KEYWORDS

datasets, neural networks, gaze detection, text tagging

ACM Reference Format:
Author1. 2018. Experience Report: Identifying common misconceptions and errors of novice programmers with ChatGPT. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX).* ACM, New York, NY, USA, 9 pages. https://doi.org/XXXXXXX.XXXXXXX

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Novice programmers often exhibit misconceptions which inhibit their ability to learn and make progress [13, 19]. Understanding the misconceptions of novice programmers is pertinent for informing instructors of the challenges faced by the novices or their students in learning computer programming. With knowledge of the common misconceptions or errors made by their students, instructors can then enact appropriate effective instructional strategies to resolve these misconceptions. There were a number of studies [1, 7, 12, 22] which investigated into difficulties [8], errors [21], mistakes [3] and misconceptions [19, 20] of novice programmers given its pedagogical importance in computer science education.

In these studies, researchers have focused on detecting different types of errors -syntax [7], semantic [11] and logical errors [9]. Syntax errors can be detected and flagged by modern compilers as compilation errors while both semantic and logical errors are more difficult to detect as they are non-syntactical and relates more to bugs which causes the program to exhibit unintended, illogical or undesirable behaviour. More recently, the advent of a popular Large Language Model (LLM) – ChatGPT has seen unprecedented research interest for its use in many application areas. Within the area of computer programming, there were studies researching into LLMs such as ChatGPT for their use in automatic program repair [10, 16], automatic code summarization [2, 23], automatic code documentation generation [14] and automatic program generation [6, 18].

The positive results from these studies suggest that LLMs such as ChatGPT, with its natural language processing and knowledge representation capabilities, are capable of both comprehending and generating programming codes. Thus, for our study, motivated by the ability of ChatGPT to analyse and understand programming codes, we investigate into the use of ChatGPT to identify and collate the programming errors and misconceptions of students from their code submissions.

We formulate the two research questions below:

- RQ1: Can we use ChatGPT to automatically collate summary of errors made by students from their code submissions?
- RQ2: To what extent is the error summary from ChatGPT useful in helping instructors identify the common misconceptions or errors of students?

2 RELATED STUDY

Denny et al. [7] analyzed the syntax errors encountered by their students in the writing of Java codes. In their study, the authors collected the data using CodeWrite – a web-based drill and practice tool developed by the authors. CodeWrite contains a repository of pre-loaded exercises with the accompanying test cases. The syntax errors encountered by students were collected automatically when the students' codes were compiled on the server. The authors concluded from the results that students encounter a small number of syntax errors more frequently than others.

In most studies within the current literature, researchers investigating into the errors or misconceptions made by novice programmers had to develop additional tools to collect the list of errors. For example, in the study by Ettles et al. [9], the authors had to develop a set of functional test scripts. The test cases within the scripts were ordered and students' programs were tested against the test scripts. The test results were then compiled into a test vector for each student. The assumption is that students who made the same logical errors would have the same test vector. The authors acknowledged that this assumption though might not apply for all cases. Moreover, to identify the logical error from the test vector, the authors still had to manually go through all the students' codes to assign a broad classification to the logical errors.

In the study by Altadmri et al. [3], a year's worth of compilation events from over 250,000 students worldwide constituting the Blackbox data set [5] was used for the analysis. The Blackbox data set was a combination of editing, compilation and execution events collected within the custom developed BlueJ Integrated Development Environment (IDE). The BlueJ IDE was designed for those learning object-oriented programming in Java. The authors then used a fixed list of 18 error types for analysing students' mistakes extracted from the Blackbox data set. For four of the error types, they were able to use the compiler error message from Blackbox's compilations to classify the errors. For the rest of the error types, they had to apply a customized parser to parse the source codes and identify the errors.

Davin and Michael [17] investigated logical errors as opposed to errors identified through diagnostic messages produced by the compiler. They argued that although the use of diagnostic messages allowed errors to be automatically collected and identified, the logical errors would be more useful for identifying the misconceptions of the students. They, however, alluded to the fact that a major hurdle with the identification of logical errors when compared to diagnostic errors is that the former cannot be automated and involved substantial manual effort to identify and classify. Similar to the study by Altadmri et al. [3], they collected the data for their study using the BlueJ IDE. In their study, the authors collected compilation events which had associated compiler diagnostic message. This suggested that logical error e.g., those relating to mismatch with the functional requirements but did not cause compilation errors were not captured. The authors' focus in their study was more on judging whether manual classification provided enhanced information over automatic classification using compiler diagnostic messages.

