Shih, JL. et al. (Eds.) (2023). Proceedings of the 31st International Conference on Computers in Education. Asia-Pacific Society for Computers in Education

# ExGen: Ready-To-Use Exercise Generation in Introductory Programming Courses

**Nguyen Binh Duong TA[a*], Hua Gia Phuc NGUYEN[a] & Swapna GOTTIPATI[a]**
[a]*School of Computing and Information Systems, Singapore Management University*
*donta@smu.edu.sg

**Abstract:** In introductory programming courses, students as novice programmers would benefit from doing frequent practices set at a difficulty level and concept suitable for their skills and knowledge. However, setting many good programming exercises for individual learners is very time-consuming for instructors. In this work, we propose an automated exercise generation system, named ExGen, which leverages recent advances in pre-trained large language models (LLMs) to automatically create customized and ready-to-use programming exercises for individual students on-demand. The system integrates seamlessly with Visual Studio Code, a popular development environment for computing students and software engineers. ExGen effectively does the following: 1) maintaining a set of seed exercises in a personalized database stored locally for each student; 2) constructing appropriate prompts from the seed exercises to be sent to a cloud-based LLM deployment for generating candidate exercises; and 3) implementing a novel combination of filtering checks to automatically select only ready-to-use exercises for a student to work on. Extensive evaluation using more than 600 Python exercises demonstrates the effectiveness of ExGen in generating customized, ready-to-use programming exercises for new computing students.

**Keywords:** introductory programming courses, exercise generation, large language models, prompt engineering, auto-filtering.

## 1. Introduction

Despite recent advances in code generation using large language models (LLMs), e.g., OpenAI Codex, ChatGPT, Google LaMDA, etc., programming is still an essential skill that computing students must master in the foreseeable future (Becker et al., 2023). It is well-known that computer programming is a challenging subject for many new university students (Keuning et al., 2018). To learn effectively, students must practice on their own frequently with suitable exercises designed for their level of programming skills and knowledge. However, it is usually impractical for instructors to manually create enough exercises for all students, let alone creating customized or personalized exercises for each individual student on-demand, i.e., when they need to practice. Exercises from Internet platforms for coding practices and interview preparations (Joshi et al., 2023) such as *LeetCode.com* might not be ready-to-use in introductory courses as they are not customized for the skill levels of new computing students. From our experience, many easy-level LeetCode problems can be quite challenging for novice programmers (Ta et al., 2022). Intermediate or harder problems found online often require advanced data structures and algorithm concepts not taught in an introductory course.

Automatic generation of programming exercises has been an important problem in technology-enabled education (Zavala and Mendoza, 2018). Recently, the application of pre-trained LLMs in supporting exercise and feedback generation has been gaining attention. Sarsa et al. (2022) explored OpenAI Codex for the purpose of creating new Python exercises and code explanations. They provided a comprehensive, mostly manual evaluation of the output generated by Codex. They showed that a large percentage of the generated exercises were sensible and new, but most of them were not directly usable by students. However, they did not address the problem of on-demand generation of ready-to-use exercises customized

for various levels of difficulty. Recently, Kazemitabaar et al. (2023), Becker et al. (2023) and Finnie-Ansley et al. (2023) discussed and evaluated the use of LLMs to generate sample solutions and to provide adaptive feedback on code submitted by students, but not exercise generation.

In this work, we propose ExGen, a software tool for automatically generating new, ready-to-use exercises for individual students learning introductory programming in Python. ExGen makes use of LLMs which have been pre-trained on a massive amount of text and code such as OpenAI's GPT-3.5-Turbo. In ExGen, we implement carefully designed LLM prompting strategies, automatic filtering of exercises, and seamless integration with Visual Studio (VS) Code. ExGen makes it convenient for students to work on new exercises that are customized for their skill levels. An innovative component of ExGen is that it makes use of LLMs not just to generate new exercises, but also to automatically check if the newly generated exercises are ready-to-use or not. In this way, ExGen is different from existing research that investigated exercise generation based on pre-defined templates (Zavala and Mendoza, 2018). We make the following contributions in this paper:

