大規模言語モデルが生成した論理エラー修正コードと授業資料を活用したプログラミング支援

上川　貴之† 　山崎　禎晃†† 　大原　剛三††

† 青山学院大学大学院理工学研究科　〒 252–5258 神奈川県 相模原市 中央区 淵野辺 5-10-1
†† 青山学院大学理工学部　〒 252–5258 神奈川県 相模原市 中央区 淵野辺 5-10-1
E-mail: †c5622208@aoyama.jp, ††{yamazaki,ohara}@it.aoyama.ac.jp

あらまし　プログラミング教育において，プログラム作成者の意図と異なる結果が出力される論理エラーは様々な要因で起こりうるため，解決方法の共通化が難しく，教員は個別に対応する必要がある．この問題に対し，近年では，問題文やエラーを含むコードを与えた際に自由度の高いアドバイスを生成できる大規模言語モデル（LLM: Large Language Model）の活用が注目を集めている．しかし，LLM が生成したアドバイスは正しいとは限らず，学習の妨げとなることが懸念されている．そこで本研究では，課題に対して正しく動作する模範コードが利用可能であることを前提とし，正しい実行結果が得られるまで LLM に与えるプロンプトを漸進的に改善する手法を提案する．さらに提案手法では，修正されたコードと元のコードの比較を通して正確なアドバイスの生成を試み，加えて，得られたアドバイスなどから抽出したキーワードを元に，授業資料中の該当ページを検索して学習者に提示する．評価実験から提案手法の授業資料ページ提示によりアドバイスの有用性が向上することが明らかになった．

キーワード　教育：e-Learning，EdTech，LLM

1　はじめに

近年，人工知能やビッグデータを用いた新たなビジネスの台頭により第 4 次産業革命が起こっている．それにともない，新たなビジネスの担い手となる IT 人材の育成が求められているが，その指導者不足が問題となっている [1]．特に，プログラミング教育においては，講義と並行して実技が必要とされるだけでなく，プログラミングの自由度の高さから生じる様々な問題への対処が求められるため，すべての学習者に行き届いた指導を実現することは困難である．

プログラミングの習得では，プログラミング言語の文法だけでなく，プログラムの動作手順をアルゴリズムとしてまとめるための論理的思考を学ぶ必要がある．それに加えて，正しい結果が返されない場合には，不具合（エラー）の原因を特定し修正するデバッグ作業が求められる．C 言語のようなコンパイル型のプログラミング言語のエラーは，文法の間違いが原因となるコンパイルエラー，segmentation fault などの実行時エラー，実行手順の論理的間違いによって起きる論理エラーの 3 種類に大別することができ，その中でもプログラミング言語の仕様とは無関係に生じる論理エラーの修正が難しいとされている．この理由としては，コンパイルエラーと実行時エラーに関しては，それぞれコンパイル時，実行時において，その解決のヒントとなるエラーメッセージやエラーの発生箇所がシステム的に出力されうるのに対し，論理エラーに関しては，エラーメッセージが出力されず，エラーの発生箇所の特定が難しいだけでなく，発生箇所を特定しても正しいアルゴリズムを理解していないと修正ができないためである．また，整数同士の除算時のデータ型の間違いなど，多くのプログラムに共通する論理エラー [2] も存在するが，プログラムは文法を守っていれば高い自由度で記述できるため，課題や学習者が作成したプログラム固有の論理エラーに対して個別の対処が求められる．特に，新たなプログラミング課題を作成した場合には，学習者が引き起こす論理エラーを事前に想定し網羅しておくことは非常に難しい．

近年，テキストからのコード作成やコードに対して自然で自由度の高いフィードバックを返すことができる大規模言語モデル（LLM: Large Language Model）が注目されている．その中でも，OpenAI が開発した対話機能を持つ ChatGPT [3] は優れたコード生成・修正能力を示しており，エラーの検出からメッセージの出力までも担えるため，プログラミング，および，プログラミング教育への活用が研究されている [4–8]．しかし，LLM は間違った解答を生成すること [3] があり，単純に利用することはできない．また，課題の答えを簡単に得られるという特性が学生の問題解決能力に悪影響を与える可能性があることが指摘されており [9]，課題に対する答えではなく解決につながるアドバイスや情報を学習者に提示する必要がある．

そこで本研究では，プログラミングの学習支援を目的として，LLM を用いて論理エラーを修正するための正確なアドバイスの生成方法，および，授業資料中の該当ページを検索して学習者に提示する方法を提案する．提案手法では，教員がプログラミング課題の模範解答と授業資料を用意することを前提とし，LLM によって生成した論理エラーを修正したコードが正しく動作するかを模範解答との比較を通して検証することで，正しいアドバイスを生成する．また，多くの授業課題では授業資料に課題のヒントが含まれるという経験則に基づき，アドバイスとともに参照すべき授業資料も提示する．それにより，講義における教員の負担の削減にも繋がると考えられる．

2 関連研究

本節では，LLM によるプログラミング支援に関する研究，および，LLM に対するプロンプトエンジニアリングに関する研究について紹介する．

2.1 LLM をプログラミングに活用する研究

LLM をプログラミングに活用する研究として，Jiang ら [4] は，LLM を用いてアルゴリズムを生成した後に，コードを生成する手法を提案している．具体的には，OpenAI が開発した Codex を用い，少数のアルゴリズム生成例と対象のプログラミング課題の課題文を用いてアルゴリズムを生成し，課題文と生成したアルゴリズムからコードを生成する．しかし，この手法は Python の 1 つのメソッド単位のソースコードを対象としており，複雑な課題には対応していない．次に，Chen ら [5] は，LLM を用いて生成したコードをモデル自身にデバッグさせるコード生成手法を提案している．この手法は，生成，説明，修正というステップに分かれている．具体的には，課題文からコードを生成し，そのコードをモデル自身に説明させる．その後，生成したコードが正しいかどうかを，ユニットテストを用いて判定した結果，もしくは，ユニットテストが無い場合はモデル自身に判断させた結果を用いてモデルに修正させる．

