大規模言語モデルは新規言語のプログラミング能力をどう獲得するか？

西潟優羽 [1] 中野 乃梨子 [2] 土田 悠佳 [2] 倉光 君郎 [2]
[1] 日本女子大学大学院 理学研究科 [2] 日本女子大学 理学部
m2116061ny@ug.jwu.ac.jp kuramitsuk@fc.jwu.ac.jp

概要

本研究では，大規模言語モデル (LLM) が文脈学習によって，事前学習に含まれない新規プログラミング言語の仕様を理解し，コード生成能力を獲得できるかを検証した．著者らが開発した教育用言語 Yui を対象に，HumanEval をベースとした評価データセットを構築し，様々な文脈情報 (言語仕様書，コード例，AI エージェントの圧縮コンテクストなど) を与えた際のコード生成性能を測定した．実験の結果，GPT-5.1 と Claude Haiku-4.5 において，AI エージェントの圧縮コンテクストを用いた場合に最高 68.9%の正答率を達成した．本研究は，LLM のプログラミング能力の中核がアルゴリズムやデータ構造の推論など言語非依存な能力であり，文脈学習のみで新規言語への転移が可能であることを示唆している．

1 はじめに

本研究は，著者らが新規に設計したプログラミング言語 Yui およびそのインタプリタを開発する過程で生じた予期せぬ観察結果に端を発している．具体的には，生成 AI エージェント（Claude Code）が，Yui 言語のインタプリタだけでなく，そのサンプルコードも生成できたという事象である．

Yui 言語は，大規模言語モデル (LLM) の事前学習に含まれておらず，コード生成が困難な言語を目指して設計された [1]．それゆえ，この現象は我々にとって想定外であった．生成 AI エージェントは，開発中のテストコードなどから言語仕様を理解し，コード生成が可能になったと考えられる．ここから，LLM は文脈学習 (In-Context Learning[2]) によっても，新規言語のプログラミング能力を獲得可能であることが示唆された．

本研究の目的は，さまざまな文脈情報を与えることで，新規プログラミング言語の知識を獲得し，既存のコード生成能力を転移可能になるかを明らかにすることである．

2 問題定義

2.1 プログラミング能力とは何か？

LLM はプログラミング能力を有し，その能力が非常に高いことが知られている [3]．実際，自然言語による指示を与えると，アルゴリズムなどの詳細を明示しなくても，必要な手順やデータ構造を推論し，最終的に実行可能なコードを生成することができる．

従来，この能力はプログラミング言語に強く依存すると考えられていた．プログラミング言語間でプログラミング能力に差があることも，学習リソース量の差 [4] と見なされてきた．しかし，我々が観察した未学習の新言語でもコードを生成できるという事実は，プログラミング能力が多層的な構造を持つ可能性を示唆している．

我々は，LLM のプログラミング能力について，次のような 3 段階の仮説を立てた．

- **（ゴールの理解）** 自然言語で与えられた仕様や目的を理解する能力
- **（手順の推論）** アルゴリズムやデータ構造などゴールを達成するための推論能力
- **（コード化）** プログラミング言語の文法や構文規則に従ってコードを出力する能力

この仮説では，プログラミング能力の中核をなすアルゴリズムやデータ構造などの推論能力は，プログラミング言語に非依存な形で獲得されていると考える．むしろ，プログラミング言語の文法に依存するコード化は，JSON や XML 形式などフォーマットを整形する程度の能力である可能性がある．

```
# クイックソート(再帰版)
# 配列の一部をソートする関数
クイックソート = 入力 配列, 開始, 終了 に対し {
    もし 開始 が 終了 より小さい ならば、{
        # パーティション操作
        境界 = パーティション(配列, 開始, 終了)

        # 左側を再帰的にソート
        クイックソート(配列, 開始, 引き算(境界, 1))

        # 右側を再帰的にソート
        クイックソート(配列, 足し算(境界, 1), 終了)
    }
    配列が答え
}

```

図 1: Yui 言語によるクイックソートの例 (抜粋)

2.2 Yui の紹介

Yui は，生成 AI 時代におけるコンピュータ科学教育 [5] という新たな文脈のもとで設計された教育用プログラミング言語である．

- 自然言語ベース：日本語ベースであり，学習者にとって直感的に理解しやすい構文を採用している (図 1)
- コンパクトな言語仕様
  - データ構造：整数，配列のみ
  - 演算：インクリメントとデクリメントのみ
  - 制御構造：条件分岐と回数指定のくり返し
- 標準ライブラリ：四則演算や文字列，小数変換と配列の変換のみ

