

修士学位論文

題目

GPT-4を用いた自動コード翻訳に対する精度調査

指導教員

肥後 芳樹 教授

報告者

著者 川渕 皓太

令和6年2月1日

大阪大学 大学院情報科学研究科

コンピュータサイエンス専攻 ソフトウェア工学講座

内容梗概

あるプログラミング言語で書かれたソースコードを別の言語に翻訳する自動コード翻訳の研究において、機械学習を用いるニューラル機械翻訳が注目されている。ニューラル機械翻訳は、事前に翻訳ルールを逐一定義するルールベース機械翻訳と比べて作成の労力が低く、人間が読みやすいコードが生成されやすいというメリットがある一方で翻訳の精度が低いという問題がある。

本研究では様々な分野で高い性能を発揮している最新の大規模言語モデルである GPT-4 を対象として、C++, COBOL, Go, Java, Rust, Python3 のソースコードに対する自動コード翻訳タスクの精度を調査した。既存のデータセットは GPT-4 が事前学習済みである可能性が高いため、新たにデータセットを作成して調査を行った。

その結果、GPT-4 が既存の事前学習モデルよりも精度が高いこと、翻訳元言語よりも翻訳先言語のほうが精度への影響が大きいこと、出力に対して静的解析ツールを利用することで精度が向上すること、プロンプトの記述言語が日本語の場合のほうが英語の場合に比べて精度が高いこと、自動コード翻訳タスクにおいては Chain-of-Thought が有効ではないことがわかった。

主な用語

GPT-4

自動コード翻訳

プロンプトエンジニアリング

目次

1	はじめに	4
2	背景	6
2.1	機械翻訳による自動コード翻訳	6
2.2	LLM	6
2.3	プロンプトエンジニアリング	7
2.4	LLMとコード関連タスク	7
2.5	LLMと自動コード翻訳	7
3	調査内容	9
4	準備	10
4.1	データセットの作成	10
4.2	精度の調査方法	10
4.3	各言語の実行環境	11
5	調査 1: ニューラル機械翻訳モデルと GPT-4 の比較	13
5.1	データセットの加工	13
5.2	調査方法	14
5.3	モデルの設定	15
5.4	結果	15
5.5	考察	16
5.5.1	TransCoder-IR の精度	16
5.5.2	関数のトークン数と翻訳の精度の相関	17
6	調査 2: 異なる特徴の言語間における翻訳の精度	18
6.1	調査方法	18
6.2	モデルの設定	18
6.3	結果	18
6.4	考察	19
6.4.1	翻訳元言語と翻訳先言語が精度に与える影響	19
6.4.2	動的型付け言語と静的型付け言語間の翻訳の精度	19
6.4.3	ソースコードのトークン数と翻訳の精度の相関	20
6.4.4	静的解析ツールによるコンパイルエラーの解消	21

7 調査 3: プロンプトの言語が翻訳の精度に与える影響	23
7.1 調査方法	23
7.2 モデルの設定	23
7.3 結果	23
7.4 考察	24
7.4.1 コード生成タスクにおけるプロンプトの言語の影響	24
8 調査 4: プロンプトに Chain-of-Thought を導入した場合の精度	25
8.1 調査方法	25
8.2 モデルの設定	27
8.3 結果	28
8.4 考察	29
8.4.1 Chain-of-Thought による逐次的な翻訳の悪影響	29
9 妥当性への脅威	30
9.1 内的妥当性	30
9.1.1 データセットに含まれるテストケースの品質	30
9.1.2 GPT-4 の出力の自然言語部分の影響	30
9.1.3 LLM の非決定性による影響	30
9.2 外的妥当性	30
9.2.1 再現性の問題	30
9.2.2 通常のプロジェクトにおける精度	31
10 おわりに	32
謝辞	34
参考文献	35

1 はじめに

自動コード翻訳とはあるプログラミング言語で書かれたソースコードを別の言語に翻訳する技術のことである。COBOL から Java などのレガシーな言語からモダンな言語への翻訳や、Python3 から C++ など低速高水準言語から高速低水準言語への翻訳などに利用されている [2, 6].

自動コード翻訳の手法の1つとして、ルールベース機械翻訳がある。ルールベース機械翻訳を用いた自動コード翻訳ツールとして、例えば Cython [2], opensourceCOBOL4j [6] などが挙げられる。これはルールや辞書に沿って翻訳を行う手法のことであり、この手法はルールの範囲内であれば正確に翻訳を行うことができる。しかし、翻訳のために大量のルール登録を手動で行う必要があり、このルール登録には非常に労力がかかる。また、人間にとって読みにくい不自然な翻訳が出力される場合もある [36].

そこで、ルールベース機械翻訳に代わる手法としてニューラル機械翻訳が提案された [5]. これは機械学習を用いて翻訳を行う手法であり、メリットとして、人間が作成したコードを学習しているため人間が理解しやすい翻訳が出力されやすいこと、ルールベース機械翻訳と比較してルール登録の手間がかからないということが挙げられる。しかし最新のニューラル機械翻訳モデルの1つである TransCoder-IR [36] においても、C++, Go, Java, Rust 間を相互に翻訳した場合の精度が平均 44.2% と低く、正確に翻訳ができないことが多いという問題がある。

2023年3月に OpenAI 社から最新の大規模言語モデル (LLM) である GPT-4 [29] が公開された。LLM とは、事前学習済み言語モデル (PLM) のモデルサイズ、学習データサイズをスケールアップすることにより、通常の PLM よりも複雑なタスクを解く能力 (創発的能力 [40]) を獲得したモデルの呼称である [35, 44]. GPT-4 は学習に自然言語、ソースコード、画像を使用しており、以下のように多くの分野で高い精度を達成している。

- Uniform Bar Examination (米国統一司法試験) を受験者の上位 10% の成績で合格 [29].
- 自然言語翻訳において既存の商用製品レベルのスコアを達成 [16].
- 既存モデルよりもコード理解、コード生成において高精度 [3, 29].

GPT-4 は高精度でコード理解、コード生成を行うことができるため、自動コード翻訳タスクも高精度で行うことができると予想できる。しかし、GPT-4 の自動コード翻訳タスクの精度を調査する研究はまだ存在しない。

そこで本研究では GPT-4 の自動コード翻訳タスクの精度を調査した。対象の言語として C++, COBOL, Go, Java, Rust, Python3 を選択した。

本研究の主な貢献は以下の通りである。

- GPT-4 (gpt-4-0613) が事前学習していないデータから新たにコード翻訳タスクのテスト用データセットを作成した.
- GPT-4 が既存の自動コード翻訳モデルよりも高精度でコード翻訳を行えることを明らかにした.
- 翻訳先言語, 翻訳元言語の組み合わせによって翻訳の精度が変化することを明らかにし, 特に翻訳先言語が翻訳の精度へ大きく影響することを明らかにした.
- 静的解析ツールを用いることが精度の向上に有効であることを明らかにした.
- プロンプトの言語が日本語の方が英語の場合よりも精度が高いことを明らかにした.
- コード翻訳タスクにおいては Chain-of-Thought [41] が有効ではないことを明らかにした.