- We considered the problem of auto-generating ready-to-use exercises customized for a certain difficulty level and programming concepts, e.g., string, lists, etc., upon requests from individual students.
- We designed zero-shot and few-shot prompting strategies to obtain appropriate responses from pre-trained LLMs. We then proposed a mechanism of chaining different filtering checks which auto-select ready-to-use exercises instead of requiring human experts to inspect the LLM output and make decisions. The checks are based on popular software engineering techniques such as unit testing combined with LLM-based checking techniques.
- We implemented a fully functional VS Code extension which makes it more convenient for students to generate new exercises and practice coding. We noted that using existing web interfaces such as ChatGPT or OpenAI's Playground requires a lot of copy/paste actions for adjusting prompts and submitting requests.
- We conducted an extensive evaluation of ExGen using more than 600 programming exercises generated via the GPT-3.5-Turbo API. Each of these exercises was manually inspected to see if they are ready-to-use; and compared to ExGen's output. The result demonstrated the effectiveness of ExGen and the proposed solution design.

## 2. Problem Statement

We are interested in auto-generating ready-to-use exercises, i.e., those that a student can work on right away without further modifications by human experts. To be considered ready-to-use, a generated exercise would need to have: 1) a clear problem statement, 2) working solutions and test cases, and 3) an appropriate difficulty level as requested by the student. This is a challenging problem, as Sarsa et al. (2022) found that most generated exercises were not ready-to-use when using OpenAI Codex which is based on the GPT-3 model.

On top of generating clear, well-defined problem statements, producing exercises with the right level of difficulty is of crucial importance to students learning programming. For instance, beginners would be encouraged by easy exercises covering the concepts taught in class as they are just getting started. On the other hand, above average students would like to be challenged with harder problems which require a certain level of higher order thinking. Many programming problems available on Internet platforms (Joshi et al., 2023) are not appropriate for students in our course due to them: 1) being very difficult, and/or 2) requiring advanced concepts such as dynamic programming, tree-based data structures, etc., which are not covered in an introductory course for first-year students.

In this work, we define three levels of difficulty for exercises which have been used in our introductory programming course at Singapore Management University. The three levels of difficulty, namely "easy", "intermediate", and "hard", are defined relative to each other. These levels can cater to a cohort of students with wide-ranging levels of skills and abilities. As the course is more about basic programming and computational thinking skills, and less about using Python, some basic exercises can be made harder by disallowing certain Python

utilities such as sorting functions. We can also impose some constraints on the number of loops due to efficiency reasons, and as a result, the difficulty is increased. In the following three examples, we explain the difficulty levels for each exercise which has been used in our course. The solutions are not included here due to space constraints.

**Example 1**: "*Prompt the user for a message. Display the message character by character on the same line but separate every two adjacent characters by a space*". This is an "easy" exercise which tests basic string manipulation skills using a "for" loop in Python. Most students, including those who are new to programming, after being taught the concepts, can handle this exercise without issues. We have been observing that a few might need help with printing out all the characters on the same line. Note that the "easy" exercises are more than just trivial programming tasks such as creating a list, adding two numbers, etc.

**Example 2**: an "intermediate" exercise for string manipulation. The number of lines of code as well as the logic for a possible solution to this exercise is more complicated than that of Exercise 1. The student would have to figure out the exact range for a loop from *m* to *n*, followed by a few complex conditional statements to get the correct output. Therefore, in the context of this paper, Exercise 2 is more difficult than Exercise 1.

```
Example 2: Define a function called dis_nums(m:
int, n: int), m <=n. The function displays all
int numbers between m and n (both inclusive)
separated by spaces. Special cases: if the number
is a multiple of 3 but not a multiple of 5,
display '-' instead. If the number is a multiple
of 5 but not a multiple of 3, display '*'
instead; 3) if the number is a multiple of both 3
and 5, display '#' instead. For instance,
dis nums(4, 16) shows: 4 * - 7 8 - * 11 - 13 14 #
```

```
Example 3: Define a function get_longest_subsq(st)
for returning the longest subsequence from st that
consists of only the letters from the English
alphabet. If there are multiple subsequences of
the same longest length, the function returns the
first one. If st doesn't contain any letter, '' is
returned. Implement this function using just one
for loop. For instance: get_longest_subsq
('ab24[AaBbCDExy0longest$]') returns 'AaBbCDExy',
and get_longest_subsq('12345 ') returns ''.
```

**Example 3**: a "hard" exercise, also for string manipulation. This exercise is not as straightforward as Exercise 1 or 2, as it requires the student to keep track of the starting and ending point of each potential subsequence of letters using a Boolean variable, and the indices in the input. There is also the additional requirement of using a single "for" loop, which makes the exercise more challenging.

