推薦論文● WISS2024

PP-Checker: LLMと協働する曖昧性を許容した自動採点システム

関口 祐豊　中村 聡史

プログラミング教育において，動的で視覚的なプログラム課題の採点およびフィードバックは従来の自動採点手法では困難であり，TA や教員にとって採点業務が大きな負担となっている．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，学生の課題の再提出時間を短縮し，課題採点業務の効率化に貢献することが示され，学生および TA，教員から高い評価を得ることができた．

In programming education, evaluating dynamic and visually interactive programming assignments and providing rapid feedback is challenging with traditional automated assessment methods.This paper proposes PP-Checker, an automated checking and feedback system designed to enhance evaluation efficiency by focusing on immediacy, scoring accuracy, and ambiguity.PP-Checker collaborates with large language models (LLMs) to enable automated evaluation of dynamic and visual programming languages.Additionally, it features real-time prompt adjustment capabilities, allowing instructors to update evaluation criteria and immediately reflect changes while students work on their assignments.Through 2,400 minutes of operation in actual lectures, PP-Checker significantly reduced assignment resubmission times, improved evaluation efficiency, and received high praise from both students and teaching staff (TAs and instructors).

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 [25]．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．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 [15] を運用しており，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，課題の迅速なフィードバックおよび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 [4]．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採点業務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 [3]．

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に対応する手法 [13] [18] や，構文木を用

PP-Checker: Ambiguity-Aware Automated Code Evaluation System with LLM Collaboration.

Yuto Sekiguchi, Satoshi Nakamura, 明治大学, Meiji University.

コンピュータソフトウェア, Vol.43, No.1 (2026), pp.15–26.
[研究論文] 2025 年 2 月 28 日受付.

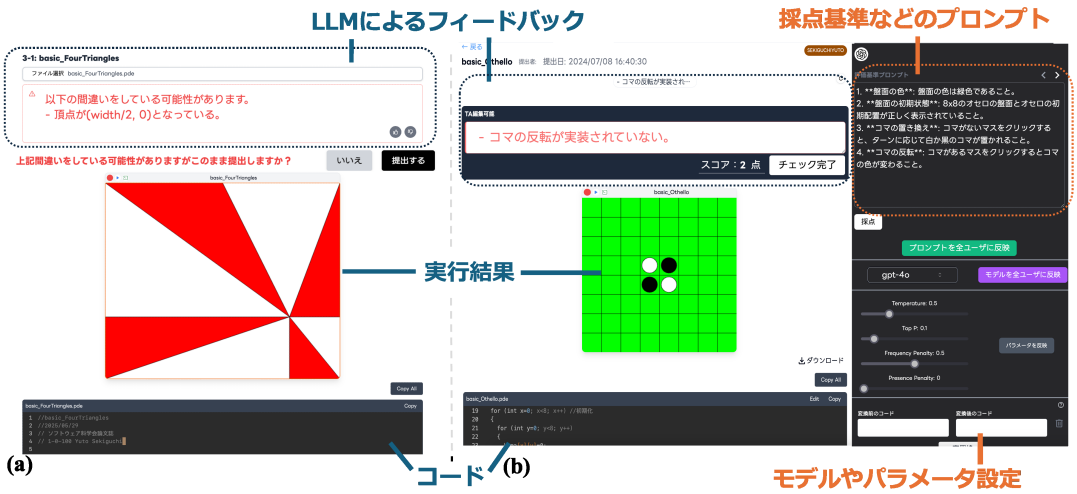

図 1　PP-Checker の 2 つの主要な画面．LLM による自動採点機能を備えた課題提出画面 (a)，手動チェック画面 (b)

いた自動採点および理解度評価の研究 [23] 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 [19] 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 [2]．しかし，Processing 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を起こすリスクが高いことが指摘されており [22]，その出力の一貫性や信頼性を確保することが難しい．そのため，動的で視覚的な要素を含むプログラムの評価においては，従来の採点手法とは異なるアプローチが求められる．

