日本語で記述された大学初年次プログラミング課題に対する ChatGPT の回答能力の調査

森畑 明昌

近年，ChatGPT をはじめとして，大規模言語モデルに基づく自動プログラミングツールが急速に発展をとげており，プログラミングの知識のない人にも手軽に利用できるものとなりつつある．このことにより，プログラミングを行う際に要求される知識と技能が大きく変わる可能性がある．本研究では，このような自動プログラミングツールの現状を調べるために，東京大学教養学部の初年次プログラミング教育で用いられている課題 40 件を ChatGPT に解かせてみた結果を報告する．特に，(1) ChatGPT はどの程度正解することができるのか，(2) ChatGPT が不得意な課題類型はあるのか，(3) ChatGPT の回答と学生の回答には教員から見て差異があるのか，の 3 つを主な問いとした．ChatGPT は 9 割弱のケースで正解プログラムを作成できた．しかし，全く正解することができなかった課題もあり，不得意な課題類型の存在を示唆した．同様の実験を行った既存研究に比べ，日本語の課題文章をそのまま入力としている点に特徴があり，情報系技術に疎い日本語話者を念頭に置いている．

1 はじめに

近年，大規模言語モデルに基づいた技術が急速な発展を遂げている．その中でも特に有名なのは OpenAI による ChatGPT [3] であり，自然言語による指示（プロンプトと呼ばれる）を行うことで，様々なタスクを実現することができる．

本稿では ChatGPT のプログラミング能力について論じる．一例を図 1 に示す．プロンプトでの指示に従ったプログラムを作成していることが見て取れる．このように，大規模言語モデルによるプログラム生成は，それ以前のプログラム自動合成技術 [8] とは異なり，自然言語からかなり精度良くプログラムを構成できる．そのため，特にプログラミングや情報系技術に疎い人にとっては，プログラムを作成するための障壁がかなり小さくなると期待されている．

大規模言語モデルがどの程度のプログラムを作成できるのかは，近年盛んに調べられている [1, 2, 6, 9, 11, 12, 17]．にもかかわらず，依然としてその能力はそれほどはっきりしていない．これには複数の理由がある．まず，大規模言語モデルの能力は，学習に用いたデータや学習モデルのパラメタ数に大きく依存する．しかも，学習データ数を増やせば単調に能力が伸びるとも限らない [5]．また，成否は目標とするプログラムの複雑さ次第だが，人間の印象とは

以下の指示に従い Python プログラムを作成してください．3 つの数が与えられたとき，2 番目に大きい数を返す関数 ex04_3 を定義せよ．3 つの数は互いに相異なると仮定してよい．

```
def ex04_3(a, b, c):
    numbers = [a, b, c]
    numbers.sort()
    return numbers[1]
```

この関数では，与えられた 3 つの数をリスト numbers に格納し，sort メソッドを使って昇順にソートします．その後，ソートされたリストの 2 番目の要素を返します．これによって，2 番目に大きい数が求められます．

図 1: ChatGPT によるプログラミング例

Investigating ChatGPT's Capability to Answer Japanese Programming Tasks for Freshperson

Akimasa Morihata, 東京大学大学院総合文化研究科, Dept. of Arts and Sciences, the University of Tokyo.

必ずしも一致しない．例えば，プログラムの作成に比べ，プログラムの実行結果を正しく示すことのほうが苦手だとされている [2] が，これは人間の場合とは逆である．さらに，成否はプロンプトとしてどのような情報を与えるかにも依存する [2]．このような事情もあり，これまでの研究が報告している結果にはかなりのばらつきがある．いずれの報告でも「ある程度のプログラムを作成できる」ことは確認されているが，具体的に，例えば**標準的な大学生と比べてどの程度のレベルにあるか**ははっきりしない．

本稿では，東京大学教養学部の初年次プログラミング教育で用いられている課題 40 件を ChatGPT に解かせてみた結果を報告する．本研究では，プログラミングや情報系技術に疎い日本語話者にとって現実的な形で，どの程度のプログラムを作成できるかを調査することを目標としている．そのため，以下のような状況で実験を行った．

- 日本語のプロンプトで指示を行う．ChatGPT は英語に比べ日本語でのプロンプトでは能力が低下すると広く信じられている．これは，学習データの量が日本語と英語では大きく異なるだろうということを考えれば，合理的ではある．しかし，著者の知る限り，日本語でのプロンプトでのプログラム作成能力を調査した研究は存在しない．
- プロンプトでの特別な工夫は行わず，課題文書をそのまま用いる．ChatGPT はプロンプトを工夫することで能力が向上する．しかし，適切な工夫（例えば部分問題への分割や，反例の提示）には，本質的にプログラミングの技術や情報系技術についての理解を必要とする．これは，今回想定しているようなユーザの能力を超える．
- 無料で利用できる ChatGPT（ChatGPT 3.5）を利用する [†1]．これは，このようなツールを本格的に利用するわけでない一般ユーザであれば，無料のものを使うだろうと想定したためである．