In the above studies, custom developed tools e.g., IDEs and specially crafted test scripts were required to identify and categorize the errors. In many of these cases, the custom tools and test scripts were only developed for a particular programming language. Translating it for use in another programming language would incur substantial development time and resources. In contrast, in our study, we did not have to use custom developed tools or test scripts. Only the students' codes together with the model solution were passed into ChatGPT for identification and categorization of the errors.

Some of the previous studies also highlighted the usefulness of logical errors for identifying the novice programmers' misconceptions. To illustrate, some students may face difficulties in iterating through the elements of an array using a for loop. A possible logical error could be that the student did not increment the index variable of the array for each iteration of the loop which might indicate that the student's misconception was not with the use of a for loop but with the concept of array indexing. The code is syntactically correct and so there would be no compilation error which would thus preclude automatic identification of such an error using compilation messages. This adds to the significance of our study in investigating whether such logical errors can be detected using LLMs such as ChatGPT.

Although we only tested that the error identification and categorization worked for PHP codes, ChatGPT has been tested to be reasonably capable of handling programming challenges in Python [24], C and Java [15] as well. Thus, we posit that our approach would likely work for multiple popular programming languages and the results should improve further with the continued advancement of LLM models.

3 METHODOLOGY

3.1 Context

The data used in this study were from two classes of a first-year programming course taught in university xxx in the year 2022. This course is conducted over a duration of 17 weeks and is compulsory for all first-year Information Systems or Computer Science students.

In this course, the students were taught how to develop both static and dynamic web applications using PHP (back-end) and HTML (front-end). Object Oriented programming and storage and retrieval of data to and from database were also covered within the

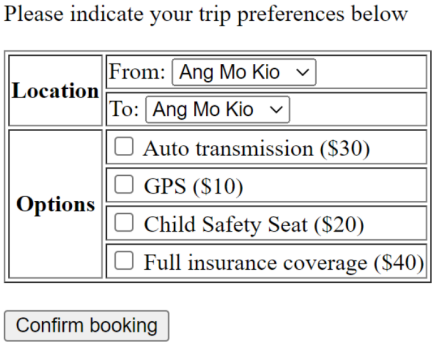

Figure 1: q2.html as displayed on a web page

You booked a vehicle for your trip from Ang Mo Kio to Jurong West

No	Description	Cost($)
1	Auto Transmission	30
2	Child Safety Seat	20
3	Mileage	31
Total cost		81

Figure 2: process.php after submission as displayed on a web page

course. The course assessment comprised of 2 lab tests and a final exam. The lab tests were conducted in week 8 and 14 of the course.

The data for this study were extractions of students' program codes that were submitted for the two lab tests. Throughout the course, the students also worked on in-class exercises provided by the instructors each week. The students were encouraged to collaborate to solve the problems in the in-class exercises and they do not need to submit these exercises. We chose to use the code extractions from the lab tests instead as the students were required to submit the lab test codes for assessment purpose. More importantly, the codes and associated errors can be attributed to the individual student's own effort (with no external help) as it was completed under the instructor's invigilation within a set duration.

This study was approved by the university's Institution Review Board (IRB) and we extracted only the program codes of students who consented to their use in this study. A total of 67 out of 78 students consented to the use of their program codes.

The objectives of the course are listed below.

- Understand how the web works
- Understand HTML and be able to create, modify and debug static webpages involving various HTML tags

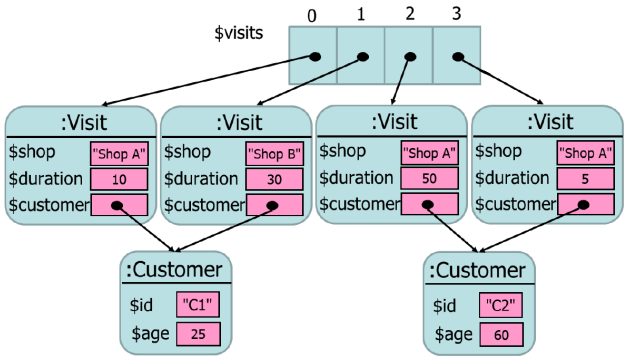

Figure 3: Memory state diagram of visits array

Shops visited by seniors:

- Shop A

Figure 4: view2.php as displayed on a web page

- Demonstrate the ability to create, modify and debug dynamic webpages using PHP
- Understand the concepts of Classes and Objects and able to use them to create dynamic webpages
- Understand how to interact with a database and be able to store, modify, and retrieve data from PHP code
- Demonstrate the ability to create web applications implementing the four main functionalities of persistent storage: create, retrieve, update, delete
- Understand session and be able to perform session management using PHP
- Demonstrate the ability to solve logic problems that can be encountered while creating web applications, and create suitable solutions using PHP

3.2 Questions

Listed below are the specifications of the questions for lab test 1 conducted in week 8 and lab test 2 conducted in week 14 of the course. Lab test 1 assesses the students on the building of both static and dynamic web applications while the coverage for lab test 2 focuses on Object Oriented Programming (OOP) and database interaction using PHP.