2 背景

2.1 機械翻訳による自動コード翻訳

2017年のアメリカにおいて、実運用で使用されている COBOL コードは 2200 億行を超え、ATM の 95% が COBOL に依存している [33]。しかし COBOL の開発者が引退するにつれてこれらのシステムをメンテナンスできる開発者が減少し、サポートが危ぶまれている [4, 19]。そのため、既存の COBOL コードを Java や C++ のような新しい言語に移行する必要があるが、レガシーシステムの移行にはソース言語とターゲット言語の両方の専門知識が必要であり、コストがかかることが多い。例えばオーストラリアのコモンウェルス銀行は、プラットフォームを COBOL から Java に変換するために約 7 億 5000 万ドルと 5 年を費やした [14]。このように、人によるレガシーシステムの移行はコストがかかるため、機械翻訳による自動化を行いたい。

事前に作成したルールに従って翻訳を行う手法であるルールベース機械翻訳を用いてレガシーシステムの移行を支援することは可能であるが、一般的にルールベース機械翻訳の出力は冗長であり可読性が低い [36]。そのため、レガシーシステムと同様に後のメンテナンスが困難になる。

それに対して、ニューラル機械翻訳は人間が書いたコードを学習しているため、可読性が高いコードを出力しやすい。そのため、ニューラル機械翻訳はレガシーシステム移行に有用であると考えられる。しかし、最新のニューラル機械翻訳モデルの 1 つである TransCoder-IR [36] においても翻訳の精度が平均 44.2% と、精度の低さが課題となっている。

2.2 LLM

事前学習済み言語モデル (PLM) とは大量のテキストコーパスで事前学習されたモデルのことである。PLM は事前学習後にタスクに応じてファインチューニングをすることで様々なタスクに対応することができる。PLM の例として、双方向 LSTM を用いた ELMo [31] や、Transformer [39] を用いた GPT-2 [32]、同じく Transformer を用いた BERT [7] などが挙げられる。

PLM は多くの場合、モデルサイズやデータサイズのスケールリングによって能力が向上すると報告されている [17]。PLM をスケールリングすることで、通常の PLM よりも複雑なタスクを解く能力 (創発的能力 [40]) を獲得したモデルを一般的に大規模言語モデル (LLM) と呼ぶ [35, 44]。LLM の例として OpenAI が開発した GPT-3.5 [28]、GPT-4 [29]、Meta が開発した Llama 2 [38]、Google が開発した Gemini [37] などが挙げられる。

LLM の応用例として GPT-3.5、GPT-4 を対話用にチューニングした ChatGPT [27] があり、高い会話能力を示している。ChatGPT の登場で LLM はより注目されるようになり、研

究論文が保存・公開されている世界最大のリポジトリである arXiv¹において、ChatGPT のリリース後に LLM に関する論文が急増している [44].

2.3 プロンプトエンジニアリング

LLM は基本的に入力としてテキストを与えると、その入力テキストに応じたテキストが出力される。LLM に与える入力テキストは一般的にプロンプトと呼ばれる。プロンプトの変更により LLM の出力の性能が変化することが報告されている [41] ため、プロンプトを工夫することが重要視されている。これをプロンプトエンジニアリング [25, 42] と呼び、例えば思考の連鎖を促すことで出力の質を上げる Chain-of-Thought [41] などのテクニックがある。

また、プロンプトを記述する言語が精度に影響を与えるという報告も行われている [8, 21, 29]。プロンプトの言語の影響はタスクによって異なることが報告されており、例えば、GPT-4 は MMLU データセット [11, 12] においてプロンプトの言語が日本語より英語の場合のほうが 5.6 % 精度が高いと報告されている [29] 一方で、ChatGPT は安全なコードの生成においてはプロンプトの言語が日本語の場合も英語の場合も精度は同等であると報告 [45] されている。

2.4 LLM とコード関連タスク

近年、ソフトウェア工学コミュニティと機械学習コミュニティの双方から LLM のコード生成の能力を評価する報告が出されている [1, 9, 24]。また、同様にコード理解も高精度で行うことができると報告されている [3, 29]。

LLM のコード理解・生成の機能をサービス化した例として GitHub Copilot [10] が挙げられる。これは統合開発環境上で動作するプログラマ支援ツールで、コードの自動補完や、自然言語からのコード生成などのさまざまな機能を持ち、2023 年 10 月時点で 100 万人以上の有償ユーザが存在すると報告されている [43]。

2.5 LLM と自動コード翻訳

関連研究 [15] では StarCoderBase [23] と GPT-3.5-turbo [28] の自動コード翻訳タスクの精度の評価を行っている。Java, Python3, C++, C#, JavaScript を含む新たなデータセットである G-TransEval を作成して評価した結果、これらの LLM は既存の事前学習モデルよりも高い精度を発揮し、特に GPT-3.5-turbo が良い性能を発揮したと報告されている。

¹<https://arxiv.org/>

しかし、作成されたデータセットに含まれるデータはLLMが事前学習の段階で学習済みの可能性があり、そのことが精度に影響を及ぼしている可能性についても同研究では述べられている。また、COBOLなどのレガシーな言語を含む翻訳性能の評価は行われていない。

3 調査内容

本研究では、GPT-4の自動コード翻訳タスクの精度に関する4つの調査を行った。

調査 1: ニューラル機械翻訳モデルと GPT-4 の比較

GPT-4がニューラル機械翻訳モデルと比較して有用かどうかを調査する。比較対象として最新のニューラル機械翻訳モデルの1つである TransCoder-IR [36] を選択する。TransCoder-IR は関数単位の翻訳を行うモデルであるため、GPT-4 も関数単位での翻訳を行って精度の比較を行う。対象の言語は C++, Go, Java, Rust の 4 言語とする。

調査 2: 異なる特徴の言語間における翻訳の精度

C++, COBOL, Go, Java, Rust, Python3 で記述されたソースコードを翻訳し、レガシーな言語とモダンな言語間や、静的型付け言語と動的型付け言語間などの異なる特徴を持つ言語間での翻訳の精度の傾向を調査する。同時にコード単位での翻訳の精度も調査する。

調査 3: プロンプトの言語が翻訳の精度に与える影響

LLM はタスクによってプロンプトの言語の影響が異なることが報告されている [29, 45, 8, 21].

調査 3 ではプロンプトが日本語の場合と英語の場合の翻訳の精度を比較し、GPT-4 の自動コード翻訳タスクにおけるプロンプトの言語の影響を調査する。

調査 4: プロンプトに Chain-of-Thought [41] を導入した場合の精度

プロンプトエンジニアリングの1つに、思考の連鎖を促すことで精度を向上させる Chain-of-Thought [41] という技術があり、その一種にプロンプトの末尾に思考の連鎖を促すプロンプトを追加する Zero-Shot Chain-of-Thought [20] という技術がある。

調査 4 では Zero-Shot Chain-of-Thought をプロンプトに導入することで精度がどのように変化するか調査する。

4 準備

本研究で使用する GPT-4 のモデル (gpt-4-0613) は 2021 年 9 月以前のデータで学習を行っている [29]。コード翻訳タスクの評価は通常 CodeXGLUE [26] 等のデータセットが使用されるが、これらのデータセットは GPT-4 が既に学習している可能性がある。そこで、本研究では GPT-4 が事前学習していないデータからデータセットを新たに作成して調査を行う。データセットは C++, COBOL, Go, Java, Python3, Rust のソースコードとテストケースを含む。