この実験を通して本研究で明らかにしたい問は以下の 3 つである．

RQ1 ChatGPT はどの程度の課題に正解することができるのか？

RQ2 ChatGPT が不得意な課題類型はあるのか？

RQ3 ChatGPT の回答と学生の回答には教員から見て差異があるのか？

まず，**RQ1** に関して言えば，ChatGPT は 90%近い課題に正解することができた．これは事前での著者の予想よりも高く，また本学の標準的な大学 1 年生の水準と同等，ないしは上回っている．不正解は一部の課題に集中する傾向があり，不得意な課題類型の存在を示唆した（**RQ2**）．具体的には，普通とは異なるプログラミングを強いる課題（例えば互除法を使わずに最大公約数を求める）や，複数のことを同時に達成することを求める課題（例えば，多数の数値列に対し，各列の最小と最大を除いた残りの値の平均値を求め，その平均値の最大値を求める）などでの不正解が目立った．また，**RQ3** に関しては，ChatGPT の作成したプログラムは，単体で見ると比較的自然であるものの，複数の課題を通して見たときにはコーディングスタイルの一貫性に欠ける様子が確認できた．

以上の結果は，日本語でのプログラミング教育や，自動プログラミング技術を用いた一般ユーザ向けツールの作成などの際に参考になるのではないかと期待している．

2 実験の方法

2.1 ベンチマーク課題

本研究で実験対象としたのは，東京大学教養学部の 1・2 年生向け全学教養教育講義「アルゴリズム入門」で用いられている課題である．当該講義は，理科生にとってはいわゆる「準必修」という位置づけになっていることもあり，理科生 1 年生のおおむね半数以上，また文科生 1 年生のおおむね 1 割程度が履修している．

講義の目的はプログラミングを通して情報科学の基礎概念を学ぶことである．講義の前半はプログラミングの基礎である変数・関数・配列 [†2]・繰り返し・

†1 実際には，実験の手間を軽減するため，有料の API を介して実験を行っている．

†2 Python で列を表す構造はリストであるが，他のプログラミング言語を学ぶ際の利便性を考え，本講義ではあえて「配列」と呼んでいる．本稿でもこの方針に従う．

条件分岐などを学ぶ．講義の後半では，プログラムを作成しながら，アルゴリズム・計算量・再帰・疑似乱数・分割統治・数値誤差などを学ぶ．より詳しい設計と内容については森畑 [19] による記事を参考にされたい [†3]．

今回の実験には，学生の自習用として整備された課題 40 件を用いた．その内訳は表 1 のとおりである．各課題には講義の指定教科書 [20] との対応があり，ex0n-m という課題名は，教科書 n 章に対応した課題の m 件目であることを表す．なお，プログラミングの観点からは，教科書の 2 章は変数，3 章は関数，4 ～ 6 章は配列・繰り返し・条件分岐，7 章は再帰と擬似乱数，8 章は二分法や分割統治法などのやや高度なアルゴリズムを主な内容としている．

それぞれの課題は，特定の仕様を満たす Python プログラムの作成を求める．図 1 ですでに示したものもその一例である．すべての課題は PLAGS-UT システム [18] 上で自動評価が行われるようになっている．このため，原則として入出力関係が明確に定まるようになっており，受講生が自由な発想でプログラムを作成するようなものはない．自習に適した課題となるよう，問題文は（少なくとも人間にとっては）仕様が明確となるように記述している．特に，コーナーケース（例えば長さ 0 のリストが入力されるなど）は原則として問題文中で明示的に排除をしている．また，講義の趣旨に照らして，一部の課題ではプログラム作成の方針をある程度指示している．具体的な指示は「`math` ライブラリを使用せよ」「再帰関数を作成せよ」などである．

課題は `ipynb` 形式で配布されており，課題内容を markdown 形式で説明したセルと，回答プログラムを記入するセルからなる．回答プログラムを記入するセル中には事前に多少のプログラムが記述されており，その中の`...` を埋めることが目標となる．ほとんどの課題では，ごくわずかな自明な要素（必須ライブラリの `import` や作成すべき関数名など）しか事前には書かれておらず，課題文章でもそのことに特段の言及はない．しかし，プログラムの大部分がすでに指定されている課題もあり，そのような課題では明示的に「プログラム中の`...` を埋めよ」との指示がある．

