PP-Checker: プログラミング教育における大規模言語モデルと協調した曖昧性のある自動採点システム

関口 祐豊 * 中村 聡史 *

概要. 本論文では，プログラミング教育における採点業務の効率化を目指し，即時性，採点精度，曖昧性に焦点を当てた自動採点システム PP-Checker を提案する．PP-Checker は，大規模言語モデル（LLM）との協調により，動的かつ視覚的なプログラミング言語の自動採点を実現する．また，リアルタイムでプロンプトを調整できる機能を備えており，学生が課題に取り組んでいる間にも採点基準を更新し，結果をすぐに反映できる．実際の講義で計 2,400 分運用した結果から，PP-Checker は課題の再提出時間を平均 3.6 分まで短縮し，課題採点業務の効率化に貢献することが示された．

1 はじめに

大学における必修プログラミング教育は，数十人から数百人規模の学生を一度に指導することが求められる [43, 29]．こうした大規模な教育現場において，大学院生が TA(Teaching Assistant) として教育補助業務を行っている [41]．例えば，我々の所属する明治大学総合数理学部先端メディアサイエンス学科では，毎年 10 名程度の TA と 4 名の教員が協力し，120 名以上の学生が受講する必修のプログラミング演習を担当しており，講義や課題提示，質問対応や課題採点などを行っている．こうしたプログラミング教育において学生・TA・教員の人数比に偏りがある場合，適切なコミュニケーションが難しく，講義運営が円滑に進まないことがある．我々の学科では，プログラムタイピングシステム [42] を開発・運用しており，初学者の基礎力向上には寄与しているが，講義運営の改善には十分でない．

プログラミング演習講義では，学生は提示された課題に取り組むことが一般的である．ここで，課題の迅速な採点が学生の学習意欲の向上につながることが知られている [6, 39]．しかし，TA や教員の人数に対して学生の人数が多い場合，随時発生する質問対応を行いつつ課題の採点を行うのは容易ではなく，全学生の課題を迅速に採点できない [4]．

自動採点は課題採点業務を効率化する方法の一つであり，フィードバックの迅速化だけでなく，学生の動機付けやスキル向上にも寄与することが知られている [8]．実際，自動採点システムとして，定義されたテストケースや短冊型問題 [30] に対応する手法 [17, 22, 31, 37] や，構文木を用いた自動採点および理解度評価の研究 [33, 36] が行われてきた．しかし，教員にとって課題の準備に加えてテストケースを用意することは，多くの時間と労力が必要となり，負担が大きい．また，Processing [23] のようなインタラクティブなプログラミング言語に対する採点では，テストケースを用意することが難しい．

ここで，大規模言語モデル（LLM）の急速な普及により，特にアルゴリズムやデータ構造の課題に対する自動採点において，様々なアプローチが試みられている [3, 19, 34]．しかし，Processing [23] などのインタラクティブなプログラミング言語においては，LLM がハルシネーション [11] を起こすリスクが高いことが指摘されており [26]，十分な精度を満たすためにはプロンプトの工夫が必要となる．

そこで本論文では，人間と LLM の協調によってインタラクティブなプログラミング言語に対応し，課題採点業務の効率化を目的とした曖昧性を許容する自動採点システムである PP-Checker（図 1）を提案する．本システムを 12 回にわたるプログラミング演習講義で実運用することで，その有用性と課題を明らかにする．

2 関連研究

宮下ら [32] の MOTIVATION や，Resnick ら [24] の Scratch，増井ら [40] が提唱した全世界プログラミングなど，プログラミング学習のモチベーションを高める研究は多く行われている．また，Greenberg ら [9] は，Processing [23] が Java と比較して学習者のモチベーションを高めることを明らかにした．Salga ら [25] は Processing.js を提案し，教育やアートの分野での広範な利用可能性を示した．Terroso ら [27] は，非プログラマ向けに p5.js [1] を用いた授業において，インタラクティブなプログラムが学生の関心や評価を高めることを明らかにした．第二著者が実施するプログラミング演習の講義においても，学生の関心を高めるため，錯視を取り入れた視覚的なプログラムの作成や，統計量の可視化，インタラクティブなゲームの作成といったように課題へ