本節では、データセットの作成方法、作成したデータセットを用いた翻訳の精度の調査方法、精度調査時の各言語の実行環境について述べる。

4.1 データセットの作成

データセットは競技プログラミングサイトである AtCoder²の問題に提出されたソースコードと、AtCoder が公式に公開している各問題のテストケース³から作成した。

対象の問題は 2022 年 1 月～2023 年 6 月までの期間に AtCoder において開催された AtCoder Beginner Contest の A 問題と B 問題とし、対象の言語は C++, COBOL, Go, Java, Python3, Rust とする。

対象の全ての問題に対して、対象の各言語で以下の処理を行いソースコードを取得する。

1. 問題に提出された正解ソースコードを行数で昇順にソートし、上位 25%を除いたソースコードのうち行数最小のものを取得。この処理を行うことで、極端に短いソースコードや極端に冗長なソースコードを除外できる。
2. 取得したソースコードを実行し、全てのテストケースに通過することを確認。

作成したデータセットのソースコード情報を表 1 に示す。トークン数の算出には ANTLR⁴を利用した。COBOL のみソースコード数が少ないのは AtCoder の対象問題のうち、COBOL の解答が提出されていない問題が存在したためである。

4.2 精度の調査方法

作成したデータセットに含まれるソースコードを翻訳し、翻訳の精度を調査する。本研究における評価指標は全てテストの合格率を指標とする計算精度 [34] を採用する。

具体的に C++ から Rust の翻訳の精度を調査する場合の流れを図 1 に例示する。

²<https://atcoder.jp/>

³<https://www.dropbox.com/sh/nx3tnilzqz7df8a/AAAY1Tq2tiEH15hsESw6-yfLa?dl=0>

⁴<https://www.antlr.org/>

表 1: データセットのソースコード情報

言語	ソースコード数	平均行数	平均トークン数
C++	140	22.5	131.5
COBOL	98	35.5	316.9
Go	140	19.7	189.8
Java	140	24.0	163.5
Python3	140	9.1	82.9
Rust	140	16.5	208.7



図 1: C++から Rust の翻訳の精度調査の流れ

1. データセットに含まれる C++のソースコードを何らかのツールを用いて Rust で記述されたソースコードに翻訳.
2. 対応する全てのテストケースを実行する. このとき, 実行時間を 10 秒に制限し, これを超えた場合はテストケースに失敗したとみなす.
3. 全てのテストケースに通過すれば翻訳は正しいとし, 正しい翻訳が出力される精度を調査する.

4.3 各言語の実行環境

各言語の実行環境を以下に示す.

C++

- OS: Windows 10 Pro for Workstations
- g++: 9.20

COBOL

- OS: Ubuntu 22.04.3 LTS

- cobb: 3.1.2.0

Go

- OS: Windows 10 Pro for Workstations
- go: 1.20.6

Java

Java は launch4j⁵を用いて jar ファイルから exe ファイルを作成して実行した。環境は以下の通り。

- OS: Windows 10 Pro for Workstations
- javac: 20.0.2
- launch4j: 3.50

Python3

- OS: Windows 10 Pro for Workstations
- python: 3.9.13

Rust

- OS: Windows 10 Pro for Workstations
- cargo: 1.75.0-nightly

⁵<https://launch4j.sourceforge.net/>

5 調査 1: ニューラル機械翻訳モデルと GPT-4 の比較

GPT-4 と最新のニューラル機械翻訳モデルである TransCoder-IR との精度比較を行う。TransCoder-IR は関数単位での翻訳を行うモデルであるため、GPT-4 でも同様に関数単位で翻訳を行う。対象の言語は TransCoder-IR が学習に使用している C++, Go, Java, Rust とする。

5.1 データセットの加工

本調査では関数単位での翻訳の精度の調査を行いたいが、先述したデータセットではソースコード単位での翻訳の精度しか調査できない。そこで関数単位の翻訳の精度を調査するために、先述したデータセットの C++, Go, Java, Rust のソースコードを加工する。

1. 各ソースコードにおいて、以下の2つの関数を含むように手作業で変更する。
 - 関数 main: 標準入力からの入力の受け取りと関数 solve の呼び出し。
 - 関数 solve: 処理と処理結果の標準出力への出力。

このとき、書き換え後のソースコードは関数 main と関数 solve 以外の関数を含まないようにする。書き換えの例を図 2 に示す。

2. 書き換え後のソースコードを実行し、全てのテストケースに通ることを確認する。この処理によって手作業で行った書き換えの正しさが保証される。

加工後のデータセットのソースコード情報を表 2 に示す。トークン数の算出には ANTLR を利用した。

このデータセットに含まれる各ソースコードの関数 solve を翻訳して精度を調査する。

表 2: 調査 1 で使用するデータセットのソースコード情報

言語	ソースコード数	関数 solve の平均行数	関数 solve の平均トークン数
C++	140	17.2	105.5
Go	140	10.7	147.7
Java	140	14.1	110.5
Rust	140	11.3	191.4

```

int main() {
    int abc, a, b, c, d;
    cin >> abc;
    a = abc / 100;
    b = (abc % 100) / 10;
    c = (abc % 100) % 10;
    d = a + b + c;
    cout << d * 100 + d * 10 + d << endl;
    return 0;
}

```



```

void solve(int abc)
{
    int a, b, c, d;
    a = abc / 100;
    b = (abc % 100) / 10;
    c = (abc % 100) % 10;
    d = a + b + c;

    cout << d * 100 + d * 10 + d << endl;
}

int main()
{
    int abc;
    cin >> abc;

    solve(abc);
    return 0;
}

```

図 2: C++のソースコードを関数 solve と関数 main に分ける処理の例

5.2 調査方法

データセットの各ソースコードの関数 solve を TransCoder-IR, GPT-4 で翻訳し, 翻訳の精度の調査を行う. C++から Rust に翻訳する場合の実験の流れを図 3 に例として示す.

1. データセットに含まれる C++のソースコードの関数 solve を取得する.
2. 関数 solve を TransCoder-IR, あるいは GPT-4 を用いて Rust で記述された関数に翻訳する. GPT-4 においては出力されたコードブロックを出力とし, 複数出力された場合は最後に出力されたコードブロックを出力とする. ここで出力された関数を関数 solve' とする.
3. 関数 solve' を含む実行可能なソースコードを作成する. 具体的には, 関数 solve' を呼



図 3: 調査 1 における C++ から Rust の翻訳の精度調査の流れ

び出す関数 `main` とライブラリのインポートを手動で補完する。このとき関数 `solve'` は一切変更しない。

4. テストケースを全て実行し、全てのテストケースに通過すれば翻訳は正しいとする。このとき、実行時間を 10 秒に制限し、これを越えた場合はテストケースに失敗したとみなす。
5. 正しい翻訳が出力される精度を調査する。

5.3 モデルの設定

TransCoder-IR は配布されているモデル⁶をそのまま使用した。GPT-4 は API を使用し、モデルは `gpt-4-0613` を選択した。C++ から Rust への翻訳を行う際のプロンプトを以下に例示する。

あなたは優秀なソフトウェアエンジニアです。
 C++ で記述された以下の関数を Rust の関数に翻訳してください。必ず翻訳結果のみを出力してください。
 ...
 {C++ で記述された関数}
 ...

