JSiSE Research Report
vol.38,no.1(2023-5)

テキスト生成システムを用いたプログラミング学習法の検討

森 幹彦[*1]
*1 法政大学

A Study of the Programming Learning Method by Using Text Generation Systems

Mikihiko Mori [*1]
*1 Hosei University

This paper investigates the method to learn programming with text generation systems. We practiced learning to reduce the influence of their proficiency level in programming languages for the people who do not specialize in computer science. The results showed that appropriate verbalization of procedures improved the quality of the program code output by the text generation systems, but that the modification process was affected by proficiency in the programming language.

キーワード: テキスト生成，プログラミング教育，ChatGPT，GitHub Copilot

1. はじめに

近年，さまざまなコンテンツを生成できる大規模言語モデル(Large Language Model; LLM)が急速に進展してきており(1)，テキスト生成システム（以降では，Text Generation System を略して TGS と呼ぶ）も実際に利用可能になってきている (2)．OpenAI が提供を始めた ChatGPT [a]は，ユーザの入力する質問（プロンプトと呼ばれる）に対して，回答（生成テキストと呼ばれる）を出力することによってテキスト応答が可能なシステムのひとつであり，急速に利用者を増やしている．ChatGPT は，様々な分野でのテキスト生成が可能で，プログラミングも例外ではない(3)．ユーザが入力するプロンプトは，日常的に使う自然言語をそのまま使えばよく，出力される生成テキストは流暢な自然言語である．一方で，ソフトウェア開発環境では，プログラミングにおける様々な場面で入力補完をする GitHub Copilot（以降，Copilot と略す）が商用提供されている．Copilot は，OpenAI が提供する GPT-3 (4) を基にしている(5) [b]．

ChatGPT と Copilot はいずれも少量のテキストから優れたテキストを生成する．これらのシステムが一般に提供されるに至り，プログラミング環境が大きく変化した．職業的なプログラマだけでなく非エンジニアのユーザにも簡単に使えるようになり，これからプログラミングを学ぶ者にも影響がある(3)．したがって，コンピュータ科学（以降，CS と略す）を専門としない学習者がプログラミングを学ぶ際にも大きな影響を与えることは容易に想像できる．

LLM をベースとした自然言語生成では，事実とは異なる出力（hallucination と呼ばれる）が常につきまとう(6)(7)．そのため，プログラミングを容易にする期待とは裏腹に，誤ったプログラムコードが生成されることがある．しかし，プログラミングにおいては，少なくともそのプログラムコードを実行すれば意図と合致するものであるかどうかを確認できる．また，プログラミングの学習では，生成されたプログラムコードを読む段階や実行する段階において，期待したものかどうかを確認できる能力が学習の一環として求められるだろう．

[a] https://chat.openai.com/
[b] ChatGPT は GPT-3.5 をベースとしているとされており，有料プランでは次バージョンである GPT-4 が試験的に提供されている（2023 年 4 月 6 日現在）．

表 1 日程と各活動にかけた時間（10 分単位で記載）

日次	時間（分）	活動内容
1 日目	120	事前課題の実施
	30	「知る」
	120	「触れる」(ChatGPT)
2 日目	150	「触れる」(Copilot)
	180	「試す」
3 日目	120	「試す」（続き）
	120	「振り返る」
	40	事後課題の実施

したがって，一般の自然言語の生成よりもプログラムコードの生成は，検証可能性に優れており学習における利用にも有利と考えた．つまり，TGS は CS を専門としないプログラミング学習者の支援が期待できる．そこで本研究では，CS の基礎が十分でない文系学生を対象にした ChatGPT と Copilot の 2 つのテキスト生成システムを用いたプログラミング学習法を提案し，その実施結果を報告する．

2. テキスト生成システムを用いた学習法

2.1 方針

CS は，「研究(study)，設計(design)，実装(implementation)して，現実の問題解決に関わる学問だという主張」[8]がある．本稿はこの主張を採用し，その中でも設計の段階に注目した．ただし，CS を専門としない学習者を対象にして，CS の専門性を問わない汎用的な技能の獲得を目指すことにした．