* 明治大学

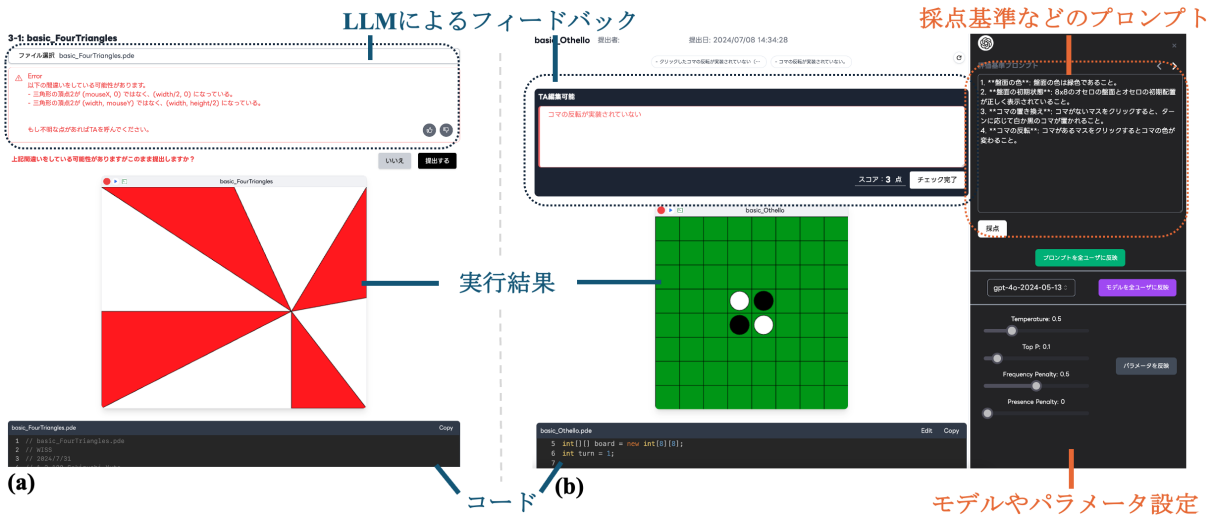

図 1. PP-Checker の 2 つの主要な画面．LLM による自動採点機能を備えた課題提出画面（a），手動チェック画面（b）．

のモチベーションを高める工夫を行っている[1]．

LLM を活用したプログラミング支援も活発に行われている．Kazemitabaar ら [15] は，AI コードジェネレータである CodeX [5] が初学者のプログラミング学習においてコードの完成率やスコアを向上させることを示したが，同時にツール依存を避け，スキル向上のための LLM の適切な使用が重要であると指摘していた．Jonsson ら [13] は，AI がプログラミング教育に有効である一方，予測不可能な振る舞いが学習体験に与える影響に注意が必要であることを示唆した．Abolnejadian ら [2] は，GPT[21, 20] を用いた教材カスタマイズが学生の理解度を向上させることを示しつつ，誤情報の訂正には教師の役割が不可欠であるとした．Kabir ら [14] も同様に，LLM の包括性と明瞭さを評価しつつ，人間と AI の協力の重要性を強調した．また，LLM によるフィードバックは，従来のテストベースのフィードバックを補完する形で効果的であることが示されている [7]．さらに，CodeAid [16]，KOGI [35]，ChotGPT [38] のように LLM を用いたエラー解決方法の研究も進展している．

これらの研究を踏まえ，PP-Checker は人間と LLM の協調による自動採点を実現し，課題採点業務の負担を軽減しつつ，迅速かつ適切なフィードバックを提供することを目指している．また，インタラクティブなプログラミング言語 Processing を使用した講義において，実運用可能な採点システムとして，プログラミング教育の円滑化を図る．

3　提案手法

学生や TA，教員の人数比の偏りにより，TA や教員が全学生の質問対応をしつつ課題採点を行うことは負荷が高い．また，学生は迅速なフィードバックを求めており [6, 39]，こうした即時性のニーズに応えることがプログラミング教育の質向上において重要な要素である．一方，Processing [23] などのインタラクティブなプログラミング言語は動的要素を含むことから，従来の採点手法では正確な評価が困難であるため，採点精度および曖昧性を考慮することが重要である．

そこで，システムの実現にあたり，即時性，採点精度，曖昧性の観点に着目した仕組みについて述べる．

3.1　フィードバックの即時性を目指した仕組み

従来の手動採点ではフィードバックが遅れることが多く，学生の学習意欲や効果に悪影響を及ぼす可能性がある．そこで，LLM を活用して自動採点を行うことにより，提出されたコードの誤りを即座に指摘することを可能とする．この仕組みにより，学生は TA の採点を待たずに自身のコードの潜在的な問題点を早期に発見し，再提出することが可能となる．

3.2　高い採点精度を目指した仕組み

先述の通り，自動採点では複雑なテストケースを作成し，提出されたコードが期待通り動作するかを評価する必要がある．しかし，視覚的・インタラクティブなプログラム課題においてはテストケースの準備に多くの時間と労力がかかり，柔軟性にも限界があるうえ，様々な記述形式が想定されるため，事前にそのすべてを予測することは困難である．

そこで，LLM とプロンプト[2]を用いた採点手法を導入することで，テストケースの作成を不要とする．しかし，LLM の出力は完全ではなく，採点精度向上にはプロンプト設計が重要である [44] ため，TA

