

Few-Shot Prompting を用いた言語系生成 AI による プログラミング演習問題の自動生成手法の検討

田中 英武 前田 悠翔 井垣 宏

プログラミング演習では、学生は与えられた問題を実装することでスキルを獲得していく。そのため、学生それぞれの理解度や技術スキルに応じた多様な演習問題が用意できることが望ましい。そのため、プログラミング演習における教員の作問負荷は非常に大きいものとなっている。そこで本研究では、自然な文章やテキストの自動生成を可能とする生成 AI 技術の一分野である言語系生成 AI を用いて作問を自動的に行う手法を検討する。本研究で提案する自動作問システムは、教員が学生に修得してほしいメソッドや文法要素と問題の難易度をパラメータとして入力することで、教員の意図に沿った問題を効率よく作成することを目指している。実際にケーススタディとして、本研究で提案する自動作問システムを使用し、既存の演習問題の題材をベースに新しく 40 の演習問題の生成を試みた結果、33 問の演習問題を作成できたことを確認した。

1 はじめに

プログラミング演習と呼ばれる授業は一般に、少数の教員やアシスタントが数十名から数百名程度の学生に対してプログラミング演習問題を提示して実装させ、問題ごとの学習内容の解説や説明を教員が行う形式で実施される [10]。

著者らの所属する大学で実施されているプログラミング演習では、授業回ごとに数問から 10 問程度、全 14 回の授業で 100 問程度の演習問題が学生に提示される。教員は各問題について、仕様や実行例、解答例コードや必要に応じて単体テストのためのテストコードなどを用意する必要がある [4]。さらに、理解度が低い学生や逆に早く実装が完了した学生向けの個別の演習問題の提供なども考えると、教員にかかる問題作成（以降作問と呼ぶ）負荷は非常に大きなものとなることが予想される。

そこで本研究では OpenAI 社の開発した ChatGPT^{†1} を利用した自動作問手法について検討する。以降ではまず 2 章において、プログラミング演習で学生に提示される演習問題の形式や作問時の課題、言語系生成 AI がプログラミング教育に及ぼす影響について述べる。その後、3 において提案する自動作問システムのフローについて詳述し、ケーススタディとして実施した作問の流れと結果を 4 章においてまとめ、5 章で結果についての考察を行う。

2 準備

2.1 プログラミング演習問題の形式

プログラミング演習問題は仕様と実行例から構成されていることが多い。学生は以降に示すような仕様と実行例を読み、指示に従ってソースコードを実装し、提出する。

プログラミング演習問題の仕様は、学生がどのようなファイルを作成する必要があるか、実装するソースコードが満たすべき条件は何かといった情報を含んでいる。以下に著者らの所属する大学で実施されている Java 言語を対象としたプログラミング演習で提示

* An Investigation into the Automatic Generation of Programming Exercise Problems Using Large Language Models with Few-Shot Prompting
This is an unrefereed paper. Copyrights belong to the Author(s).

Hidetake Tanaka, Haruto Maeda, Hiroshi Igaki, 大阪工業大学情報科学部, Osaka Institute of Technology.

†1 <https://chat.openai.com/>

されている演習問題の仕様例を示す。この問題では、`java.util.Scanner` クラスの `next` メソッドと `nextLine` メソッドを利用したプログラム `Work45.java` を「自己紹介」という題材で実装することを目的としている。

例：プログラミング演習問題の仕様

問題 4-5 自己紹介

- `Scanner` クラスの `next` メソッドと `nextLine` メソッドを利用して、自己紹介する内容を表示するプログラムを作成しましょう。

仕様

次のクラスを作成します。

1. `Work45` クラス

* 自己紹介【イニシャル、年齢、好きなプログラミング言語】の内容を表示するプログラムを作成しましょう。

* 実行例と同じになるように `Work45.java` を完成させてください。

* このプログラムは以下の手順で入出力されます。

a. イニシャルと年齢の入力を一行（空白区切り）で受け付けます。

b. 好きなプログラミング言語は別の一行で受け付けます。

c. イニシャルと年齢、好きなプログラミング言語を別々に表示します。

[注意]`Scanner` クラスの `next` メソッドと `nextLine` メソッドを混在させるとバグが生じやすいため、注意しましょう。実装の方針を以下に示します。