5.4 結果

結果を表 3 に示す。表における縦軸は翻訳元言語、横軸は翻訳先言語を示す。結果から、GPT-4 が全てのペアにおいて TransCoder-IR よりも精度が高いことがわかる。このことから、GPT-4 は既存のニューラル機械翻訳モデルよりも高精度であると言える。

⁶<https://github.com/facebookresearch/CodeGen/blob/main/docs/TransCoder-IR.md>

表 3: TransCoder-IR と GPT-4 における関数単位の翻訳の精度 (TIR: TransCoder-IR)

	C++		Go		Java		Rust		平均	
	TIR	GPT-4								
C++	-	-	0.45	0.93	0.54	0.93	0.24	0.85	0.41	0.90
Go	0.29	0.89	-	-	0.33	0.86	0.16	0.84	0.26	0.86
Java	0.41	0.91	0.34	0.94	-	-	0.19	0.92	0.31	0.92
Rust	0.07	0.84	0.05	0.88	0.04	0.83	-	-	0.05	0.85
平均	0.26	0.88	0.28	0.92	0.30	0.87	0.20	0.87	0.26	0.88

5.5 考察

5.5.1 TransCoder-IR の精度

TransCoder-IR に関する報告 [36] では C++, Go, Java, Rust の 4 言語で翻訳の精度を調査した結果、精度の平均は 0.44 と報告されている。しかし、本実験では同様の言語、同様の指標であるにも関わらず精度の平均が 0.26 と大幅に低下している。原因を調査するために TransCoder-IR の評価用データセット (以下, TransCoder-test) と本実験で作成したデータセットの比較を行う。

TransCoder-test は TransCoder [34] の評価時に作成されたデータセットを拡張したもので、プログラミング学習サイトである GeeksForGeeks⁷に提出されたソースコードから作成されている。TransCoder-test に含まれる関数を確認したところ、以下の特徴が確認できた。

- 全ての関数は 1 つの数値, または bool を返す。
- 関数内では標準出力への出力を行わない。
- 関数の引数の型が限定的である。

「関数の引数の型が限定的である」とは、例えば、Rust における関数の引数の型の種類は `mut i32` (可変 32bit 符号付き整数), `*mut i32` (i32 型のデータへの可変ポインタ), `mut u32` (32bit 符号なし整数), `mut f32` (32 ビットの浮動小数点数), `mut f64` (64 ビットの浮動小数点数) の 5 種類のみしか含んでおらず, `usize` (環境依存符号なし整数), `Vec<->` (動的配列), `String` (文字列) などの一般的に使用される型が含まれていないことを示す。

また、関連研究において、TransCoder-test は単純な関数が多く含まれており、それによって不当に精度が高く報告されている可能性について報告されている [15]。

⁷<https://www.geeksforgeeks.org/>

表 4: 関数のトークン数と翻訳の精度の相関係数

(a) TransCoder-IR における関数のトークン数と翻訳の精度の相関係数					(b) GPT-4 における関数のトークン数と翻訳の精度の相関係数				
	C++	Go	Java	Rust		C++	Go	Java	Rust
C++	-	-0.36	-0.29	-0.19	C++	-	-0.21	-0.26	-0.29
Go	-0.19	-	-0.29	-0.33	Go	0.02	-	-0.03	-0.17
Java	-0.28	-0.31	-	-0.28	Java	-0.04	-0.31	-	-0.22
Rust	-0.07	-0.14	-0.06	-	Rust	-0.08	-0.13	-0.11	-

対して、本研究で作成したデータセットは関数の引数の型を限定しておらず、元のソースコードで記述された型をそのまま使用しているため、引数の型において TransCoder-test よりも一般的であると考えられる。また、本研究で作成したデータセットには複数の値を標準出力に出力する関数も含むため、戻り値が1つに限定されている TransCoder-test の関数よりも処理的に複雑な関数が含まれていると考えられる。

つまり、本研究で作成したデータセットは TransCoder-test よりも一般的で複雑な関数を含むため、TransCoder-IR に関する報告よりも TransCoder-IR の精度が低下したと考えられる。

5.5.2 関数のトークン数と翻訳の精度の相関

関数単位の翻訳におけるトークン数と翻訳の精度の相関を調査する。翻訳が正しい場合は1、不正な場合に0として、翻訳元の関数のトークン数との相関係数を算出した。この相関係数は-1に近づくほど、トークン数が少ないほど翻訳の精度が向上する傾向にあることを示す。トークン数の算出には ANTLR を利用した。

調査結果を表4に示す。表における縦軸は翻訳元言語、横軸は翻訳先言語を示す。また、相関係数の絶対値が0.3以上の場合に背景色を緑としている。結果から、トークン数と翻訳の精度には一部弱い相関関係があるが、全体を通して大きな相関関係がないことが確認できた。つまり TransCoder-IR, GPT-4 のいずれにおいても、今回のデータセットに含まれるような短い関数においては、関数のトークン数は翻訳の精度に大きな影響はないということがわかる。

6 調査 2: 異なる特徴の言語間における翻訳の精度

C++, COBOL, Go, Java, Rust, Python3 のソースコードを翻訳し、レガシーな言語とモダンな言語間や、静的型付け言語と動的型付け言語間などの異なる特徴を持つ言語間での翻訳の精度の傾向を調査する。各言語の主な特徴を表 5 に示す。

また、同時にコード単位での翻訳の精度を調査する。

6.1 調査方法

先述したデータセットに含まれるソースコードを翻訳し、精度を調査する。C++から Rust に翻訳する場合の実験の流れを例示する。

1. データセットに含まれる C++ のソースコードを GPT-4 を用いて Rust で記述されたソースコードに翻訳。このとき、GPT-4 が出力したコードブロックを出力とし、複数出力された場合は最後に出力されたコードブロックを出力とする。
2. 各問題のテストケースを実行し、全てのテストケースに通過すれば翻訳は正しいとする。このとき、実行時間を 10 秒に制限し、これを超えた場合はテストケースに失敗したとみなす。
3. 正しい翻訳が出力される精度を調査する。

6.2 モデルの設定

GPT-4 は API を使用し、モデルは gpt-4-0613 を選択した。C++ から Rust への翻訳を行う際のプロンプトを以下に例示する。

```
あなたは優秀なソフトウェアエンジニアです。  
C++ で記述された以下のソースコードを Rust のソースコードに翻訳してください。必ず翻訳結果のみを出力してください。  
...  
{C++ で記述されたソースコード}  
...
```

6.3 結果

実験結果を表 6 に示す。また、各翻訳におけるコンパイルエラーの割合を表 7 に示す。表における縦軸は翻訳元言語、横軸は翻訳先言語を示す。

表 5: 各言語の主な特徴

言語	型付け	登場時期	特徴
C++	静的	1983 年	C 言語を高水準言語として使用できるように機能を拡張した言語.
COBOL	静的	1959 年	仕様が古く, 一般的にレガシーな言語と言われる. モダンな言語に比べて機能が少ない.
Go	静的	2009 年	シンプルな記法と並列性を重視した言語.
Java	静的	1995 年	JVM 上で動作するためプラットフォームに依存せずに動作可能.
Python	動的	1991 年	動的型付け言語であり, 実行時に型付けが行われる.
Rust	静的	2010 年	所有権などの独自の仕組みによってガベージコレクションなしでメモリ安全性を達成.

6.4 考察

6.4.1 翻訳元言語と翻訳先言語が精度に与える影響

表 6 から, 翻訳元言語別の平均精度はばらつきが少ない (分散 = 0.001) のに対して, 翻訳先言語別の平均精度はばらつきが大きい (分散 = 0.08) ことが確認できる. このことから, 翻訳元言語よりも翻訳先言語の方が精度に影響することがわかる.