[1] https://lecture.nkmr.io

[2] https://github.com/YutoSekiguchi/ExPrompt

図 2. 課題一覧画面

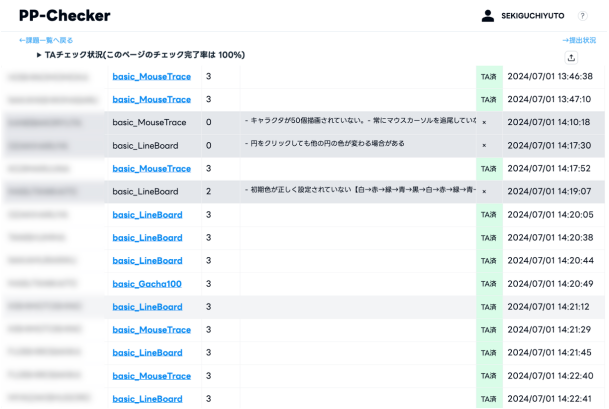

図 3. 提出物一覧画面

や教員が LLM のプロンプトをリアルタイムに変更できる機能も導入する．具体的には，採点中にプロンプトを修正すると，更新されたプロンプトに基づいた採点が未採点の課題に即座に反映される仕組みを構築する．これにより教員は，講義開始前に完璧なプロンプトを準備する必要がなく，採点精度を講義中にリアルタイムで調整することが可能になる．

3.3 曖昧な判定を許容する仕組み

LLM による自動採点は当然判定ミスも多くなると考えられる．そこで，学生に対して LLM の指摘を受け入れるか否かを選択する機会を提供する．これにより，学生は LLM の指摘を参考にしつつ，自らの判断を信頼して学習を進めることが可能になる．このアプローチは，LLM などシステムへの過度な依存を避け，自立性と判断力の強化にもつながることが期待される．

さらに LLM を用いた指摘において，具体的な正答を示してしまうと学生の問題解決能力を高めることができない．そこで LLM の出力結果を工夫することで，学生が自ら考えを深め，問題解決能力を高めること [18] を目指す．

4 PP-Checker

提案手法を講義において実運用することを目指し，課題提出，実行，採点，コード管理を一元化した Web アプリケーションとして実装を行った．本システムにより TA や教員は複数のツールを使い分けることなく，一つのプラットフォーム上で全ての課題採点業務を行うことができ，業務の効率化が期待される．PP-Checker は，課題一覧画面，課題提出（自動採点）画面，提出物一覧画面，手動チェック画面の 4 つの主要な画面から構成されている．

4.1 課題一覧画面

課題一覧画面（図 2）は，PP-Checker のホーム画面であり，学生は各課題の進捗状況，提出状況，フィードバックの有無，過去に提出したコードを確認できる．各課題項目をクリックすることで課題提出画面に移動し，コードの提出を行うことができる．これにより，学生は課題管理が容易になり，学習の効率が向上することが期待される．

4.2 課題提出（自動採点）画面（学生用）

課題提出画面（図 1a）では，学生が自身のプログラムコードを提出し，LLM が採点を行う．具体的には，学生はコードを含むフォルダをアップロードし，提出前にその場で LLM による自動採点を受けることができる．LLM はアップロードされたコードを即座に採点し，誤りの可能性がある箇所を指摘するフィードバックを生成する．このフィードバックをもとに学生は自身のコードの潜在的な問題点を TA に提出する前に把握し，コードの修正や再提出が可能となる．また，この画面には提出されたコードや実行結果も表示され，学生は自分のコードがどのように動作するかを改めて確認できる．

4.3 提出物一覧画面

提出物一覧画面（図 3）は，TA や教員が提出された課題を一覧で確認できる．この画面では，各課題の提出状況，採点状況，フィードバックの有無，どの TA がどの課題をチェックしているかがリアルタイムで表示される．各提出物の課題名をクリックすることで，手動チェック画面に移動し，LLM が行った初期採点を TA や教員が最終確認できる．

4.4 手動チェック画面（TA・教員用）

手動チェック画面（図 1b）は，TA や教員が LLM による採点結果を確認し，必要に応じて補足や修正を行うことができる．画面左側には，LLM が生成したフィードバックを確認・修正できるインタフェースがあり，その下にコードや実行結果も表示されるため，TA は学生のコードの動作を確認しながら課題採点を行うことができる．また，画面上部には，

他の提出物に対するフィードバックコメントを選択できるインタフェースが設置されており，これを再利用することで，入力の手間を省き，効率的な課題採点業務をサポートする．画面右側では，LLM のプロンプトやモデルの変更，パラメータの調整を行うことができ，リアルタイムで採点精度を高めることが可能である．また，プロンプトは TA や教員が容易に変更できるように，課題の満たすべき条件を箇条書きで記述できる形式にしている．

4.5 実装

PP-Checker は，フロントエンドに Next.js と TypeScript，バックエンドに FastAPI，データベースに MySQL を用いて実装した．LLM は OpenAI 社の gpt-3.5-turbo，gpt-4-turbo-preview，gpt-4-turbo，gpt-4o を選択できるような設計にしている．また，リアルタイム通信には Socket.io を利用しており，学生と TA の間での迅速なやり取りを可能にしている．