これらの研究では，LLM のプログラミングに関するコード生成性能，および，バグ修正性能の向上を目的とした手法を提案しているが，プログラミング教育に役立てることは行っていない．

2.2 LLM によってコードからアドバイスを生成する研究

LLM によってコードからアドバイスを生成する研究として，Kiesler [6] らは，間違ったコードを修正するためのアドバイスを ChatGPT に生成させ，そのアドバイスの質を調査している．具体的には，33 個の間違ったコードの修正を 3 回ずつ ChatGPT に依頼し，エラーの原因の説明，修正結果，コード等の種類に分類している．計 99 件の回答の中で，80 件以上の回答にテキストによるエラーの原因と修正に関する説明が含まれており，修正後のコードも 65 件の回答に含まれていた．しかし，61 件の回答には不正確な内容が含まれており，21 件の回答にはエラーを特定できないという内容が含まれていた．また，各コードに対して生成した 3 回の回答は大きく異なっており，ランダム性が大きいことも明らかにしている．そのため，著者らは，ChatGPT は有用な回答を行うこともあるが，フィードバックのランダム性や正確性に問題があるため，適切な指導の元で利用する必要があると結論付けている．フィードバックの正確性を向上させる研究として，Phung [7] らは，文法エラーに対して生成したフィードバックを検証する手法を提案している．具体的には，文法エラーを含むコードから複数の修正コードを生成し，その中で最も差分が小さいコードと元のコードから修正内容のフィードバックを生成する．その後，生成したフィードバックを用いてエラーを含むコードを修正し，修正後のコードとフィードバック生成時に利用したコードが一致するかによって，フィードバックの正しさを検証している．さらに，Phung [8] らは，GPT-4，GPT-3.5 モデルをそれぞれ教員と学生に見立てたフィードバック生成手法を提案している．最初に，既存研究 [7] の手法に従ってエラーを含むコードを修正し，修正したコードから教員役モデルが詳細な説明と学生に提供する簡潔なフィードバックを生成する．次に，より弱い性能の学生役モデルが，詳細な説明を用いた場合，用いない場合，それぞれについてエラーを含むコードの修正を実行する．その際，修正コードをそれぞれ複数生成し，正しい修正コードの数を比較することによって，フィードバックの正しさと有用性を検証している．しかし，この方法では 1 つのコードのフィードバックを生成するために多数のリクエストが必要となることから，API 利用における金銭的・時間的コストが高く，大人数の授業環境には適していない．

2.3 プロンプトエンジニアリングに関する研究

プロンプトエンジニアリングとは，言語モデルに入力するプロンプトと呼ばれる文書を最適化することによって，モデルの推論能力を向上させる手法である．Wei ら [10] は，大規模言語モデルの推論能力を向上させる Chain of Thoughts (CoT) という手法を提案している．具体的には，少数のタスクの段階的な解法をプロンプトに含めることで，モデルに与えるタスクを段階的に考えさせ，推論能力を向上させている．Kojima ら [11] は，解法を必要とせずに大幅に推論能力を向上させる Zero-shot-CoT という手法を提案している．この手法は，「Let's think step by step」という単純な指示をプロンプトに追加するだけであり，それにより段階的な思考を誘発し，多様な論理推論タスクの精度を向上させている．

3 提案手法

本研究では，プログラミング授業における受講者のプログラミング支援を目的とし，LLM による正確なアドバイスの生成，および，授業資料を十分に活用するための該当ページ検索手法を提案する．また，アドバイス生成は授業補助者の代替となることが期待されるため，単に高性能な手法を追い求めるだけでなく，利用料金，実行時間などの観点からの利用コストの削減も考慮する．

提案手法の概要を図 1 に示す．提案手法は大きく，アルゴリズム生成，コード修正，アドバイス生成の 3 つから構成される．アルゴリズム生成は，授業準備段階で実施することを想定し，コード修正，および，アドバイス生成は授業中に学生からコードが提出された際に発生する処理となる．図 1 における LLM-1，LLM-2 は，それぞれ利用コストが高いが性能の高い LLM，利用コストは低いが性能が LLM-1 より劣る LLM を表す．アルゴリズム生成では，利用コストの低い LLM-2 の性能向上につながるアルゴリズムを利用コストの高い LLM-1 を用いて授業開始前に生成する．コード修正に関しては，授業課題の課題文，動作例における標準入力と標準出力，および模範解答コードを入力として利用する．また，LLM が誤ったアドバ