3.2.1 Lab Test 1. The question for the first lab test required the student to complete the codes for calculating car rental fees.

Students were provided with q2.html in Figure 1 and process.php. The file process.php contains the codes for initialization of two arrays. The first array provides a mapping between the selected locations to mileage in kilometres (km) and the second array provides the mapping of the options e.g., GPS to the cost.

The students were provided the formula for the calculation of mileage cost below.

- A flat cost of $5 for the first 10 km.
- For total mileage (x) between 10 and 30 km (exclusive), every additional (x-10) km is charged at $2 per km.
- For total mileage (x) more than 30 km (inclusive), every additional (x-10) km is charged at $1 per km.

The students were tasked to complete the codes for printing the overall cost breakdown and the total cost for the booking in a table within process.php as shown in Figure 2.

In addition, the following checks would have to be performed.

- If no options are selected and user clicks "Confirm booking", only the mileage cost is calculated and displayed.
- If process.php is visited without going through q2.html, redirect to q2.html.

3.2.2 Lab Test 2. In this question, the students were tasked to complete view2.php. The file Visit.php contained Visit class definition while the file view2.php which had template codes that loaded an array of Visit object with the data from a database. Both Visit.php and view2.php were provided to the students. A memory state diagram of the visits array (Figure 3) was also provided to the students. The students were tasked with completing view2.php to implement the following functionalities:

- Iterate through the Visit objects in the visits array

```
| S.No | Error | Original Code Line | Corrected Code Line | Type of Error |
|---|---|---|---|---|
| 1 | Assumed submission from the form without conducting a check | `$from = $_POST['from'];` |
`if(isset($_POST['from'])) $from = $_POST['from'];` | Error handling logic |
| 2 |  The location specified in the echo statement is not changing | `echo "You booked a vehicle for your
trip from" . $location_mileage["amkjw"][0] ." with the cost breakdown below.";` | `echo "You booked a
vehicle for your trip from" . $location_mileage[$from.$to][0] ." with the cost breakdown below.";` |
Incorrect values |
```

Figure 5: Sample prompt response from ChatGPT

- Identify the visits that were made by a senior citizen using the wasMadeByASenior() method (within the Visit class)
- Identify the shops that were visited by a senior and display them in a bulleted list
- If a shop was visited more than once, show the shop only once in the list

The final expected output of view2.php is shown in Figure 4.

3.3 Workflow

For both lab tests, the students were provided with a minimal code template for completion, and they were instructed to key in their names and email addresses within the first three lines of comments. To anonymize the extracted students' programming code submissions, we renamed the source file to a generic name e.g., LT1_XX where XX is a running sequence of numbers and removed the first three lines of comments (containing the students' names and email addresses).

We subscribed to OpenAI API and developed a Python script to construct the prompts and interface to ChatGPT's GPT-4 model directly through the API. We designed the prompt passed into GPT-4 as shown below.

```
Given the following model code solution
enclosed within ``` below.
```<Model code>```

Identify the errors in the code below that performs
the same function and enclosed within ``` below.

Show the error, original code lines,
corrected code lines and type of error in a tabular form.
```<Student's code>```
```

The prompt was formulated for all the students' submitted codes before being passed into GPT-4. An example prompt response from GPT-4 is shown in Figure 5.

From the prompt responses, we wrote a Python script to parse and collate the prompt responses from GPT-4 for summarization of the different error types. We then evaluate each of the prompt responses for only the first lab test manually with the results of the manual verification shown in Table 1.

Table 1: Verification results for errors picked up by GPT-4

Errors classified correctly? (Yes/No)	No. of errors (percentage)
Yes	348 (93.3%)
No	25 (6.7%)
Total	373 (100%)

As seen from Table 1, 93.3% of the errors picked up by GPT-4 were correct (as verified manually). We did not constrain the error types to a pre-specified list in the prompt passed into GPT-4 as we wanted to explore whether GPT-4 model can generate more descriptive and precise error types. Comparing passing in the natural language instructions that detailed specifically what is to be achieved by the student for the question as opposed to the model code or solution into the prompt, we opted for the latter. The reason being that for questions similar to lab test 1, an image of the web page instead of lengthy descriptions using text would be more efficient and effective for illustrating the required look and feel. However, ChatGPT does not support the input of images as of the date of this study.