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．本システムでは，LLM を活用した自動採点により，コードの不備や仕様の未達成部分をフィードバックしたうえで，その判定結果を学生が採用，却下，保留を選択できるフローを組み込んでいる (図 1a)．この「LLM 判定を唯一解とせず，最終的に人間が結果を取捨選択できる運用上の柔軟さ」を，本論文では「曖昧性の許容」と呼ぶ．これにより，学生は TA や教員の採点を待つことなく自身のコードの問題点を早期に判断・把握し，修正に取り組むことができると考えられる．さらに，再提出までの時間 (本論文において「再提出」とは，学生が一度提出を試みたコードに対して，フィードバックを受け，修正し，再度同一課題の提出を試みる行為を指す) が短縮されるとともに，教員や TA はスクリーニングを通過した提出物のみを確認すればよくなり，採点業務の負担の軽減が期待される．本研究では，PP-Checker 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．

本研究の貢献を以下に示す．

- LLM を活用した曖昧性を許容する新しい自動採点システム PP-Checker を実装し，動的で視覚的なプログラミング課題にも柔軟に対応できる採点手法を提案・運用したこと
- 120 名以上の学生を対象に 12 回の講義で実運用を行い，SUS スコアは 76.4 と高評価であり，TA の負担を軽減するとともに，従来手法と比較して学生の平均再提出時間を約 60% 短縮できることを示したこと

本論文は，WISS2024 にて発表した論文[21] を基に，加筆したものである．

2 関連研究

2.1 自動採点システム

従来の自動採点システムでは，あらかじめ用意したテストケースによる出力の完全一致で正誤判定を行う方式が主流である[13][18]．しかし，これらはテストコードの実装が必要であり，教員の準備負担が大きい．そのため，完璧なテストケースを用意することなく，課題要件を満たしていない提出物をある程度識別可能なスクリーニング手法の研究も進められている．

Zhiyu ら[5] は模範解答と学生提出の概念グラフの差分から要件達成度を評価する手法を提案し，複数の模範解答があれば高精度な採点が可能であることを示した．Umar ら[1] は，問題文と模範解答を LLM に入力することで，正答・誤答の判別に有効なテストケースを自動生成する手法を提案した．この手法では，生成したテストケースが教員作成のものに比べ，誤答に対する識別精度が向上することが示された．

しかし，これらは，事前定義された複数の模範解答や限られた数のテストケースを前提とするため，非決定的な挙動や多様な解法を十分にカバーできない．また，採点基準を途中で変更したい場合でも，既存の方式では模範解答やテストケースを再設計する必要があり，講義中にリアルタイムで柔軟に修正を反映することが難しい．一方，PP-Checker は，課題仕様をプロンプトとして与えるだけで，非決定的な挙動をする提出物や多様な解答が存在する課題に対してもスクリーニングを行える仕組みを目指す．また，採点観点の追加・修正も講義中にプロンプトを編集するだけで全提出物に反映できるようにすることで，ユニークな解法や新たな要件の発生にも迅速に対応することを目指す．

2.2 LLM を活用したプログラミング支援

インタラクティブなプログラミング課題に対応した自動採点を実現する上で，OpenAI 社の GPT[17] などの LLM の活用が重要な役割を果たしている．さらに，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が[6]，誤情報の提供やフィードバックの正確性には改善の余地があり，この点を考慮することが求められている．Kazemitabaar ら[12] は，AI コードジェネレータである CodeX 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．Kabir ら[11] 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．さらに，NotebookGPT[7]，ChotGPT[16] 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，迅速かつ適切なフィードバックにより，課題採点業務の負担を軽減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における採点業務の効率化を図る．

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[4][9]，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 などのインタラクティブなプログラミング言語は，図形やアニメーションなど，時間的変化を伴う動的な出力を含むことから，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，GitHub Copilot や Cursor のような自動補完とは異なり，提出物が一度学生の手を離れ，TA や教員による指導的介入が可能となる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，速やかに修正や再提出を行うことが可能となる．本研究では，この再提出までの時間の短縮を，迅速なフィードバックの効果を測る主要な指標の一つとして位置づける．

3.2 高い採点精度を目指した仕組み

先述の通り，既存の自動採点ではテストケースや模範解答を用意し，出力の一致や類似度などに基づいて提出コードを評価する手法が一般的である．しかし，視覚的・インタラクティブな課題では，実装や表現の多様性が高く，模範解答との単純な比較や出力一致による評価が成り立たない場合が多い．また，チェックリスト形式による自己確認もスクリーニング手法の一つとして考えられるが，その記入内容と実際の課題達成度が一致するとは限らず，完了していないタスクの項目をチェック (Failed checks) や，完了している項目をチェックしていない (False checks)，途中までしか行っていないにもかかわらずチェックする (Inaccurate checks) といった非準拠な行動をとることが知られている [14].

