This paper appears in the 2023 IEEE 35th International Conference
on Software Engineering Education and Training (CSEE&T 2023)
https://doi.org/10.1109/CSEET58097.2023.00028

Toward AI-assisted Exercise Creation for First Course in Programming through Adversarial Examples of AI Models

W. K. Chan*, Y. T. Yu*, Jacky W. Keung*, and Victor C.S. Lee†
* *Department of Computer Science, City University of Hong Kong Kowloon Tong, Hong Kong*
{*wkchan, csytyu, jwkeung*}*@cityu.edu.hk*
† *Department of Electrical and Electronic Engineering, The University of Hong Kong, Pokfulam, Hong Kong*
csvlee@eee.hku.hk

***Abstract*—We propose a new methodology, the Exercise Creation Methodology (ECM), that leverages recent AI technology advancements to create ChatGPT-assisted programming exercises for beginners. ECM takes an existing exercise as input and mutates it by removing some contents into semantically equivalent but syntactically different versions. The pair of versions are labeled as answered correctly and misleadingly by ChatGPT. The removed contents are re-inserted incrementally with further mutation, ensuring the labels remain unchanged. Using the version with the misleading answer and the ChatGPT elaboration on the other version, we construct a ChatGPT-assisted exercise. The latter version may also serve as a solution. We illustrate ECM using a case study.**

Keyword*—*ChatGPT, mutation, exercise, adversarial example

1. Overview

Recent advances in Artificial Intelligence (AI) have demonstrated its ability to simplify the collection and distillation of knowledge. ChatGPT [4] is an AI-powered conversation tool that has garnered significant attention. However, some universities have announced that they would not allow students to use ChatGPT for coursework, while many university educators are discussing how to thrive on this new technology available to learners. Such a tool significantly simplifies students conducting meaningful processes to find answers to questions, bypassing a core value in teaching and learning that motivates students to engage in deep reflection and systematic analysis to solve problems. Although ChatGPT provides quick answers, yet if students are unaware of their (in-)correctness or how they were obtained, these answers may be misleading. If so, such a tool will undermine students' ability to learn the principles behind the coursework exercises.

We thus ask the following *research questions*: How can ChatGPT create programming exercises with solutions that ChatGPT itself cannot solve? What is the amount of human effort required in so doing?

In this paper, we conduct an exploratory study to explore the impact of ChatGPT on a first course in programming for undergraduate computer science or engineering students by producing ChatGPT-assisted programming exercises with solutions. For this purpose, we are developing a novel Exercise Creation Methodology (ECM) as summarized in Section 3 (see also Figure 1).

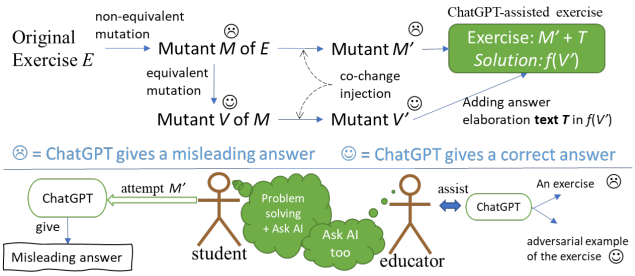


Figure 1: ECM Overview

Suppose ChatGPT can produce a correct answer to a programming exercise E, which contains either a textual question to ask students to develop a program or a question with a piece of code for students to further work on specific tasks. For example, suppose E is the sample exercise with code shown in Figure 2 and its list of errors shown in Figure 3 provided to students for them to fix. ChatGPT produces a correct answer shown in Figure 4, which we refer to as $f(E)$. We wonder whether it is feasible to *slightly* modify E to produce an exercise version M' (such as the one in Figure 7b) that ChatGPT will suggest a misleading program $f(M')$ (Figure 6a), while with a further slightly modified exercise V' (Figure 7a), ChatGPT could suggest a correct program $f(V')$ (Figure 8) similar to the original solution. If this can be done, we can use the exercise version M' (that misleads ChatGPT) together with the correct elaboration associated with the answer $f(V')$ to create an exercise. Moreover, the solution of this ChatGPT-assisted exercise could be $f(V')$. In this way, we leverage advanced AI models to produce a pair of an exercise and its solution.

By the above exercise design, students can attempt an exercise question similar to the past and yet cannot simply use ChatGPT to find answers without critical learning.