* イニシャルと年齢を `next` メソッド、好きなプログラミング言語を `nextLine` メソッドで受け付けましょう。

* この場合、イニシャルと年齢を受け付けた `next` メソッドの後に、何も受け取らない `nextLine` メソッドを挿入するとうまくいきます。

実行例は対象のプログラムを実行した際の入出力を示す。ターミナル画面での入出力に付け加えて、どれがユーザによる入力であるかを示す記号が指示されていることも多い。以下に示す実行例の場合、`<` という記号が右側にある行はユーザによる入力を示している。

通常、教員はプログラミング演習問題ごとにこの仕様、実行例と解答例コードを用意しておく必要がある。ここで解答例コードとは与えられた仕様、実行例を満たすソースコードを指し、教員やアシスタントが学生サポートを行う際に参考にされる。

例：プログラミング演習問題の実行例

実行例は下記のとおりです。右に`<`という記号がある行はユーザによる入力を示しています。

```
$ java Work45
```

```
イニシャルと年齢を入力してください
```

```
t.k 20 <
```

```
好きなプログラミング言語を入力してください
```

```
Java <
```

```
イニシャルは t.k です
```

```
年齢は 20 歳です
```

```
好きなプログラミング言語は Java です
```

```
$
```

2.2 プログラミング演習のための作問における課題

プログラミング演習において、様々な学習内容や学生の技術レベルに対応した多くのプログラミング演習問題を作成することは教員にとって困難な課題である。そのため、従来より教員の作問を支援する様々な仕組みが検討されている。

山本ら[7]は作問の支援を目的として既存のプログラミング演習問題と類似している問題を異なるデータセットから検索する手法を提案している。宮澤[9]は標準規格に準拠したプログラミング演習問題を作成する出題モジュールを開発している。宮澤の手法では、短冊形と呼ばれる形式のプログラミング演習問題を一定のフォーマットに従って出力することができる。

システムによる作問の多くは短冊形に代表される穴埋め形式や一定のフォーマットのみを対象としている。これはプログラミング演習における仕様を学生が理解しやすい形で柔軟に作成することが困難であるためである。そのため、学生同士で仕様の作成や解答例コードの作成を行うシステムの提案も行われている[5]。この種のシステムでは、作問に学生自身が作成に関与することで、より多様で自由度が高い問題の作成が可能となる。学生同士による作問はその教育的な側面も注目されているが、作成される成果物の品質のばらつきが大きいという課題があることも知られている[1]。

そこで昨今では、言語系生成 AI を利用して問題やコードの説明を作成する手法に注目が集まっている[6]。

2.3 言語系生成 AI とコード生成

言語系生成 AI とは、大規模言語モデルに自然言語処理技術を応用して文章を生成することができる人工知能 (AI) プログラムの一種であり、代表的なも

```

1 import java.util.Scanner;
2
3 public class Work45 {
4     public static void main(String[] args) {
5         Scanner scanner = new Scanner(System.in);
6         System.out.println("イニシャルと年齢を入力してください");
7         String initial = scanner.next();
8         int age = scanner.nextInt();
9         scanner.nextLine();
10        System.out.println("好きなプログラミング言語を入力してください");
11        String favoriteLanguage = scanner.nextLine();
12        System.out.println("イニシャルは" + initial + "です");
13        System.out.println("年齢は" + age + "歳です");
14        System.out.println("好きなプログラミング言語は" + favoriteLanguage + "です");
15        scanner.close();
16    }
17 }

```

図 1 ChatGPT が生成したコード（コメント及び改行の一部省略）

のとして OpenAI 社の開発した ChatGPT が広く利用されつつある。

言語系生成 AI を利用することで、ユーザは自然言語による入力（プロンプト）を通じて多様なタスクの実施を AI に指示し、結果を文章として得ることが可能となる。ここで ChatGPT を始めとする多くの言語系生成 AI は、ユーザの入力した文章にもとづいてコードを実装することが可能であることも知られている。