2.2 ChatGPT への指示とその正しさの評価方針

ChatGPT に対しては，以下の 3 点を 1 つのプロンプトにまとめて提示した．

- 「以下の指示に従って Python プログラムを作成してください.」という 1 文
- 課題内容を説明するセルに記載された markdown 記法での文章すべて
- 課題が「プログラム中の`...` を埋めよ」と指示しているものについては，回答プログラム記入セルの内容

モデルとしては `gpt-3.5-turbo` を用いた．モデルの実行パラメタはすべてデフォルトのままとした．各課題について 10 回ずつ独立した対話を行い，10 通りの返答を得た．これらの実験は 2023 年 7 月 24 日から 26 日にかけて行った．

ChatGPT の返答に含まれるプログラムに対し，PLAGS-UT システムでの自動評価を参考としつつ，人手で正しさを検証した．正しいかどうかの判断基準は以下のとおりとした．

- 入出力関係が正しいかだけでなく，課題の指示に従っていることも求める．また，現実的な時間では明らかに実行が終了しないプログラムも正しくないとする．
- 返答中に正しいプログラムが含まれていれば，返答のそれ以外の部分に関わらず正しいとする．例えば，プログラムが正しければ，その実行例が間違っていてもよい．

3 実験結果

3.1 RQ1 についての分析

40 課題 × 10 回 = 400 回 の対話において，350 回で正しいプログラムが作成された．よって正答率は 350 / 400 = 87.5% であった．

図 2 に課題ごとの正答数の分布を示す．1 度も正解できなかった課題 2 件，2 度しか正解できなかった課題 1 件，そして半分程度（6 回）しか正解できなかっ

†3 当該記事の時代には Ruby を用いていたが，2018 年から Python を用いるようになった．とはいえ，講義の趣旨はほとんど変化していない．

表 1: 実験に用いた課題と教科書との対応

教科書	2 章	3 章	4 章	5 章	6 章	7 章	8 章
課題数	8	7	9	3	3	6	4

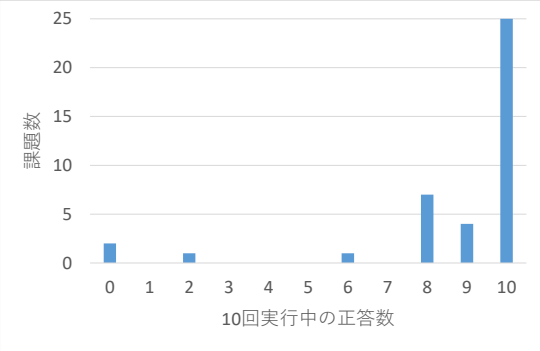

図 2: 課題ごとの正答数の分布

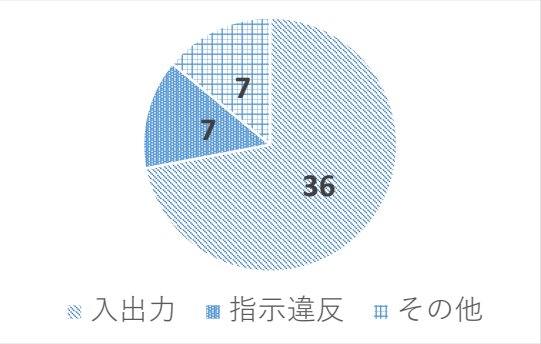

図 3: 誤答の分類.「入出力」はプログラムが入力に対して適切な出力を返せないこと,「指示違反」は入出力関係は正しいがそれ以外の指示を満たさないことを表す.

た課題 1 件を除いた 36 件については，ほとんど完璧に回答できていることが見て取れる．この観点からは，36 / 40 = 90% の課題に正答できていると言えよう．

この結果は,「アルゴリズム入門」の平均的な受講者と同等，ないしは上回っていると言える．確かに前半（特に 2・3 章）はかなり初歩的であり，正解するのも不思議はない．しかし，後半の課題に関しては，なかなか正解に至らない受講者も少なくない．これらのほとんどに正解するためには，講義内容をよく理解することが求められる．

3. 2　RQ2 についての分析

次に，誤答の内容について述べる．

図 3 に誤答の分類を示す．50 件の誤答の中で，36 件は得られたプログラムが入力に対して適切な出力を返せない状況であった．また，次に多かった誤答の類型は，作成すべきアルゴリズムが指示されていたにもかかわらずこれを守れていなかったものであった．それ以外の誤答の理由としては，プログラム作成の方針のみ示してプログラムは作成しなかった，動作に必要なライブラリを `import` しなかった，現実的な時間での実行が不可能だった，などであった．