そこで，LLM と課題の必要要件を列挙したプロンプトを用いた採点手法 †1 を導入することで，テストケースや必要以上の模範解答の作成を不要とする．しかし，LLM の出力は完全ではないため，TA や教員が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，修正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とともに，その修正内容が妥当かどうかを事前に確認できるテスト用フィードバック機能を備えている．この仕組みにより，LLM の挙動に対して違和感が複数の TA や教員の間で共有された場合，完全な合意形成を経ずとも，各採点者が柔軟にプロンプトを更新し，その影響を確認・共有しながら採点基準を調整することが可能となる．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 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 による指摘において，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 の出力を工夫することで，学生が考えを深め，問題解決能力を高めることを目指す．

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，課題設定画面，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 5 つの主要な画面から構成されている．

4.1 課題設定画面 (TA・教員用)

課題登録画面 (図 2) は，PP-Checker において教員や TA が課題を登録するためのインターフェースである．ユーザは，課題名やプロンプト，出題日，締切時刻などを登録することができる．また，チェックボックスを利用することで，マルチメディアファイルや標準出力の有無を選択でき，プログラム実行時に多様な課題要件の適応性を確保する．

†1 運用時のプロンプトデータ: https://github.com/YutoSekiguchi/ppc_prompt

図 2　課題設定画面

図 3　課題一覧画面

4.2　課題一覧画面

課題一覧画面 (図 3) 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．

4.3　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．

図 4　提出物一覧画面

4.4　提出物一覧画面

提出物一覧画面 (図 4) 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．

4.5　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．

4.6 実装

PP-Checker は，フロントエンドに Next.js，バックエンドに FastAPI，データベースに MySQL を用いて実装した．全体のコード量は約 34,000 行程度であり，そのうちフロントエンドが約 24,000 行，バックエンドが約 10,000 行程度で構成されている．サーバはさくらインターネットの VPS(6 コア，8 GB RAM，780 GB SSD) 上に構築し，Docker Compose を用いて一括で管理している．LLM のモデルは OpenAI 社の任意の GPT モデルを選択できる設計にしている．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 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 [20] 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．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，迅速なフィードバックが学生の課題修正行動を促進し，TA や教員の採点業務負担が軽減されるかを評価する．具体的には，PP-Checker で収集した提出ログを用いて，再提出するまでの時間を分析するとともに，学生・TA・教員それぞれに対するアンケート調査を併せて実施した．再提出までの時間は，学生がフィードバックを受けてから修正・改善に至るまでのスピードを示す指標であり，フィードバックの即時性が学習サイクルをどれほど加速させているかを測る上で有効である．また，課題達成までの時間は，学生が最終的に正答に到達するまでに必要とした時間を表し，学習プロセス全体の効率性を評価する指標となる．これらの指標を通じて，LLM フィードバックが TA や教員の採点作業の負担軽減に寄与しているかを検証する．

5.2 運用結果と分析

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．

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，自主的な提出取り下げの回数や再提出までの時間および課題達成までの時間の変化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，学生が自主的に提出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 5 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

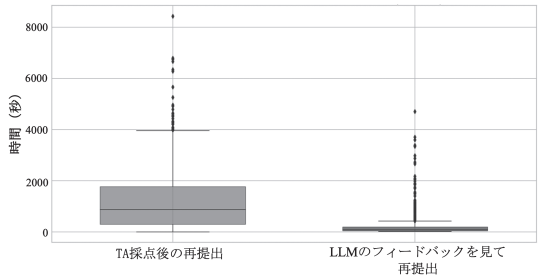

図 5 再提出までの時間の比較

れた件数は 634 件 (約 41.7%) であり，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 (約 58.3%) であった．再提出までの平均時間は，TA 採点後の再提出が約 20.3 分であり，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 分であった．さらに採点ログを分析した結果，LLM は比較的判定が容易な提出物を迅速に処理し，複雑な判断が必要な提出物については TA が時間をかけて評価しており，提出物の内容に応じた役割分担がなされていた．