特に, C++, Java が翻訳先言語である場合は精度がそれぞれ 0.81, 0.85 と比較的精度が高いのに対して, COBOL が翻訳先言語である場合は精度が 0.01 と非常に低いことがわかる. また, 表 7 から翻訳先言語が COBOL の場合のコンパイルエラーの割合が他言語と比べて高いことがわかる. このことから, GPT-4 は COBOL のようなレガシーな言語のソースコードを構文的に正しく生成する能力が他言語と比べて低いことが考えられる.

6.4.2 動的型付け言語と静的型付け言語間の翻訳の精度

関連研究 [15] において, C++, C#, Java, JavaScript, Python3 の 5 言語で事前学習モデルの翻訳の精度調査が行われており, 一般的な事前学習モデルでは動的型付け言語から静的型付け言語への翻訳の精度が低下すると報告されている. 本実験では Python3 のみが動的型付け言語であり, 残りの言語は静的型付け言語であるため, 静的型付け言語から動的型付け言語への翻訳の精度の平均は, 翻訳先言語が Python3 の場合の精度の平均である 0.65 であり, 動的型付け言語から静的型付け言語への翻訳の精度の平均は, 翻訳元言語が Python3 の場合の精度の平均である 0.60 となる. これらの精度に大きな差は見られないため, GPT-4

表 6: GPT-4 のソースコード単位の翻訳の精度

GPT-4	C++	COBOL	Go	Java	Python3	Rust	平均
C++	-	0.01	0.70	0.90	0.63	0.63	0.59
COBOL	0.53	-	0.29	0.65	0.58	0.54	0.52
Go	0.90	0.01	-	0.94	0.67	0.58	0.62
Java	0.91	0.01	0.60	-	0.56	0.64	0.54
Python3	0.84	0.02	0.60	0.87	-	0.66	0.60
Rust	0.86	0.00	0.62	0.90	0.74	-	0.62
平均	0.81	0.01	0.56	0.85	0.65	0.61	0.58

表 7: GPT-4 のソースコード単位での翻訳におけるコンパイルエラーの割合

GPT-4	C++	COBOL	Go	Java	Python3	Rust	平均
C++	-	0.76	0.16	0.01	0.00	0.10	0.21
COBOL	0.07	-	0.21	0.05	0.00	0.10	0.09
Go	0.01	0.72	-	0.01	0.00	0.08	0.16
Java	0.00	0.65	0.25	-	0.00	0.11	0.20
Python3	0.01	0.69	0.16	0.01	-	0.17	0.21
Rust	0.02	0.71	0.23	0.01	0.00	-	0.19
平均	0.02	0.71	0.20	0.02	0.00	0.11	0.18

においては関連研究で報告されている動的型付け言語から静的型付け言語への翻訳の精度が低下するような傾向は無いと考えられる。

6.4.3 ソースコードのトークン数と翻訳の精度の相関

ソースコード単位での翻訳におけるトークン数と翻訳の精度の相関の調査を行う。調査 1 の考察における実験と同様に、翻訳が正しい場合は 1、不正な場合に 0 として、翻訳元のソースコードのトークン数との相関係数を算出した。この相関係数は -1 に近づくほど、トークン数が少ないほど翻訳の精度が向上する傾向にあることを示す。トークン数の算出には ANTLR を利用した。

調査結果を表 8 に示す。表における縦軸は翻訳元言語、横軸は翻訳先言語を示す。結果か

表 8: ソースコードのトークン数と翻訳の精度の相関係数

GPT-4	C++	COBOL	Go	Java	Python	Rust
C++	-	-0.05	-0.10	-0.29	-0.10	-0.16
COBOL	-0.09	-	0.03	-0.13	-0.19	-0.04
Go	0.02	-0.03	-	0.08	-0.12	-0.02
Java	-0.15	-0.03	-0.05	-	0.10	-0.25
Python	-0.07	-0.13	0.03	-0.11	-	-0.25
Rust	-0.06	0.00	-0.25	-0.08	-0.11	-

表 9: 調査 2 の結果に静的解析ツールを適用した場合の精度

(a) goimports を適用した場合の精度

GPT-4 (goimports)	Go
C++	0.82 (+0.12)
COBOL	0.32 (+0.03)
Java	0.70 (+0.10)
Python3	0.68 (+0.08)
Rust	0.71 (+0.09)
平均	0.65 (+0.09)

(b) cargo-fix を適用した場合の精度

GPT-4 (cargo-fix)	Rust
C++	0.65 (+0.01)
Go	0.58 (± 0.00)
COBOL	0.54 (± 0.00)
Java	0.66 (+0.02)
Python3	0.66 (+0.01)
平均	0.62 (+0.01)

ら、全てのコードのペアにおいて相関係数の絶対値が 0.3 以上のものが存在しない、つまり相関がないことが確認できる。このことから、ソースコード単位での翻訳においても関数単位の翻訳と同様に、トークン数と翻訳の精度に大きな相関関係がないことがわかる。

6.4.4 静的解析ツールによるコンパイルエラーの解消

翻訳先言語が Go である場合のコンパイルエラーを調査すると未使用のパッケージをインポートしていることによるコンパイルエラーが多く含まれていた。これは goimports⁸という静的解析ツールを使うことで自動解決できる。また、翻訳先言語が Rust である場合のコンパイルエラーを調査すると、Rust のコンパイラが提案した修正を自動で適用するツールで

⁸<https://pkg.go.dev/golang.org/x/tools/cmd/goimports>

表 10: 調査 2 の結果に静的解析ツールを適用した場合のコンパイルエラーの割合

(a) goimports を適用した場合のコンパイルエラーの割合 (b) cargo-fix を適用した場合のコンパイルエラーの割合

GPT-4 (goimports)	Go	GPT-4 (cargo-fix)	Rust
C++	0.03 (−0.13)	C++	0.09 (−0.01)
COBOL	0.12 (−0.09)	COBOL	0.10 (±0.00)
Java	0.04 (−0.21)	Go	0.07 (−0.01)
Python3	0.06 (−0.10)	Java	0.09 (−0.02)
Rust	0.11 (−0.12)	Python3	0.15 (−0.02)
平均	0.07 (−0.13)	平均	0.10 (−0.01)

ある cargo-fix⁹を用いることで自動解決できるエラーが含まれていることが確認できた。これらのツールを用いることでどのように精度が向上するか調査する。

調査 2 で出力された Go コードと Rust コードにそれぞれ goimports と cargo-fix を適用した場合の精度を表 9 に、コンパイルエラーの割合を表 10 に示す。表における縦軸は翻訳元言語、横軸は翻訳先言語を示す。また、括弧内の数値は各ツール適用前との差分を示し、表 9 においては精度が上がった場合は背景色を緑、精度が変化しなかった場合は背景色を灰とし、表 10 においては割合が低下した場合は背景色を緑、割合が変化しなかった場合は背景色を灰としている。

結果から、静的解析ツールを利用することで一部のコンパイルエラーの解消、精度の向上が可能であることが確認できる。特に、goimports を適用した場合には精度が平均+0.09、コンパイルエラーの割合が平均−0.13 と大幅に性能が向上している。