以下，特に誤答の多かった 4 件の課題について詳細に述べる．

3. 2. 1　ex02_6

次に示す課題 ex02_6 は，10 回の実行中 6 回が正答，4 回が誤答であった．

> T 国では 9^3 円玉，9^2 円玉，9^1 円玉，9^0 円玉の 4 種類の硬貨を用いている．1000 円をおつりがなく使う硬貨が最小になるように（つまり金額の大きな硬貨をできるだけ沢山）支払うとき，支払いに用いる 9^3 円玉，9^2 円玉，9^1 円玉，9^0 円玉の枚数をそれぞれ変数 ex02_6_1，変数 ex02_6_2，変数 ex02_6_3，変数 ex02_6_4 に保持するプログラムを作成せよ．

4 件の誤答のうち 3 件は変数に束縛している値が正しくなかったもので，特に 9^3 円玉の枚数に誤りが目立った．また，1 件は図 4 次に示すプログラムであった．実行が結果は正しいものの，10^{12} 回以上の繰り返しが必要であり，標準的な Python 処理系の性能では現実的な時間での計算は到底不可能である．

```
min_count = 1000  # 最小の硬貨の枚数
for x1 in range(1001):  # 9^3 円玉の枚数を 0 から 1000 まで試す
    for x2 in range(1001):  # 9^2 円玉の枚数を 0 から 1000 まで試す
        for x3 in range(1001):  # 9^1 円玉の枚数を 0 から 1000 まで試す
            for x4 in range(1001):  # 9^0 円玉の枚数を 0 から 1000 まで試す
                if 729 * x1 + 81 * x2 + 9 * x3 + x4 == 1000:
                    # 支払い金額が 1000 円になる場合
                    count = x1 + x2 + x3 + x4  # 硬貨の総枚数
                    if count < min_count:  # より少ない枚数で支払える場合
                        min_count = count  # 枚数を更新
                        ex02_6_1 = x1  # 9^3 円玉の枚数
                        ex02_6_2 = x2  # 9^2 円玉の枚数
                        ex02_6_3 = x3  # 9^1 円玉の枚数
                        ex02_6_4 = x4  # 9^0 円玉の枚数
```

図 4: 課題 ex02_6 に対する誤答例

大規模言語モデルは，複数の部分問題が組み合わさった課題を苦手としていることが知られている [2]. この課題は，4 種類の硬貨の枚数を順に求めてゆく必要があるために，この苦手な課題類型に合致したのではないかと推測している．また，図 4 の誤答は，ChatGPT は作成したプログラムのテスト実行を全く行っていないことを示唆している．他の課題でも，import がないために実行できないプログラムや，どんな入力に対してもリストの添字範囲外アクセスを起こすプログラムなどを作成しており，この推測を裏付けている．

3.2.2 ex04_9

次に示す課題 ex04_9 は，10 回中 2 回しか正答することができなかった．

> ある競技では，複数の審査員がつけた点数のうち，最大の値 1 つと最小の値 1 つを除いた残りの点数の平均が，その競技者の点数となる．例えば，審査員の点数が 80, 60, 70, 72, 76, 90 であれば，60 と 90 を除いた残り 4 つの平均である 74.5 がその競技者の点数である．各競技者に対する審査員の点数が配列で与えられるとする．この時，多数の競技者の中で，最も高い点数を得た競技者の点数を返す関数 ex04_9 を定義せよ．競技者は 1 名以上いるとしてよい．

特に目立った誤答（8 件中 5 件）は，入力が数値列だとし，その最大・最小値以外の平均値を求めるものである．つまり，[80，60，70，72，76，90] に対して 74.5 を返す．1 人の競技者の点数は正しく計算できているが，多数の競技者の点数の最大値を求めることはできていない．

この課題も，課題 ex02_6 と同様，複数の部分問題が組み合わさってできており，苦手な問題類型であった可能性がある．また，審査員がつけた点数と競技者の点数，各競技者に対する点数中の最大・最小と競技者の点数の最大，というように，関連する別の内容を同じ言葉で表しており，誤読を誘発したかもしれない．さらに，提示された具体例が部分問題に対するものだったのも誤読を引き起こした可能性がある．

3.2.3 ex06_2

次に示す課題 ex06_2 は全く正答できなかった．

> 与えられた 2 つの整数 n と m（$n, m \geq 2$）に対し「n または m から 1 つずつ数を減らしながら n と m の両方が割り切れるか試してゆき，両方割りきれる数が見つかった時点で終了する」という考えに基づき，n と m の最大公約数を求める関数 ex06_2 を定義せよ．（教科書練習問題 6.2）