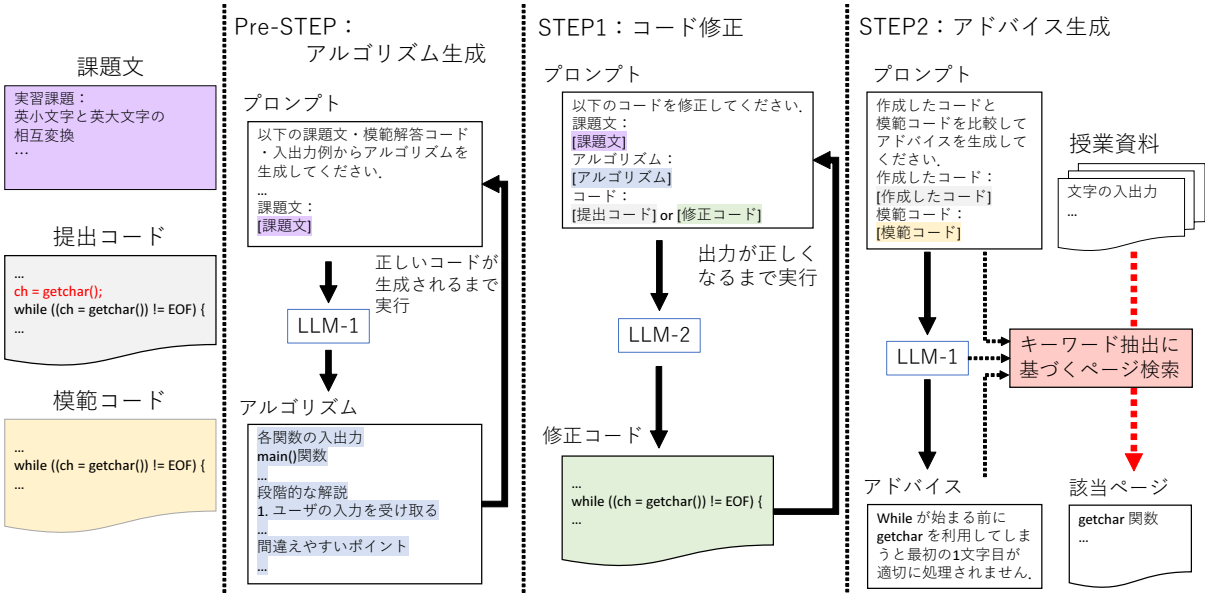


図 1: 提案手法の概要

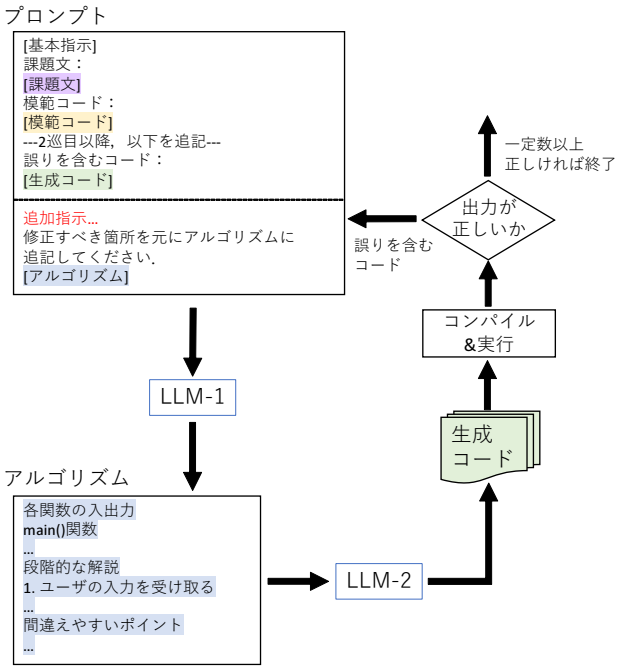


図 2: アルゴリズム生成の概要

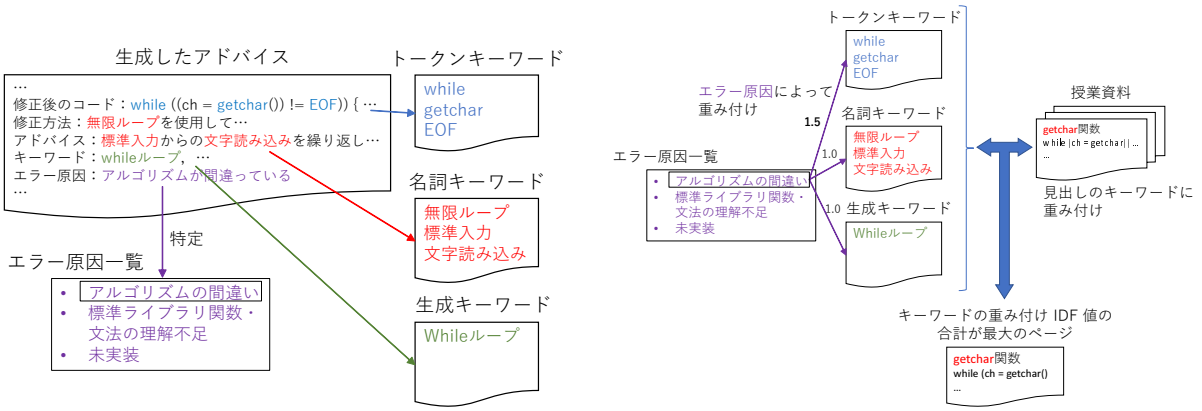
イスを生成することを防ぐために，誤りを含んだコードが適切な出力を返すように修正されるかをチェックし，出力が正しくなるまで修正コードの生成を繰り返す．そのため，ここでは利用コストの低い LLM-2 を用いる．アドバイス生成では，元の誤りを含むコードと正しく修正できたコードを LLM-1 への入力として用いてアドバイスを生成し，生成されたアドバイスから抽出したキーワードを用いて授業資料中の参考となるページを検索する．