We note that it is not practical to manually create a lot of such ready-to-use exercises on-demand, i.e., when students need them, that are personalized to individual skill levels and learning concepts, as many students might be working concurrently on different kinds of problems at various difficulty levels. In the below section, we describe our approach to realizing automatic generation of ready-to-use exercises.

## 3. ExGen: Design and Implementation

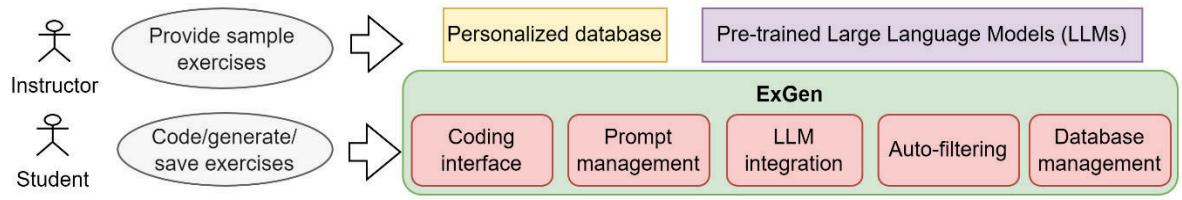



*Figure 1.* Design and Implementation of ExGen

Figure 1 shows the interactions between students, instructors and ExGen in our programming course. As ExGen aims to minimize instructor's involvement, the only role of the instructor is to provide a limited number of sample (seed) exercises with three difficulty levels: "easy", "intermediate", and "hard". Each student in the course can download these exercises into a personalized database which is saved locally. Students can load any exercises from their own

database, work on them, and use ExGen to generate more exercises to practice. Students can choose to save new exercises into their personalized database for reference later.

ExGen provides functionalities for managing a personalized database of exercises for each student, generating new exercises via pre-trained LLMs, and providing a suite of auto-filtering checks to give students ready-to-use exercises at a difficulty level suitable for their knowledge and programming skills. In the below, we provide a brief description of each ExGen component as shown in Figure 1, and relevant implementation details.

### *3.1 Prompt management and LLM integration*

ExGen generates new exercises by first constructing a "prompt" as a query to be sent to a pre-trained LLM such as OpenAI GPT-3.5-Turbo. It obtains a list of candidate exercises for further filtering before presenting a few selected ones in an IDE, e.g., VS Code, for the student to work on. Constructing good prompts for LLMs in many application domains, i.e., "prompt engineering", is currently an active area of research (Liu et al., 2023). LLM prompts may consist of questions, instructions, examples, etc., so that the model would reply with an output which has some desired qualities and/or quantities. In "zero-shot" prompting, we can query the model without providing any example of expected results. On the other hand, "few-shot" prompts provide several actual examples to the model.

In the GPT-3.5-Turbo model, which is used in ExGen's implementation, input prompts can be given as a series of messages with different parameters. Each message is an object with a role, which is either "system", "user", or "assistant", together with the content of the message. Usually, the first message would be a "system" message which helps define the behavior of the AI assistant. We can then alternate between "user" and "assistant" messages, in which "user" messages provide specific instructions for the AI assistant, and "assistant" messages can be used to provide examples of desired outputs, i.e., the "shots" in few-shot prompting. All messages must be constructed and passed to the LLM in a single API call, so that the model would have the right context when generating the new exercises. Note that OpenAI's web APIs do not have the memory of past requests. In this work, we consider two different strategies of prompting for exercise generation using GPT-3.5-Turbo, namely:

- **Zero-shot Prompting** (referred to as "zero-shot"): provides keywords on the programming concepts and difficulty levels required by a student, without using any examples of exercises. The prompt starts with a system message, e.g., "*You are a helpful teaching assistant for undergraduates who are learning introductory programming in Python*", and then adds a user message, e.g., "*Give me three easy Python string exercises with this keyword: house*".

```
Message 1: {"role": "system", "content": "You are a helpful teaching assistant for
undergraduates who are learning introductory programming in Python."}
Message 2: {"role": "user", "content": "Give me a hard Python string exercise."}
Message 3: {"role": "assistant", "content": "Here is one hard Python string exercise:
{example of a relevant exercise from the database}"}
Message 4: {"role": "user", "content": "Good. I want one more hard Python string
exercise. Print the result with the same format as the previous ones."}
... (more messages for remaining examples)
Last message: {"role": "user", "content": "Good. I want {N} more hard Python string
exercises using this keyword: {keyword}. Print the result with the same format as the
previous ones."}
```