Yui 言語はチューリング完全であり，プログラミングの記述能力は Python に劣るものではない．しかし，言語仕様が限られているため，Python の書き方をそのまま 1 対 1 に変換できない．たとえば，四則演算や論理演算 (and, or, not) が存在しないため，関数や if 文の組み合わせに置き換える必要がある．ループも，繰り返しの回数を事前に予測するなど，高度なプログラミング能力が求められる．

2.3 本論文の目的

我々は，対象とする LLM が少なくとも Python など既存のプログラミング言語に関するプログラミング知識を保有していることを前提とする．

本研究の目的は，LLM の事前学習に含まれない未学習の Yui 言語が，プログラミング能力をどのように獲得するかを検証することである．言い換えると，Python などのプログラミング能力を Yui 言語にどのように転移させるかを明らかにすることである．

なお，LLM に知識を転移させる方法には微調整やアライメントなどさまざまな手法があるが，本研究では文脈学習 [2] に注目し，与える文脈情報の種類と転移されたコード生成能力の関係を分析する．

3 文脈情報の設計

本節では，Yui 言語のプログラミング能力の獲得を目指し，文脈情報の設計について述べる．

3.1 AI エージェントの文脈

我々は，Yui 言語のインタプリタを AI エージェント Claude Code で開発 [1] し，その開発過程で Yui 言語によるバブルソートなどの複雑なサンプルコードが生成できる点に気づいた．

我々は AI エージェントに対して「なぜコード生成が可能なのか」と問いかけた．その応答は「これまでの対話で与えられた例，README，テストコードを通じて言語仕様をおおよそ理解したため，コード生成が可能になった」というものであった．

我々は，Yui インタプリタ開発時の AI エージェントの文脈を圧縮して出力した．出力された文脈を人手で検証したところ，部分的に言語仕様の誤解が含まれていたため，それらを修正した後，`AGENTS` とラベル付けした文脈情報とした．

3.2 フォーマルな言語仕様

プログラミング言語は，構文や意味論など数理的に定義する形式仕様がある．これらの仕様書は，主に言語開発者が用いるもので，一般的なプログラマは参照しないが，処理系の動作を正確に再現するときに必要となる．

- `EBNF`：拡張バッカスナウア記法による構文定義
- `DENOTE`：Python 言語との対応付けによる表示的意味論 (denotational semantics)
- `RULES`：Python から Yui への構文変換規則 (Python サブセットの Yui による表示的意味論)

3.3 インフォーマルな言語仕様

我々は，開発者や学生が利用することを想定し，Yui 言語のプログラミングを理解するための文書を以下に用意した．

- `INTRO`：著者が書き下ろした授業用の Yui の教材．プログラミングの概念から解説される．
- `DNCL`：Yui 言語仕様．大学入試センター編の

DNCL と構成，内容ともほぼ同じ．

- **CHEATSHEET**：Yui 言語仕様とサンプルを 1 枚にまとめたサバイバルガイド
- **PYTHON**：Python プログラマのための Yui 入門．Yui の言語仕様が Python コードとともに解説される
- **C**：C プログラマのための Yui 入門．Yui の言語仕様が C コードとともに解説される

これらの文書は，**INTRO** 以外，**AGENTS** を文脈に入れた AI エージェントで生成したのち，人手で内容を修正したものである．

4 実験

4.1 実験デザイン

本実験の目的は，「LLM が文脈情報を与えることで，Yui の言語仕様を学習し，どのようにプログラミング能力を獲得するか」調べることである．

次のとおり，実験を行い検証する．

1. Yui 言語未学習の LLM を用意する．(Yui は新規言語であるため，未学習であると想定する．)
2. プロンプトの指示で，Yui コードを生成する．
3. 生成された Yui コードの正しさを確認し，正答率を算出する．

生成された Yui コードの評価は，実行ベースの評価指標 pass@1[6] を用いる．我々が開発した Yui インタプリタで実行して，テストケースと同じ正解だった場合をパスとする．正答率が高いほど，Yui 言語の仕様書を理解し，プログラミング能力を獲得したものとする．

4.2 評価データセット

我々は, 実験に先立ち, 評価データセットの開発を行なった．評価データセットは，Python コードの生成の標準ベンチマークの HumanEval[7](指示文は日本語で行なったので正確に言えば，JHumanEval[8]) をベースして，問題文とテストケースはそのまま採用し，Yui コードを生成するようにした．