The main contribution of this paper is as follows: It presents the Exercise Creation Methodology. This work is the *first work* to show the feasibility of creating programming exercises with solutions using a pair of equivalent

```
#include <iostream>
using namespace std;
int main()
{
   double x,y;
   cin  << "2␣Inputs␣=␣";
   cin  >> x >> y;
   cout << "Yes,␣" << "␣or␣" << "␣no␣"
        << "␣or␣" << "␣maybe." << endl;
   cout << x + y =  (x + y) << endl;
   return 0;    // exit program
}
```

Figure 2: Code in sample exercise E for students to work

	Code	Description
abc	E0349	no operator "<<" matches these operands
abc	E2140	expression must have integral or unscoped enum type
abc	E0299	cannot determine which instance of function template "std::endl" is intended
⊗	C2676	binary '<<': 'std::istream' does not define this operator or a conversion to a type acceptable to the predefined operator
⊗	C2563	mismatch in formal parameter list
⊗	C2568	'<<': unable to resolve function overload

Figure 3: Error list in E to students for fixing the errors

mutants (syntactically slightly different yet semantically equivalent code versions) with co-injection of the same changes. One mutant in the pair is an adversarial example of the AI tool. The other is a mutant of an original exercise that may serve as a solution and enrich the question text.

The rest of this paper is organized as follows. Section 2 presents the preliminaries. Section 3 describes our novel methodology ECM. We walk through ECM in Section 4 with case study results and discuss lessons learned. Section 5 summarizes the paper with related and future work.

2. Preliminaries

2.1. Adversarial Examples of AI models

AI models should ideally have high accuracy and be robust to noise when given a clean input, also known as benign input. In the context of this work, benign input is a question text or a code listing to which an AI model can reasonably answer. However, it is widely recognized that even minor alterations to benign inputs may produce *adversarial examples* [3] that can cause many advanced AI models to behave improperly. We speculate that ChatGPT may be susceptible to the same issue when it comes to coding exercises. Our **insight** is: Educators can use this observation to design exercises that encourage students to adopt investigative problem-solving approaches.

2.2. Mutation Analysis

In the real world, many programs produce desirable results with occasional failures in output. Mutation analysis [5] is a software testing methodology to create a program version (mutant) by slightly modifying such a program. Mutation analysis is based on a set of mutation operators which transform the given program into a mutant. Each mutation operator is designed to simulate a simple syntactic change based on the assumption that a program bug results from a series of *simple* programming mistakes.

There are a few errors in the code:

1. The input operator << is used instead of the output operator >> for the prompt message.
2. The statement `cout << x + y = (x + y) << endl;` is incorrect. The left-hand side of the assignment operator = should be a variable, not an expression. Additionally, the parentheses are not needed in this case since addition has higher precedence than assignment.
3. The prompt message is not asking a question or providing context for the "Yes", "No", or "Maybe" output.

Here is the corrected code:

```
#include <iostream>
using namespace std;

int main()
{
   double x,y;
   cout << "Please enter two numbers: ";
   cin >> x >> y;
   cout << "Do you like the sum of " << x << " and " << y
<< "? (Yes, No, Maybe)" << endl;
   cout << "The sum of " << x << " and " << y << " is "
<< x + y << endl;

   return 0;    // exit program
}
```

This code will prompt the user to enter two numbers, then ask if they like the sum of those numbers. Finally, it will print the sum of the two numbers.

Figure 4: ChatGPT output to fix errors in E (Figure 2)

For illustration, consider the program in E (Figure 2). One mutation operator is to delete a statement randomly. When applied to E, it may delete the first `cin` or `cout` statement. Another mutation operator is to change variable data type(s). For instance, the variables `x` and `y` can be changed to `char` and `int` types, respectively. A mutation operator may also delete a statement fragment, such as deleting `x + y =` from the second `cout` statement. Applying the above mutations to the sample program in E produces a mutant M, as shown in Figure 5a.

3. ECM: Exercise Creation Methodology

The current version of *Exercise Creation Methodology* (ECM), as summarized in Figure 1, requires a human test oracle ($\mathcal{O}$). A test oracle [2] is a mechanism to judge the correctness of the output of the program under test. From hereon, we reserve the term "judge" to mean applying $\mathcal{O}$.