Processing [23] などのインタラクティブな言語は，動的な要素を含むため静的なコードよりも複雑な評価が求められる．そこで，Processing.js [25] を用いて実行画面を描画することで，学生が提出したコードの自動採点と視覚的な確認を可能にするとともに，マウスやキーボードのクリックや画像，学生ごとに異なるデザインを含む課題でも正確なフィードバックを提供する．Processing.js で対応していない circle や enum などの関数については，自作の関数を作成し対応している．なお，学生や TA を識別するため，大学発行のメールアドレスによる認証を用いて，ログインを行うようにしている．

5 運用と分析

5.1 運用形態

PP-Checker を，明治大学総合数理学部先端メディアサイエンス学科 1 年次対象の必修科目であるプログラミング演習 I（100 分 2 コマ）において，2024 年 4 月 15 日から 7 月 22 日までの 12 回分の講義（計 2400 分）で運用した．履修者は学部 1 年生と再履修者の 123 名で，TA は筆頭著者を含む大学院生 10 名，教員は第二著者を含む 4 名であった．

講義では，教員が冒頭の数十分間でスライドを用いて説明を行い，その後，4〜5 つの課題が提示される．課題は難易度に応じて 2 種類に分かれ，基本課題は講義時間内に，難易度の高い発展課題は，次回授業開始時までに提出する必要がある．これらの課題の提出先を PP-Checker にすることで運用を行った．

5.2 運用結果と分析

運用期間中，基本課題 34 問と発展課題 17 問の計 51 問の課題を実施した．本研究では，授業内での採点業務の効率化を目的としているため，分析対象を基本課題に限定する．基本課題において，PP-Checker を通じて合計 6,415 回の提出が行われた．

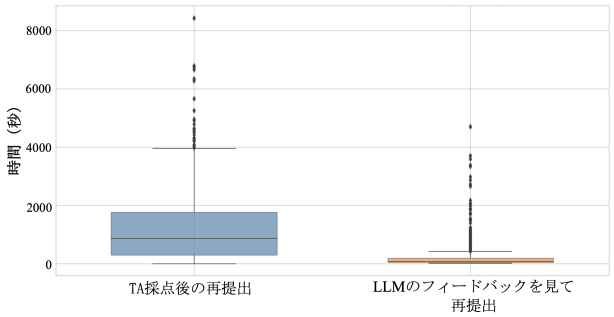

図 4. 再提出までの時間の比較

PP-Checker の導入による迅速なフィードバックの効果を定量的に評価するため，学生がフィードバックを受けてから再提出までの時間的変化や提出回数について分析を行った．分析の結果，6,415 回の提出のうち，1,491 回（約 23.2%）は TA 採点前に LLM によるフィードバックを受けた時点で，学生が自主的に課題を取り下げてコードの改善を行っていることが明らかになった．図 4 に，再提出までの時間の分布を箱ひげ図で示す．TA 採点後に再提出された件数は 634 件であったのに対し，LLM によるフィードバックを受けて自主的に修正し TA に再提出された件数は 886 件であり，学生が早期フィードバックを活用していることが明確となった．再提出までの平均時間は，TA 採点後の再提出が約 20.3 分であったのに対し，TA 採点前に LLM のフィードバックを受けて修正した場合は約 3.6 分であり，再提出のスピードが著しく向上していることがわかった．

プロンプト変更機能の使用状況について，収集を開始した第 5 回講義以降の基本課題 24 問について分析を行った結果，合計 82 回のプロンプト変更が観察された．プロンプト変更前後の LLM の採点正解率を比較したところ，変更前が 60.4%（標準偏差：23.7），変更後は 60.7%（標準偏差：17.8）とわずかな向上が見られたものの，有意な差は確認されなかった．全体的な正解率に有意な差は見られなかった一方で，課題ごとの分析では，プロンプト変更の効果に顕著な変動があることが確認された．具体的には，初期精度が 60%未満のプロンプトに変更を加えた場合，90%近くのケースで精度が向上することが確認された．最も改善した課題では，初期精度 33.3%のプロンプトが変更後に 72.9%まで向上し，約 40%の精度向上が確認された．一方，初期精度が 80%以上あった課題では，プロンプト変更後にその精度が低下するケースが大半であった．

5.3 SUS と学生アンケートの結果

114 名の学生を対象に，System Usability Scale（SUS）[12] を用いた PP-Checker のユーザビリティ

表 1. TA アンケート結果

質問項目	評価値の分布					平均
	-2	-1	0	1	2	
PP-Checker は全体的に使いやすいと思いましたか？	0	0	0	2	7	1.78
PP-Checker をこれからも利用したいと思いますか？	0	0	0	1	8	1.89
PP-Checker 導入前後で、課題採点業務の作業効率は向上しましたか？	0	0	0	0	6	2.00