そこで，「設計」にかかわる計算論的思考[9]（Computational Thinking, 以降では CT と略す）を獲得することを目指して学習設計することにした．そのための道具立てとして TGS を用いたプログラミング活動を行う．したがって，本稿では特定のプログラミング言語の仕様を把握することや，プログラミング技術を駆使して高度なアルゴリズムを組み立てるといったことは目標にはしていない．

対象となる学習者が実務の中でプログラミングに大きくかかわる機会として，自らが求める情報システムをつくりあげようとするときであることは容易に想像できる．CS の専門性を求めないことから，対象者が実務で利用するシステムについて，特にシステム設計の場面でかかわることになる．その際には，システム構築を受け持つシステムエンジニアとの間で円滑なコミュニケーションを進めて具体化していく[10]必要があるため，CS への理解や CT に対する学習は実務への貢献も大きい．

ChatGPT は，適切な CS 課題を提示すれば正確な回答が期待できる[11]．同様に Copilot も，状況に応じたコードやコメントなどのプログラミングに関わるテキスト生成を適切に行えるとの期待が持てる[12]．実際，ChatGPT を用いて自然言語での要求に対する自動生成したプログラムコードを実行するような仕組みがサービスとして提供し始めている（例えば，replit [13]）．

したがって，これらの TGS を道具として用いていく際には，システム設計の中でも特に要件定義や基本設計を適切に指示できることが学習者に求められる．これは CT であり，「手順的な自動処理」[14] にも関わる．本稿では，直接コンピュータに指示するよりは抽象的または曖昧と言える，人間に対する「手順的な自動処理」の指示を想定して，そのレベルでの達成を目指すことにした．

2.2 学習設計

次のような学習活動を行う．

(1) 事前学習

(2) 実践

- (a) 知る：TGS の概要説明をする．
- (b) 触れる：TGS を利用してみる．
- (c) 試す：プログラミング課題について，TGS を用いて実装を試す．
- (d) 振り返る：想定と生成結果の関係を考える．コードレビューをする．

事前学習では，Python の初歩的な知識を得ることを目標とした．参加希望者には実践開催日までに，変数とオブジェクト，条件分岐と繰り返しの機能と記述法に触れておくように伝えた．基準として，東京大学が公開する Python 入門[15]の 4 節までを確認することとした．なお，参加を呼びかけた対象者は，正規授業で学ぶことが可能であった．また，この授業を受講していない場合や Python を独りで学習することが難しい

場合も想定し，paiza [c]で提供されている講座を案内した．paiza を利用する場合は，「Python 体験編」程度を必須として，「新・Python 入門編」と「Python3 入門編」の受講を推奨する旨を伝えた．

実践においては，「知る」「触れる」「試す」「振り返る」の 4 つのステップを踏むことにした．

「知る」では TGS についての概要を学んだ．その後に，ChatGPT へのプロンプト（質問）と生成テキスト（回答）の関係を把握することを目的に行った．「触れる」では，各サービスのアカウントを作成して，実際に様々なプロンプトを入力した結果を確認させた．いずれも，プログラミングに限定しなかった．

「試す」では，プログラミング課題に応じた実装を試させた．特定のソフトウェアの完成を目標として与え，TGS を自由に用いてよいとして実装させた．ChatGPT へのプロンプトを意図に合わせて修正できるようになること，Copilot に適切なきっかけを与えて入力補助に用いることができるようになることを期待した．「振り返る」では，実際にできあがったプログラムコードを見せ合い，評価し合うとともに，主として教授者によるコードレビューを行った．いずれのステップでも，学習者間で意見を出し合うこと，相談すること，プログラムコードやプロンプトを見せ合うことを推奨した．