ECM first passes the programming exercise (or its solution) E to ChatGPT f, which generates a suggested code $f(E)$. The oracle $\mathcal{O}$ judges whether $f(E)$ reasonably serves the purpose of a reference solution for E. Next, ECM reduces E through mutation analysis (see Section 2.2) one by one to produce non-equivalent mutants having fewer and slightly different contents than E until the tool f suggests a misleading answer on such a mutant M (i.e., $f(E) \neq f(M)$). To ensure the misleading mutant M is

```cpp
#include <iostream>
using namespace std;
int main()
{
    char x; int y;
    cin >> x >> y;
    cout << (x + y) << endl;
    return 0;    // exit program
}
```

(a) Mutated program M

```cpp
#include <iostream>
using namespace std;
int main()
{
    char x; int y;
    cin >> x >> y;
    cout << x + y << endl;
    return 0;    // exit program
}
```

(b) Mutant N, semantically equivalent to M

```cpp
#include <iostream>
using namespace std;
int main()
{
    int y; char x;
    cin >> x >> y;
    cout << x + y << endl;
    return 0;    // exit program
}
```

(c) Mutant V, semantically equivalent to M

Figure 5: Three semantically equivalent mutants dervied from E. M deletes a few code fragments from E; N adds brackets surrounding `x + y` in M; and V changes the order of variable declaration in N.

```cpp
#include <iostream>
#include <string>
using namespace std;
int main()
{
  char x;
  int y;
  cout << "Enter a character and an integer: ";
  cin >> x >> y;
  string result = string(1, x) + to_string(y);
  // concatenate the char and int into a string
  cout << "Concatenated string: " << result << endl;
  return 0;
}
```

(a) as C1 for mutant M (incorrect)

```cpp
#include <iostream>
#include <string>
using namespace std;
int main()
{
 char x;
 int y;
 cout << "Enter a character and an integer: ";
 cin >> x >> y;
 string result = string(1, x) + to_string(y);
 // concatenate the char and int into a string
 cout << "Concatenated string: " << result << endl;
 return 0;
}
```

(b) for mutant N (incorrect, same as C1)

```cpp
#include <iostream>
using namespace std;
int main()
{
   char x;
   int y;
   cout << "Enter a character and an integer: ";
   cin >> x >> y;
   cout << x + y << endl;
   return 0;    // exit program
}
```

(c) for mutant V (correct)

Figure 6: Code returned by ChatGPT

```cpp
#include <iostream>
using namespace std;
int main()
{
    int y; char x;
    cin  << "2 Inputs = ";
    cin >> x >> y;
    cout << "Yes, " << " or " << " no "
        << " or " << " maybe." << endl;
    cout x + y << endl;
    return 0;    // exit program
}
```

(a) Mutant V' (V with inserted errors)

```cpp
#include <iostream>
using namespace std;
int main()
{
    char x; int y;
    cin  << "2 Inputs = ";
    cin >> x >> y;
    cout << "Yes, " << " or " << " no "
        << " or " << " maybe." << endl;
    cout (x + y) << endl;
    return 0;    // exit program
}
```

(b) Mutant M' (M with inserted errors)

Figure 7: Co-changing the pair of mutants M and V

meaningful to the exercise E, ECM constructs an exercise variant V that is semantically equivalent to but syntactically different from M (i.e., $M \approx V$), such that f suggests a code judged to be similar to the reference solution of E (note $f(V) \neq f(M)$). As such, the variant V represents f's adversarial example (see Section 2.1). ECM then incrementally injects those previously removed contents from the original exercise E and makes further mutations one by one into both M and V to produce a pair of revised versions (M' and V', respectively) such that the tool f suggests different codes for them (i.e., $f(M') \neq f(V')$) and the code $f(V')$ is judged to resemble $f(V)$ (i.e., $f(V') \approx f(V)$). The above process thus results in a ChatGPT-assisted exercise containing (1) E's exercise instruction and other information (e.g., error list) with the mutant M' and f's elaboration message for $f(V')$, and (2) V or $f(V')$ serving as the reference solution.

4. Case Study

4.1. Background

Our case study adopts the teaching material from the first tutorial session of the course *Computer Programming* offered by the City University of Hong Kong. This course teaches students C++ programming. As the impact of AI tools on students is still uncertain, we involve no students in this case study. The exercises cover creating C++ projects in an Integrated Development Environment (IDE) from a major vendor, writing individual statements, fixing syntactic errors, analyzing and solving simple problems, and formatting inputs and outputs. We ran ChatGPT in early March 2023.