We did also experiment with just passing in the program specifications instead of the model solution codes. This, however, resulted in GPT-4 generating program codes which differed from what was required for the question. Further clarification prompts were required to tune the GPT-4 generated codes to the required specifications. It is plausible that different clarification prompts would be required for different program specifications or questions which may entail even possibly the need to provide GPT-4 with examples. In contrast, use of the model code solution allows for a standardized prompt to be passed in to the GPT-4 model which will not need to vary for different programming questions.

We recognize that one constraint with the use of a model code solution is that it may not accommodate students' codes that satisfy the required specifications using programming constructs or algorithms different from that used in the provided model solution. In our study, however, the students were provided with a code template that were partially filled as well as the related accompanying codes. That constrained the variations of solution codes which satisfy the program specifications. The generalization of LLM in recognizing or understanding more open or different versions of programming codes which fulfil the same set of specifications is not within the scope of this study and can be investigated in a future study.

4 RESULTS

Table 2 shows the code errors that were correctly picked up by GPT-4. We chose these samples of code errors as they were difficult to detect even with the use of custom code parsers and this also explains why most of the sample code errors listed are logical errors.

In the first row of the table, GPT-4 was able to detect that the name within 'POST' array did not match with the input text field

name in the HTML file and both the generated error type and error description was apt.

In the second row, hard coding of the locations instead of retrieving them from the HTML form was correctly picked up as well. Rows 3 to 5 relate to the detection of computation errors which may require the writing of custom test scripts for their detection before the advent of LLMs.

The error in the last row of the table points to a deficiency in the student's code where duplicate shop names would be displayed. The error description correctly and specifically advises that the list of unique shops should be generated first instead before the loop to display the unique shop names. It is worth highlighting here that GPT-4 was able to understand from the student's code that duplicate shop names would be displayed which would not fulfill the requirements of the question. Another finding from the results is that the GPT-4 generated error description aptly describes the specific error latent within each erroneous block of codes.

Table 3 shows the codes errors that were either wrongly picked up or labelled with incorrect error types by GPT-4. The error in row 1 which relates to the use of addition instead of concatenation operator is incorrectly classified as a syntax error and should be a run-time error instead.

The error listed in the second row should not be an error. GPT-4 was not able to infer that age is discrete and thus the condition that age is greater than 59 is the same as age greater than or equals to 60. For ease of discussion, we will refer to this as the "discrete age" issue.

The error listed in the third row should be a run-time error instead of a syntax error as there is nothing wrong with the syntax of the student's code. The student's code in the last row is correct but differs from the model solution codes with the swapping of the condition check for age. GPT-4 was not able to establish that the student's codes is actually similar to the solution in this case. For ease of discussion, we will refer to this as the "condition check swap" issue.

The top three collated types of errors across the two lab tests are shown in Table 4. The types of error generated by GPT-4 were not granular enough for identifying the specific difficulty or misconception that students were facing. The error description (error column in the prompt response) as seen in Figure 5, would fit the purpose but it would be onerous for the instructor to go through all the error descriptions to identify the misconceptions. Thus, for an automated way of summarizing the error descriptions, we passed in the list of error descriptions and prompted GPT-4 to group them into clusters as shown in the prompt below.

```
Given the list enclosed within ``` below.
Cluster all the items in the list into minimal
number of clusters. Display the output in JSON format
```

Table 2: Sample errors or error types correctly identified up by GPT-4

Error description	Erroneous code	Correct Code	Error type
The form field name in the 'POST' array does not match the form field name in HTML file.	'$from = $_POST["from"]; $to = $_POST["to"];'	'$from = $_POST["start_loc"]; $to = $_POST["end_loc"];'	Logical Error
Locations are hard-coded in PHP whereas they should be obtained from HTML form	'if ($from == 'AMK' && $to == 'Jurong')'...	'$startLoc = $_POST['start_loc']; $endLoc = $_POST['end_loc']; $combinedLoc = $startLoc . $endLoc; $distance = $location_mileage[$combinedLoc][1];'...	Logical Error
$totalcost is calculated twice	'$totalcost=$mileagecost+$addon; $totalcost=$mileagecost+$addon;'	'$totalCost = $mileageCost; $totalCost += $addOnItemCost;'	Logical error