このような学習設計は，プログラミング学習の活動において従来からよくみられた．それらとの違いは，TGS によってプログラムコードを学習者自身が考えなくてよいことである．一見すると思考力の低下を招くようにも考えられるが，プログラミングにおけるつまずき[16]を軽減する方法と捉えることもできる．本稿の学習法は，一般的なプログラミング学習の課程においてつまずく場面を置き換えるまたは容易化する方法とも言える．

2.3 評価法

学習活動を始める前に，事前課題として次の 4 問に解答させた．

問 1：ジャムとバターの塗り方を指示する説明書の作成する [d]

問 2：飯ごう炊さんでカレーを作る

問 3：自転車操縦手順の修正をする

問 4：九九表作成プログラムの出力例とプログラムコードの修正をする

問 1 から問 3 までは，手順的な指示能力の確認を意図した設問である．問 4 は，今回の学習活動の中で実際に出会う可能性のある問題に対する解決能力を見ようとした．プログラムで実現したいことを手順として示した上で，出力結果とプログラムコードが誤ったものを掲載して，この出力結果とプログラムコードのそれぞれを修正させた．

問 4 は，九九の数値が偶数か奇数かを表形式で表示させるための手順が自然言語で示してある．このとき，行と列に見出しとして掛ける数を表示することになっている．また，この手順を実装したとするプログラムコードと，そのプログラムが出力するはずとする出力結果が示されている．しかし，プログラムコードは，手順とは異なっている．また，出力結果にも誤りがあり，行と列の見出しが表示されていない．また，偶数・奇数の表示に失敗があって数値が表示されていて，その数値も誤りがある．なお，この問 4 の元となった手順，プログラムコード，出力結果は，筆者が ChatGPT に何度かプロンプトを与えた中で誤ったときの生成テキストを基にしている．

事後課題は，事前課題で提出したものを修正させた．学習活動を通じて CT やプログラミングについて理解が深まったなら，適切に修正されることを期待した．事前課題・事後課題はともに，Microsoft Word 形式のファイルの中で実施した．最初に課題を示し，その末尾に解答を記入させた．

3. 実践結果

3.1 実践設定

課外学習の機会として文系学部生に参加を呼びかけた．日程と 2.2 節で示した各活動にかけた時間を表 1

[c] https://paiza.jp/
paiza ラーニング学校フリーパスを利用した．

[d] YouTube に掲載された次の動画からアイディアを得た．
Josh Darnit: Exact Instructions Challenge - THIS is why my kids hate me.
https://www.youtube.com/watch?v=cDA3_5982h8

表 2 参加者のプログラミング経験

参加者	授業	自習
A	同年度秋学期にプログラミング科目において Python を学習	なし
B	なし	なし
C	なし	Paiza で「Python 体験編」「Python 入門編」を受講
D	なし	1 年前に 1 年前に東大の教材[15]の基礎部分

に示す．ChatGPT は無料版，すなわち GPT-3.5 ベースのものを使った．Copilot は GitHub Education のうち GitHub Student Developer Pack として無料で提供されたものを使った．

3.2 実践結果

本実践は，2023 年 3 月 20 日から 21 日の 3 日間で行った．実践参加者は，5 名の 2 年生であった．このうち，4 名が最終日まで参加した．それぞれを A, B, C, D と呼ぶことにする．それぞれの Python への習熟度を表 2 に示す．

事前課題は，1 問ごとに全員が回答し終わるまで待つようにして進めていった．長いもので 40 分程度かかった．解答の間は，参加者間での会話は禁じた．

「触れる」ステップでは，ブロック崩しゲームを題材に，Pygame を用いて実装するデモンストレーションをしながら ChatGPT と Copilot での実現可能性を示した．

「試す」ステップでは，To-Do リスト管理ソフトウェアを Python で作成することを目標とした．一般に Python で GUI を実現する場合，標準で提供されている tkinter パッケージが使える．また，「触れる」でも示した Pygame でも可能ではある．