と信頼性を評価するアンケート調査を実施した．

調査の結果，PP-Checker の SUS スコアは 76.4 となった．これは平均スコア（68.0）を大きく上回っており，システムのユーザビリティが高く評価された．しかし，標準偏差は 12.2 であり学生間の評価には一定のばらつきが見られた．

PP-Checker の利用に関するアンケートの分析では，授業の進行に伴い，学生が LLM のフィードバックからコードの誤りに気づいたり，修正のヒントを得たりする経験が徐々に減少する傾向にあった．しかし，学期後半においても 57%以上の学生が，フィードバックの半分以上が有用だったと回答していた．参考になったフィードバックには,「円の動作が毎フレーム X 方向に 3 ピクセル、Y 方向に 2 ピクセル動いていない（Y 方向に 2 倍動いている）。」や「A さんのダイスの範囲が 1 から 6 ではなく、1 から 5 になっている。」といった具体的な誤りの箇所を含むフィードバックがあげられた（図 5）．クリックするたびにじゃんけんの結果を標準出力する課題では,「judgeJanken 関数内の条件で、パーとグーの勝敗判定が間違っている。」といった，複数回操作しないと気づきにくい誤りを指摘するフィードバックも有用と評価された．さらに,「枠線が削除されていない」といった，一見しただけでは見落としがちな問題を指摘するフィードバックも高く評価された．

一方で，参考にならないと評価されたのは，正確性に欠けるフィードバックや課題を解く上でプログラムの挙動に影響しない指摘であった．例えば,「標準偏差の計算において、平均値を毎回再計算しているため、効率が悪い。」といった課題の評価に影響しない指摘は，参考にならなかったと評価された．

5.4 TA アンケートの結果

PP-Checker の運用終了後，筆頭著者を除く 9 名の TA を対象にアンケート調査を実施した．アンケートは，PP-Checker の使用感や作業効率に関する定量評価項目（表 1）と，質的データ収集のためのインタビュー項目で構成した．定量評価は 5 段階のリッカート尺度（-2：全くそう思わない〜2：非常にそう思う）を用いた．インタビューでは，LLM との協調や PP-Checker 導入による TA 業務の変化に焦点を当てた．PP-Checker 導入前後の比較は，導入前の TA 経験を持つ 6 名のみを対象とした．

表 1 より，PP-Checker の使用感や作業効率に関

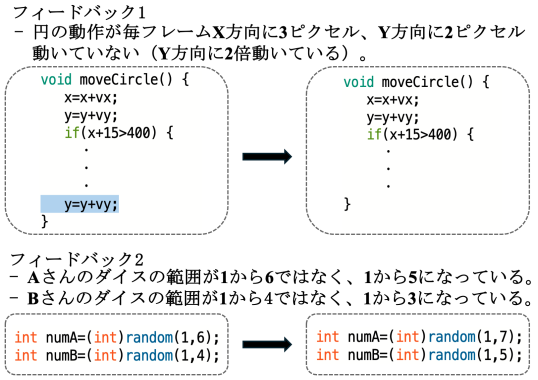

図 5. 参考になった LLM のフィードバックと修正の例

する全ての項目において高い評価を得ることができていることがわかった．インタビュー項目では,「採点作業の効率化により，採点にかけていた時間を質問対応に回せるようになった」や「GPT のフィードバックがあることで基礎的な質問が減少した」,「プロンプトの変更により採点精度が向上した」といったポジティブな意見が多く得られた．

一方で，動作順序が重要な課題において精度にばらつきが多く見られ，課題内容による LLM の精度の差に関するネガティブな意見も一部見られた．

5.5 教員アンケートの結果

PP-Checker の運用終了後，第二著者を除く 2 名の教員を対象にアンケート調査を実施した．アンケートは，PP-Checker の利点や改善点，TA や学生に感じた変化などを調査するための項目で構成した．調査の結果，PP-Checker の利点として，LLM のフィードバックが学生自身の確認時に注目すべきポイントを明確となる点や LLM のフィードバックによって学生が誤りに気づいて修正する周期が早くなったという意見が得られた．一方，段階的な減点条件を定めたい場合のプロンプト作成が難しかったという意見が得られた．

TA に対して感じた変化としては，以前は多かった氏名や課題名の記入のような誤りの指摘をしなくて良くなったことでストレスが減少したという意見が得られた．また，不備を指摘することが機械であることで，TA や教員側だけでなく学生側のストレスも下がっているという意見が得られた．しかし，自発的な興味が確立していない初学者に対する LLM