4.2. Generating ChatGPT-assisted Exercise

This section walks through our methodology ECM.

Figure 2 shows the code of an exercise E from the tutorial session. E requires students to fix three errors listed in Figure 3. When we input the code to ChatGPT, it produces the message displayed in Figure 4. The message includes the corrected code listing between the sentences "Here is the corrected code" and "This code will prompt ...". The returned code fixes all three errors and adds additional sentences to polish the Console input and output messages that are observable to students.

With ECM, we create adversarial examples from the exercise trying to deceive ChatGPT through mutation analysis. We start with our reference solution code of E, which fixes the three errors but does not polish the input/output wording. We use the program output of our reference solution code as ground truth. For instance, if we input 1 and 2, our reference solution code outputs "`Yes, no or maybe.`" in one line and "`1 + 2 = 3`" in the next line. As presented in Section 2.2, applying the following sequence of mutation opertaions to modify E creates a mutant M (see Figure 5a): (i) Delete the statement `cin << "2 Inputs = ";`. (ii) Delete the statement `cout << "Yes, " << "or " << "no " << "or " << "maybe." << endl;`. (iii) Delete the statement fragment `x + y =`. (iv) Change the type declaration statement `double x, y;` to `char x; int y;`.

We input the mutant M (Figure 5a) to ChatGPT to generate a suggested code C1, as shown in Figure 6a. Rather than relying solely on inspection, ECM executes C1 with test cases and judges the outputs against the ground truth.

As we can interpret from the code listing, code C1 treats the semantics of the program as if it intends to concatenate two strings. By inputting `1` and `2` as test cases, the program outputs the string `12`, which is different from the ground truth (which outputs a value of 3). This result shows that ChatGPT suggests a misleading answer on M.

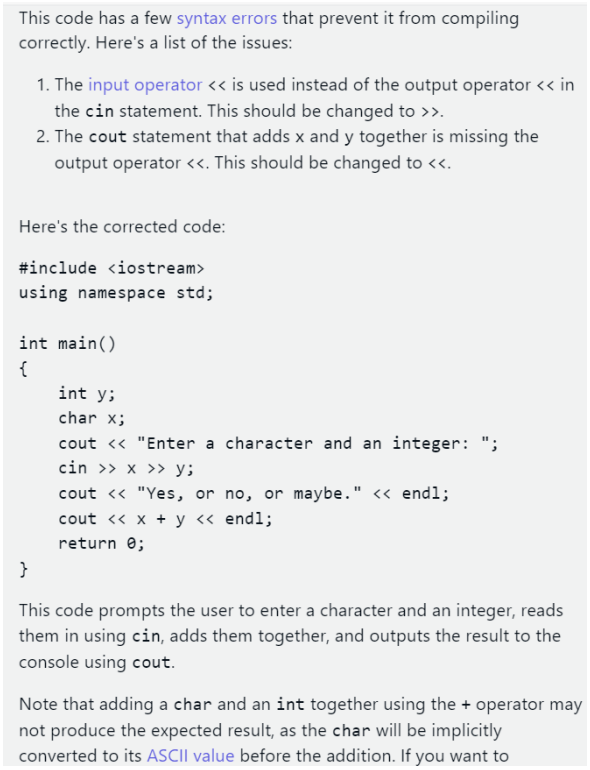
ECM aims to create semantically equivalent program versions of mutant M that exploit the property of adversarial examples incurred by the AI model in ChatGPT. Two program versions equivalent to M are exemplified as N and V, shown in Figures 5b and 5c, respectively. N changes `(x + y)` to `x + y`, while V further swaps the order of the two variable declarations (changed from `char x; int y;` to `int y; char x`) on top of N. ChatGPT produces the code listings in Figures 6b and 6c, respectively.

ECM tests the two pairs of mutants, (M, N) and (M, V). It reveals the pair (M, V) producing semantically different codes in their answers, meaning V is an adversarial example perturbed from M as perceived by ChatGPT.