誤答は以下の 2 種類に大別できた．まずは，ユークリッドの互除法を用いて最大公約数を正しく求めるプログラム（10 件中 6 件，図 5 上）である．これは，「1 つずつ減らしながら n と m の両方が割り切れるか試す」という指示に従っていない．他方は，n と m の値を両方 1 ずつ減らしながらお互いを割る，と

```
def ex06_2(n, m):
    while True:
        if n % m == 0:
            return m
        if m % n == 0:
            return n
        if n > m:
            n = n % m
        else:
            m = m % n


def ex06_2(n, m):
    while n >= 2 and m >= 2:
        if n % m == 0:
            return m
        elif m % n == 0:
            return n
        else:
            n -= 1
            m -= 1
    return 1
```

図 5: 課題 ex06_2 に対する 2 種類の誤答例

いうプログラムである（10 件中 4 件，図 5 下）．n でも m でもない変数を用意しその値を 1 ずつ減らしてゆく，という形式のプログラムは皆無であった．

この結果には大きく 2 つの理由が考えられる．まず第一に，最大公約数を求めるには，ユークリッドの互除法という非常によく知られた手法があることである．大規模言語モデルは，深く関連する課題に対するよく知られたコード例があると，それに引きずられやすいことが知られている [2]．この課題はまさにそのような場合に合致する．また，大規模言語モデルが「最大公約数」という概念を理解していないことも大きな理由である．最大公約数を知っていれば，n と m を両方割り切れるできる限り大きな値を探す，というのは自然な発想である．むしろ，n や m の値を変更しながら最大公約数を探そうとはまず考えないだろう．一方 ChatGPT は，そのような前提を理解していないため,「n または m から 1 つずつ数を減らし」という指示を,「変数 n や変数 m の値を 1 ずつ減らす」と誤読した可能性が高い．

3.2.4 ex08_4

次に示す課題 ex08_4 にも全く正答できなかった．

配列 x と整数 n（$1 \leq n \leq$ (x の長さ)）が与えられたとき，x 中の小さい方から n 個（$n \geq 1$）の要素を抽出する再帰関数 ex08_4 を定義せよ．この関数は，併合整列法を参考にして，配列を再帰的に分割し，それぞれの部分の小さい方から n 個の要素を抽出し，それを併合してゆくことで最終的な結果を得るものとせよ．なお，x 中に同じ大きさの値が複数あったとしても，それらは別のものとしてよい．例えば，[2,1,2,1,3] から 3 個抽出する場合，結果は [1,1,2] となる．なお，結果として出力される要素は値の小さい順に整列すること.（教科書練習問題 8.5）

10 件の誤答のうち，8 件は正しい結果を返さないものであった．誤りの理由は様々であったが，特に目立ったものは，分割した部分列から n 未満の個数の要素（例えば $n/2$ 個）を抽出しているもの（4 件，図 6 (a)）と，ベースケース（配列 x の長さが n 以下）で x を整列せずに返しているもの（2 件，図 6 (b)）であった．また，残りの 2 件は，実行結果は正しいが課題の指示を満たさないものであった．具体的には，1 件は併合ソートを行い上位 n 件を返すもの（図 7 (c)），1 件は最小値を繰り返し抽出するもの（図 7 (d)）であった．

この課題は，40 件の中でも最も高度なものであり，これまで挙げた理由が複数絡まり合って誤答に至っていると思われる．まず，この課題は明らかに部分問題の組み合わせになっており，大規模言語モデルが苦手とする類型に合致する．次に，このような分割統治の課題では，配列を 2 つに分けてゆくアルゴリズムも，数値を半分にしてゆくアルゴリズムも，どちらもよくある．今回のように，配列と数値を両方入力としつつ，配列のみを半分にし，数値は変更しないようなものは比較的まれであろう．そのため，よくあるプログラム例に引きずられて誤った可能性がある．さらに，問題を正しく理解すれば，どちらの部分列からも n 件抽出しなければならないことは比較的容易にわかる．しかし，問題を理解せずにそれっぽいコード列を合成するだけでは，n を半分にするのがむしろ自然に思えたとしても不思議はない．

```python
def ex08_4(x, n):
    # ベースケース
    if len(x) <= n:
        return sorted(x)

    # リストを分割する
    mid = len(x) // 2
    left = x[:mid]
    right = x[mid:]

    # 左半分の小さい方から n 個の要素を抽出する
    left_sorted = ex08_4(left, n)

    # 右半分の小さい方から n − len(left_sorted) 個の要素を抽出する
    right_sorted = ex08_4(right, n - len(left_sorted))

    # 左右の部分リストを併合し、小さい方から n 個の要素を抽出する
    merged = merge(left_sorted, right_sorted)
    return merged[:n]
```