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．プロンプト変更を実施した講義回において，講義開始時と終了時の LLM の採点正解率を比較したところ，開始時が 60.4% (標準偏差：23.7)，終了時は 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．

また，課題内容の検討に時間をかけた結果，講義開始 30 分前から 5 つの課題を登録するような状況もあったが，PP-Checker はこのような直前の課題登録にも問題なく対応できた．

5.3 従来の採点手法との比較

PP-Checker の利点を従来の採点手法 †2 と比較してさらに評価するため，2023 年度と 2024 年度の共通するプログラミング課題を用いて，再提出までの時間および課題達成時間を比較する分析を行った．2024 年度には PP-Checker を導入し，2023 年度は従来の採点手法を用いていた．従来の採点手法では，学生は Google Drive の指定されたフォルダに提出物をアップロードし，Google Form を通じて学年，クラス，学生番号，氏名，提出する課題の詳細を申請する必要があった．このプロセスは，提出や再提出を行うたびに繰り返す必要があった．また，TA や教員は，Google Drive と同期した Processing で作成された専用アプリケーションを用い，学生と課題を選択した後，Processing エディタ上で提出物を開き，コードを手動で実行・確認した．その後，課題の要件を満たしているか評価し，点数とフィードバックコメントを Google スプレッドシートに手動で入力していた．なお，課題提出が行われるたびにスプレッドシートにタイムスタンプが自動的に記録される仕様であったため，このタイムスタンプ情報を利用して分析を行う．

本分析では，計算と変数について扱う講義回の 1 問目である basic_FootStep†3 を対象とした．この課題は，両年度において同じ講義回の 1 問目に実施され，共通して出題された唯一の課題であった．また，前年度だけでなく，そのさらに前の年度にも提示されており，特定の年度の影響を受けていないことから，両年度の比較を行ううえで適した課題として選定した．2023 年度は 115 名の学生が従来手法で取り組み，2024 年度は 123 名が PP-Checker を利用して課題に取り組んだ．従来の採点手法と PP-Checker を比較した結果を図 6 に示す．図 6 より，学生がコードの誤りに気付き，再提出するまでの平均時間は，2023 年度の約 30.9 分に対し，2024 年度は約 12.3 分と大

†2 https://note.com/nkmr/n/n6edcb28582a7
†3 https://lecture.nkmr.io/2023/programming1_03_sub.pdf

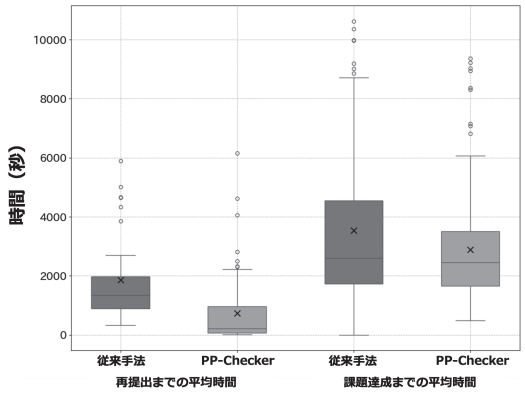

図 6　PP-Checker 導入前後における再提出までの平均時間および課題達成までの平均時間の比較

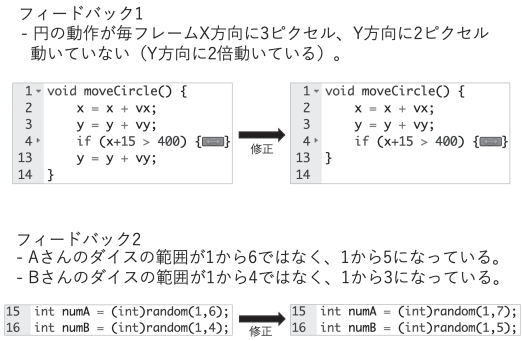

図 7　参考になった LLM のフィードバックと修正の例

幅に短縮されたことが明らかになった．また，再提出回数に関しては，PP-Checker を利用した 2024 年度の学生の提出回数 (79 回) は 2023 年度 (33 回) より増加したものの，課題が提示されてから最終的な提出物を提出するまでの平均時間は，2023 年度の約 58.9 分に対し 2024 年度は約 48.2 分と短縮されたことがわかった．