この結果は、GPT-4 などのコード生成可能な LLM に静的解析ツールを組み込むことで、さらに高精度にコード生成を行うことができるようになることを示す。

⁹<https://doc.rust-lang.org/cargo/commands/cargo-fix.html>

7 調査 3: プロンプトの言語が翻訳の精度に与える影響

コード翻訳タスクにおけるプロンプトの言語の影響を調査するために、プロンプトの言語が英語の場合と日本語の場合の翻訳の精度を比較する。

7.1 調査方法

プロンプトの言語を英語に変更して実験を行い、調査 2 の結果と比較する。GPT-4 に与えるプロンプト以外の条件は調査 2 と同じとする。

7.2 モデルの設定

GPT-4 は API を使用し、モデルは gpt-4-0613 を選択した。C++ から Rust への翻訳を行う際のプロンプトを以下に例示する。

```
You are an excellent software developer.  
Translate the following source code, written in C++, into Rust source code. Make  
sure that you only output the results of the translation.  
...  
{C++で記述されたソースコード}  
...
```

7.3 結果

プロンプトを英語に変更して実験した精度を表 11 に、コンパイルエラーの割合を表 12 に示す。表における縦軸は翻訳元言語、横軸は翻訳先言語を示す。また、括弧内の数値はプロンプトが日本語の場合の精度との差分を示し、表 11 においては精度が上がった場合は背景色を緑、精度が下がった場合は背景色を赤、精度が変化しなかった場合は背景色を灰、表 12 においては割合が減少した場合は背景色を緑、割合が増加した場合は背景色を赤、割合が変化しなかった場合は背景色を灰としている。

調査 3 の結果と調査 2 の結果に対して T 検定を行った結果、精度には有意差が存在 ($p = 0.002 < 0.05$) し、コンパイルエラーの割合には有意差が存在しなかった ($p = 0.07 > 0.05$)。つまり、プロンプトが日本語のほうが精度が高くなり、コンパイルエラーの割合は差がないことが確認できる。

この結果から、コード翻訳タスクに関しては日本語話者はプロンプトを英語に翻訳する必要がないことがわかる。

表 11: プロンプトを英語に変更した場合の精度

GPT-4	C++	COBOL	Go	Java	Python3	Rust	平均
C++	-	0.01 (±0.00)	0.66 (-0.04)	0.91 (+0.01)	0.67 (+0.04)	0.58 (-0.05)	0.57 (-0.02)
COBOL	0.51 (-0.02)	-	0.17 (-0.12)	0.58 (-0.07)	0.54 (-0.04)	0.46 (-0.07)	0.45 (-0.06)
Go	0.91 (+0.01)	0.03 (+0.02)	-	0.91 (-0.03)	0.63 (-0.04)	0.56 (-0.02)	0.61 (-0.01)
Java	0.92 (+0.01)	0.01 (±0.00)	0.58 (-0.02)	-	0.50 (-0.06)	0.63 (-0.01)	0.53 (-0.01)
Python3	0.79 (-0.05)	0.00 (-0.02)	0.39 (-0.21)	0.84 (-0.03)	-	0.69 (+0.04)	0.54 (-0.06)
Rust	0.84 (-0.02)	0.01 (+0.01)	0.59 (-0.03)	0.89 (-0.01)	0.57 (-0.17)	-	0.58 (-0.04)
平均	0.79 (-0.02)	0.01 (±0.00)	0.49 (-0.07)	0.83 (-0.02)	0.58 (-0.07)	0.58 (-0.03)	0.55 (-0.03)

7.4 考察

7.4.1 コード生成タスクにおけるプロンプトの言語の影響

LLMが出力するソースコードの安全性に関する調査 [45] において、プロンプトが英語の場合も日本語の場合も生成されるコードの安全性は基本的に変化せず、一部の実験で日本語の場合のほうが有意に安全であったと報告されている。この報告と本実験での結果を考慮すると、コード生成に関するタスクにおいてはプロンプトを日本語で記述した方が英語で記述した場合よりも高精度な出力が返ってくる可能性が考えられる。

日本語と英語以外の言語においても精度の差があると考えられるため、コード生成タスクにおけるプロンプトの言語の影響の詳細な調査は今後の課題である。

表 12: プロンプトを英語に変更した場合のコンパイルエラーの割合

GPT-4	C++	COBOL	Go	Java	Python3	Rust	平均
C++	-	0.77 (+0.01)	0.15 (-0.01)	0.02 (+0.01)	0.00 (±0.00)	0.15 (+0.05)	0.22 (+0.01)
COBOL	0.09 (+0.12)	-	0.31 (+0.10)	0.07 (+0.02)	0.00 (±0.00)	0.26 (+0.16)	0.15 (+0.06)
Go	0.01 (±0.00)	0.64 (-0.08)	-	0.01 (±0.00)	0.00 (±0.00)	0.07 (-0.01)	0.15 (-0.01)
Java	0.02 (+0.02)	0.74 (+0.09)	0.24 (-0.01)	-	0.00 (±0.00)	0.09 (-0.02)	0.22 (+0.02)
Python3	0.04 (+0.03)	0.66 (-0.03)	0.19 (+0.03)	0.03 (+0.12)	-	0.16 (-0.01)	0.22 (+0.01)
Rust	0.02 (±0.00)	0.74 (+0.03)	0.16 (-0.07)	0.01 (±0.00)	0.00 (±0.00)	-	0.19 (±0.00)
平均	0.04 (+0.02)	0.71 (±0.00)	0.21 (+0.01)	0.03 (+0.01)	0.00 (±0.00)	0.15 (+0.04)	0.19 (+0.01)

8 調査 4: プロンプトに Chain-of-Thought を導入した場合の精度

Chain-of-Thought をプロンプトに導入することで精度がどのように向上するか調査する。本調査では単純にプロンプトの末尾に思考の連鎖を促すプロンプトを追加する Zero-Shot Chain-of-Thought [20] を採用する。

8.1 調査方法

プロンプトに Chain-of-Thought を導入して調査 2 の結果と比較する。

1. データセットに含まれる C++ のソースコードを GPT-4 を用いて Rust で記述されたソースコードに翻訳。プロンプトには Chain-of-Thought を含む。
2. プロンプトに Chain-of-Thought を導入することで断片的にソースコードが出力される場合があり、そのような場合は単純にコードブロックの結合を行う。このとき以下のルールを適用する。
 - 必要なコードブロックを手作業で選択して結合を行う。

COBOLのプログラムからRustのプログラムへの変換は以下の通りです：

1. ファイルの読み込み：

```
use std::fs::read_to_string;
```

2. 文字列の変換：

```
use std::str::to_uppercase;
```

3. 主要な処理部分：

```
fn main() -> std::io::Result<()> {  
    let mut inp = read_to_string("sysin"); // ファイルの読み込み  
    inp = inp.trim().to_string(); // 文字列のトリム  
    println!("{}", inp.to_uppercase()); // 文字列の大文字変換と表示  
    Ok(())  
}
```



コードブロックの結合

```
use std::fs::read_to_string;  
use std::str::to_uppercase;  
fn main() -> std::io::Result<()> {  
    let mut inp = read_to_string("sysin"); // ファイルの読み込み  
    inp = inp.trim().to_string(); // 文字列のトリム  
    println!("{}", inp.to_uppercase()); // 文字列の大文字変換と表示  
    Ok(())  
}
```

図 4: GPT-4 が断片的に出力したコードブロックを結合する例

- コードブロック内は一切変更しない.
- コードブロックの順序は入れ替えない.