(a)

```python
def ex08_4(x, n):
    if len(x) <= n:
        return x

    def merge(xs, ys):
        res = []
        i = j = 0
        while i < len(xs) and j < len(ys):
            if xs[i] < ys[j]:
                res.append(xs[i])
                i += 1
            else:
                res.append(ys[j])
                j += 1
        if i < len(xs):
            res.extend(xs[i:])
        if j < len(ys):
            res.extend(ys[j:])
        return res

    def sort_and_take_n(x):
        if len(x) == 1:
            return x
        mid = len(x) // 2
        left = sort_and_take_n(x[:mid])
        right = sort_and_take_n(x[mid:])
        return merge(left, right)[:n]

    return sort_and_take_n(x)
```

(b)

図 6: 課題 ex08_4 に対する 4 種類の誤答例 (1)

```python
def merge_sort(x):
    # リストの長さが 1 以下の場合、そのまま返す
    if len(x) <= 1:
        return x

    # リストを 2 つに分割する
    mid = len(x) // 2
    left = x[:mid]
    right = x[mid:]

    # 分割したリストを再帰的にソートする
    left_sorted = merge_sort(left)
    right_sorted = merge_sort(right)

    # 分割されたリストを併合する
    return merge(left_sorted, right_sorted)

def merge(left, right):
    merged = []
    i = 0
    j = 0

    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            merged.append(left[i])
            i += 1
        else:
            merged.append(right[j])
            j += 1

    # left または right の要素が残っている場合、それを merged に追加する
    merged.extend(left[i:])
    merged.extend(right[j:])

    return merged

def ex08_4(x, n):
    # merge_sort 関数を使ってリストを整列する
    sorted_x = merge_sort(x)

    # ソートされたリストの先頭から n 個の要素を抽出する
    extracted = sorted_x[:n]

    return extracted
```

(c)

```python
def ex08_4(x, n):
    if n == 1:
        return [min(x)]
    elif len(x) == 0 or n == 0:
        return []
    else:
        pivot = min(x)
        count = x.count(pivot)
        left = [i for i in x if i < pivot]
        right = [i for i in x if i > pivot]
        if n <= len(left):
            return ex08_4(left, n)
        elif n <= len(left) + count:
            return left + [pivot] * (n - len(left))
        else:
            return left + [pivot] * count + ex08_4(right, n - len(left) - count)
```

(d)

図 7: 課題 ex08_4 に対する 4 種類の誤答例 (2)

3.2.5 まとめ

以上の結果をまとめると，今回の実験において，典型的に ChatGPT が苦手とした課題類型は以下の 3 種類である．

- 複数の部分問題が組み合わさっている課題
- 深く関連するよく知られたプログラムがあり，それとは異なる方針でのプログラムが求められる課題
- 問題に対する数学的・論理的な理解や分析を求める課題

これらの要素は，先行研究で報告されていたものと一致しており，プロンプトが日本語であったことの影響は即座には見て取れない．

なお，誤答の少なかった課題と多かった課題が明確に別れていることを考えると，苦手な課題類型の場合，日本語であれば英語に比べてより顕著に不得手となっている可能性はある．この点についての調査は今後の課題である．

3.3 RQ3 についての分析

次に示す課題 ex04_4 に対して得られたプログラム例を図 8 に示す．

> 数値の配列が与えられたとき，その中の正の値の総和を返す関数 ex04_4 を定義せよ．

いずれのプログラム例もよく書けている．ある程度プログラミングに習熟した学生のものとの見分けはつきそうもない †4．

その一方で，これらのプログラム例は，一人の学生が様々な課題の回答を ChatGPT で作成した場合，回答のコーディングスタイルに一貫性がないだろうということを示唆する．例えば，(b) のプログラムでは変数名や空白を豊富にとっており，(a) や (d) と同一人物が書いたとは思いにくい．同様に，(c) のように細かくコメントを書く受講生が (a) や (d) のようなプログラムを書くとは思いにくい．(d) のプログラムを書ける受講生が，その次の似た課題で (a) のようなプログラムを書くのも不自然だろう．

実は，受講生が (d) のようなプログラムを書く可能性はかなり低い．これは教科書では取り上げていないジェネレータ式を用いているからである．似たケースは他の課題でも散見された．例えば，本講義のサポートを目的として ita ライブラリ †5 を整備しており，これを講義でも紹介し利用を勧めている．しかし，当然のことではあるが，ChatGPT の生成したプログラムは 400 件中 1 件たりともこのライブラリを使用していない．このように講義内容から乖離したプログラムは不自然であり，疑いをもつ理由になるだろう．

