K-013

大規模言語モデルを用いた非専門家向けプログラミング支援ツールの開発と実践

Development and Preliminary Evaluation of a Programming Support Tool for Non-Experts Using Large Language Models

出口 大地† 仙田 朋也† 梶原 祐輔†
Daichi Deguchi Tomoya Senda Yusuke Kajiwara

1. はじめに

幼稚園から高校までの K-12 プログラミング教育では、問題解決能力、論理的思考、創造性などの計算思考（CT）の育成を目指している[1]。従来の小学生を対象としたプログラミング教育では、視覚的プログラミング言語（Visual Programming Languages：VPL）が主流である[2]。ただしアルゴリズム的な思考の養成は VPL より疑似言語によるプログラミング教育のほうが優秀である[3]。たとえば Scratch において、学生はあらかじめブロックを組み合わせてプログラムを作成することに対して、テキストベースに比べ、創造性が制限されている[4]。これらの点で VPL は課題が残っていた。

本研究では、大規模言語モデル（以下、LLM）を活用し、プログラミングの非専門家を主な対象とする支援ツール「AI Programmer」の開発を行った。さらに、非専門家と LLM の協働における課題を明らかにすることを目的として、小中学生を対象としたアプリ開発セミナーを実施した。本稿では、開発したツールの概要を紹介するとともに、セミナーを通じて得られた知見に基づき、LLM を用いたプログラミング支援における課題について検討する。

2. 支援ツール「AI Programmer」

本研究では、プロンプトプログラミングが可能となるよう、図 1 に示すアプリケーション開発プラットフォーム「AI プログラマー」を構築した。図 2 には AI プログラマーのユーザインターフェース（UI）を示している。この AI プログラマーを使用することで、開発者は自然言語を使ってアプリケーションを作成することができる。LLM には ChatGPT を使用した。ChatGPT の優れた点は、OpenAI の API を利用して自由に組み込むことができる点である。これにより、プログラマーはモデルをアプリケーションに組み込むことができる。以下に、AI プログラマーを使用してアプリケーションを開発する手順を説明する。

1. 開発者は図 2 のシステム初期画面または開発環境の UI ボタンを押す。これにより、特定のアクションがトリガーされ、開発プロセスが開始される。
2. ボタンが押されると、該当する UI に対して、対応する処理が実行される。各 UI 要素にはそれぞれ異なるイベント処理が割り当てられており、それがアプリケーションの異なる部分に対応する。
3. イベントハンドリングがトリガーされると、関連するデータがクエリされる。ここで参照されるデータとは、プロダクトバックログ（PBL）、テスト仕様書、修正指示書のことを指し、これらは Excel ファイルで管理されている。
4. クエリ応答によって取得したデータに、「事前プロンプト」を付与して記述し、その後にユーザが入力したプロンプトの統合を行う。このプロセスにより、システムに与えるべき指示（プロンプト）が生成される。
5. 統合したプロンプトをネットワーク部で API リクエストに変換する。この段階で、自然言語による要求をシステムが解釈できる API リクエストに変換する。
6. 生成された API リクエストを送信する。API リクエストがサーバーやクラウドサービスに送信され、処理が行われる。
7. API からの応答として、ソースコードが返される。
8. 受け取ったソースコードを Python インタプリタで実行し、アプリケーションを自動生成する。
9. 自動生成されたアプリケーションを開発者に提供する。最終的に、開発者は完成したアプリケーションを利用またはテストすることができる。

これらの機能の操作ログおよび内容はロギング機能によって、タイムスタンプで管理され、記録される。このように、AI プログラマーでは自然言語を利用して要件定義や修正指示を行い、API を通じてソースコードを生成し、自動的にアプリケーションを作成することができる。このプラットフォームにより、ユーザーはプログラミング知識がなくても、複雑なアプリケーション開発を効率的に進めることが可能となる。また、UI を通じた操作やイベント処理の仕組みによって、柔軟で段階的な開発がサポートされている。

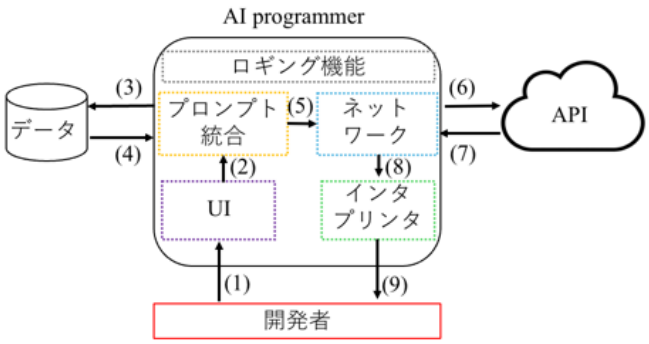
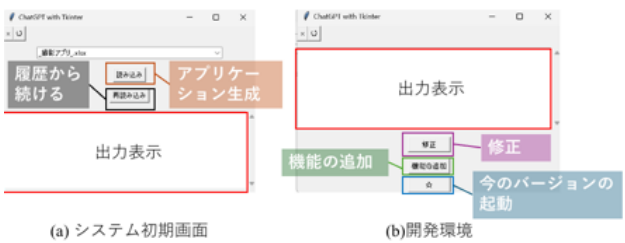