*Figure 2*. Messages for Few-shot Prompting

- **Few-shot Prompting** (referred to as "few-shot"): provides a few actual examples on how the desired output should look like. In essence, when requested by students, ExGen constructs a series of messages following the GPT-3.5-Turbo model as shown Figure 2. ExGen first inspects the current exercise that the student is working on. Second, it searches for other exercises on the same concept, e.g., string, list, etc. and having a similar level of difficulty in the student's database. ExGen then sequentially appends the relevant examples, each with a problem statement, solution, and test cases, with the

appropriate roles into the list of messages. The result will be like a conversation in which the LLM plays the role of a teaching assistant guided via a multi-turn conversation with several concrete examples. The last message is a user message requesting the LLM to generate N new exercises. Students can also specify some new keywords, e.g., "person", "movies", etc., so that the generated exercises would have more variations. Note that we can specify the difficulty levels, e.g., "easy", "hard", etc., and concepts, e.g., "string", "list", etc. in the "few-shot" prompt messages from Message 2 onwards in Figure 2.

In the implementation of "few-shot", ExGen uses three examples (3-shot) with the default parameters of the GPT-3.5-Turbo model. We deploy our own cloud instance for running LLM inferences using Azure OpenAI Service for better control and billing purposes. We also note that previous work in exercise generation, e.g., (Sarsa et al., 2022), mostly made use of the OpenAI Codex API, which has been deprecated since Mar 2023. The output of this stage is N candidate exercises, each having: 1) a problem statement, 2) a solution, and 3) test cases. ExGen will conduct auto-filtering for these exercises in the next stage.

### *3.2 Auto-filtering of generated exercises*

In ExGen, we implement several methods to filter exercises generated by LLMs. The methods are chained, i.e., a newly generated exercise must successfully clear the previous filter before it is checked by the next filter, as shown in Figure 3. We consider standard software engineering techniques as well as LLM-based approaches for filtering. First, ExGen will check if the Python solution code can be compiled to bytecode without any errors. Then it will run all the unit test cases which are generated by the LLM together with every new exercise.

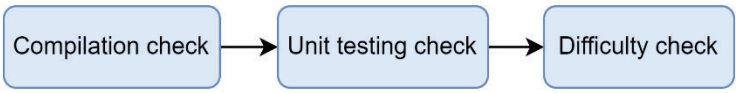



*Figure 3*. Chain of Filters in ExGen

After a new exercise can pass the first two filters, ExGen will do an additional check regarding the difficulty level of the exercise. The rationale for implementing this final check is that LLMs such as GPT-3.5-Turbo are probabilistic models, and their answers might be different from one interaction to the next. Therefore, it could be beneficial to ask an LLM to verify its own answers generated previously.

```
Message 1: {"role": "system", "content": "You are a classification model that will
classify the difficulty of Python exercises."}
Message 2: {"role": "user", "content": "I want to you classify this exercise: {example
exercise}"}
Message 3: {"role": "assistant", "content": "Difficulty: " {example's difficulty level}}
... (more messages for remaining examples)
Last message: {"role": "user", "content": "I want you to classify this exercise:
{candidate exercise}"}
```

*Figure 4*. Example Messages for Difficulty Classification

For the difficulty check, ExGen will perform a classification task by first constructing an appropriate prompt as shown in Figure 4, and then asking the LLM to classify the generated exercise in terms of difficulty levels. The prompt has two examples for each level of difficulty in a series of user and assistant messages. For the last user message, we ask the LLM to provide a classification for a new candidate exercise based on the provided examples. If the difficulty of the new exercise matches the required difficulty level, it will be shown to the student as a ready-to-use exercise using ExGen's VS Code extension. In addition, we observed that GPT-3.5-Turbo can generate exercises covering the concepts specified in the prompt such as lists, strings, etc. most of the time, so ExGen does not need to check if a requested programming concept is included in the output.