5.4　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) [10] を用いた PP-Checker のユーザビリティ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は，学期末に Google Form を用いて実施した．このアンケートでは，PP-Checker の実際のフィードバック例を参照しながら，時期ごと (授業初回，学期前半，学期後半) の LLM フィードバックの有用性や修正経験について振り返ってもらった．その結果，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 7)．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，「text 関数の位置が適切でないため、パラメータの表示が曲線に重なる可能性がある。」といった課題の評価に影響しないと思われる指摘は，参考にならなかったと評価された．

5.5　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

表 1 TA アンケート結果．なお，PP-Checker 導入前後に関する評価は，導入前の TA 経験者 6 名のみを対象とした

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

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」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．

5.6 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，主にプロンプトの修正および採点業務を行った 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 のフィードバックを学生自身が確認する時に注目すべきポイントが明確となる点や 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 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影響を与えることが示唆された．LLM によるフィードバックは簡易的なスクリーニングであるのに対し，TA はより複雑な誤りに対する修正を求められるため，LLM の方が再提出までの時間が短くなるのは妥当と判断できる．しかし，その即時性により，学生は自身の誤りを素早く認識し，その場で修正に取り組むことができ，結果として学習サイクルの加速につながっていると考えられる．その上で，LLM の即時性と TA や教員の専門性を組み合わせた適切な役割分担ができていたと考えられる．また，PP-Checker と従来手法における再提出時間および課題の達成時間を比較した結果からも，学生はフィードバックを待つ時間を減らし，提出物の改善により多くの時間を費やすことができるようになったと考えられる．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．

プロンプト変更が LLM の採点精度に与える影響を検証した結果，全体的な正解率に対して有意な精度向上は見られなかった．これは，TA や教員が採点精度

をリアルタイムで把握できず，プロンプトに修正を加えた時点で，精度の変化を即座に確認できなかったため，修正がどの程度影響を与えたのか判断しづらかったことが要因であると考えられる．一方で，特定の課題においては大きな精度向上が見られるものもあった．その中で，興味深い事例として，ある基本課題において，提出開始から最初の 20 分は採点精度が 30% 程度と低かったものの，その後プロンプトが 2 回修正され，70% 近くまで精度が向上したケースがあった．このプロンプト変更は，初期の段階で提出された課題の自動採点結果を踏まえ，それらの評価傾向をもとに行われたものであった．プロンプト修正の例として，「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に，未定義のパラメータを明確に記述する修正があった．また，出力形式の統一や数式の具体的な指定といった変更も行われていた．さらに，採点基準の一貫性を高めるために，例外規定の明示 (例：「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。」) などの修正が見られた．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っていた可能性が考えられる．

今後は，プログラミング講義に限らず，レポート課題など他形式の提出物にも対応を広げることで，教育機関内の多様な成果物を PP-Checker 上で一元的に管理できる環境の構築を目指す．

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．今後は，採点業務のさらなる効率化を目指し，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 [24] への懸念を示す声もあり，2024 年度の課題提出回数が 2023 年度と比べて大幅に増加していた点 (5.3 節) も，その影響を受けている可能性がある．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システム構築を目指す．

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を許容す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] Alkafaween, U., Albluwi, I., and Denny, P.: Automating Autograding: Large Language Models as Test Suite Generators for Introductory Programming, *Journal of Computer Assisted Learning*, Vol. 41, No. 1(2024).

[2] Balse, R., Valaboju, B., Singhal, S., Warriem, J. M., and Prasad, P.: Investigating the Potential of GPT-3 in Providing Feedback for Programming Assessments, in *Proceedings of the Conference on Innovation and Technology in Computer Science Ed-*