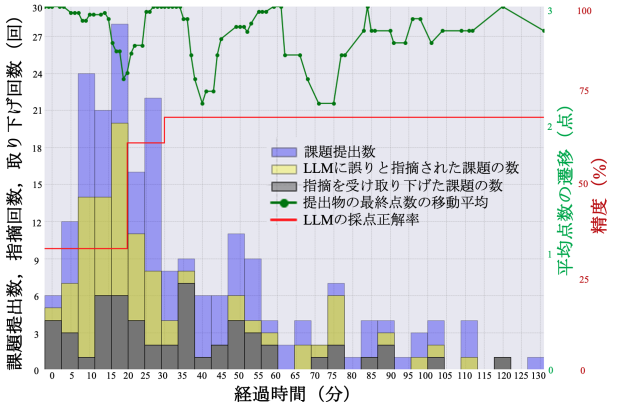

図 6. ある課題における提出数（青棒），LLM の指摘数（黄棒），課題取り下げ回数（灰棒），点数推移（緑），LLM フィードバックの正解率（赤）

の依存を危惧するような意見もあった．

PP-Checker を今後も運用したいかという質問では，2 名ともに非常にそう思うと肯定的な回答をした．

6　考察と議論

6.1　学生に与える影響に関する考察と議論

PP-Checker を利用することで，再提出までの時間が大幅に短縮されたことから，即時性のあるフィードバックが学生の学習プロセスに大きな影響を与えることが示唆された．これは，学生が自身の誤りを素早く認識し，改善する機会を得ており，学習サイクルの加速につながっていると考えられる．LLM からのフィードバックを受けて自ら修正を行うプロセスは，単にプログラミングスキルの向上だけでなく，問題解決能力全般の改善にもつながると考えられる．さらに，SUS の結果から，システムへの満足度や有用性が高いことが示唆された．

図 6 は，ある基本課題における時間経過に伴う提出状況，採点精度，点数の推移を示している．この結果では最初の 20 分は精度が 30%程度と低いが，30 分までに 2 回プロンプトの修正がなされ 70%近くまで向上している．このプロンプトの修正は，最初の 20 分程度で提出された課題の自動採点およびその結果を無視して提出されたものにおける自動採点の結果を参考に行われている．プロンプト修正の主な例として，「**クリック時の再描画**: 画面をクリックするたびに **a, b, c, d の値が 1〜20 の間でランダムに決定され**、曲線が再描画されていること。」という条件の太字部分のような未定義パラメータの具体化，出力形式の明確化，数式の具体的な指定といった変更が挙げられる．さらに，採点精度向上のため，明示的な例外規定（例：「1. 円が端付近に到達すると跳ね返るように実装されていない。円の半径分を考慮していない場合でも、課題の要件には明記されていないため、減点対象にはしません。」）などのプロンプト変更が行われていた．これらの修正により，プロンプトの曖昧さを授業の進行に伴い軽減し，LLM の反復的な誤評価を抑制することで，教育効果の向上につながる可能性が考えられる．

プロンプトがまだ不十分な段階で精度が低い時に課題を提出する学生は基本的にプログラミング能力が高く，また自信があるため，精度が低くても大きな問題はない．一方で，中盤に課題を提出するプログラミングにおける自信があまりない学生は精度が低い場合にその影響を受けてしまう可能性がある．つまり，今回のように採点精度は時間とともに徐々に向上することが望ましく，今回の手法はこうした講義運営において適切であったと考えられる．

6.2　TA・教員に与える影響に関する考察と議論

TA へのアンケート結果から，PP-Checker の導入が課題採点業務の効率化につながることが示唆された．また，LLM による即時フィードバックにより，学生は初歩的なミスを自ら修正し，より完成度の高い状態で提出を行うようになった．これにより，TA が同じような間違いを含む提出物を繰り返し確認する手間が減少し，学生の質問対応に多くの時間を割けるようになることが示唆された．一方で，プロンプトのリアルタイム変更に関する分析結果から，課題によって精度が大きく異なることや精度の高いプロンプトを変更することで精度が下がってしまう可能性があること確認された．そこで今後は，学生が LLM の判定を無視したときに，その情報を利用してプロンプトの自動修正を行う仕組みを実現する予定である．

また教員へのアンケート結果から，PP-Checker が教員の期待に十分に応えていることが示唆された．しかし，初学者の学生が LLM に依存してしまうこと [28, 10, 15] への懸念を示した意見があったことから，今後はこのような不安を解消しつつ，LLM，学生，TA，教員がそれぞれの役割を最大限に発揮できるようなインタフェース設計を目指す．

7　まとめ

本論文では，プログラミング教育における課題採点業務の効率化を目指した曖昧性のある自動採点システム PP-Checker を提案し，その有用性を検証した．実運用の結果，PP-Checker は学生，TA，教員の全てから高評価を得ており，課題採点業務の効率化とフィードバックの迅速化に貢献した．特に，LLM による早期フィードバックが学生の再提出までの時間を大幅に短縮し，TA が学生の質問対応に時間を多く割けるようになった．

今後は，本システムの欠点を徹底的に洗い出し，その改善や新たな知見の深掘り，新たなプロンプト構築手法の検討を行っていきたいと考えている．

謝辞

本システムを講義で活用してくださった明治大学総合数理学部先端メディアサイエンス学科の 12 期生および，対応していただいた TA，教員のみなさまに感謝します.