我々は，HumanEval-Yui 版を生成するにあたり，参照コード (canonical_solution) を Python コードから Yui に移植し，すべて動作検証した．移植作業は，Claude Code と Yui インタプリタを用い，テストケースをパスするまで自動修正した．一部，人手による修正も含まれている．最終的に HumanEval に含まれる 164 件の問題を Yui に移植し，動作検証ができた．

```
### 問題

整数 x の絶対値を返す関数 absolute_value を実装してください。

実行例
>>> absolute_value(-5)
5
>>> absolute_value(3)
3
>>> absolute_value(0)
0

### 回答
```yui
標準ライブラリを使う

absolute_value = 入力 x に対し {
    もし x が 0 より小さい ならば {
        result = 0
        result = 差(result, x)
        resultが答え
    }
    そうでなければ {
        xが答え
    }
}
```

問題
{PROBLEM}
```

図 2: 実験で用いた one-shot プロンプトの例

4.3 プロンプト

プロンプトは，コード生成の対象を Yui 言語にする以外，標準的なコード生成プロンプトを用意する．

我々は，文脈情報の効果を比較するため，代表的な文脈学習である few-shot プロンプトも 5 例まで用意した．図 2 は，one-shot プロンプトの例である．

文脈情報は，zero-shot プロンプトに対して追加した．つまり, one-shot プロンプトの代わりに文脈情報を配置した．文脈解釈を促進する Chain-of-Thought (CoT) は今後の検討課題である．

4.4 モデル

最後に，今回の実験で用いたモデルを述べる．

- GPT-5.1(gpt-5.1-2025-11-13): コード能力が高く，推論性能に優れたモデル
- Haiku(claude-haiku-4-5): コード生成能力が高く，トークン効率に優れたモデル

4.5 実験結果

表 1 は，実験結果をまとめたものである．トークン数は LLM のトークナイザーに依存する値であるが，文脈情報を与えるときはトークン効率も重要であるため，本稿では Claude 4.5 のトークナイザーで算出している．

表 1: 文脈情報からの Yui 言語仕様理解とプログラミング能力 (Pass@1)

文脈情報	トークン数	GPT-5.1	Haiku
なし	0	0.00	0.00
1SHOT	180	1.22	0.61
2SHOT	441	6.10	6.10
3SHOT	682	14.20	17.07
4SHOT	924	15.24	21.95
5SHOT	1306	18.90	21.95
EXAMPLES_9	1856	28.05	24.39
EXAMPLES_18	3385	42.07	44.51
EXAMPLES_36	7605	57.93	56.71

文脈	トークン数	GPT-5.1	Haiku
EBNF	1139	1.83	1.83
DENOTE	6960	35.98	28.66
RULES	1123	22.56	22.56
INTRO	7838	34.15	20.73
DNCL	4041	60.98	54.88
CHEATSHEET	2375	58.54	50.00
PYTHON	4449	65.24	47.56
C	3840	32.32	27.44
AGENTS	7357	68.90	66.46

まず，実験したモデル GPT-5.1 と Claude Haiku-4.5 の両方において，Yui 言語が未学習であることを確認しておこう．文脈「なし」では，どちらも 0.00 であり，コード生成は全くできていない．

最も高い性能を示したのは，AGENTS(圧縮コンテクスト) である．Yui インタプリタの開発後，AI エージェントの開発コンテクストから言語仕様のみを圧縮した情報であるが，両モデルともに最高スコアを示した．この正解率は，2024 年の Claude 3 Haiku の Python におけるコード生成能力に相当する．これはソフトウェア開発の実務への適用が始まったレベルであるため，文脈学習のみで実用的なコード生成能力を獲得できたといえる．

フォーマルな言語仕様 (EBNF, DENOTE, RULES) とインフォーマルな文書 (PYTHON, DNCL, CHEATSHEET 等) を比較したとき，総じて一般向けの文書の方がスコアが良かった．文書の種類ごとにスコアが分かれたため，今後，コード生成能力の獲得度から文書の理解しやすさの分析にも応用が期待できる．

単純な few-shot プロンプトでも，例を増やすとスコアが向上した．興味深いのは，EXAMPLES_9，_18，_36 で，これは問題文なしのコード実例集となる．実例だけを 9 個，18 個，36 個と増やすと，コード生成の性能も有意に向上する．経験ある人間のプログラマでもコード例から言語仕様を推測することは可能であるため，文脈学習でも同等の推論を行っていることが示唆される．

5 関連研究

LLM によるプログラミング能力の研究は，コード生成・コード実行推論など多岐にわたって進められている．特に近年では，低リソースプログラミング言語への適応や，未知の言語に対する汎化的な推論能力の評価が重要な研究課題となっている．