実践中の発言から，参加者たちは TGS を全般的に知らず，本実践ではじめて ChatGPT の存在も知ったようだった．当然ながら利用経験はなく，プロンプト・エンジニアリング[17]も知らなかった．なお，本実践中には，プロンプト・エンジニアリングについて触れていない．

3.2.1 事前・事後課題の内容

事前・事後課題の内容を確認したところ，次のようなことが分かった．まず，4 問のうち手順的な記述を求める最初の 3 問は，事前課題の段階で人間に伝えるには十分な粒度の記述がなされていた．事後課題として修正させた結果，多くの場合に手直しされて厳密になった．例えば，問 2 において，作業ごとに対応する作業分担者の名前を記入していた．

問 4 のプログラム修正では，次のようになった．まず事前課題において，参加者 B と D は修正しなかった．参加者 A と C は，出力結果の部分を正しく修正していたものの，プログラムの修正は十分ではなかった．参加者 A は，`for` 文の二重構造を分解して 2 つの単独ループにしてしまい，元のプログラムよりも出力結果が期待よりも遠くなった．参加者 C は，改行が適切に行われないために，1 行にすべてを出力するコードだった．

次に，問 4 の事後課題について修正点を確認した．参加者 A は，ほぼ正しいプログラムコードを書いていた．ただし，`print` 関数が標準で末尾で改行することを理解できていなかったためか，`¥n` を `print` 文に与えてしまって余計な改行が入った．参加者 B は出力結果の部分を修正しなかった．また，行見出しを表示するプログラムコードの追加に成功したが，列見出しを表示するプログラムコードの追加をしなかった．また，同じ変数名にすべきところを別にしたために，偶数・奇数の判定コードが動作しなかった．参加者 C は，行・列の見出しはないものの，偶数・奇数の表示が正しくできるプログラムコードだった．参加者 A と同様に`¥n` を `print` 文に与えて余計な改行をしていた．参加者 D は，出力結果の修正は，部分的にできていた．ただし，偶数・奇数の表示が掛け算の結果と一致しない修正をしていた．プログラムコードは修正しなかった．

なお，`¥n` をプログラムコードに入れがちだった原因として，問 4 の設問文中に示した手順の中に「各行の

末尾に改行（¥n）を入れること.」があったことが考えられる．改行コード（¥n）が改行であることは文意として理解したときに，それが print 関数の標準的な動作と結びついていなかった可能性がある．

3.2.2 コードレビューの内容

参加者たちが作成した To-Do リスト管理ソフトウェアを「振り返り」としてコードレビューした．いずれの参加者も TGS の生成したコードをおおよそそのまま使い，大幅な修正をしていなかった．各参加者のプログラムコードは次のようになっていた．

参加者 A は，最初に生成したプログラムコードが思い通りの結果にならず，最後の 5 分でもう一度 ChatGPT に生成させたと話した．当初のプログラムは参加者 A の要件を満たそうとするときに根本的な修正が必要になり，多くの修正を加えたが要件を満たせなかった．最後に生成したものは期待した動作をしていた．参加者 B は，生成されたプログラムコードを修正してカテゴリごとにリストを分けようとしたが，カテゴリ数の分だけリスト作成部分のプログラムコードをコピーして実装に失敗していた．参加者 C は，生成されたプログラムコードに対して多くの修正を加えていた．参加者 D は，ラベルを修正する程度の修正にとどまっていた．

それぞれの参加者が提示したプログラムコードに対して，各行やブロックにおいてどのような動作であるかを尋ねたところ，いずれの参加者も答えられないことが多かった．ChatGPT や Copilot が生成したものが動いたから問題ないと判断したといった回答をされた．

3.2.3 アンケート結果

アンケートで尋ねたことは次のとおりである．まず，学習経験を尋ねた後に，今回の課題について機能，注力した点，その実装方法について尋ねた．その後に，つまずいた点，コードレビューを通じて自分のプログラムコードについて感じたこと，プログラミングにおいて ChatGPT が上手く使えるときや場面の気づき，プログラミングにおいて Copilot が上手く使えるときや場面の気づき，検索エンジンなど他のサービスが上手く使えるときや場面の気づき，TGS を授業で使うとしたら教員に尋ねる場合との使い分け方を尋ねた．最後に，プログラミングを今後進めるときにどのような知識やスキルが必要になるかを尋ねた．