参考文献

[1] p5.js. `https://p5js.org/` (2024/8/31 確認).

[2] M. Abolnejadian, S. Alipour, and K. Taeb. Leveraging ChatGPT for Adaptive Learning through Personalized Prompt-based Instruction: A CS1 Education Case Study. *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems(CHI EA'24)*, (521):1–8, 2024.

[3] R. Balse, B. Valaboju, S. Singhal, J. M. Warriem, and P. Prasad. Investigating the Potential of GPT-3 in Providing Feedback for Programming Assessments. *In Proceedings of the Conference on Innovation and Technology in Computer Science Education (ITiCSE'23)*, pp. 292–298, 2023.

[4] D. Boud and E. Molloy. *Feedback in Higher and Professional Education*. Routledge, 2012.

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

[6] J. Clune, V. Ramamurthy, R. Martins, and U. A. Acar. Program equivalence for assisted grading of functional programs. *In the Proceedings of the ACM on Programming Languages*, 4(171):1–29, 2020.

[7] H. Gabbay and A. Cohen. Combining LLM-Generated and Test-Based Feedback in a MOOC for Programming. *In Proceedings of the Eleventh ACM Conference on Learning @ Scale (L@S'24)*, pp. 177–187, 2024.

[8] A. Gordillo. Effect of an Instructor-Centered Tool for Automatic Assessment of Programming Assignments on Students' Perceptions and Performance. *Sustainability*, 11(20):1–24, 2019.

[9] I. Greenberg, D. Kumar, and D. Xu. Creative coding and visual portfolios for CS1. *In Proceedings of the 43rd ACM technical symposium on Computer Science Education (SIGCSE'12)*, pp. 247–252, 2012.

[10] M. Jakesch, A. Bhat, D. Buschek, L. Zalmanson, and M. Naaman. Co-Writing with Opinionated Language Models Affects Users' Views. *In Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI'23)*, (111):1–15, 2023.

[11] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. J. Bang, A. Madotto, and P. Fung. Survey of Hallucination in Natural Language Generation. *In ACM Computing Surveys*, 55(12):1–38, 2023.

[12] B. John. SUS: A 'Quick and Dirty' Usability Scale. *Usability Evaluation In Industry*, p. 207–212, 1996.

[13] M. Jonsson and J. Tholander. Cracking the code: Co-coding with AI in creative programming education. *In Proceedings of the 14th Conference on Creativity and Cognition (CC'22)*, pp. 5–14, 2022.

[14] S. Kabir, D. N. Udo-Imeh, B. Kou, and T. Zhang. Is Stack Overflow Obsolete? An Empirical Study of the Characteristics of ChatGPT Answers to Stack Overflow Questions. *In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'24)*, pp. 1–17, 2024.

[15] M. Kazemitabaar, J. Chow, C. K. T. Ma, B. J. Ericson, D. Weintrop, and T. Grossman. Studying the effect of AI Code Generators on Supporting Novice Learners in Introductory Programming. *In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'23)*, (455):1–23, 2023.

[16] M. Kazemitabaar, R. Ye, X. Wang, A. Z. Henley, P. Denny, M. Craig, and T. Grossman. CodeAid: Evaluating a Classroom Deployment of an LLM-based Programming Assistant that Balances Student and Educator Needs. *In Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems (CHI'24)*, (650):1–20, 2024.

[17] S. Krusche and A. Seitz. ArTEMiS: An Automatic Assessment Management System for Interactive Learning. *In Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE'18)*, pp. 284–289, 2018.

[18] T. J. Ngoon, C. A. Fraser, A. S. Weingarten, M. Dontcheva, and S. Klemmer. Interactive Guidance Techniques for Improving Creative Feedback. *In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI'18)*, (55):1–11, 2018.

[19] H. Nguyen and V. Allan. Using GPT-4 to Provide Tiered, Formative Code Feedback. *In Proceedings of the 55th ACM Technical Symposium on Computer Science Education (SIGCSE'24)*, pp. 958–964, 2024.

[20] OpenAI. GPT-4 Technical Report. *OpenAI Blog*, pp. 1–100, 2023.

[21] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, and R. Lowe. Training language models to follow instructions with human feedback. *In Advances in Neural Information Processing Systems*, pp. 1–15, 2022.

[22] R. A. P. Queirós and J. P. Leal. PETCHA: a programming exercises teaching assistant. *In Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education (ITiCSE'12)*, pp. 192–197, 2012.

[23] C. Reas and B. Fry. Processing: a learning environment for creating interactive Web graphics. *In ACM SIGGRAPH 2003 Web Graphics(SIGGRAPH'03)*, 2003.