例えば、2.1 節で示した仕様及び実行例を ChatGPT^{†2} にプロンプトとして入力すると、図 1 に示すようなコードが出力される。このコードは正常にコンパイル・実行が可能であり、与えられた仕様及び実行例と対応づいていることが確認できる。

鈴木ら [8] は大学 1 年生を対象とする Python によるオブジェクト指向プログラミングの基礎及び応用の 2 つの授業で出題されたプログラミング演習問題に ChatGPT-3.5 及び ChatGPT-4 を適用する実験を行っている。結果として ChatGPT-3.5 は基礎授業における問題の 9 割以上で正解し、ChatGPT-4 は応用授業にも十分に合格することが示されている。

AI によるコード生成が教育に及ぼす影響について、Becker ら [2] は特にプログラミング初学者向け授業への影響について述べている。そのなかで、言語系生成

AI が教員の代わりにコードを説明したり、より高度なアルゴリズムの理解を助けたりするのに役立つといったメリットと学生らによる不正行為やコードの再利用についての（著作権、ライセンス等の倫理的な）問題、AI が生成するコードが初学者にとって適切で、安全であるとは限らない点などについてのデメリットがあることが示されている。

2.4 言語系生成 AI による作問支援とその課題

Sarsa ら [6] は Python のプログラミング演習を対象とし、プログラミング演習問題の作成及びコードの解説を OpenAI 社の Codex モデル^{†3} を利用して実施する手法を提案している。Sarsa らの演習問題作成手法では、既存の仕様、解答例コード、テストケースと function や cars, parameters といったキーワードをプロンプトとして OpenAI Codex に与え、新しい演習問題を生成している。

大規模言語モデルで学習済みの言語系生成 AI にタスクの指示のみをプロンプトとして与える方法を Zero-Shot prompting と呼び、タスクの指示にあわせて、出力例をプロンプトとして与える方法を Few-Shot prompting と呼ぶ [3]。Sarsa らの手法もこの Few-Shot prompting に近い方法でプロンプトを作成し、演習問題の生成を行っている。

本研究では Sarsa らの手法を参考に、著者らの所

^{†2} ChatGPT August 3 Version

^{†3} <https://openai.com/blog/openai-codex>

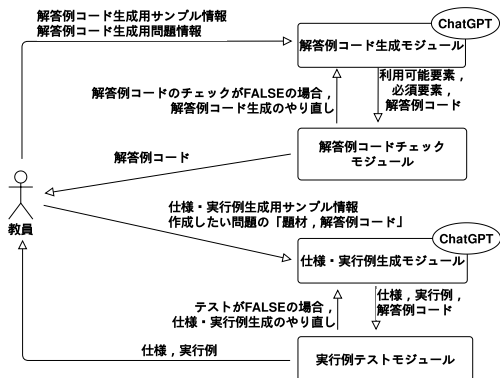


図 2 ChatGPT を用いた作問フロー

属する大学で実施している Java 言語向けプログラミング演習のための問題を Few-Shot prompting を用いて言語系生成 AI により作成する手法を検討する。

3 Few-Shot Prompting を用いた ChatGPT による自動作問システムの検討

本稿で提案する自動作問システムでは作問を「解答例コード生成」と「仕様・実行例生成」の 2 段階で実施する。教員はまず作成したいプログラミング演習問題の解答例コードを作成し、次にその解答例コードから仕様や実行例を生成する。

以降では図 2 に示す作問フローについて詳述する。

3.1 解答例コード生成

解答例コードの生成は解答例コード生成モジュール及び解答例コードチェックモジュールの 2 つのモジュールで実施される。

解答例コード生成モジュール

このモジュールには Few-Shot prompting のための解答例コード生成用サンプル情報と解答例コードを作成したい演習問題情報が入力として与えられる。モジュール内では OpenAI 社の ChatGPT API (gpt-3.5-turbo) が利用されている。

解答例コード生成用サンプル情報には、具体例として完成された「クラス名, 題材, 利用可能要素, 必須要素, 解答例コード」が含まれる。サンプル情報の例

を付録 A の図 3 に示す。ここで「題材」は 2.1 節で述べた「自己紹介」のような対象の演習問題の特性を示す一行程度の文章で構成されており、「利用可能要素」はその問題で利用することが可能な文法要素やメソッドを、「必須要素」はその問題で必ず使わなければならない文法要素やメソッドを指定する。