Incorrect mileage cost calculation	'if ($distance > 20) {
	$distance = $distance - 20;
	$cost = $cost + 40;
	$cost = $distance + $cost; }'
		'if ($totalMileage>30) {
		$mileageCost = 5 + ($totalMileage-10) * 1;
		} else {
		$mileageCost = 5 + ($totalMileage -10) * 2;
		}'
			Mathematical Error

Calculation of No for table entries is incorrect	$count=$count+count($option);	$num +=1;	Logical error
The list of unique shops should be generated first before echoing the shop names. The provided solution echos the shop names directly in the loop.	'foreach($visits as $visit) {... echo " $shopname "; ...}'	'foreach($visits as $visit){... $visited[] = $shopname; ...}' and 'foreach ($visited as $shop){ echo " $shop "; }'	Logical Error

Table 3: Sample errors or error types wrongly identified up by GPT-4

Error description	Erroneous code	Correct Code	Error type
Wrong calculation for $trip. It should be concatenation not addition	‘$trip = $start + $end;‘	‘$combinedLoc = $startLoc . $endLoc;‘	Syntax Error
Incorrect comparison operator	‘if($age > 59)‘	‘if($age>=60)‘	Logical Error
Wrong method to get age	$CusAge = $this->getAge($this->customer);	$CusAge = $this->customer->getAge();	Syntax Error
Different structure of condition in the function "wasMadeByASenior"	‘$customer = $this->customer; $age = $customer->getAge(); if($age<60) return False; else return True; ‘	‘if ($this->customer->getAge()>=60) return true; else return false; ‘	Logical error

```
as in {'cluster':, 'issues':[]}.
```
```
<List of error descriptions>
```

The clustering result output by GPT-4 for lab test 1 and lab test 2 are shown in Tables 5 and 6 respectively. The cluster labels in Tables 5 and 6 are the actual cluster labels as generated by GPT-4 i.e. we did not rename the cluster labels.

5 DISCUSSIONS

To answer RQ1, we manually verified the summary of error generated by the GPT-4 model of ChatGPT on lab test 1. The verification results as shown in Table 1 demonstrated that ChatGPT is able to achieve 93.3% accuracy for the generation of error summary. Although ChatGPT did not achieve perfect accuracy of 100%, the accuracy of 93.3% is sufficiently high for aggregating error summaries across the many students' code submissions to offer us a glimpse into the common errors made by students. With the proposed methodology, we were also able to classify the error summaries from ChatGPT into the top error types ranked by frequencies as seen in Table 4.

In the prompt that we passed into GPT-4 model, we included both the students' submitted codes and the model solution for generation of the error description, original code, corrected code and type of error. We did not constrain the types of errors to a specific list of error types.

For most of the discovered erroneous code lines though, the GPT-4 model managed to categorize most of them as syntax, logical and semantic errors. There were also some of the other error types generated which were more specific e.g., error handling logic where some exceptions were not handled, run-time error and omission error which depicted that some functionalities specified in the question were not implemented. We suspect that the model was able to discover the omission of functionalities and wrong or missing handling of exceptions because we passed in the model solution codes which allowed the model to compare between the functionalities implemented in the model solution with that implemented in the students' codes. However, we are not sure whether the same capability can be achieved with just the inclusion of program specifications in the prompt.

GPT-4 model was able to identify a number of logical errors which can only be detected with custom code parsers and test scripts. Some of these (listed in Table 2) include hard coding (fixing of values within the programming codes) errors, computation errors and even errors relating to program output which did not conform to the specifications. GPT-4 was able to comprehend the codes and interpret the intent of the programmer e.g., in identifying the incorrect computation of mileage cost. This is also evident in row 5 of Table 2 where GPT-4 identifies that the correct intent of the student was to display the sequence number of each entry (which should be increased by 1 for every consecutive entry) in the HTML table. This matching to the correct block of codes within the model solution is even possible when the variables are named differently between the student's codes and the model code solution e.g., $distance instead of $totalMileage (in row 5 of Table 2). Considering that we only passed in the model solution code (without code comments to explain the intent of the blocks of codes) and not the problem specification, it is thus credible that GPT-4 comprehends the codes and is able to match it correctly with the original intent and the correct block of codes within the model code solution. More importantly, this would provide further support for the generalizability of our study to different programming problems.