To create a faulty mutant, we aim to insert all three errors related to the error list shown in Figure 3 into the mutant M one after another and judge ChatGPT's reply after each insertion. Figure 7b shows the result of inserting two errors into mutant M, resulting in a faulty mutant M' that ChatGPT produces the code the same as the one in the answer for M. However, upon inserting yet another error `x + y =`, ChatGPT can produce a correct result similar to the original reference solution.

Although mutant M' can fool ChatGPT and build up students' programming abilities, students cannot benefit from using the AI tool directly. Recall that mutant V is M's equivalent mutant that produces mutant V'. So, while injecting each error into M, ECM also injects the same error into V. Thus, when M' is produced, V is transformed into a new mutant V' (see Figure 7a).

ECM passes V' to ChatGPT, which produces the message ($Message_{V'}$) shown in Figure 8, and judges that V' resembles the reference solution so that $Message_{V'}$ is relevant to the exercise. $Message_{V'}$ contains a few sections: (a) a list of elaborations on the two errors, (b) a piece of returned code, (c) the suggested purpose of the code, (d) a reminder about the wrong usage of + operator, and (e) the remaining contents are not shown in the figure.



This code has a few syntax errors that prevent it from compiling correctly. Here's a list of the issues:

1. The input operator << is used instead of the output operator << in the `cin` statement. This should be changed to >>.
2. The `cout` statement that adds x and y together is missing the output operator <<. This should be changed to <<.

Here's the corrected code:

```
#include <iostream>
using namespace std;

int main()
{
    int y;
    char x;
    cout << "Enter a character and an integer: ";
    cin >> x >> y;
    cout << "Yes, or no, or maybe." << endl;
    cout << x + y << endl;
    return 0;
}
```

This code prompts the user to enter a character and an integer, reads them in using `cin`, adds them together, and outputs the result to the console using `cout`.

Note that adding a `char` and an `int` together using the `+` operator may not produce the expected result, as the `char` will be implicitly converted to its ASCII value before the addition. If you want to

Figure 8: $Message_{V'}$: ChatGPT output for mutant V'

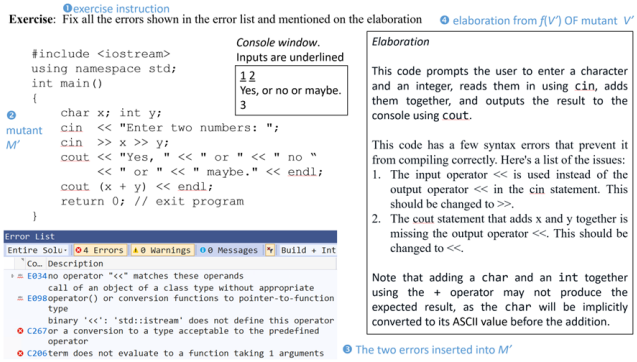


Figure 9: A ChatGPT-assisted exercise with elaboration

Finally, ECM constructs the ChatGPT-assisted exercise, shown in Figure 9, which consists of mutant M', sample input and output, an error list, $Message_{V'}$'s sections (a) and (c)–(d), and the exercise instruction. Also, $f(V')$ (see the code in Figure 4) is the created exercise's solution. (Generally, V' or $f(V')$ may serve as a solution.)

4.3. Results on All Exercises and Lessons Learned

We apply ECM to all the programming exercises (including the case presented in the last subsection) in the tutorial session in early March to explore its feasibility. Besides judging the codes, we assess the required time spent.

Exercise 1 requires students to create a C++ project in the IDE software, including project creation steps, a project file, and a reference to teaching materials. When we input the text of Exercise 1 into ChatGPT, the returned message provides a more abstract list of steps than Exercise 1. So, we skip applying ECM to Exercise 1.

Exercise 2 asks students to declare variables in various data types, perform arithmetic calculations and assignments, output results to the Console, and correct some syntax errors. It consists of several sub-questions, one of which is the exercise we have walked through in Section 4.2. We find that ChatGPT can directly answer simple questions in Exercise 2. We attempt to generate equivalent mutants for these simple questions but fail to find a pair of required equivalent mutants in 10 attempts. More complex questions like those in Section 4.2 can produce a ChatGPT-assisted exercise within 30 minutes.