解答例コード生成用問題情報には作成したい問題の「クラス名, 題材, 利用可能要素, 必須要素, 難易度」が含まれる。「難易度」は行数と循環的複雑度を利用して指定する。循環的複雑度 (以下 CCN とする) とは、ソフトウェアの複雑度を表す指標である。著者らの所属する大学で実施されている過去の演習問題では、特定の授業回内で難易度の低い問題から高い問題まで様々な問題が出題される。そのため、作問時にも難易度を指定できるようにすることで、より多様な問題を作成できるようになることが期待される。本研究で指定する難易度については、過去の演習問題を参考に表 1 に示すように 4 段階で構成されており、1 が最も単純なコードで、4 が最も複雑なコードとなる。なお、ここで行数はコメント行や空行も含むソースコード行数とする。

表 1 難易度と行数・CCN の対応

難易度	行数	CCN
1	10 行程度	2
2	20 行程度	3
3	30 行程度	4
4	40 行程度	5

以上の情報を解答例コード生成モジュールに入力として与えることで、解答例コードを出力として得ることができる。

解答例コードチェックモジュール

解答例コードのチェックモジュールでは、「利用可能要素, 必須要素, 解答例コード」を受け取り、下記のチェックを行う。なお、このモジュールでは ChatGPT API は利用しない。

- C1. 解答例コードに含まれているすべてのメソッド及び文法要素が必須要素あるいは利用可能要素に含まれているか

C2. すべての必須要素が解答例コードに含まれているか

C1 及び C2 のいずれかの結果が FALSE の場合、解答例コード生成モジュールに解答例コードを再生成するよう指示を送る。いずれも TRUE の場合は解答例コードを教員に渡す。

3.2 仕様・実行例生成

仕様・実行例の生成は仕様・実行例生成モジュール及び実行例テストモジュールの2つのモジュールで実施される。解答例コードの生成と同様に、仕様・実行例生成モジュールのみ ChatGPT API である GPT-3.5(gpt-3.5-turbo) を利用する。以降各モジュールの入出力及び処理内容について述べる。

仕様・実行例生成モジュール

仕様・実行例の生成モジュールでは、生成用サンプル情報として、完成された「題材、仕様、実行例、解答例コード」と、作成したい問題の「題材、解答例コード」を入力として与える。サンプル情報の例を付録 A の図 3 に示す。このモジュールに以上の情報を入力として与えることで、仕様と実行例を出力する。

実行例テストモジュール

実行例テストモジュールでは、仕様・実行例生成モジュールによって生成された実行例から JUnit 単体テストを生成し、解答例コード生成モジュールによって生成された解答例コードのテストを実施する。ここで、すべての単体テストが正常に実行できれば、仕様・実行例を教員に返却し、一つでも Fail した場合は仕様・実行例生成モジュールに戻ってやり直す。

以上の処理が実施されることで、指定の題材と演習問題のフォーマットに従った解答例コード及び仕様・実行例を教員は受け取ることができる。以降ではケーススタディとして、著者らの所属する大学で実施した Java 言語の過去の題材から実際に新しい演習問題を生成することができるか検証を行う。

4 ケーススタディ

自動作問システムにより、実際にプログラミング演習問題をどの程度生成可能かを検証するため、ケーススタディを実施した。今回のケーススタディでは、

著者らの所属する大学で開講されている Java 演習と呼ばれるプログラミング演習の既存問題 10 問の題材から、新しい問題を生成できるか検証した。以降ではケーススタディの手順と問題の生成結果について述べる。

自動作問システムは 3 節で述べたとおり、解答例コード生成及び仕様・実行例生成の 2 段階に分けて入力を行う。ケーススタディにおける各入力内容を以下に示す。

4.1 解答例コード生成モジュールへの入力

付録 A の図 3 に示す解答例コード生成用サンプル情報のうち、「クラス名、題材、利用可能要素、必須要素、解答例コード」を解答例コード生成モジュールに入力する。さらに作成対象の問題情報の入力項目を表 2 に示す。ケーススタディでは Java 演習の過去の演習問題を参考に Work41 から Work4a までの 10 問分の題材を用意した。さらに各問題について、必ず含めるべき文法要素やメソッド名を必須要素として定義した。表 3 に示す利用可能要素については、Work41 生成時点のもので、Work42 以降については Work41 生成時点の利用可能要素に必須要素を追加したものとす。例えば Work42 の場合、表 3 に示す利用可能要素に Work41 の fill を追加したものが Work42 の利用可能要素となる。これら 10 問それぞれについて難易度 1~4 を指定し、合計 40 問分の解答例コードの生成を行う。