We noted, however that in some of the code submissions (listed in Table 3), GPT-4 model mistakenly identified codes that were correct as errors e.g. "discrete age" and "condition check swap" issues. In addition, some of the codes were also assigned with incorrect error type labels. This points to the need for further tuning of the GPT-4 model to achieve higher accuracy in identifying and classifying the errors. One suggestion to rectify issues of false positives where

Table 4: Top three collated types of error across the two lab tests

Types of error	Lab test 1	Lab test 2
Logical error	169	43
Syntax error	58	36
Semantic error	11	7

Table 5: Clustering of error description for logical and syntax errors found in students' codes for lab test 1

Types of error	Cluster label	Subset of error description
Logical error	Incorrect calculation	- Wrong calculation for mileage - The mileage calculation for distances greater than 30 is incorrect
	Form validation and handling	- The option field name in the 'POST' array does not match the form field name in HTML file - Elements 'from' and 'to' not defined
	Incorrect loop usage	- The incrementing of the count variable '$i' is missing after adding the mileage cost to the 'add_on'_array - For loop for fetching requested options is incorrect
	Incorrect array usage	- Incorrect array element $add_ons element index should be a number - Incorrect variable in array push
Syntax error	Syntax error	- The 'echo' statement at the end does not have a semi-colon which would lead to syntax error - Missing closing parenthesis
	HTML tag error	- The tr tag in the foreach loop for creating the table rows is not closed correctly - The opening HTML tag left unclosed
	Variable definition error	- Undefined '$price' variable - No variable '$msg' and '$mileage' defined

Table 6: Clustering of error description for logical and syntax errors found in students' codes for lab test 2

Types of error	Cluster label	Subset of error description
Logical error	Redundancy issues	- Shops' names are repeatedly printed, regardless if it's already printed before - The shops are redundant and it is necessary to remove the duplicates
	HTML placement and format Issues	- echo "ul" is placed inside the loop, it should be outside the loop so that it is not repeatedly printed for each iteration - The unordered list and closing tags ' ' and ' ' are inside the foreach loop used to list out the different shops, which causes an error where every shop is displayed as a separate unordered list
	Code structure and logic errors	- The code doesn't allow for a dynamic number of shops. It assumes only shops A, B, and C exist - The shops are predefined in the code but they should be determined dynamically from the visits - Not properly checking if an item is in the '$shops' array before adding it.
Syntax error	Syntax error	- Missing braces in the foreach loop - Missing semicolon at the end of the line
	HTML tag error	- Incorrect HTML tag for unordered list: Tag should be ul not ui - Incorrect closing tag placement of the unordered list
	Variable misuse	- Not initializing the $shops array - Variable naming mismatch. Variable $shops used instead of $shop

GPT-4 wrongly identifies working codes as erroneous could be to develop test scripts and couple that with the use of GPT-4 for error identification and classification. The test scripts can either be manually written or possibly auto-generated from GPT-4 by passing in the program specifications. The prompts can then be re-designed to incorporate the passing in of both the test scripts and the model solution for more precise error identification and classification. This should rectify issues such as "discrete age" and "condition check swap" highlighted in the results section.

From the results, we also discovered that the error type summary alone, does not suffice in enabling instructors to understand the misconceptions of students for enacting appropriate teaching strategies. To illustrate, we were able to surmise from Table 4 that there were more significantly more logical errors made by students for the first lab test as compared to the second one. However, we were still clueless as to which part of the program logic did majority of the students encounter difficulties with implementing. As such, we will require more granularity into the errors made by students to investigate into the use of the error summaries for identifying the misconceptions of students (RQ2).

After clustering of the error descriptions, we were able to get more granular summary of the errors made by the students within their code submissions and these were summarized in Table 5 and 6. From the clustering results as listed in Table 5 and 6, we surmise that the errors are reasonably assigned to their relevant clusters with adequate clustering labels (generated by GPT-4).

From Table 5, the logical errors made by students for lab test 1 can be traced to incorrect computation of costs, form validation and handling and incorrect array and loop usage. The students find it challenging to translate from the given cost computation formula (for both mileage and the selected options e.g. GPS, child seat cost) in the specification into the correct program codes.

Another misconception is with HTML form handling where students used the label for the drop-down boxes instead of the name attribute to retrieve the selected value for the start and end location.

The final misconception had to do with the processing of arrays and iteration of the elements of the array using a loop. It appears that many of the students were not proficient with the iteration of elements of an array using loops (e.g. using non-numerical variables as indexes for array) and how to insert new elements at the end of an array using array_push function. For syntax errors, the common mistakes made by students were missing semi-colons, missing parenthesis, use of undefined variables and non-matching of HTML opening and closing tags.