ucation (ITiCSE'23), (2023), pp. 292–298.
[3] Boud, D. and Molloy, E.: *Feedback in Higher and Professional Education*, Routledge, 2012.
[4] Clune, J., Ramamurthy, V., Martins, R., and Acar, U. A.: Program equivalence for assisted grading of functional programs, *In the Proceedings of the ACM on Programming Languages*, Vol. 4, No. 171 (2020), pp. 1–29.
[5] Fan, Z., Tan, S. H., and Roychoudhury, A.: Concept-Based Automated Grading of CS-1 Programming Assignments, in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2023, New York, NY, USA, Association for Computing Machinery, 2023, pp. 199–210.
[6] Gabbay, H. and Cohen, A.: 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)*, (2024), pp. 177–187.
[7] George, S. and Dewan, P.: NotebookGPT – Facilitating and Monitoring Explicit Lightweight Student GPT Help Requests During Programming Exercises, in *Companion Proceedings of the 29th International Conference on Intelligent User Interface (IUI'2024 Companion)*, (2024), pp. 62–65.
[8] Gordillo, A.: Effect of an Instructor-Centered Tool for Automatic Assessment of Programming Assignments on Students' Perceptions and Performance, *Sustainability*, Vol. 1, No. 20 (2019), pp. 1–24.
[9] 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, Vol. 44, No. 1 (2020), pp. 105–113.
[10] John, B.: SUS: A 'Quick and Dirty' Usability Scale, *Usability Evaluation In Industry*, (1996), pp. 207–212.
[11] Kabir, S., Udo-Imeh, D. N., Kou, B., and Zhang, T.: 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)*, (2024), pp. 1–17.
[12] Kazemitabaar, M., Chow, J., Ma, C. K. T., Ericson, B. J., Weintrop, D., and Grossman, T.: 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)*, No. 455 (2023), pp. 1–23.
[13] Krusche, S. and Seitz, A.: ArTEMiS: An Automatic Assessment Management System for Interactive Learning, in *Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE'18)*, (2018), pp. 284–289.
[14] Kulp, L., Sarcevic, A., Zheng, Y., Cheng, M., Alberto, E., and Burd, R.: Checklist Design Reconsidered: Understanding Checklist Compliance and Timing of Interactions, in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI '20, New York, NY, USA, Association for Computing Machinery, 2020, pp. 1–13.
[15] 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*）, Vol. 2020-HCI-189, No. 1 (2020), pp. 1–8.
[16] 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*), Vol. 2024-CE-174, No. 7 (2024), pp. 1–8.
[17] OpenAI: GPT-4 Technical Report, *OpenAI Blog*, (2023), pp. 1–100.
[18] Queirós, R. A. P. and Leal, J. P.: PETCHA: a programming exercises teaching assistant, in *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education (ITiCSE'12)*, (2012), pp. 192–197.
[19] Reas, C. and Fry, B.: Processing: a learning environment for creating interactive Web graphics, in *ACM SIGGRAPH 2003 Web Graphics(SIGGRAPH'03)*, (2003).
[20] Salga, A., Hodgin, D., Sobiepanek, A., Downe, S., Medel, M., and Leung, C.: Processing.js: sketching with ¡canvas¿, in *ACM SIGGRAPH 2011 Talks (SIGGRAPH '11)*, No. 15(2011), pp. 1.
[21] 関口祐豊, 中村聡史: 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, 第 *32* 回インタラクティブシステムとソフトウェアに関するワークショップ（*WISS2024*）, (2024), pp. 1–8.
[22] Singla, A.: Evaluating ChatGPT and GPT-4 for Visual Programming, in *Proceedings of the 2023 ACM Conference on International Computing Education Research (ICER'23)*, Vol. 2 (2023), pp. 14–15.
[23] 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*）, Vol. 2021-CE-162, No. 16 (2021), pp. 1–6.
[24] Wang, X., Kim, H., Rahman, S., Mitra, K., and Miao, Z.: 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)*, No. 303(2024), pp. 1–21.
[25] Yan, L., McKeown, N., and Piech, C.: 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)*, (2019), pp. 119–125.

関口祐豊

2000 年生．2025 年明治大学大学院先端数理科学研究科博士前期課卒業．現在，同研究科博士後期課程在学中．生成 AI を活用したインタラクションや筆圧を利用した手書きインタフェース等の研究活動に従事．修士 (工学).

中村聡史

1976 年生．2004 年大阪大学大学院工学研究科博士後期課程修了．同年独立行政法人情報通信研究機構専攻研究員．2006 年京都大学大学院情報学研究科特任助手，2009 年同特定准教授，2013 年明治大学総合数理学部先端メディアサイエンス学科准教授，2018 年同教授，現在に至る．サーチとインタラクションや，ネタバレ防止技術，平均手書き文字等の研究活動に従事．博士 (工学).