今回の課題についての質問に対しては，それぞれの参加者は自身のソフトウェアへの機能や注力した点について具体的に回答していた．注力した点や実装方法も具体的であった．参加者 A は，修正した箇所を示していた．参加者 B と D は，意図した生成が行われるようにプロンプトを試行錯誤しており，プロンプトの与え方について具体的に回答していた．実践中に，参加者 C は期限入力を選択式にしようとして方法がわからず困っていた．その機能を提供するウィジェット名を知らなかったためである．そこで，教授者として立ち会った筆者が Combobox を使えばできる旨を伝えた．アンケートでは，その情報を得て Google 検索を用いて使い方を調べたと回答していた．

コードレビューを通じて感じたことについての回答として，参加者 A は修正しようとした際に根本的な修正が必要になり修正が簡単ではなかったとしていた．参加者 C は，大体は分かっていても詳細に説明させられるとよく分かっていないことや，全部を理解しなくても動かせることを回答していた．参加者 B は，理解していなくても書けるとはいえ，書かれているコードが理解できれば追加機能の実装ができたと回答していた．参加者 D は，コードの修正をするときに，Python の知識が足りないために回り道をして時間がかかったと回答していた．

ChatGPT や Copilot を上手く使える場面と他のサービスとの使い分けについての回答からは次のようにまとめられる．ChatGPT について，最初のコードを作成する段階で ChatGPT を使うと時間短縮ができる．漠然とした要求ではなく具体的な指示をすると大きな力になる．自分の中で具体化していくと ChatGPT の回答も良くなる．Copilot については，書き方がうろ覚えのときに助けになった．標準的な手続きの場合に自動で次のプログラムコードを出力するため楽だった．検索エンジンについては，過去に確実に動くプログラムコードを得るために使える．ある程度出来上がった後では，ChatGPT に尋ねなくても解説が詳しく書かれた Web ページが見つけられる．

教員と TGS の使い分けにおいて教員に尋ねる意義についての回答として，的確に解決に向かいたいとき，ChatGPT への質問する内容自体が分からないときに教員に尋ねるとしていた．一方で，多くの質問に対し

て ChatGPT の回答で満足が得られると感じてもいるようで，ChatGPT へ質問しても上手く解決できないときに教員に尋ねたいとの意見が出された．その背景には，教員が常に回答できる状況とは限らないことが挙げられていた．

今後のプログラミングに必要なこととして次のような回答があった．参加者 A は，その言語の基礎知識を得て何ができるかを知っている必要があるとした．参加者 B は，エラーの読み方，書かれているプログラムコードの読み方を知る必要があるとした．参加者 C は基礎知識と，書かれている内容の理解の向上を挙げた．参加者 D は，ChatGPT と教員のどちらに対しても，実行したい「ビジョン」が明確でないと得たい結果に繋がらないと答えた．

3.3 考察

本実践の参加者は，言葉で手順を伝える能力が相応にあったことが事前課題の結果から言える．参加者たちが期待するプログラムコードを的確に ChatGPT が生成できた要因であろう．手順を的確に説明できる能力は，TGS が質の高いテキストを生成するために必要であると言える．したがって，CT のうち本稿で注目した「手順的な自動処理」の考え方は TGS によるプログラムコードの質を向上させると考えられる．ただし，参加者 A のように，実際に修正しようとする際には，少しの修正のために根本的な修正が要求される可能性は排除できない．