From the clustering results as shown in Table 6, the logical errors made by students for lab test 2 can similarly be traced to duplicate printing of shops, HTML placement and format issues and hard-coding of the list of shops. Students were not sure how to iterate through the visits array to get a unique list of shops visited by the customers i.e. some shops may be visited by multiple customers but the shop name should only be displayed once.

One surprising discovery here though is that the GPT-4 model even manages to pick up hard-coding of the list of shops, a feat which is unattainable with the use of existing techniques e.g. code parser. These students pre-defined or hard-coded an array containing the values 'Shop A', 'Shop B' and 'Shop C' and use that to filter out the visits for each in turn instead of retrieving the shop names dynamically from the visits array. The syntax errors for lab test 2 were mostly similar to that made by students in lab test 1.

6 THREATS TO VALIDITY

The results presented are based on two PHP lab tests and involved only the code submission from 67 students. This thus limits the generalization of our results for other programming languages and for a larger variety of programming problems.

In addition, in our study, the students were not required to write the programming codes from scratch but were instead provided with code templates where they had to fill in missing parts of the programming codes. This likely constrained the search space for error identification and classification within the submitted programming codes. The results presented thus may not apply for cases where the students are free to develop their own programming codes to fulfill the specifications.

We did not empirically evaluate the clustering results generated by GPT-4 model. These will entail the verification that all the errors are assigned to a cluster (coverage), the errors within each cluster are cohesive, the clusters are well separated and lastly, the clusters are relatively balanced in size. It is thus possible that the clustering results reported in this study may not be optimal and there are other clustering or non-clustering algorithms which can possibly perform better than GPT-4 in categorizing the error descriptions.

7 FUTURE WORK

We would suggest possible extensions to this study as listed below.

- Investigating the generalization of the error identification capability across different programming languages and code complexity.
  In this study, we used GPT-4 for error identification of PHP programs. Conceivably, the ability of LLMs to understand and generate programming codes across different languages should allow it to identify and categorize errors across different programming languages and for codes with different complexities. This would, however, require validation from future studies.
- Refining the clustering and categorization of error descriptions.
  The error descriptions were clustered into the different categories using only GPT-4 model. It is plausible that the clustering by GPT-4 may not be optimal and other clustering techniques may offer better categorization of the error descriptions.
- Seeding the model with inputs other than the model code solution to further enhance identification and categorization of the errors.
  We passed in both the students' codes and model code solution into GPT-4 for the error identification and categorization and this was validated to work well in this study. With further enhancement in the capability of LLMs e.g. interpreting images,we can possibly pass in other inputs such as program specifications in the form of images and investigate into its viability for enhanced error identification.

- Investigating the impact of automated error identification and categorization on teaching of computer programming. Brown and Altadmri [4] concluded in their study (comparing instructors' opinions versus actual students' misconceptions in Java programming) that instructors failed to form an accurate consensus on the programming misconceptions of their students. Qian and Lehman [19] further posited that with an accurate picture of the students' misconceptions, instructors can then formulate effective teaching strategies and tools to address these misconceptions. We thus suggest future extensions of this study to investigate and evaluate the various teaching strategies which may be devised with this enhanced understanding of the students' misconceptions.

8 CONCLUSION

The identification of students' misconceptions is imperative for informing instructors on the difficulties faced by their students in the learning of computer programming. It is however onerous for the instructors to manually go through each student's erroneous programming codes for compilation of the common errors and misconceptions. The current state of the art techniques for identification of the errors require custom tools and scripts to be developed and adapted for the different programming languages and this may not be feasible to implement for every educational institution. In this study, we proposed the use of ChatGPT in identifying and summarizing the errors and misconceptions of students through seeding the model with the students' code submissions and the model code solution. The results demonstrated that ChatGPT is able to not only identify syntax but also the more difficult category of logical errors. In addition, ChatGPT is also capable of more granular categorization of the errors, providing the instructors with insights into the common misconceptions of their students.

REFERENCES

[1] Marzieh Ahmadzadeh, Dave Elliman, and Colin Higgins. 2005. An analysis of patterns of debugging among novice computer science students. In *Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education*. 84–88.

[2] Toufique Ahmed and Premkumar Devanbu. 2022. Few-shot training LLMs for project-specific code-summarization. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 1–5.

[3] Amjad Altadmri and Neil CC Brown. 2015. 37 million compilations: Investigating novice programming mistakes in large-scale student data. In *Proceedings of the 46th ACM technical symposium on computer science education*. 522–527.