図 1 AI Programmer の機能と流れ



(a) システム初期画面 (b)開発環境

† 公立小松大学 Komatsu University

図２　AI Programmer の UI

３．実験方法

AI を使ったアプリ開発と銘打ったセミナーを開催した。参加者には、AI Programmer を用い、予めこちらで設定した「カメラアプリを作る」という課題に沿って、アプリ開発をしてもらった。初めに、ツールの利用法を説明し、学んでもらった。その後、コアシステムを開発したあと、こちらの用意したいくつかの機能を追加してもらい、テストを行う。失敗していたらエラーの内容を記載してもらうことで、LLM による修正を実行する。成功していた場合は、次の機能の追加を実行する。このように開発を進めていく。こちらの用意した機能の追加を終えた参加者から、自由に好きな機能を追加してもらった。ただし、演習を円滑に進めるため、コアシステムを開発するプロンプトは事前に用意し、統一したものを利用した。

本実験には、9 歳から 13 歳までの 11 人の学生が石川県内から集められた。全員事前に LLM を利用した経験はなく、2 時間のセミナーにご参加いただいた。

４．実験結果と考察

まず、統一されたプロンプトから生成されたコアシステムを状態によって分類し、その数を表 1 に示した。

ID	状態	生成数
1	撮影後、編集後の画像が元のプレビュー上に一瞬だけ表示されるが、キャプチャボタンを押しても変化しない。	3
2	撮影後、ウィンドウが下方向に展開し、撮影画像がそこに表示される。	3
3	キャプチャボタンを押すと別ウィンドウが開き、編集が可能となる。	2
4	最初からプレビュー下に空のフレームがあり、撮影後にそこへ画像が表示される。	2
5	撮影後、別ウィンドウは表示されず、プレビュー画面も撮影画像に切り替わらない。	1

表 1　コアシステムのパターンと生成された数

LLM なので、当然ではあるのだが、同一の呼びかけでも多様な返事をするように、同一のプロンプトでも多様なアプリが生成される。教育という観点では、統一的に進めるのが難しくなるという課題になると感じた。

次に、発生したエラーをカテゴリー（エラー名）に分類した。そして、それぞれ発生した人数と回数を記して、表 1 にまとめた。

カテゴリ（エラー名）	人数	回数
属性エラー（AttributeError）	6	16
名前解決エラー（NameError）	3	5
（UnboundLocalError）	1	3
値エラー（ValueError）	2	2
ライブラリエラー（cv2.error）	1	2
（tkinter.TclError）	1	12
ファイルシステムエラー（FileNotFoundError）	1	4

表 1　エラー毎の発生した人数・回数

とくに多かった属性エラー（AttributeError）は、オブジェクトに存在しない属性（メソッドやプロパティ）を参照または代入しようとしたときに発生するもの。それ以外も、文法ミスや名前の指定のミスなど、初歩的なエラーが多い。出力したコードにこのような初歩的な誤りが含まれていた場合であっても、非専門家であるため、自ら修正することは出来ない。とくに、1 人が 12 回起こしたライブラリエラー（tkinter.TclError）に注目したい。これは、tkinter を利用する際に発生するエラーであるが、ここでは内容よりも、同じエラーを何度も起こしたことに注目する。被験者は非専門家であるため、そもそもアプリの起動が出来ないという状況においては、LLM にエラー内容の伝達を行えない。そのような場合において、「起動しない」というようなエラーの伝え方では修正が上手くいかないことがある。そして、曖昧な表現への応答として LLM が幻覚（hallucination）に近い挙動を示し、問題の本質とは異なる解決策を提示したためと考えられる。

５．まとめ

本研究では大規模言語モデル（LLM）を用いた非専門家向けのプログラミング支援ツール「AI Programmer」を開発し、小中学生を対象にしたセミナーを通じて、その有効性と課題を検証した。実験の結果、同一のプロンプトであっても生成されるアプリの仕様にばらつきがあり、プログラムの一貫性や安定性の確保が困難であることが明らかとなった。また、出力コードに含まれる文法的誤りや、ユーザからの曖昧な修正指示に対して適切な応答が得られないといった問題も観察された。特に非専門家においては、エラーの内容を正確に理解・伝達できないことが、開発の進行を妨げる要因となる。これらの課題を踏まえ、今後は、生成コードの品質を評価・検証する機構の導入や、曖昧なプロンプトへの応答精度を高めるためのフィードバック設計の改善が求められる。プロンプトの構造化やテンプレート化によって出力の一貫性を担保し、典型的なエラーに対する自動修正機能の整備を進めることも有効である可能性がある。今後はこれらの機構を組み込み、より確実で柔軟な自然言語によるプログラミング支援の実現を目指す。

参考文献

[1] Denner, J., Werner, L., & Ortiz, E. (2012). Computer games created by middle school girls: Can they be used to measure understanding of computer science concepts?. Computers & Education, 58(1), 240-249.
[2] Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12?. Computers in human behavior, 41, 51-61.
[3] Garlick, R., & Cankaya, E. C. (2010, June). Using Alice in CS1: A quantitative experiment. In Proceedings of the fifteenth annual conference on Innovation and technology in computer science education (pp. 165-168).
[4] Weintrop, D., & Wilensky, U. (2015, June). To block or not to block, that is the question: students' perceptions of blocks-based programming. In Proceedings of the 14th international conference on interaction design and children (pp. 199-208).