Exercise 3 guides students to create a program to find the area of a rectangle in specific input and output formats. It consists of 11 steps, including how to analyze the problem and work out a solution plan. We input the question texts of Exercise 3 into ChatGPT. The tool produces correct solutions but not in the required input and output (I/O) formats. We input the original solution of Exercise 3 into ChatGPT to get the elaboration on the solution and extract the elaboration on the I/O format to construct a ChatGPT-assisted exercise within 10 minutes. (We cannot produce a ChatGPT-assisted exercise for Exercise 3 once the word "area" or the concept of computing the product of height and width appears in the text to ChatGPT.)

Exercise 4 refers to the Body Mass Index (BMI) calculation in the teaching materials not included in the exercise and requires students to display the result in a specific output format. We remove all the textual indications about BMI from the text and refer students to another teaching material about BMI. ChatGPT produces a misleading program. We then rename the word "tut2" (a folder name) into "bmi", and the tool produces a correct program with an elaboration of steps. We can compose a ChatGPT-assisted exercise within 10 minutes.

Exercise 5 states to write a program to convert an input number of seconds into the clock time format. ChatGPT returns the correct code. Revising the question to "Write a program to show the clock time based on the given input" can fool ChatGPT into suggesting a wrong code, while if we append "(in seconds)" to the revised question, ChatGPT suggests a correct code. We use the pair to construct a ChatGPT-assisted exercise, which takes 10 minutes to find the mutant pairs. ChatGPT generates an elaboration text without any code if we start from the reference solution instead. Like Exercise 3, we use the revised question and this elaborative text to create a ChatGPT-assisted exercise.

5. Summary with Related and Future Work

Our case study shows that the preliminary ECM version applies to most but not all of the exercises in a tutorial session. When successful, a ChatGPT-assisted exercise is often created within 10 minutes. Inputting the original reference solution of the exercise to the AI tool, the elaboration provided by ChatGPT can be integrated into the question text to enhance the exercise and guide students. Finding mutant pairs for more complex exercises is feasible. It requires creative ideas and insights into where the AI tool may encounter adversarial examples. It requires both domain knowledge in education and an understanding of adversarial attacks of AI models like ChatGPT. A reliable test oracle is essential in ECM. Online AI tools tend to evolve continually, and we find ChatGPT also behaves differently over time. Indeed, in May, we found ChatGPT has resolved some misleading answers it suggested in March. Hence, reproducing the same results in class sessions can be challenging.

Using a solver approach, Verifix [1] generates a corrected solution for student submission that contains errors. Clef [6] trains a model to return a repaired version of a code submission as feedback. Many blogs presenting the use of ChatGPT to fix the code are published [7]. Our ECM primarily produces exercise questions with elaborative descriptions in addition to answers through finding equivalent mutant pairs, in which one mutant is an adversarial example of the AI model and the other in the pair produces a solution.

In the future, we will refine ECM to help students easily kick-start their programming learning journeys. We will study methods for generating required mutants and developing new and repeatable adversarial-example approaches to synthesizing programming exercises that effectively leverage AI-based technologies like ChatGPT.

Acknowledgments

This paper is supported in part by the City University of Hong Kong (project no. 9678180).

References

[1] U. Z. Ahmed, Z. Fan, J. Yi, O.I. Al-Bataineh, A. Roychoudhury, "Verifix: Verified Repair of Programming Assignments," *ACM Trans. Soft. Engg. and Method.* 31(4): Article 71, 31 pages, 2022.

[2] T. Y. Chen, S. C. Cheung, S. M. Yiu, "Metamorphic testing: A new approach for generating next test cases," *Technical Report HKUST-CS98-01*, Department of Computer Science, The Hong Kong University of Science and Technology, Hong Kong, 1998. arXiv:2002.12543.

[3] I. J. Goodfellow, J. Shlens, C. Szegedy, "Explaining and harnessing adversarial examples.", In *ICLR 2015*, 2015. arXiv:1412.6572

[4] OpenAI, *ChatGPT*, URL: https://openai.com/blog/chatgpt

[5] R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "Hints on test data selection: Help for the practicing programmer," *IEEE Computer*, 11(4):34–41. April 1978.

[6] J. Zhang, D. Li, J. C. Kolsar, H. Shi, R. Piskac, "Automated feedback generation for competition-level code," In *ASE 2022*, Article 13, 13 pages, 2022.

[7] Lablab.ai, "How to easily improve your coding skills with ChatGPT," URL: https://lablab.ai/t/chatgpt-tutorial-how-to-easily-improve-your-coding-skills-with-chatgpt