### *3.3 Coding interface and database management*

ExGen provides a convenient interface for students to work and generate exercises right in VS Code. We have used ChatGPT as well as OpenAI's Chat Playground extensively, and we observed that it takes a lot of time and effort to copy/paste prompts and exercises for generation tasks. A screenshot of ExGen's interface is shown in Figure 5. Students can click on "+ More exercise" at the right bottom of the IDE to generate new ready-to-use exercises. They also have the option to change the problem statement, solutions or test cases used as examples in the prompt before clicking "Submit". ExGen will automatically connect to an LLM deployment, generate, and filter the exercises.

Students can also save selected ready-to-use exercises into his/her own personalized database. To make it easy for students, we use the TinyDB package, which is a portable Python document-oriented database, instead of setting up a fully functional SQL database engine like MySQL. The database comes bundled in ExGen so students can just use it right away without further setup or installation.


*Figure 5.* ExGen's VS Code Interface

## 4. Evaluation

In this section, we aim to answer the following research questions (RQs):

- **RQ1**: Which is the best prompting strategy for generating ready-to-use exercises?
- **RQ2**: How effective is the auto-filtering approach implemented in ExGen?
- **RQ3**: How much time does it take to generate a ready-to-use exercise?

### *4.1 Dataset and performance measures*

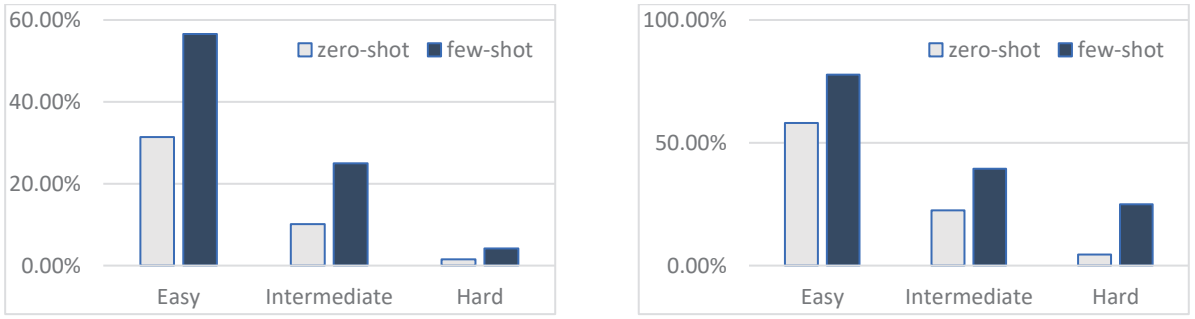
We used a seed set that has 49 manually crafted Python exercises including all levels of difficulty and introductory programming concepts such as string, list, dictionary, tuple, etc. There were 18 "easy", 17 "intermediate", and 14 "hard" exercises in the seed set. We then generated a total of 613 exercises (205 "easy", 270 "intermediate", and 138 "hard") using ExGen via the API of GPT-3.5-Turbo LLM. We then manually inspected each of these exercises to determine if they are ready-to-use. A ready-to-use exercise must satisfy the following criteria used in the manual inspection: 1) a clear problem definition, 2) a correct solution and test cases, and 3) a correct difficulty level as requested in the prompt.

We then measured the performance, i.e., accuracy, of each prompting strategy and filtering check when compared to the manual inspection results. For example, when "few-shot" prompting produces 5 ready-to-use out of 10 generated exercises, the performance of "few-shot" is calculated as 50%. On the other hand, the performance of a filtering check is calculated by how often it agrees with the manual inspection result. For example, the performance of the unit testing check is calculated as 70% if it provides the same result, i.e., ready-to-use or not, as the manual inspection for 7 out of 10 generated exercises. We measured the individual performance of each filtering check, as well as the overall filtering performance.

*4.2 Results and discussion*

**RQ1-Comparing different prompting strategies**: Figure 6(a) shows the percentages of ready-to-use exercises for the two prompting strategies and three difficulty levels. The "few-shot" prompting performed much better than "zero-shot" prompting in all cases. This is because "few-shot" incorporates examples to better guide the LLM in generating new exercises with appropriate levels of difficulty. We observed from the collected data that the "zero-shot" prompt usually produced many more trivial exercises, such as creating a list of several numbers, printing out a message with certain parameters, etc. These are not even at the "easy" level. This is understandable, as the "zero-shot" prompt does not have enough information about how an easy or hard exercise would be like. As a result, about 57% of "easy" exercises generated by "few-shot" prompting were ready-to-use, compared to just 31% of them in "zero-shot" prompting.