[4] Neil CC Brown and Amjad Altadmri. 2014. Investigating novice programming mistakes: Educator beliefs vs. student data. In *Proceedings of the tenth annual conference on International computing education research*. 43–50.

[5] Neil Christopher Charles Brown, Michael Kölling, Davin McCall, and Ian Utting. 2014. Blackbox: A large scale repository of novice programmers' activity. In *Proceedings of the 45th ACM technical symposium on Computer science education*. 223–228.

[6] Arghavan Moradi Dakhel, Vahid Majdinasab, Amin Nikanjam, Foutse Khomh, Michel C Desmarais, and Zhen Ming Jack Jiang. 2023. Github copilot ai pair programmer: Asset or liability? *Journal of Systems and Software* 203 (2023), 111734.

[7] Paul Denny, Andrew Luxton-Reilly, and Ewan Tempero. 2012. All syntax errors are not equal. In *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education*. 75–80.

[8] Benedict Du Boulay. 1986. Some difficulties of learning to program. *Journal of Educational Computing Research* 2, 1 (1986), 57–73.

[9] Andrew Ettles, Andrew Luxton-Reilly, and Paul Denny. 2018. Common logic errors made by novice programmers. In *Proceedings of the 20th Australasian Computing Education Conference*. 83–89.

[10] Zhiyu Fan, Xiang Gao, Martin Mirchev, Abhik Roychoudhury, and Shin Hwei Tan. 2023. Automated repair of programs from large language models. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 1469–1481.

[11] Austin Henley, Julian Ball, Benjamin Klein, Aiden Rutter, and Dylan Lee. 2021. An inquisitive code editor for addressing novice programmers' misconceptions of program behavior. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*. IEEE, 165–170.

[12] James Jackson, Michael Cobb, and Curtis Carver. 2005. Identifying top Java errors for novice programmers. In *Proceedings frontiers in education 35th annual conference*. IEEE, T4C–T4C.

[13] Lisa C Kaczmarczyk, Elizabeth R Petrick, J Philip East, and Geoffrey L Herman. 2010. Identifying student misconceptions of programming. In *Proceedings of the 41st ACM technical symposium on Computer science education*. 107–111.

[14] Junaed Younus Khan and Gias Uddin. 2022. Automatic code documentation generation using gpt-3. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 1–6.

[15] Wei Ma, Shangqing Liu, Wenhan Wang, Qiang Hu, Ye Liu, Cen Zhang, Liming Nie, and Yang Liu. 2023. The Scope of ChatGPT in Software Engineering: A Thorough Investigation. *arXiv preprint arXiv:2305.12138* (2023).

[16] Ehsan Mashhadi and Hadi Hemmati. 2021. Applying codebert for automated program repair of java simple bugs. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, 505–509.

[17] Davin McCall and Michael Kölling. 2014. Meaningful categorisation of novice programmer errors. In *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*. IEEE, 1–8.

[18] Eng Lieh Ouh, Benjamin Kok Siew Gan, Kyong Jin Shim, and Swavek Wlodkowski. 2023. ChatGPT, Can You Generate Solutions for my Coding Exercises? An Evaluation on its Effectiveness in an undergraduate Java Programming Course. *arXiv preprint arXiv:2305.13680* (2023).

[19] Yizhou Qian and James Lehman. 2017. Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Transactions on Computing Education (TOCE)* 18, 1 (2017), 1–24.

[20] Zahava Scherz, D Goldberg, and Z Fund. 1990. Cognitive implications of learning Prolog—Mistakes and misconceptions. *Journal of Educational Computing Research* 6, 1 (1990), 89–110.

[21] Derek Sleeman. 1986. The challenges of teaching computer programming. *Commun. ACM* 29, 9 (1986), 840–841.

[22] James C Spohrer and Elliot Soloway. 1986. Novice mistakes: Are the folk wisdoms correct? *Commun. ACM* 29, 7 (1986), 624–632.

[23] Weisong Sun, Chunrong Fang, Yudu You, Yun Miao, Yi Liu, Yuekang Li, Gelei Deng, Shenghan Huang, Yuchen Chen, Quanjun Zhang, et al. 2023. Automatic Code Summarization via ChatGPT: How Far Are We? *arXiv preprint arXiv:2305.12865* (2023).

[24] Haoye Tian, Weiqi Lu, Tsz On Li, Xunzhu Tang, Shing-Chi Cheung, Jacques Klein, and Tegawendé F Bissyandé. 2023. Is ChatGPT the Ultimate Programming Assistant–How far is it? *arXiv preprint arXiv:2304.11938* (2023).

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009