LLM が複数のプログラミング言語を同時に学習することで，言語非依存のプログラミング概念を獲得しているという仮説は，複数の研究 [9, 10] によって支持されている．ただし，学習データが限られた低リソース言語では性能が著しく低下することも知られている [4, 11]．

Singha ら [12] は，Ada, Swift, Prolog, Clojure, Dart, Elixir の 6 言語を対象に，文脈学習 (RAG)，エージェント，ツール呼び出し，実行誘導生成の 4 つの適応技術を比較評価し，低リソース言語に対してはツール呼び出しが効果的であると報告した．

Patel ら [13] は，仕様が非常に限られた言語について，基本的な構文や機能を少数の例や記述から文脈学習できる兆候が見られることを報告している．ただし，HumanEval と同等のコード生成能力の評価は本研究が初めてである．

6 むすびに

本研究では，文脈学習によって，未学習のプログラミング言語の仕様を理解し，コード生成の性能を獲得することを示した．文脈にコード例を含めることが，言語仕様の理解をコード生成能力に結びつける傾向も確認できた．

今後は，Yui 言語以外のバリエーションも試すことで，LLM の言語理解とコード生成の原理の解明に努めていきたい．Yui が自然言語ベースであることを活かし，自然言語の言語獲得の分析，言語獲得における文脈学習と事前学習の効率比較などを行いたい．

参考文献

[1] 倉光君郎, 西潟優羽, 中野乃梨子, 土田悠佳. yui: 生成 ai で生成できない新しいプログラミング言語. 情報処理学会第 67 回プログラミング・シンポジウム. 情報処理学会, 2026.

[2] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Tianyu Liu, Baobao Chang, Xu Sun, Lei Li, and Zhifang Sui. A survey on in-context learning, 2024.

[3] James Finnie-Ansley, Paul Denny, Brett A. Becker, Andrew Luxton-Reilly, and James Prather. The robots are coming: Exploring the implications of openai codex on introductory programming. In **Proceedings of the 24th Australasian Computing Education Conference**, ACE '22, p. 10–19, New York, NY, USA, 2022. Association for Computing Machinery.

[4] Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, Ming-Ho Yee, Yangtian Zi, Carolyn Jane Anderson, Molly Q Feldman, Arjun Guha, Michael Greenberg, and Abhinav Jangda. Multipl-e: A scalable and extensible approach to benchmarking neural code generation, 2022.

[5] Kimio Kuramitsu, Waka Itoh, Kaoru Kagami, and Ayako Masamune. Prompts first で始めたプログラミング教育の再構築. 第 2025 巻, pp. 64–71. 情報教育シンポジウム論文集, Aug 2025.

[6] Sumith Kulal, Panupong Pasupat, Kartik Chandra, Mina Lee, Oded Padon, Alex Aiken, and Percy S Liang. Spoc: Search-based pseudocode to code. **Advances in Neural Information Processing Systems**, Vol. 32, , 2019.

[7] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. **arXiv preprint arXiv:2107.03374**, 2021.

[8] 佐藤美唯, 高野志歩, 梶浦照乃, 倉光君郎. Llm は日本語追加学習により言語間知識転移を起こすのか？ 言語処理学会第 30 回年次大会発表論文集, pp. 2897–2900. 言語処理学会, 2024.

[9] Yuha Nishigata, Waka Itoh, and Kimio Kuramitsu. Non-english code generation with cross-lingual chain of thought. In **Proceedings of the 39th Pacific Asia Conference on Language, Information and Computation**, Hanoi, Vetonum, December 2025.

[10] A. Madaan, S. Zhou, U. Alon, Y. Yang, and G. Neubig. Language models of code are few-shot commonsense learners. **arXiv preprint arXiv:2210.07128**, 2022.

[11] Federico Cassano, John Gouwar, Francesca Lucchetti, Claire Schlesinger, Anders Freeman, Carolyn Jane Anderson, Molly Q Feldman, Michael Greenberg, Abhinav Jangda, and Arjun Guha. Knowledge transfer from high-resource to low-resource programming languages for code llms. **Proceedings of the ACM on Programming Languages**, Vol. 8, No. OOPSLA2, pp. 677–708, 2024.

[12] A. Singha, M. Singh, H. Hasanbeig, A. Radhakrishna, and S. Gulwani. Language model adaptation for low-resource programming languages: A comparative study. In **39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop**, 2025.

[13] Arkil Patel, Siva Reddy, Dzmitry Bahdanau, and Pradeep Dasigi. Evaluating in-context learning of libraries for code generation. In **Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)**, pp. 2908–2926, 2024.