4 議論

本稿では，大学 1・2 年生向け全学教養教育で用いているプログラミング課題 40 件について，ChatGPT に回答させてみた結果を報告した．約 90%については正答しており，しかもそのプログラムは（少なくとも単独では）プログラミングに習熟した学生のものとはほとんど区別がつかない．

以上の結果の影響は小さくはない．まず直接的なこととして，今回用いたような課題は成績をつけるのに使えないことがわかる．受講生は ChatGPT を用いることで，講義内容を全く理解していなくても，ほとんどの課題に容易に正解できてしまう．この事実は，さらに根本的な変革を要求する．現状の課題ですら，ChatGPT を用いない初学者にとっては十分に難しい．これを，ChatGPT では容易に解けないような課題に変更した場合，今度は ChatGPT なしでは講義についていけなくなるだろう．ということは，ChatGPT の利用を何らかの手段で禁止ないし排除するか，または ChatGPT の利用を前提・推奨した新たな講義カリキュラムを作成するかのどちらかが必須となる．しかし，ChatGPT の利用を禁止・排除するのは原理的に難しく，しかも将来の技術発展を考えると適切な方針かどうか疑わしい．一方で，後者の方針の現時点での採用には，現状の技術がまだ発展途上であることを考えると，運用に際してかなりの労力が求

†4 実際のところ，現実の学生のものと比べると，**きれいすぎる**印象はある．学生のプログラムの場合，変数名の付け方やスペースの入り方がもっと雑になるのが普通であろう．とはいえ，プログラムがきれいすぎるというのは，自作のプログラムではないことを疑う理由としては弱いだろう．

†5 https://pypi.org/project/ita/

```python
def ex04_4(arr):
    total = 0
    for num in arr:
        if num > 0:
            total += num
    return total
```

(a)

```python
def ex04_4(numbers):
    sum_positive = 0

    for num in numbers:
        if num > 0:
            sum_positive += num

    return sum_positive
```

(b)

```python
def ex04_4(numbers):
    # 総和を保存する変数を初期化
    total = 0

    # 配列の要素を順に調べる
    for num in numbers:
        # 要素が正の場合，総和に加算する
        if num > 0:
            total += num

    return total
```

(c)

```python
def ex04_4(arr):
    return sum(x for x in arr if x > 0)
```

(d)

```python
def ex04_4(numbers):
    positive_numbers = [x for x in numbers if x > 0]
    return sum(positive_numbers)
```

(e)

図 8: 課題 ex04_4 に対する ChatGPT の回答プログラム例

められる．アルゴリズム入門のように多数の教員が協調して行っている大規模講義では，これは現実的とは言い難い．

また今回，大規模言語モデルは日本語のプロンプトであっても基本的・定型的なプログラムであれば作成できることがわかった．このことからは，簡単なプログラミング課題であれば，プロンプトの多少の差異は回答の精度にそれほど影響しないだろうと予想できる．これは，プログラミングに疎い利用者向けシステムの作成を考えた際には，大きな利点となりうる．現時点で実用化されている，自動プログラミングに基づく最も有名なシステムは Microsoft Excel の FlashFill [4, 7] であろう．FlashFill では，ユーザの指示に従いシステムが自動的に文字列操作プログラムを作成することで，だれでもプログラミングの恩恵を享受することができる．今回の結果は，似たようなシステムの可能性を示していると言えよう．

その一方で，今回の実験を通して，どの程度正確に仕様を特定し指示しなければならないのかは全く評価できていない．今回用いた課題は，学生の自習用ということもあり，不明瞭・曖昧な指示がないよう複数の教員で確認している．しかし，プログラミングに疎いユーザには，どのような指示をすべきなのか，コーナーケースとしてどのようなことを考えなければならないのか，また得られたプログラムの正しさをどうやって確認すればよいのか，といったことは分からないだろう．しかし，今回の実験結果からも示唆されるとおり，ChatGPT が苦手とするような課題類型に対処するためには，部分問題への分割や，数学的・論理的な知識・推論結果の提示などを，ユーザが適切に行うことが求められる．この点の検証は今後の課題である．

また，今回は大規模言語モデルを単に利用するだけでどの程度のプログラムが作成できるかを調査した．近年，大規模言語モデルを一部品とした，更に発展的な自動プログラミングシステムが次々と提案されている [10, 13–16]．現時点ではこれらはまだ一般ユーザからは縁遠い印象もあるが，将来にはこれらも含めて考えてゆく必要があるだろう．

謝辞 当研究の方向性についての示唆を頂いた東京大学の佐藤重幸氏に感謝する．本研究は JSPS 科研費 23K11044 の助成を受けたものである。

参 考 文 献

[1] Alet, F., Lopez-Contreras, J., Koppel, J., Nye, M. I., Solar-Lezama, A., Lozano-Pérez, T., Kaelbling, L. P., and Tenenbaum, J. B.: A large-scale benchmark for few-shot program induction and synthesis, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, Meila, M. and Zhang, T.(eds.), Proceedings of Machine Learning Research, Vol. 139, PMLR, 2021, pp. 175–186.