From Figure 6(a), it is observed that generating ready-to-use "intermediate" and "hard" exercises is quite challenging, even with recent advances in LLMs. In the experiments, only around 10% and 25% of "intermediate" exercises generated by "zero-shot" and "few-shot" prompting were considered ready-to-use, respectively. Similarly, we obtained only several ready-to-use "hard" exercises using "few-shot".




(a) ready-to-use exercises (%)

(b) exercises with correct difficulty levels (%)

*Figure 6.* Comparing two Prompting Strategies

Figure 6(b) shows the percentages of generated exercises which have the correct difficulty levels (but might not be ready-to-use). By providing more detailed information about difficulty levels with examples in the prompt, "few-shot" produced more exercises with the desired difficulty levels. Around 40% and 25% of exercises generated by "few-shot" could be considered as "intermediate" and "hard", respectively. However, quite a number of these exercises did not have clear problem statements, correct solutions and/or test cases, so they were not considered ready-to-use. Nonetheless, we observed that it's possible to make some small modifications so that many of these more challenging exercises would be ready-to-use.

Table 1. *Performance of the Auto-Filtering Checks*

| | zero-shot (unit test) | zero-shot (difficulty) | zero-shot (chain) | few-shot (unit test) | few-shot (difficulty) | few-shot (chain) |
|---|---|---|---|---|---|---|
| Easy | 68.6% | 37.1% | **69.5**% | 83.8% | 96% | **83.8**% |
| Intermediate | 51.4% | 79% | **90.6**% | 55.3% | 78.8% | **90.2**% |
| Hard | 51.5% | 90.9% | **98.5**% | 54.2% | 77.8% | **97.2**% |

**RQ2-Evaluating ExGen's auto-filtering approach**: The standalone performance of the unit testing and difficulty checks, as well as the overall filtering performance (chain of all checks) are shown in Table 1 (higher percentage is better). From the experiments, we noted that the solutions of most generated exercises could pass the compilation check so that its result is not included in Table 1. The unit testing check when used alone could correctly identify if a

candidate exercise is ready-to-use or not in many cases, especially for "easy" exercises generated with "few-shot" prompting (83.8% accuracy when compared to manual inspection's result). However, unit testing became less effective as the only filter for harder exercises as shown in Table 1. This is because although around 50-70% of candidates for harder exercises had working solutions and test cases in our experiments, they were not at the right levels of difficulty, i.e., "intermediate" or "hard".

Table 1 also shows the performance of the difficulty check when used alone as an exercise filter. It is not very effective when filtering "easy" exercises generated by "zero-shot" prompting as there could be many trivial exercises. The difficulty check performed better when filtering "intermediate" and "hard" exercises, compared to the unit testing check.

We observed good performance for our auto-filtering approach when chaining all the three filters, i.e., compilation followed by unit testing followed by difficulty check. The accuracy of the chained filters improved significantly for all prompting strategies when filtering "intermediate" and "hard" exercises. Overall, we noted that "few-shot" combined with the chained filters produced the best performance. Using this combination, we are more confident that ExGen can provide ready-to-use exercises for students to work on right away.

It was also noted that the LLM used in ExGen tends to provide partially correct solutions and/or test cases when generating "intermediate" and "hard" exercises. Although these candidate exercises did meet the difficulty requirement, they failed ExGen's chained filters in the end. We believe that with a few manual modifications by human experts, they can be ready-to-use by students.

**RQ3-Exercise generation time**: We measured the average time to produce a ready-to-use exercise using the two prompting strategies combined with the chained filters for various difficulty levels. We noted that the generation time was quite reasonable for "easy" exercises, which was around 10 and 20 seconds per exercise with "few-shot" and "zero-shot" prompting, respectively. An "intermediate" exercise required 53 seconds for "few-shot", and 90 seconds for "zero-shot". Using a less effective prompt like "zero-shot" increased the average generation time as there would be more candidate exercises which were not ready-to-use.