解答例コード生成後は解答例コードチェックモジュールで生成された解答例コードのチェックを行う。チェック項目は 3.1 節で述べたとおりだが、False の場合のやり直し回数の上限を 10 回とし、10 回を超えた場合は失敗として、以降の生成を行わないものとする。

4.2 仕様・実行例生成モジュールへの入力

付録 A の図 3 に示す解答例コード生成用サンプル情報のうち、「題材、仕様、実行例、解答例コード」を入力する。作成対象問題については、表 2 に示す「題材」と解答例コード生成モジュールで生成された「解答例コード」を入力する。

次に仕様・実行例生成モジュールにより得られた

表 2 作成対象問題の入力項目

クラス名	題材	必須要素
Work41	配列の表示	fill
Work42	偶数/奇数の数を表示	for, if
Work43	最大値/最小値/平均値の表示	for, if
Work44	出現順位×要素値の最小値/最大値の表示	for, if
Work45	自己紹介	next, nextLine
Work46	集合間での要素総当たりの割り算	for
Work47	ソート（並べ替え）	for, if
Work48	三角形の構成判定	for, if
Work49	健康チェック	format
Work4a	配列の要素同士の演算	next

表 3 利用可能要素

println	next
if	nextLine
for	close
while	parseInt
switch	length
equals	sort
max	

「実行例」から単体テストを生成し、「解答例コード」に対してテストを実施する。このとき、単体テストが1つでも Fail すると仕様・実行例生成の最初からやり直し、やり直し回数が5回を超えた場合は解答例コードの生成からやり直し。また、やり直し回数が10回を超えた場合は仕様・実行例生成を失敗とみなす。

4.3 結果

10個の題材それぞれについて4種類の難易度で問題の作成を行った。解答例コード、実行例テストそれぞれについてチェックを行っているため、基準を満たす解答例コードを生成するのに要した回数を表4に、仕様・実行例を生成するのに要した回数を表5に示す。表では縦軸はクラス名、横軸は難易度を表している。

解答例コードの生成において、10回生成しても解答例コードが基準を満たさなかった場合は10+と表記している。40問中36問が生成できており、Work45とWork49を除く8問はすべての難易度において生成できている。

仕様・実行例の生成において、解答例コードを10回以内に生成できず仕様・実行例の生成に移行できなかった場合は×、10回生成しても実行例が基準を満たさなかった場合は10+と表記している。解答例コードの生成に成功した36問のうち、基準を満たす実行例を生成できたのは33問であった。Work44, Work47は解答例コード生成時は少ない回数になっているが、仕様・実行例生成時には多くの試行回数を要

していることが分かる。

表 4 基準を満たす解答例コードの生成に要した回数

	1	2	3	4
Work41	1	2	4	3
Work42	1	1	1	1
Work43	1	1	1	3
Work44	1	1	1	1
Work45	4	10+	10+	10+
Work46	1	1	1	1
Work47	2	1	1	1
Work48	2	1	1	2
Work49	2	10+	9	3
Work4a	4	1	4	7

生成された解答例コードの行数を表6に、CCNを表7に示す。いずれの表も、解答例コードを10回以内に生成できなかったものについては×と表記している。指定した難易度通りに生成できている解答例コードについてはアンダーラインを付与している。ただし、行数については指定した難易度の行数との差が10以内、CCNについては指定した難易度のCCNとの差が1以内であれば誤差として許容している。生成された解答例コードのうち、行数が難易度の条件を満たしているものは36問中18問であった。また、CCNが難易度の条件を満たしているものは36問中15問であった。