3.1 アルゴリズム生成

GPT-3.5 のような利用コストの低い LLM は，GPT-4 のような利用コストの高い LLM と比較すると，性能は低いものの，推論のヒントを与えることで，生成結果の質を向上させることができることが知られている．そこで提案手法では，利用コストの低い LLM-2 の生成結果の質を向上させるために，利用コストの高い LLM-1 を用いて，LLM-2 のよりよいコード生成につながるアルゴリズムを事前に生成する．ここでは，各課題に対して 1 つのアルゴリズムを生成することを目的とし，アルゴリズム生成とコード生成を交互に実行し，正しい出力を返すコードが生成されるまで繰り返す．その概要を図 2 に示す．最初に，対象の課題文，動作例における標準入出力，模範解答コードを入力として LLM-1 を用いてアルゴリズムを生成する．具体的には，各関数の目的，入出力，外部への出力，段階的な解説，および，全体を通して間違えやすいポイントについてまとめさせる．次に，課題文，動作例における標準入出力，生成したアルゴリズムから LLM-2 を用いて 10 個のコードを生成する．生成したコードが一定以上正解すればそのアルゴリズムを採用する．一定以上正解しない場合は，模範解答と間違ったコードを LLM-1 に比較させ，間違えた箇所をアルゴリズムの間違えやすいポイントに追記するように再度生成させる．その際，生成した中で模範解答コードとのトークンのレーベンシュタイン距離が最大となる間違ったコードを選択して比較する．これらのアルゴリズム生成とコード生成を繰り返すことによってアルゴリズムを改善する．

3.2 コード修正

提案手法におけるコード修正について説明する．課題からは課題文と動作例における標準入出力，間違ったコードからはコードと動作例における標準出力を取得し，事前に生成したアルゴリズムとともにプロンプトを生成する．さらに，正しい標準出力と間違ったコードによる標準出力を比較し，内容が異なる最初の 1 行を抽出し，明示的にプロンプトに記載する．こうして生成したプロンプトを LLM-2 に入力し，間違ったコードに対する修正コードを出力させる．そして，修正コードが正



(a) 生成したアドバイスからのキーワード抽出　　(b) エラー原因による重み付けと授業資料の検索

図 3: 授業資料のページ検索の概要

しいかどうかを動作例の実行によって判定し，出力結果が正しければ次のアドバイス生成に進み，正しくなければ，間違ったコードに関するプロンプトの内容を修正コードのものと入れ替えたプロンプトを生成し，再度 LLM-2 を用いて修正コードを生成する．これらの操作を一定回数まで繰り返すことによって正しい修正コードを生成する．

3.3 アドバイス生成

提案手法の最終段階のアドバイス生成は，LLM-1 による自然言語によるアドバイス生成，および，その際に生成した情報に基づいた授業資料ページの検索から構成される．アドバイスは 2 段階で生成し，1 段階目では，課題文，動作例における正しい入出力，生成したアルゴリズム，間違ったコードとその出力を LLM-1 に入力し，コードを比較させ，間違えている箇所を生成させる．その際，修正箇所の説明，修正方法，および修正内容の分類（必要な修正またはリファクタリング）を生成させる．2 段階目では，1 段階目の対話に加えて，最初に修正すべき箇所の説明，修正方法，該当箇所の修正前後のコード，簡潔な 1 つのアドバイス，関連する用語,「アルゴリズムの間違い」「文法や基本的概念の理解不足」「未実装」の 3 種類の分類となるエラー原因を生成させる．このように，段階的にアドバイスを生成することで，アドバイスの正確性の向上が期待できる．

次に，授業資料のページ検索の概要を図 3 に示す．授業資料のページ検索に関しては，図 3a に示すように次の 3 つの方法で取得したキーワードを利用する．1 つ目は，LLM-1 で生成した修正方法と簡潔なアドバイスから，普通名詞，複合名詞を抽出するものである．これにより，修正に必要な概念に関する名詞キーワードを取得する．2 つ目は，生成した修正前後のコードをトークン化し，それぞれ関数，演算子，制御構文，定数を抽出し，修正後のコードに含まれるものと修正前のコードにしか含まれていないものを抽出する．このようにして抽出したトークンキーワードにより，課題のヒントとなるサンプルコードが授業資料に含まれる場合，その検索が期待できる．3 つ目は，前述のアドバイス生成 2 段階目で生成した関連する用語を利用するものである．この生成キーワードは，名詞キーワードでは取得できない関連概念に対応することが期待できる．この名詞キーワードは,「○○関数」のような英数字・日本語を含む場合にそれらを分割し，キーワードとして追加する．これにより，より抽象的な概念や具体的な関数名を用いたコード部分とのマッチングが期待できる．

次に，パワーポイント形式の授業資料に含まれるテキストをページごとに抽出し，その際，検出したテキストを含む図形の中心の位置が最も上となる図形のテキストをそのページの見出しとする．その後，対象とする課題の回までの授業資料において，検索するキーワードの Inverse Document Frequency (IDF) 値を計算し，図 3b に示すように 2 つの条件で重み付けする．IDF とは，文書集合内における対象単語の希少性を表す値であり，その単語を含む文書数の逆数を用いて定義される．IDF が大きいほど，その単語がより少ない文章にのみ出現することを意味する．1 つ目の条件は，キーワードがスライドの見出しに含まれているかどうかであり，含まれていればより関連するスライドだとみなし，そのキーワードを重みづける．2 つ目の条件は，ページに関係なく LLM-1 に生成させたアドバイス箇所のエラー原因に基づいたものである．具体的には，エラー原因が「アルゴリズムの間違い」であれば，授業資料の中で類似した構造をもつサンプルコードが,「文法や基本的概念の理解不足」であれば，そのような文法・概念を解説したページが参考になると考えられる．そのため，エラー原因が「アルゴリズムの間違い」の場合は，主にサンプルコードとのマッチングに向いているトークンキーワードを重み付けし,「文法や基本的概念の理解不足」の場合は，名詞キーワード・生成キーワードを重み付けする．ただし，エラー原因が「未実装」の場合は，どちらも原因である可能性が考えられるため，いずれのキーワードも重み付けしない．最後に，ページごとに含まれているキーワードの重み付け IDF 値の合計が最も高いページを検索結果として提示する．