[2] Austin, J., Odena, A., Nye, M. I., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C. J., Terry, M., Le, Q. V., and Sutton, C.: Program Synthesis with Large Language Models, *CoRR*, Vol. abs/2108.07732(2021).

[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, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H.(eds.), 2020.

[4] Cambronero, J., Gulwani, S., Le, V., Perelman, D., Radhakrishna, A., Simon, C., and Tiwari, A.: FlashFill++: Scaling Programming by Example by Cutting to the Chase, *Proc. ACM Program. Lang.*, Vol. 7, No. POPL(2023), pp. 952–981.

[5] Chen, L., Zaharia, M., and Zou, J.: How is ChatGPT's behavior changing over time?, *CoRR*, Vol. abs/2307.09009(2023).

[6] Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W.: Evaluating Large Language Models Trained on Code, *CoRR*, Vol. abs/2107.03374(2021).

[7] Gulwani, S.: Automating string processing in spreadsheets using input-output examples, *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*, Ball, T. and Sagiv, M.(eds.), ACM, 2011, pp. 317–330.

[8] Gulwani, S., Polozov, O., and Singh, R.: Program Synthesis, *Found. Trends Program. Lang.*, Vol. 4, No. 1-2(2017), pp. 1–119.

[9] Hendrycks, D., Basart, S., Kadavath, S., Mazeika, M., Arora, A., Guo, E., Burns, C., Puranik, S., He, H., Song, D., and Steinhardt, J.: Measuring Coding Challenge Competence With APPS, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, Vanschoren, J. and Yeung, S.(eds.), 2021.

[10] Jain, N., Vaidyanath, S., Iyer, A. S., Natarajan, N., Parthasarathy, S., Rajamani, S. K., and Sharma, R.: Jigsaw: Large Language Models meet Program Synthesis, *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*, ACM, 2022, pp. 1219–1231.

[11] Lai, Y., Li, C., Wang, Y., Zhang, T., Zhong, R., Zettlemoyer, L., Yih, S. W., Fried, D., Wang, S. I., and Yu, T.: DS-1000: A Natural and Reliable Benchmark for Data Science Code Generation, *CoRR*, Vol. abs/2211.11501(2022).

[12] Liu, J., Xia, C. S., Wang, Y., and Zhang, L.: Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation, *CoRR*, Vol. abs/2305.01210(2023).

[13] Liventsev, V., Grishina, A., Härmä, A., and Moonen, L.: Fully Autonomous Programming with Large Language Models, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2023, Lisbon, Portugal, July 15-19, 2023*, Silva, S. and Paquete, L.(eds.), ACM, 2023, pp. 1146–1155.

[14] Nijkamp, E., Pang, B., Hayashi, H., Tu, L., Wang, H., Zhou, Y., Savarese, S., and Xiong, C.: CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis, *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*, OpenReview.net, 2023.

[15] Xia, C. S., Wei, Y., and Zhang, L.: Automated Program Repair in the Era of Large Pre-trained Language Models, *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*, IEEE, 2023, pp. 1482–1494.

[16] Xia, C. S. and Zhang, L.: Less training, more repairing please: revisiting automated program repair via zero-shot learning, *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022, Singapore, Singapore,*

November 14-18, 2022, Roychoudhury, A., Cadar, C., and Kim, M.(eds.), ACM, 2022, pp. 959–971.

[17] Xu, F. F., Alon, U., Neubig, G., and Hellendoorn, V. J.: A systematic evaluation of large language models of code, *MAPS@PLDI 2022: 6th ACM SIGPLAN International Symposium on Machine Programming, San Diego, CA, USA, 13 June 2022*, Chaudhuri, S. and Sutton, C.(eds.), ACM, 2022, pp. 1–10.

[18] 佐藤重幸, 犬伏貴之: PLAGS UT: 自動評価付き Python プログラミング課題管理システム, 第 *24* 回プログラミングおよびプログラミング言語ワークショップ, *PPL2022*, 2022.

[19] 森畑明昌: 東京大学における全学プログラミング教育 (〈 特集 〉 プログラミング入門をどうするか), 情報処理, Vol. 57, No. 4(2016), pp. 358–361.

[20] 森畑明昌: *Python* によるプログラミング入門 東京大学教養学部テキストアルゴリズムと情報科学の基礎を学ぶ, 東京大学出版会, 2019.