一方で，プログラムコードを読む力は，Python に関する事前学習による習熟度の影響があることが事前課題の問 4 やコードレビューにおける状況から見える．アンケートの回答からも，習熟していない場合には，生成されたプログラムコードを修正する段階で困ることも分かった．プログラミングが不要になるとの見立てがある[18]が，完全に不要になるにはまだかかるか，新たなサポートが必要になるだろう．現状では，プログラミング言語への習熟度は，プログラムコードの適切な修正に大きく影響し，作成されるソフトウェアの質にも影響すると言える．

昨今，ChatGPT へのプロンプトの内容によって，得られる生成テキストの質が異なることが言われており，適切な質問をいわゆる AI に入力できることが成果の違いを生む（例えば，文献[19]）との言説に繋がっている．それもあって，生成結果の精度向上の方法（例えば OpenAI Cookbook [20]）や，プロンプト・エンジニアリング[17]といったテクニックが話題に挙がることが多い．しかし，本実践ではこのようなテクニックに一切触れなかった．プロンプト・エンジニアリングとして紹介されているようなテンプレートを自ら見つけ出すことが学習者に求められることだと考えたためである．この判断による大きな不利益はなかったように見える．ただし，修正が難しいためにつまずいた場面で，プロンプト・エンジニアリングなどのプロンプトの工夫によって解決できた可能性は否定できない．

これらのことから，CT やプログラミング言語への習熟度が高いほど，TGS を用いたときにプログラムコードの質も高まることが示唆された．プログラミング言語への習熟度は求められるものの詳細な記憶は不要になり，アルゴリズムを自然言語で記述すれば初期のプログラムコードが生成できる．つまり，従来よりも抽象的な思考，すなわち CT が重要になる．

本実践における「試す」で設定した To-Do リスト管理ソフトウェアは，チュートリアルによく用いられるものである．つまり，様々な場所で関連する文章が書かれている可能性が高い．参加者たちがプロンプトを入力したとき，相応に適切なプログラムコードが生成されたのは，例が多いことも一因であろう．プログラミング課題を学習者に委ねて実装させる制作を授業で設定すると，ありふれた制作目標を掲げた学習者が簡単に実装でき，独創的であるほど目標の難易度だけでなく例示されにくいための難易度も高まってしまう可能性がある．先述した習熟度と質の関係とは異なり，習熟していないからこそ成果の質が高まる可能性も考えられる．授業においてこのようなプログラミング課題を与える場合には，一定の独創性を要求することが必要になるかもしれない．

本実践の参加者たちは，生成されたプログラムコードを十分に理解しているとは言えなかった．その一方で，プログラムを完成させられたことにも注意が必要である．このような乖離は，コピー＆ペーストでプログラミングしていった場合にも起きていた．その場合にこれまでは，変数名の不整合などで上手くいかないことも多かった．TGS によってそのような不整合が起

きにくくなったことは大きな進歩と言える．一方で，学習者にとっては，失敗（エラー）によって理解が促される機会の喪失とも言える．従来であれば，「プログラムが動かない」ことは大きなつまずきであり，プログラミング関連授業においては離脱を引き起こす要因であった．したがって，授業の中で利用を許す場面や段階などを十分に授業設計として折り込んでいくことによって，学修成果を引き出す道具として TGS を使えるだろう．また，CT のトレーニングの一環としても，ソフトウェアの要件を絞り込み設計を十分に行った上でそれに沿った実装をさせるような授業設計が考えられる．

3.4 本研究の限界

本稿での学習活動は，正規授業ではない．ノンフォーマル教育[21]と呼ばれる正課外であるが構造を持った教育として捉えられる．正規授業で実施する場合には，時間数としてもテーマの広さとしても十分ではない．ただし，簡単に TGS を試せる時間数でもあり，ノンフォーマル教育としては妥当な学習設計と考える．

本実践では 4 名が最後まで参加した．人数が少ないため数量的な分析はできなかった．一方で，詳細に活動を確認できた．一般的な教育現場では，数十名規模が想像に難くない．その場合には，グループ編成を行うことなどにより対応可能であると考える．