On the other hand, generating ready-to-use "hard" exercises could take too long to be considered on-demand (about 8 minutes for "few-shot", and 14 minutes for "zero-shot"). This was because many candidate exercises would have to be generated before ExGen could find one that met the "hard" difficulty requirement and has correct solutions/test cases. To this end, we believe that "hard" exercise generation should be best done in a hybrid way. That is, ExGen can provide initial generation and filtering support for course instructors to quickly modify the LLM's output and make it ready-to-use.

**Discussion:** From the careful evaluation using more than 600 auto-generated exercises, we believe that ExGen can provide students with ready-to-use programming exercises customized at the right difficulty levels. LLM prompting strategies having sufficient details such as examples have been shown to generate more exercises that are ready-to-use. In most cases, our auto-filtering approach with a chain of software engineering and LLM-based checks could work well regardless of the prompting strategy used. However, it is noted that automatically generating truly ready-to-use, "hard" exercises is still a problem in AI-enabled education despite recent progress in LLM and AI development.

**Threats to Validity**: We note that there are several limitations which may affect the validity of this study. First, the output from LLMs could be non-deterministic, which might impact the quality of generated exercises. We have tried to take this into account by evaluating many exercises. Second, manual evaluation of exercises by human experts could be subjective, e.g., when assigning a difficulty level. For this, we have used the opinion of two different programming experts on the set of generated exercises. Finally, as more advanced LLMs such as GPT-4 API are still in limited availability, it is possible that the results obtained from these models are different from what we have seen in this study. We plan to do further investigations using exercises in Python and other programming languages once access to more powerful LLMs becomes widely available.

## 5. Related Work

Much research has been done in automatic generation of formative feedback and reference solutions to code produced by students (Keuning et al. 2018, Koutcheme 2022, Ta et al. 2022). Such feedback and reference solutions could be generated by current pre-trained LLMs to help novice programmers know how to proceed when facing coding issues. On the other hand, not much has been done in generating ready-to-use programming exercises for students to do more practices. Kurdi et al. (2020) conducted a systematic review of automatic exercise generation in many different domains such as analytical reasoning, geometry, history, logic, relational databases, programming, and science.

Zavala and Mendoza (2018) used Automatic Item Generation (AIG) to address the problem of creating many similar programming exercises using pre-defined templates which are used for quizzes. The main goal was to ensure consistency in testing many students with questions of the same level of difficulty. On the other hand, ExGen focuses on generating ready-to-use exercises for a specific difficulty level and concept that the student is currently working on. We leveraged the latest advances in LLMs to autogenerate many novel exercises and filter them to ensure that they are suitable for students. Exercises considered in ExGen are different from other kinds of programming practices such as faded Parson problems (Fromont et al. 2023), which require students to fill in code in partially scrambled solutions.

The most relevant work in automatic exercise generation using pre-trained LLMs has been done by Sarsa et al. (2022). The authors explored OpenAI Codex (which has been deprecated since Mar 2023) for the purpose of creating new programming exercises and code explanations. They found that many Codex-generated exercises were sensible and novel, but may have confusing problem statements, missing or faulty test cases. Sarsa et al. (2022) did not consider the implementation of an auto-filtering tool to supply students with ready-to-use exercises at various difficulty levels.

Recently, there has been an increase in the number of works leveraging pre-trained LLMs for educational purposes. Kasneci et al. (2023) discussed the potential benefits, for instance content generation and personalized learning, as well as challenges, e.g., model biases, system brittleness, etc., when applying LLMs to education. Similarly, Becker et al. (2023) elaborated the educational opportunities of AI code generation, and how educators should act quickly given these developments. Finnie-Ansley et al. (2023) reported the performance of OpenAI Codex on real questions from programming courses when compared to that of actual students. Denny et al. (2023) studied GitHub Copilot, a plug-in for IDEs like VS Code, which is based on the Codex model, to see what kinds of problem it would not perform well. The authors also found that prompt engineering can play an important role in interacting with AI tools like Copilot. MacNeil et al. (2023) found that code explanations such as line-by-line, high-level summary, and lists of important concepts generated by Codex and GPT-3 were helpful to most students. Kazemitabaar et al. (2023) studied how Codex's code generation capability could assist novice programmers via a controlled experiment for young students. We note that our work focuses on automatic exercise generation and filtering, not feedback and explanation for code submitted by students.