[24] M. Resnick, A. M.-H. John Harold Maloney, N. Rusk, E. Eastmond, K. A. Brennan, A. Millner, E. Rosenbaum, J. S. Silver, B. S. Silverman, and Y. B. Kafai. Scratch: Programming for All. *Communications of ACM 52*, pp. 60–67, 2009.

[25] A. Salga, D. Hodgin, A. Sobiepanek, S. Downe, M. Medel, and C. Leung. Processing.js: sketching with ¡canvas¿. *In ACM SIGGRAPH 2011 Talks (SIGGRAPH '11)*, (15):1, 2011.

[26] A. Singla. Evaluating ChatGPT and GPT-4 for Visual Programming. *In Proceedings of the 2023 ACM Conference on International Computing Education Research (ICER'23)*, 2:14–15, 2023.

[27] T. Terroso and M. Pinto. Programming for Non-Programmers: An Approach Using Creative Coding in Higher Education. *In Third International Computer Programming Education Conference (ICPEC'22)*, 102(13):1–8, 2022.

[28] X. Wang, H. Kim, S. Rahman, K. Mitra, and Z. Miao. Human-LLM Collaborative Annotation Through Effective Verification of LLM Labels. *In Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems (CHI'24)*, (303):1–21, 2024.

[29] L. Yan, N. McKeown, and C. Piech. The PyramidSnapshot Challenge: Understanding student process from visual output of programs. *In Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE'19)*, pp. 119–125, 2019.

[30] 角田博保, 久野靖. 短冊型問題: プログラミングの技能を評価可能な試験出題形式. 夏のプログラミング・シンポジウム 2016「教育・学習」報告集, 2016:73–80, 2016.

[31] 久野靖. 短冊型問題を用いたプログラミング学習アドバイスツール. 情報処理学会 プログラミングシンポジウム 報告集, pp. 129–139, 2020.

[32] 宮下芳明, 中橋雅弘. 学習者のモチベーション向上のための好意的解釈を行うフィジカルコンピューティング環境のデザイン. ヒューマンインタフェース学会論文誌, 13(4):303–313, 2011.

[33] 漆原宏丞, 本多佑希, 岸本有生, 兼宗進. 抽象構文木を利用したプログラミング理解度採点の試み. 情報処理学会 研究報告コンピュータと教育（CE）, 2021-CE-162(16):1–6, 2021.

[34] 若谷彰良, 前田利之. 生成 AI を用いた C 言語プログラミング学習のための助言システムの試作. 甲南大学紀要 知能情報学編, 16(2):7–16, 2024.

[35] 小原有以, 佐藤美唯, 倉光君郎. KOGI: ChatGPT を Colab に統合したプログラミング演習支援. 情報処理学会 情報教育シンポジウム論文集, pp. 141–148, 2023.

[36] 小川弘迪, 小林亜樹. プログラミング課題の自動採点に向けた構文木上のカーネル法による類似度関数の提案. 情報処理学会 第 80 回全国大会講演論文集, 2018(1):661–662, 2018.

[37] 新田章太, 小西俊司, 竹内郁雄. 複数言語に対応しやすいオンラインプログラミング学習・試験システム track. 情報処理学会 情報教育シンポジウム論文集, pp. 114–121, 2019.

[38] 森陽菜, 松澤芳昭. ChatGPT を利用したプログラミング教育支援システム「ChotGPT」の提案と評価. 情報処理学会 研究報告コンピュータと教育（CE), 2024-CE-174(7):1–8, 2024.

[39] 石原浩一, 泰山裕. フィードバックと振り返りが学習者の認知欲求に及ぼす影響の検討. 日本教育工学会論文誌, 44(1):105–113, 2020.

[40] 増井俊之, 塚田浩二. 全世界プログラミング. 情報処理学会 プログラミングシンポジウム, pp. 1–8, 2006.

[41] 文部科学省. 中央教育審議会 大学分科会 制度部会（第 22 回（第 3 期第 7 回））議事録・配付資料 ［資料 2 － 1］ ティーチング・アシスタント（TA）について. `https://www.mext.go.jp/b_menu/shingi/chukyo/chukyo4/003/gijiroku/07011713/001/002.htm` (2024/8/31 確認).

[42] 又吉康綱, 中村聡史. typing.run: 初学者のプログラミング学習を支援するプログラムタイピングシステムの提案と実践. 情報処理学会 研究報告ヒューマンコンピュータインタラクション（HCI）, 2020-HCI-189(1):1–8, 2020.

[43] 又吉康綱, 中村聡史. askTA: 消極性を考慮したオンライン演習講義支援システム. コンピュータソフトウェア, 39(1):55–71, 2022.

[44] 尹子旗, 王昊, 堀尾海斗, 河原大輔, 関根聡. プロンプトの丁寧さと大規模言語モデルの性能の関係検証. 言語処理学会 第 30 回年次大会, pp. 1–6, 2024.