「手順的な自動処理」を確認するために用いた事前・事後課題の妥当性が示せていない．また，「試す」でのプログラミング課題は，本実践からも分かるように TGS によってほぼ自動的に生成可能である．課題の設定方法について今後検討していく必要がある．多少の難しさは，進展する TGS によってすぐに自動で解けてしまう可能性が高い．自動で解けてしまってさえも課題として成立するために求められる要件を考えていかなければならない．

4. まとめ

本稿では主に文系学部生を想定し，非専門家としての CS への関わりを考えて，CT の獲得を目指して TGS を用いたプログラミング学習法を提案した．4 人の学習者による実践の結果，「手順的な自動処理」の思考は TGS が生成するプログラムコードの質を高めること，その一方で修正段階ではプログラミング言語の習熟度の影響を受けることがわかった．ただし，本実践では，すべての参加者が手順を言葉として表す能力を一定程度持っていた．そのため，事前に想定していた「試す」「振り返る」でのフォローアップをしなかった．今後，本学習法を再実践するにあたり，言語化の能力の違いに注目しながら，本学習法の精緻化を進めたい．

現在，TGS にかかわる進展は非常に速い．その中でも「手順的な自動処理」のような CT にかかわる技能は重要さが保たれると考えている．一方で，CS の専門でなかったとしても，プログラミング言語の仕様を理解することから始める学習が不要であるとは言い切れない．適切な内容を考えていかなければならない．このときに，TGS を用いた自動生成を導入として，言語仕様の理解に繋げる学習も考えられる．様々な形態を模索していく必要がある．

実施時点では，国内でプロンプト・エンジニアリングに言及する記事は一般向けにほとんどなかった．しかしその後，この 1 か月にはそれに触れる記事が Web や新聞・雑誌で大量に現れている．今後は，このテンプレート化された「命令」または「質問」との向き合い方も学習設計の中に取り込む必要がある．今後もプロンプト・エンジニアリングが継続的に利用可能であると推察するものの，技術の進展によって求められるものは大きく変化し続けると考える．プロンプト・エンジニアリングを学習の中にどの程度とりいれるべきかが今後の検討課題である．

本実践では，実施中に偶然にも ChatGPT の障害があった[22]．このような問題によって学習活動が滞ることは避けられないとはいえ，バックアップ手段が求められる．例えば，今回の障害のように Web UI の問題なら API 経由では問題ない．API 経由でアクセスするシステムを構築すれば，障害を避けられるだけでなく学習者の活動の記録を残せて，より詳細な分析が可能になるだろう．しかし，これはサービス全体が使えなくなる場合の回避策にはなっていない．実利用が本格化する場合には，耐障害性についても検討しなければならない．この場合には，ローカルに動かす LLM を用いてもよいかもしれない．例えば，LLaMA[23] などが考えられる．今後のコンピューティング環境の変化に依存すると考える．

参 考 文 献

(1) Cao, Y., Li, S., Liu, Y., Yan, Z. et al.: “A Comprehensive Survey of AI-Generated Content (AIGC): A History of Generative AI from GAN to ChatGPT”, arXiv (2023). doi: 10.48550/arXiv.2303.04226.

(2) 吉川和輝: “AI に人間らしさをもたらした大規模言語モデル”, 日経サイエンス, vol. 53, no. 5, pp. 32-39 (2023).

(3) Dwivedi, Y. K., Kshetri, N., Hughes, L. et al., “‘So what if ChatGPT wrote it?’ Multidisciplinary perspectives on opportunities, challenges and implications of generative conversational AI for research, practice and policy”, International Journal of Information Management, vol. 71, no. 102642, pp. 1–63 (2023). doi: 10.1016/j.ijinfomgt.2023.102642.

(4) Brown, T. B., Mann, B., Ryder, N., Subbiah M. et al.: “Language Models are Few-Shot Learners”, arXiv (2020). doi: 10.48550/arXiv.2005.14165.