結合の例を図 4 に示す.

3. 各問題のテストケースを実行し、全てのテストケースに通過すれば翻訳は正しいとする。このとき、実行時間を 10 秒に制限し、これを超えた場合はテストケースに失敗したとみなす。
4. 正しい翻訳が出力される精度を調査する。

表 13: プロンプトに Chain-of-Thought を導入した場合の精度

GPT-4	C++	COBOL	Go	Java	Python3	Rust	平均
C++	-	0.01 (±0.00)	0.71 (+0.01)	0.89 (-0.01)	0.66 (+0.03)	0.55 (-0.08)	0.56 (-0.03)
COBOL	0.47 (-0.06)	-	0.16 (-0.13)	0.40 (-0.25)	0.48 (-0.10)	0.34 (-0.20)	0.37 (-0.15)
Go	0.89 (-0.01)	0.02 (+0.01)	-	0.86 (-0.08)	0.66 (-0.01)	0.53 (-0.05)	0.59 (-0.03)
Java	0.89 (-0.02)	0.01 (±0.00)	0.59 (-0.01)	-	0.54 (-0.02)	0.63 (-0.01)	0.53 (-0.01)
Python3	0.71 (-0.13)	0.01 (-0.01)	0.54 (-0.06)	0.76 (-0.11)	-	0.63 (-0.03)	0.53 (-0.07)
Rust	0.86 (±0.00)	0.01 (+0.01)	0.59 (-0.03)	0.87 (-0.03)	0.67 (-0.07)	-	0.60 (-0.02)
平均	0.76 (-0.05)	0.01 (±0.00)	0.52 (-0.04)	0.76 (-0.09)	0.60 (-0.05)	0.54 (-0.11)	0.53 (-0.05)

8.2 モデルの設定

GPT-4 は API を使用し、モデルは gpt-4-0613 を選択した。C++ から Rust への翻訳を行う際のプロンプトを以下に例示する。プロンプト末尾の「段階的に考えてください」の箇所が Zero-Shot Chain-of-Thought に該当する。

あなたは優秀なソフトウェアエンジニアです。C++ で記述された以下のソースコードを Rust のソースコードに翻訳してください。

...

{C++ で記述されたソースコード }

...

段階的に考えてください。

表 14: プロンプトに Chain-of-Thought を導入した場合のコンパイルエラーの割合

GPT-4	C++	COBOL	Go	Java	Python3	Rust	平均
C++	-	0.68 (-0.08)	0.09 (-0.07)	0.01 (±0.00)	0.00 (±0.00)	0.11 (+0.01)	0.18 (-0.03)
COBOL	0.08 (+0.01)	-	0.29 (+0.08)	0.20 (+0.15)	0.00 (±0.00)	0.18 (+0.08)	0.15 (+0.06)
Go	0.01 (±0.00)	0.68 (-0.04)	-	0.04 (+0.03)	0.00 (±0.00)	0.16 (+0.08)	0.18 (+0.02)
Java	0.03 (+0.03)	0.74 (+0.09)	0.21 (-0.04)	-	0.00 (±0.00)	0.15 (+0.04)	0.23 (+0.03)
Python3	0.05 (+0.04)	0.65 (-0.04)	0.12 (-0.04)	0.01 (±0.00)	-	0.15 (-0.02)	0.20 (-0.01)
Rust	0.04 (+0.02)	0.69 (-0.02)	0.13 (-0.10)	0.03 (+0.02)	0.00 (±0.00)	-	0.18 (-0.01)
平均	0.04 (+0.02)	0.69 (-0.02)	0.17 (-0.03)	0.06 (+0.04)	0.00 (±0.00)	0.15 (+0.04)	0.19 (+0.01)

8.3 結果

プロンプトに Chain-of-Thought を導入した場合の精度を表 13 に、コンパイルエラーの割合を表 14 に示す。表における縦軸は翻訳元言語、横軸は翻訳先言語を示す。また、括弧内の数値はプロンプトが日本語の場合の精度との差分を示し、表 13 においては精度が上がった場合は背景色を緑、精度が下がった場合は背景色を赤、精度が変化しなかった場合は背景色を灰、表 14 においては割合が減少した場合は背景色を緑、割合が増加した場合は背景色を赤、割合が変化しなかった場合は背景色を灰としている。

調査 4 の結果と調査 2 の結果に対して T 検定を行った結果、精度には有意差が存在 ($p = 0.0003 < 0.05$) し、コンパイルエラーの割合には有意差が存在しなかった ($p = 0.43 > 0.05$)。

つまり、コード翻訳タスクにおいては Chain-of-Thought を導入しない方が精度が高くなることが言える。

8.4 考察

8.4.1 Chain-of-Thought による逐次的な翻訳の悪影響

表 13 の結果から、特に翻訳元言語が COBOL の場合に大きく精度が低下していることがわかる。Chain-of-Thought を導入することでより逐次的に翻訳を行うようになり、そのことがパラダイムや仕様が大きく違う言語間の翻訳において悪影響を及ぼしていると考えられる。

9 妥当性への脅威

9.1 内的妥当性

9.1.1 データセットに含まれるテストケースの品質

データセットに含まれるテストケースの網羅性が低いと実験の信頼性が失われる。本研究で作成したデータセットのドメインである AtCoder では、コンテスト中にテストケースの不備が発覚した場合にコンテスト後に不備を補うテストケースが追加される。

本研究で作成したデータセットはそのようなテストケースも含んでいるため、テストケースの不備による妥当性の脅威は小さいと考えられる。

9.1.2 GPT-4 の出力の自然言語部分の影響

本実験では GPT-4 の自然言語部分を考慮しておらず、自然言語部分に含まれる指示に従うことで精度が向上する可能性が考えられる。

例えば、調査 4 の COBOL から Java への翻訳において図 5 のような結果が出力された。今回の調査 4 では単純な結合のみを行った結果、コンパイルエラーとなったが、自然言語部分の指示に従って適切にコードブロックの結合を行い、インポートを補完することでコンパイルエラーを解決することができる。

このように、出力の自然言語部分の指示に従って適切にコードブロックの統合、補完をすることで精度が向上する可能性が考えられる。

9.1.3 LLM の非決定性による影響

LLM は非決定的であり [22]、同一モデル、同一プロンプトであっても複数回実行することで出力が変化する。LLM によるコード生成における非決定性の脅威は深刻であり、科学的結論の妥当性に対する重大な脅威が存在すると関連研究で報告されている [30]。

本研究における実験では、各翻訳を一度ずつしか実行していないため LLM の非決定性による妥当性の脅威は存在すると考えられる。

9.2 外的妥当性

9.2.1 再現性の問題

LLM は同じモデルとプロンプトであっても出力が一定ではないため、本研究の結果を完全に再現することは不可能である。

```
COBOL の各部分を Java に翻訳すると次のようになります。

まず、データ部を考えます。

```java
public class Main {
 // 中略
}
```

次に、PROCEDURE DIVISIONを考えます。
Javaではメソッドとして表現するのが理にかなっています。

```java
public static void main(String[] args) throws IOException {
 // 中略
}
```

次に、このPROCEDURE内のループについて考えてみましょう。