4 評価実験

実際の大学のプログラミング授業において学生が作成したソースコードを用いて，提案手法の論理エラー修正性能を評価

表 1: アルゴリズム生成結果

課題	反復回数ごとの正解数					合計入力	合計出力	合計 API
	1	2	3	4	5	トークン数	トークン数	利用料金（$）
素数の判定	10/10	-	-	-	-	12,326	2,431	0.046
相互変換	10/10	-	-	-	-	15,700	3,136	0.058
カレンダー	3/10	4/10	8/10	-	-	68,947	22,965	0.39

表 2: コード修正結果（アルゴリズムあり）

課題	総正解数	反復回数ごとの正解数				修正コードとの	模範コードとの
		1	2	3	4	平均距離	平均距離
素数の判定	24/25	23/25	1/2	0/1	0/1	31.5	52.7
相互変換	15/17	13/17	1/4	1/3	0/2	53.1	71.8
カレンダー	10/14	3/14	4/11	1/7	2/6	231.3	304.9

表 3: コード修正結果（アルゴリズムなし）

課題	総正解数	反復回数ごとの正解数				修正コードとの	模範コードとの
		1	2	3	4	平均距離	平均距離
素数の判定	24/25	23/25	0/2	1/2	0/1	28.7	52.7
相互変換	15/17	10/17	1/7	4/6	0/2	51.3	71.8
カレンダー	3/14	2/14	1/12	0/11	0/11	168.0	301.0

した．

4.1 実験設定

評価実験では，青山学院大学理工学部情報テクノロジー学科 2 年生を対象とした C 言語を用いたプログラミング実習科目の 2022 年度受講生が作成したプログラムを利用した．ここでは，同科目で出題された課題のうち 3 つの課題に対する提出プログラムを対象とした．具体的には，標準入力で与えられた整数が素数か否かを判定する課題，標準入力で与えられた半角英字の大文字と小文字を相互に変換する課題，標準入力で与えられた年月のカレンダーを出力する課題である．素数判定の課題は，単純なアルゴリズムの実装が必要な低難度の課題，大文字・小文字相互変換の課題は ASCII コードの知識と適切な条件分岐が必要な中難度の課題，カレンダーの課題は，月の開始する曜日の計算，適切な日数での改行が必要な高難度の課題となっている．本研究では，論理エラーを対象としているため，文法エラー，実行時エラーの発生しているコードと正解コードは除外した．本実験での利用に同意した受講生から提出されたコードのうち，これらを除外した後のコード数は，それぞれ素数判定の課題が 25，大文字・小文字相互変換の課題が 17，カレンダーの課題が 14 となった．また，取得したコードは，コメントを削除し，linux の lint コマンドを用いて整形して使用した．整形後のコードの平均行数は，それぞれ 30.9 行，35.5 行，64.7 行となった．

本実験では，LLM として ChatGPT を利用し，高コストかつ高性能な LLM-1 としては GPT-4-1106-preview を用い，利用コストの低い LLM-2 としては GPT-3.5-turbo-1106 を使用した．また，ChatGPT には，設定できるパラメータの 1 つとして temperature が存在する．temperature は，モデルの出力の決定性に関わるパラメータであり，低い方がより決定性が高まり $[0.0, 2.0]$ の範囲の値を指定できる．アルゴリズムの生成，アドバイス生成では 0.0，コードの生成，修正では 0.5 を指定した．また，生成したアルゴリズムを採用する正解コード数の割合は 0.8，アルゴリズム追記の反復回数は 5，コード修正の際の反復回数は 4 に設定した．

表 4: コード修正のコスト（アルゴリズムあり）

課題	平均入力	平均出力	平均 API	平均実行時間（s）
	トークン数	トークン数	利用料金（$）	
素数の判定	1,840	618	0.0031	13.4
相互変換	3,648	792	0.0044	17.8
カレンダー	8,682	2,456	0.014	49.3

表 5: コード修正のコスト（アルゴリズムなし）

課題	平均入力	平均出力	平均 API	平均実行時間（s）
	トークン数	トークン数	利用料金（$）	
素数の判定	870	687	0.0022	15.6
相互変換	1,505	1,023	0.0036	24.0
カレンダー	4,273	2,790	0.0099	55.5

4.2 評価指標

アルゴリズム生成では，アルゴリズムの精度として生成したコードの反復ごとの正解数，コストとして全体の入出力トークン数と API 利用料金を評価した．コード修正では，修正精度として修正したコードの反復ごとの正解数，最終的な正解数を評価した．また，修正前のコードと修正後のコードおよび模範解答コードのトークンのレーベンシュタイン距離を計算することによって，ChatGPT による修正がより個人のコードに沿った修正になっているかどうかを評価した．さらに，1 コードあたりの平均入出力トークン数，平均 API 利用料金，平均実行時間を実行コストとして評価した．アドバイス生成においても，1 コードあたりの平均入出力トークン数，平均 API 利用料金，平均実行時間を実行コストとして評価し，また，アドバイスの有用性を被験者実験によって評価した．

4.3 アルゴリズム生成とコード修正の実験

アルゴリズム生成とコード修正の実験では，正解数，および，コストによる定量評価を行った．アルゴリズム生成の結果を表 1 に示す．高難度のカレンダーの課題以外は，最初に生成した段階ですべて正解となっている．カレンダーの課題で最初に生成されたアルゴリズムと最終的なアルゴリズムを比較すると，「合計日数を計算する際に、入力された年の 1 月 1 日から入力された月の 1 日までの日数を加算するが、これを二重に加算してしまうと曜日の計算がずれる。」のように追記されていた．このように，間違いやすい部分を強調することによってコードの生成・修正精度が高められていると考えられる．