(5) Dohmke, T.: “GitHub Copilot X: The AI-powered developer experience”, The GitHub Blog (2023). https://github.blog/2023-03-22-github-copilot-x-the-ai-powered-developer-experience/ (閲覧日 2023 年 4 月 6 日)

(6) Dziri, N., Milton, S., Yu, M. et al.: “On the Origin of Hallucinations in Conversational Models: Is it the Datasets or the Models?”, arXiv (2022). doi: 10.48550/arXiv.2204.07931.

(7) Ji, Z., Lee, N., Frieske, R. et al.: “Survey of Hallucination in Natural Language Generation,” ACM Comput. Surv., vol. 55, no. 12, pp. 1–38 (2023). doi: 10.1145/3571730.

(8) 赤堀侃司, “プログラミング教育の現状についての考察”, CRET 年報, no. 2, pp. 19-34 (2017).

(9) Wing, J. M., 中島秀之(訳): “Computational Thinking 計算論的思考”, 情報処理, vol. 56, no. 6, pp. 584-587 (2015).

(10) 深沢隆司: “SE の教科書”, 完全版. 技術評論社 (2009).

(11) Bordt, S. and von Luxburg, U.: “ChatGPT Participates in a Computer Science Exam”, arXiv (2023). doi: 10.48550/arXiv.2303.09461.

(12) Sareini, M.: “Building Ghostwriter Chat”, Replit Blog (2023). https://blog.replit.com/ghostwriter-building (閲覧日 2023 年 4 月 5 日).

(13) Wermelinger, M.: “Using GitHub Copilot to Solve Simple Programming Problems”, in Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1, in SIGCSE 2023 pp. 172–178 (2023) doi: 10.1145/3545945.3569830.

(14) 情報処理学会情報処理教育委員会: “日本の情報教育・情報処理教育に関する提言 2005（2006.11 改訂/追補版）” (2006). http://www.ipsj.or.jp/12kyoiku/teigen/v81teigen-rev1a.html (閲覧日 2023 年 3 月 29 日)

(15) 東京大学数理・情報教育研究センター: “Python プログラミング入門” (2020). https://utokyo-ipp.github.io/ (閲覧日 2023 年 4 月 7 日).

(16) 岡本雅子, 喜多一: “プログラミングの「写経型学習」における初学者のつまずきの類型化とその考察”, 滋賀大学教育学部附属教育実践総合センター紀要, vol. 22, pp. 49–53 (2014).

(17) Wei, J., Wang, X., Schuurmans, D. et al.: “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models”, arXiv (2023). doi: 10.48550/arXiv.2201.11903.

(18) Welsh, M.: “The End of Programming”, Commun. ACM, vol. 66, no. 1, pp. 34–35 (2023). doi: 10.1145/3570220.

(19) サム・ポトリッキオ: “ChatGPT は大学教育のレベルを間違いなく高める――「米国最高の教授」がそう言い切るこれだけの理由”, Newsweek 日本版 (2023). https://www.newsweekjapan.jp/sam/2023/03/chatgpt.php (閲覧日 2023 年 4 月 5 日)

(20) OpenAI: “OpenAI Cookbook”, OpenAI (2022). https://github.com/openai/openai-cookbook (閲覧日 2023 年 4 月 7 日)

(21) Eshach, H.: “Bridging In-school and Out-of-school Learning: Formal, Non-Formal, and Informal Education”, J Sci Educ Technol, vol. 16, no. 2, pp. 171–190 (2007). doi: 10.1007/s10956-006-9027-1.

(22) Cole, S.: “ChatGPT Users Report Being Able to See Random People’s Chat Histories”, Vice (2023). https://www.vice.com/en/article/5d9zd5/chatgpt-users-report-being-able-to-see-random-peoples-chat-histories (閲覧日 2023 年 4 月 5 日)

(23) Touvron H., Lavril, T., Izacard, G. et al.: “LLaMA: Open and Efficient Foundation Language Models”, arXiv (2023). doi: 10.48550/arXiv.2302.13971.