表 5 基準を満たす仕様・実行例の生成に要した回数

	1	2	3	4
Work41	1	1	1	1
Work42	1	1	1	1
Work43	1	1	1	1
Work44	5	6	10+	10+
Work45	1	×	×	×
Work46	2	1	1	6
Work47	1	1	5	10+
Work48	1	1	1	1
Work49	1	×	2	7
Work4a	1	1	1	1

表 6 生成された解答例コードの行数

	1	2	3	4
Work41	22	<u>13</u>	20	14
Work42	26	<u>24</u>	19	25
Work43	48	36	<u>36</u>	<u>37</u>
Work44	23	35	<u>29</u>	<u>39</u>
Work45	32	×	×	×
Work46	30	<u>15</u>	17	13
Work47	<u>21</u>	<u>21</u>	<u>34</u>	<u>31</u>
Work48	24	<u>20</u>	<u>29</u>	<u>38</u>
Work49	37	×	<u>33</u>	<u>31</u>
Work4a	23	<u>24</u>	<u>21</u>	25

また、自動作問システムにより生成された問題のうち、クラス名が Work41、題材を「配列の表示」、難易度指定 2 の場合の生成例を付録の図 4 に示す。必須要素や利用可能要素の条件を満たしており、難易度も行数が 13 行、CCN が 2 と指定通りの問題が生成できている。

5 考察

ケーススタディで実施した解答例コード生成及び仕様・実行例生成の結果と本手法の制限と今後の課題について述べる。

表 7 生成された解答例コードの CCN

	1	2	3	4
Work41	<u>2</u>	<u>2</u>	2	2
Work42	<u>3</u>	<u>3</u>	<u>3</u>	3
Work43	7	5	<u>5</u>	<u>5</u>
Work44	4	5	7	<u>6</u>
Work45	<u>3</u>	×	×	×
Work46	5	<u>4</u>	<u>4</u>	3
Work47	5	5	6	<u>6</u>
Work48	5	5	6	13
Work49	10	×	6	8
Work4a	<u>3</u>	<u>3</u>	<u>3</u>	3

5.1 解答例コード生成

表 4 より、Work45 は難易度 2~4 において解答例コードの生成に失敗している。この問題は、必須要素に `java.util.Scanner` クラスの `next` メソッドと `nextLine` メソッドの両方を含んでいるが、生成された解答例コードのほとんどにおいて `next` メソッドが使用されていない。これは、言語系生成 AI の学習データに含まれるコードに `next` と `nextLine` 両方を含むものが少ないというバイアスによるものと考えられる。

また、Work49 では難易度 2 の解答例コードの生成に失敗している。生成時のログを確認すると、利用可能要素や必須要素に含まれていない `parseDouble` メソッドや `nextDouble` メソッドを含む解答例コードが複数回生成されていた。これは、言語系生成 AI が「健康チェック」という題材から BMI 値の算出を想定し、入力として与える身長や体重といった数値を `double` 型で処理する必要があると判断したためであると考えられる。

5.2 仕様・実行例生成

Work44 は難易度 3 と難易度 4 において基準を満たす実行例の生成に失敗している。Work44 は配列を扱う問題であるが、生成された問題を確認すると、実行例の表示内容が実際の実行結果と異なるものや、配列の要素数よりも大きい要素番号が使用されて範囲外アクセスとなってしまうものなどがあつた。

また、Work47 は難易度 4 において基準を満たす実行例の生成に失敗している。Work47 はソートを題材とした問題であるが、生成された問題を確認すると、実行例でソートが行われないうま出力されているものや、入力回数が実行例と解答例コードで一致していないものが見られた。生成された解答例コードを確認すると、配列の要素数を最初にユーザに入力させ、その後各要素の値を入力させるという形式となっており、当初に入力した値に応じて配列のサイズや要素が変わるプログラムとなっていた。

このことから、ユーザ入力によって、出力が大幅に影響を受けるようなケースに置いて、生成 AI による実行例生成が困難となる可能性があると考えられる。

5.3 難易度設定

生成された解答例コードの行数・CCN を確認すると、結果として半分程度指定した難易度の範囲に収まっているが、同じクラスであれば同程度の行数及び CCN のコードが生成されているケースが多いことも確認された。そのため、解答例コードの生成において、難易度の指定はそこまで考慮されていない可能性がある。今後、より複雑な問題の生成などを通じて難易度指定の有効性を検証していきたい。