次に，生成したアルゴリズムの有無によるコード修正の結果を表 2，表 3 に示す．カレンダーの課題は正解数が大幅に向上しており，それ以外の課題でも正解数は変わらないが，アルゴリズムがある場合の方が修正の反復回数が減少していることがわかる．これらの結果から，提案手法で生成したアルゴリズムがコード修正精度の向上に有効だと考えられる．また，ChatGPT で修正したコードは，模範コードより元のコードとの平均レーベーンシュタイン距離が小さくなっており，より個別のコードに沿った修正ができているといえる．さらに，コー

表 6: 標準的な手法のアドバイス生成のコスト

課題	平均入力トークン数	平均出力トークン数	平均 API 利用料金（$）	平均実行時間（s）
素数の判定	816	72	0.010	7.5
相互変換	932	76	0.012	5.9
カレンダー	1,584	94	0.018	6.8

表 7: 提案手法のアドバイス生成のコスト

課題	平均入力トークン数	平均出力トークン数	平均 API 利用料金（$）	平均実行時間（s）
素数の判定	4,762	1,082	0.080	97.3
相互変換	5,324	974	0.082	96.1
カレンダー	7,897	1,374	0.120	125.1

表 8: 標準的な手法と提案手法の評価結果

課題	正確性				情報の量	有用性
	間違いから正しい	正しいから正しい	正しいから間違い	間違いから間違い		
標準的な手法	10	0	0	0	−0.37	3.96
提案手法	10	0	0	0	−0.28	3.88

表 9: 授業資料ページの評価結果

ページに関連する概念やコードの構造が含まれているか		ページ単体の有用性	アドバイスと組み合わせた有用性
含まれている	含まれていない		
5	5	3.41	4.23

ド修正のコストをそれぞれ表 4，表 5 に示す．アルゴリズムを用いた修正のほうが，アルゴリズムを入力に含むため入力トークン数が増加し，それに応じて API 利用料金が高くなっているが，修正の反復回数が減少しているため，実行時間は減少していることが確認できる．

4.4 アドバイス生成の実験

アドバイス生成の実験では，コストの定量評価と被験者実験による評価を行った．具体的には，実験データを収集した授業において，ティーチングアシスタントとして経験のある学生 7 人を対象に被験者実験を実施し，生成したアドバイスとマッチングした授業資料ページを評価した．対象としたコードは被験者の負担を考慮して合計で 10 コードであり，各課題のデータ数に応じて素数の判定の課題から 5 コード，相互変換の課題から 3 コード，カレンダーの課題から 2 コードをランダムに選択した．具体的な評価手順は 2 段階に分かれている．1 段階目では，間違ったコード，模範コード，および，入出力例を入力としてアドバイスを生成する標準的な手法と提案手法に関して，生成したアドバイスの正確性，アドバイスとしての情報の量，および，アドバイスの有用性を評価した．この時，提示する両手法によるアドバイスはコードごとにランダムな順序で提示した．2 段階目では，マッチングした授業資料ページの評価であり，ページ内にコード修正に関連する概念およびコードの構造が含まれているか，および，検索したページ単体の提示とアドバイスと組み合わせた際の有用性を評価した．正確性の評価では，コードの状態が「間違った状態から正しい状態（必要な修正）」「正しい状態から正しい状態（リファクタリング）」「正しい状態から間違った状態」「間違った状態から間違った状態」のいずれの状態であるかを評価して貰った．ただし，標準的な手法と提案手法はどちらも最初に修正すべき箇所のアドバイスを生成させている．そのため，この正確性の分類では，アドバイス箇所のエラーを正しく指摘できているかについて注目しているため，コード全体で他に間違っている箇所が残っているかについては考慮していない．次に，アドバイスとしての情報の量は，LLM が簡単に答えを得られるという特性を考慮し，答えを教えすぎていないかという点を「不足」「やや不足」「適切」「やや過剰」「過剰」の-2 から+2 の 5 段階で評価した．最後に，これらの有用性は，どのくらい修正に役立つかを「まったく有用ではない」「あまり有用ではない」「どちらともいえない」「やや有用である」「とても有用である」を 1 ～ 5 の 5 段階で評価した．また，実験の最後に全体を通してアドバイスおよび授業資料ページの良かった点，改善すべき点についての自由記述を用意した．

アドバイス生成における標準的な手法と提案手法のコストをそれぞれ表 6 と表 7 に示す．提案手法の方が，標準的な手法より入力トークン数が 5 倍以上，出力トークン数が 12 倍以上，API 利用料金が 8 倍以上，実行時間が 13 倍以上かかっている．これは，標準的な手法が 1 回の入出力でアドバイスを生成しているのに対して，提案手法では，一度間違い箇所の一覧を出力し，さらにその入出力を入力として対象とする箇所のアドバイスを生成しているためである．また，提案手法では，授業資料の検索に利用するためにアドバイスの他に修正方法，キーワード，エラー原因といった情報も出力しており，さらにコストが増加している．

次に，標準的な手法と提案手法の評価結果と提案手法のマッチングした授業資料ページの評価結果を，それぞれ表 8 と表 9 に示す．正確性，ページに関連する概念およびコードの構造が含まれているかの評価は，コードごとに被験者の分類結果を多数決で決定しており，情報の量と有用性はそれぞれ被験者の評価結果の平均値を記載している．実験の結果，正確性の評価では，どちらの手法もすべてのコードで間違った状態から正しい状態という結果になっており，標準的な手法でも十分に正しいアドバイスを生成できると考えられる．次に，情報の量はどちらの手法も少し不足しており，過剰にアドバイスを与えすぎてはいないといえる．最後に，提案手法の授業資料とアドバイスを組み合わせた際の有用性は，各手法のアドバイスのみ，および授業資料ページのみに比べて向上しており，標準的な手法に対して t 検定を行った結果，有意差（$p < 0.05$）があることを確認した．そのため，生成したアドバイスとともに授業資料ページを提示することは，アドバイスのみを提示するよりも有用だと考えられる．