## 6. Conclusion

We have implemented and evaluated ExGen, a new software tool that automatically generates ready-to-use exercises for computing students. ExGen manages a personalized database of seed exercises and constructs appropriate LLM prompts to obtain candidate exercises for a requested difficulty level and learning concept. ExGen incorporates a novel combination of auto-filtering checks which reduce the tedious work of manually inspecting all the generated output. Our extensive evaluation with more than 600 programming exercises generated using GPT-3.5-Turbo API demonstrated that ExGen can be used to provide students with ready-to-use exercises. We plan to investigate new approaches for automatic prompt optimization so that harder exercises can be generated more easily. As a VS Code extension, ExGen can be released to our incoming freshmen for further user evaluation in the next academic year.

## Acknowledgements

This research is supported by the Ministry of Education, Singapore, under its Tertiary Education Research Fund (Award No. MOE2021-TRF-014). Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not reflect the views of the Ministry of Education, Singapore.

## References

Becker, B. A., Denny, P., Finnie-Ansley, J., Luxton-Reilly, A., Prather, J., & Santos, E. A. (2023). Programming Is Hard-Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (pp. 500-506).

Denny, P., Kumar, V., & Giacaman, N. (2023). Conversing with Copilot: Exploring prompt engineering for solving CS1 problems using natural language. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (pp. 1136-1142).

Finnie-Ansley, J., Denny, P., Luxton-Reilly, A., Santos, E. A., Prather, J., & Becker, B. A. (2023, January). My AI Wants to Know if This Will Be on the Exam: Testing OpenAI's Codex on CS2 Programming Exercises. In *Proceedings of the 25th Australasian Computing Education Conference* (pp. 97-104).

Fromont, F., Jayamanne, H., & Denny, P. (2023). Exploring the Difficulty of Faded Parsons Problems for Programming Education. In *Proceedings of the 25th Australasian Computing Education Conference* (pp. 113-122).

Joshi, I., Budhiraja, R., Dev, H., Kadia, J., Ataullah, M. O., Mitra, S., ... & Akolekar, H. D. (2023). ChatGPT--a Blessing or a Curse for Undergraduate Computer Science Students and Instructors?. *arXiv preprint arXiv:2304.14993*.

Kasneci, E., Seßler, K., Küchemann, S., Bannert, M., Dementieva, D., Fischer, F., ... & Kasneci, G. (2023). ChatGPT for good? On opportunities and challenges of large language models for education. *Learning and Individual Differences*, *103*, 102274.

Kazemitabaar, M., Chow, J., Ma, C. K. T., Ericson, B. J., Weintrop, D., & Grossman, T. (2023). Studying the effect of AI Code Generators on Supporting Novice Learners in Introductory Programming. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (pp. 1-23).

Keuning, H., Jeuring, J., & Heeren, B. (2018). A systematic literature review of automated feedback generation for programming exercises. *ACM Transactions on Computing Education (TOCE), 19*(1), 1-43.

Koutcheme, C. (2022). Towards Open Natural Language Feedback Generation for Novice Programmers using Large Language Models. In *Proceedings of the 22nd Koli Calling International Conference on Computing Education Research* (pp. 1-2).

Kurdi, G., Leo, J., Parsia, B., Sattler, U., & Al-Emari, S. (2020). A systematic review of automatic question generation for educational purposes. *International Journal of Artificial Intelligence in Education, 30*, 121-204.

Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., & Neubig, G. (2023). Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys, 55*(9), 1-35.

MacNeil, S., Tran, A., Hellas, A., Kim, J., Sarsa, S., Denny, P., ... & Leinonen, J. (2023). Experiences from using code explanations generated by large language models in a web software development e-book. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (pp. 931-937).

Sarsa, S., Denny, P., Hellas, A., & Leinonen, J. (2022). Automatic generation of programming exercises and code explanations using large language models. In *Proceedings of the 2022 ACM Conference on International Computing Education Research-Volume 1* (pp. 27-43).

Ta, D., Shar, I. K., & Shankararaman, V. (2022). AP-coach: Formative feedback generation for learning introductory programming concepts. In *Proceedings of IEEE International Conference on Teaching, Assessment, and Learning for Engineering* (TALE).

Zavala, L., & Mendoza, B. (2018). On the use of semantic-based AIG to automatically generate programming exercises. In *Proceedings of the 49th ACM technical symposium on computer science education* (pp. 14-19).