5.4 提案手法における制限と今後の課題

プログラミング演習の既存問題で使用された 10 個の題材を利用し、難易度の異なる 40 問の演習問題を提案手法に基づいて作成した。実際のプログラミング演習で本稿で提案する自動作問システムを利用するにあたって、現時点での制限と今後の課題について述べる。

必須要素と利用可能要素の制限

本稿で述べる自動作問システムでは入力として演習問題ごとに利用可能な文法要素やメソッド、学習してほしい必須要素を指定する。特にメソッドの指定において、異なるクラスに属する同一メソッド名の考慮ができていない。例えば、nextInt メソッドは java.util.Scanner クラスにも java.util.Random クラスにも属するメソッドであり、単に nextInt メソッドを指定しただけではどちらのクラスに属するメソッド

かを区別できない。これらの区別を行うためには、必須要素や利用可能要素を指定する際にクラス情報も含めて指定し、メソッドが属するクラスを考慮した解答例コードチェックを行う仕様に変更する必要がある。

また、特に必須要素において、より柔軟な指定に対応できていない。現状の解答例コードチェックモジュールは、必須要素に指定されたすべての要素を含む場合に True を返す仕様となっている。しかし、この仕様では「next メソッドか nextLine メソッドのどちらかを利用してほしい」といったケースに対応できない。このようなケースに対応するために、必須要素の指定方法や解答例コードチェックの基準を見直す必要がある。

解答例コードの規模の制限

今回のケーススタディにおいて、自動作問システムが生成する問題は単一クラスのみからなるものを想定している。実際のプログラミング演習では複数クラスの作成を行う問題も数多く存在するため、今後は複数クラスを対象とした仕様・実行例・解答例コード・単体テストの生成についても検証を行っていきたい。

生成される仕様の定性的な評価に関する制限

本稿で実施したケーススタディでは生成された実行例と解答例コードの対応が取れているかしか検証されていない。そのため、作成された仕様の文章が演習問題として適切であるかは評価できていない。実際、付録 A の図 4 の例でもわかる通り、仕様の文章が解答例コードにもとづく具体的な変数名や処理手順を挙げた説明文になっている。この種の仕様は難易度の低い初期の問題においては望ましいが、難易度が高い後半の問題では必ずしも望ましいとは言えない。そのため、実際のプログラミング演習で使用されるような抽象度の高い説明文を生成する方法を検討する必要がある。また、出題する教員の視点で仕様の品質を評価する手法を確立できていないため、仕様の評価手法についても検討していきたい。

6 おわりに

本稿では、言語系生成 AI を用いたプログラミング演習問題の自動生成について提案した。また、実際に自動作問システムを試作し、難易度や必須要素、利用

可能要素を指定した作問を実施した。その結果、指定した内容に基づいた多くの問題を作成できることがわかった。今後は複数クラスを対象とした作問の検証や、仕様の評価手法について検討していきたい。