次に，提案手法のアドバイスと授業資料ページを提示した際の評価が最も高かった（平均 5.0）事例について考察する．この事例では，相互変換の課題で図 4 のように while ループの条件式が間違っているコードに対して,「代入と比較の優先順位に注意して、条件式を見直してください。」というアドバイスと

```c
#include <stdio.h>
int lower2upper(int c);
int upper2lower(int c);
int main()
{
    int ch;
    while (ch = getchar() != EOF) {
        if (ch == 10)
            putchar(ch);
        else if (97 <= ch && ch <= 122)
            printf("%c", (char) lower2upper(ch));
        else if (65 <= ch && ch <= 90)
            printf("%c", (char) upper2lower(ch));
        else
            putchar('*');
    }
    return 0;
}
```

図 4: 最高評価のアドバイスのコード

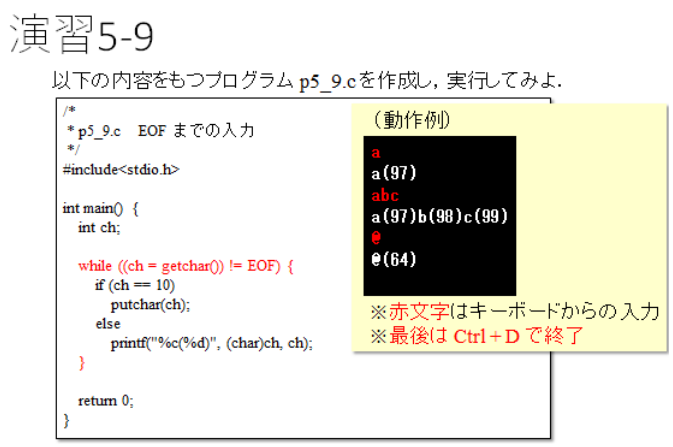


図 5: 最高評価の際に提示されたページ

```c
#include<stdio.h>
int isprime(int i);
int main()
{

}

int isprime(int n)
{
    int m, ans;
    for (n = 1, m = 2; m != n; m++) {
        ans = n % m;
        if (ans == 0) {
            return 0;
        } else {
            return 1;
        }
    }
}
```

図 6: 最低評価のアドバイスのコード

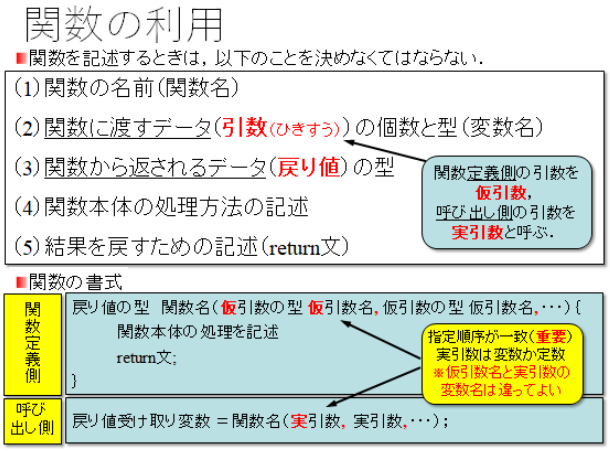


図 7: 最低評価の際に提示されたページ

図 5 のような授業資料ページを提示している．このページには，正しいループ構造の例が含まれているため，高評価となったと考えられる．また，同様に最も低かった事例では，素数の判定の課題で図 6 のように関数名が課題の指定と異なっているコードに対して,「関数の宣言と呼び出しで使用されている名前を確認し、課題文の指定に合わせてください。」というアドバイスと図 7 のような授業資料ページを提示している．低評価となった原因として，課題の指定とは異なるが被験者が普段指導する際に重視する箇所ではないこと，および，isprime 関数の引数 n を関数内で上書きしているといったさらに重大な論理エラーを指摘できていないことが評価を下げる原因になったと考えられる．ただし，提示されたページは関数の使い方に関するページであり，アドバイスに関連した内容を提示することはできている．最後に，最も正確性の評価が分かれており，さらに，提案手法のアドバイスのみの評価が標準的な手法より最も下がった事例について考察する．この事例では，図 8 のようにカレンダーの課題のコードに対して，標準的な手法では,「カレンダーの日付を出力するためのループ処理が不足しています」というアドバイスを提示している．提案手法では,「入力された年と月が指定された範囲内にあるかを確認し、範囲外であれば再入力を促すループを追加する。」というアドバイスと図 9 のような授業資料ページを提示している．提案手法のアドバイスの評価が最も分かれた原因として，模範コードでは存在するが入出力例には示されていない入力の検証機構がリファクタリングだと考える被験者も多かったことが考えられる．また，提示された授業資料は，scanf() 関数から受け取った標準入力を条件ごとに処理するコードになっており，アドバイスとの関連は，標準入力によって分岐するといった部分的なものになっている．