参考文献

- [1] Abdi, S., Khosravi, H., Sadiq, S., and Demartini, G.: Evaluating the Quality of Learning Resources: A Learnersourcing Approach, *IEEE Transactions on Learning Technologies*, Vol. 14, No. 1(2021), pp. 81–92.
- [2] Becker, B. A., Denny, P., Finnie-Ansley, J., Luxton-Reilly, A., Prather, J., and Santos, E. A.: Programming Is Hard - Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation, *Proceedings of the 54th ACM Technical Symposium on Computer Science Education, SIGCSE 2023*, Vol. 1, 2023, pp. 500–506.
- [3] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D.: Language Models are Few-Shot Learners, *CoRR*, Vol. abs/2005.14165(2020), pp. 1–75.
- [4] Lobb, R. and Harlow, J.: Coderunner: A Tool for Assessing Computer Programming Skills, *ACM Inroads*, Vol. 7, No. 1(2016), pp. 47–51.
- [5] Pirttinen, N., Kangas, V., Nikkarinen, I., Nygren, H., Leinonen, J., and Hellas, A.: Crowdsourcing Programming Assignments with CrowdSorcerer, *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE 2018*, 2018, pp. 326–331.
- [6] Sarsa, S., Denny, P., Hellas, A., and Leinonen, J.: Automatic Generation of Programming Exercises and Code Explanations Using Large Language Models, *Proceedings of the 2022 ACM Conference on International Computing Education Research, ICER '22*, Vol. 1, 2022, pp. 27–43.
- [7] 山本大貴, 松尾春紀, 沖野健太郎, 近藤将成, 亀井靖高, 鶴林尚靖: プログラミング作問支援に向けた類似問題検索手法の評価, *ソフトウェア工学の基礎ワークショップ論文集*, Vol. 28(2021), pp. 115–120.
- [8] 鈴木智也, 神谷年洋: ChatGPT によるプログラミング授業の課題の解答生成の評価, *信学技報*, Vol. 123, No. SS-123(2023), pp. 55–60.
- [9] 宮澤芳光: 標準規格に準拠したプログラミング問題とデータ活用問題の出題モジュールの評価, *人工知能学会全国大会論文集*, Vol. JSAI2023(2023), pp. 1R4OS10a04–1R4OS10a04.
- [10] 横原絵里奈, 藤原賢二, 井垣宏, 吉田則裕, 飯田元: 初学者向けプログラミング演習のための探索的プログラミング支援環境 Pockets の提案, *情報処理学会論文誌*, Vol. 57, No. 1(2016), pp. 236–247.

A 付録: Few-Shot Prompting のための サンプル情報と生成 AI による出力例

[クラス名] Work25
[題材] 文字列の比較
[必須要素] equals
[利用可能要素] close, next, println, nextLine, if, parseInt, equals
[仕様]

課題 2-5 文字列の比較

- 標準入力から文字列を 2 回読んで、同じ文字列かどうかを判別してみましょう。
- 下記仕様と実行例に従って 'Work25' クラスを持つ 'Work25.java' を作成しましょう。

仕様

次のクラスを作成します。

1. Work25 クラス (Work25.java)

- String 型の変数 input1, input2 を宣言します。
- input1, input2 には標準入力から読み込んだ文字列がそれぞれ入ります。
- input1, input2 が同じ文字列かどうかを判別し、等しいか、等しくないか (違うか) を実行例のように出力します。

[実行例]

```
'''sh
$ java Work25
hogehoge <-
hogehoge <-
1 番目と 2 番目の文字列は等しいです。
```

```
$ java Work25
java は簡単 <-
java は楽しい <-
1 番目と 2 番目の文字列は違います。
```

\$

'''

[解答例コード]

```
'''java
public class Work25 {
    public static void main(String[] args) {
        java.util.Scanner scanner = new java.util.Scanner(System.in);

        String str1 = scanner.nextLine();
        String str2 = scanner.nextLine();

        // 文字列比較
        if (str1.equals(str2)) {
            System.out.println("1 番目と 2 番目の文字列は等しいです。");
        } else {
            System.out.println("1 番目と 2 番目の文字列は違います。");
        }
        scanner.close();
    }
}
```

'''

図 3 Few-Shot prompting のための解答例生成及び仕様・実行例生成用サンプル情報

課題 4-1 配列の表示

- 配列に値を設定し、その値を表示するプログラムを作成してみましょう。
- 下記仕様と実行例に従って 'Work41' クラスを持つ 'Work41.java' を作成しましょう。

仕様

次のクラスを作成します。

1. Work41 クラス (Work41.java)

- int 型の配列 numbers を宣言します。要素数は 5 とします。
- numbers には全て 10 が入るように設定します。('java.util.Arrays.fill()' メソッドを使用します。)
- for 文を使用して、配列 numbers の各要素を 1 行ずつ表示します。

実行例

```
'''sh
$ java Work41
10
10
10
10
10
$
'''
```

解答例コード

```
'''java
public class Work41 {
    public static void main(String[] args) {
        int[] numbers = new int[5];

        // 配列に値を設定
        java.util.Arrays.fill(numbers, 10);

        // 配列の表示
        for (int i = 0; i < numbers.length; i++) {
            System.out.println(numbers[i]);
        }
    }
}
'''
```

図 4 自動作問システムにより生成された問題