4.5 自由記述の考察

自由記述では，アドバイスの良かった点として,「直接的に答えを教えるのではなく考えさせるアドバイスがあってよかった」といった意見があった．ただし，改善すべき点として,「記述を促すだけのアドバイスが参考にならない」「何回もやりとりする必要がある」といった意見もあり，ヒントの量が少なすぎるアドバイスがあることを指摘していた．そのため，ヒントの量が少ないと感じた際にインタラクティブに LLM にヒントを生成させるような手法が必要だと考えられる． また，授業資料ページ良かった点として,「ベースとなる知識を提示できていて良かった」や「コードの枠組みが分からない人にコード例を提示できていて良かった」という意見があった．しかし改善すべき点として,「まったく関係ないページが迷いを生じさせる可能性がある」といった意見や「コード例のページの際にどこに注目すれば良いか分かりにくい」といった意見があった．このこ

```c
#include <stdio.h>
int is_leap_year(int year);
int get_days(int year, int month);
int main()
{

    int year, month, day, a, i = 5;

    printf("西暦を入力してください（2000年以降） : ");
    scanf("%d", &year);
    printf("月を入力してください : ");
    scanf("%d", &month);

    printf("%d年%d月¥n", year, month);
    printf("Sun Mon Tue Wed Thu Fri Sat¥n");

    day = get_days(year, month);

    return 0;
}
```

図 8: 最も分類が分かれたアドバイスのコード

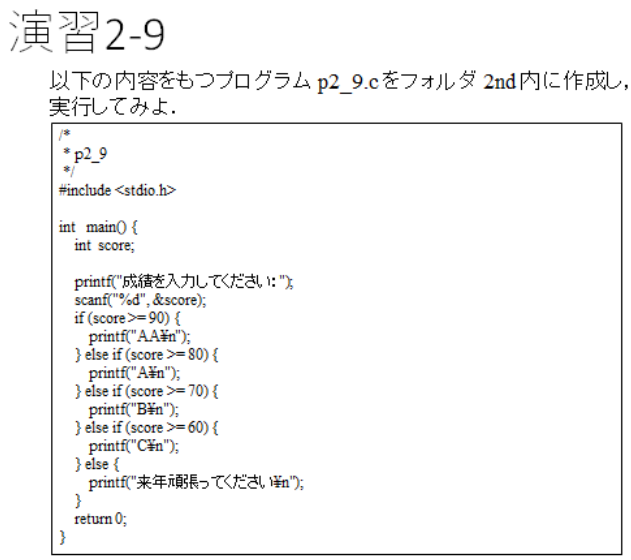


図 9: 最も分類が分かれたアドバイスとともに提示されたページ

とから，マッチングした授業資料ページをさらに LLM に入力として使い，LLM に関係あるページかどうか判断させる手法，どこに注目すべきかといったより充実したアドバイスを生成させる手法などが考えられる．

5 ま と め

本研究では，論理エラーを含む学生のプログラムコードに対する修正アドバイスを，LLM を用いて生成する手法を提案した．提案手法では，間違ったコードに対する修正コードを生成し，そのうえでアドバイスを生成する．提案手法におけるコード修正の性能を評価した結果，課題に対して事前に生成したアルゴリズムが，コード修正において正解数の向上，反復回数の減少に寄与することが明らかになり，それにより 7 割以上のコードを修正できることを確認した．また，アドバイス生成に関しても，被験者実験の結果から授業資料をアドバイスとともに提示することが有用だと明らかになった．今後の課題として，授業資料の検索精度の向上，アドバイス生成の 1 段階化によるコスト削減が挙げられる．提案手法では，テキスト情報とスライドの見出しの情報しか用いていないが，授業資料では，強調したい情報は文字色の変更，フォントサイズの変更などの操作が行われることが多い．そのため，抽出したテキストの文字色やフォントサイズを用いることによって，授業資料のマッチング精度の向上が期待できる．また，今回標準的な手法と提案手法のアドバイスの正確性は変化がなかった．そのため，1 段階でのアドバイス生成による実行コストの削減が考えられる．

本研究の一部は，一般財団法人ホワイトロック財団の補助による．

文 献

[1] NTT ラーニングシステムズ株式会社．文部科学省委託事業次世代の教育情報化推進事業『平成 30 年度教育委員会等における小学校プログラミング教育に関する取組状況等について』の調査．https://www.mext.go.jp/component/a_menu/education/micro_detail/__icsFiles/afieldfile/2019/05/28/1417283_002.pdf.

[2] Andrew Ettles, Andrew Luxton-Reilly, and Paul Denny. Common logic errors made by novice programmers. In *Proceedings of the 20th Australasian Computing Education Conference (ACE)*, p. 83–89, 2018.

[3] OpenAI. Introducing chatgpt. https://openai.com/blog/chatgpt.

[4] Xue Jiang, Yihong Dong, Lecheng Wang, Zheng Fang, Qiwei Shang, Ge Li, Zhi Jin, and Wenpin Jiao. Self-planning code generation with large language models. arXiv preprint arXiv:2303.06689, 2023.

[5] Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. arXiv preprint arXiv:2304.05128, 2023.

[6] Natalie Kiesler, Dominic Lohr, and Hieke Keuning. Exploring the potential of large language models to generate formative programming feedback. arXiv preprint arXiv:2309.00029, 2023.

[7] Tung Phung, José Cambronero, Sumit Gulwani, Tobias Kohn, Rupak Majumdar, Adish Singla, and Gustavo Soares. Generating high-precision feedback for programming syntax errors using large language models. arXiv preprint arXiv:2302.04662, 2023.

[8] Tung Phung, Victor-Alexandru Pădurean, Anjali Singh, Christopher Brooks, José Cambronero, Sumit Gulwani, Adish Singla, and Gustavo Soares. Automating human tutor-style programming feedback: Leveraging gpt-4 tutor model for hint generation and gpt-3.5 student model for hint validation. arXiv preprint arXiv:2310.03780, 2023.

[9] Md. Mostafizer Rahman and Yutaka Watanobe. Chatgpt for education and research: Opportunities, threats, and strategies. *Applied Sciences*, Vol. 13, No. 9, p. 5783, 2023.

[10] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. arXiv preprint arXiv:2201.11903, 2022.

[11] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 35, pp. 22199–22213, 2022.