```java
// ループを2重に実装しましょう。
for(i = 0; i < 7; i++) {
 // 中略
}

// フラグの値に応じて出力を変えています。
if (flg == 1) System.out.println("Yes");
else System.out.println("No");
```

これでCOBOLのプログラムをJavaへ翻訳できました。
お手数となりましたが、上記の片段を統合して完成したプログラムを作成してください。
```

図 5: 自然言語による指示の影響があると考えられる出力 (ソースコード部分は一部省略)

また、GPT-4 は定期的に新しいモデルが発表され、古いモデルは非推奨になる。今後非推奨のモデルが非公開になる可能性も考えられ、その場合には本研究で利用したモデルを用いた実験を行うことは不可能になる。

9.2.2 通常のプロジェクトにおける精度

本研究で作成したデータセットのドメインは競技プログラミングサイトに提出された解答に限られている。そのため、通常のプロジェクトのソースコードにおける翻訳の精度は本研究で報告した精度とは異なる可能性がある。

10 おわりに

本研究では GPT-4 が事前学習していないデータからデータセットを新たに作成して GPT-4 のコード翻訳タスクの精度の調査を行った。具体的には既存のニューラル機械翻訳モデルと GPT-4 の精度比較 (調査 1), 異なる特徴の言語間における翻訳の精度の調査 (調査 2), プロンプトに使用する言語が翻訳の精度に与える影響の調査 (調査 3), プロンプトに Chain-of-Thought を導入した場合の精度の調査 (調査 4) を行った。

調査 1 では GPT-4 が最新のニューラル機械翻訳モデルである TransCoder-IR よりも高精度でコード翻訳を行えることがわかった。調査 2 ではコード翻訳においては翻訳元言語よりも翻訳先言語の影響が大きいことを明らかにし, 特に翻訳先言語が COBOL である場合の精度が低いことがわかった。また, 出力された結果に対して静的解析ツールを利用することで精度が向上することもわかった。調査 3 ではプロンプトの言語が日本語の方が英語の場合よりも高精度で自動コード翻訳を行えることがわかった。調査 4 では Chain-of-Thought は自動コード翻訳タスクにおいては有効ではないことがわかった。

今後の課題として以下が挙げられる。

実プロジェクトにおける精度調査

本研究ではデータセットのドメインが競技プログラミングサイトに提出された解答に限られているため, 通常のプロジェクトにおけるソースコードと性質の乖離がある可能性がある。競技プログラミングサイトに提出された解答と通常のプロジェクトにおけるソースコードの乖離を調査し, 大きく乖離している場合は実プロジェクトから新たにデータセットを作成して精度を調査することが考えられる。

出力されたコードの自然さの調査

GPT-4 が出力したソースコードが自然であるかどうか調査を行う。自然さの調査は関連研究 [13] で述べられている以下の手法を用いて行う。

1. 対象の言語のデータセットから n-gram 言語モデルを学習する。
2. 学習済み n-gram 言語モデルを用いて GPT-4 が出力したソースコードの自然さを計算する。

プロンプトの言語が翻訳の精度に与える影響の詳細な調査

本研究によってプロンプトの言語が日本語のほうが英語の場合よりも翻訳の精度が高いことが明らかになった。自然言語に関するタスクでは, プロンプトの言語が英語の方が日本語

の場合よりも高精度であると報告されている [29] ことから、ソースコードに関するタスクと自然言語に関するタスクではプロンプトの言語の影響が異なる可能性がある。日本語と英語以外のプロンプトで自動コード翻訳を行い、精度の傾向を調査することが考えられる。

プロンプトに Recursively Criticizes and Improves (RCI) [18] を導入した場合の精度の調査

RCI とは LLM の出力を LLM 自身で再帰的に批判することで改善を試みるプロンプトエンジニアリングの手法である。これをプロンプトに導入することで精度がどのように変化するか調査する。また、Chain-of-Thought と同時に使用することで、より性能が向上するという報告 [18] もあることから、Chain-of-Thought と RCI を組み合わせたプロンプトで翻訳の精度の調査を行うことも考えられる。

LLM の非決定性による妥当性の脅威の軽減

内的妥当性の脅威に関する節で述べたように、LLM の出力は非決定的であり、そのことが本実験の結論の妥当性を脅かす可能性がある。このような妥当性の脅威の軽減のために、関連研究 [30] で述べられているような、複数回翻訳を実行して精度の平均・分散・分布などを調査することが挙げられる。

謝辞

本研究を行うにあたり，調査の妥当性に関する御助言など日頃から様々な御指導を賜りました，大阪大学大学院情報科学研究科コンピュータサイエンス専攻 肥後芳樹 教授に，心から深く感謝の意を表します。

本研究の全過程を通して，研究の方向性の検討から本論文の執筆に至るまで，手厚く御指導，御助言を賜りました，大阪大学大学院情報科学研究科コンピュータサイエンス専攻 松下誠 准教授に心より深く感謝いたします。

本研究を行うにあたり，研究活動など幅広い点で適切な御指導，御助言を賜りました，大阪大学大学院情報科学研究科コンピュータサイエンス専攻 神田哲也 助教に，心より深く感謝いたします。

最後に，その他様々な御指導及び御助言を頂いた大阪大学大学院情報科学研究科コンピュータサイエンス専攻肥後研究室の皆様ならびに事務職員 軽部 瑞穂氏に心より深く感謝いたします。

参考文献

- [1] Y. Bang, S. Cahyawijaya, N. Lee, W. Dai, D. Su, B. Wilie, H. Lovenia, Z. Ji, T. Yu, W. Chung, Q. V. Do, Y. Xu, and Pa. Fung. A multitask, multilingual, multi-modal evaluation of chatgpt on reasoning, hallucination, and interactivity. *eprint arXiv:2302.04023*, Feb. 2023. DOI:10.48550/arXiv.2302.04023.
- [2] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. S. Seljebotn, and K. Smith. Cython: The best of both worlds. *Computing in Science & Engineering*, Vol. 13, No. 2, pp. 31–39, Mar. 2011. DOI:10.1109/MCSE.2010.118.
- [3] S. Bubeck, V. Chandrasekaran, R. Eldan, J. Gehrke, E. Horvitz, E. Kamar, P. Lee, Y. T. Lee, Y. Li, S. Lundberg, H. Nori, H. Palangi, M. T. Ribeiro, and Y. Zhang. Sparks of artificial general intelligence: Early experiments with gpt-4. *eprint arXiv:2303.12712*, Mar. 2023. DOI:10.48550/arXiv.2303.12712.
- [4] D. Carr and R. J. Kizior. The case for continued cobol education. *IEEE Software*, Vol. 17, No. 2, pp. 33–36, 2000.
- [5] X. Chen, C. Liu, and D. Song. Tree-to-tree neural networks for program translation. In *Neural Information Processing Systems*, pp. 2552–2562, Montréal, Canada, Dec. 2023.
- [6] OSS Consortium. opensource cobol ダウンロード. <https://www.osscons.jp/ossco-bol/download/>. Accessed: Dec, 2023.
- [7] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In J. Burstein, C. Doran, and T. Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. DOI:10.18653/v1/N19-1423.
- [8] J. Etxaniz, G. Azkune, A. Soroa, O. L. de Lacalle, and M. Artetxe. Do multilingual language models think better in english? *eprint arXiv:2308.01223*, Aug. 2023. DOI:10.48550/arXiv.2308.01223.
- [9] Y. Feng, S. Vanam, M. Cherukupally, W. Zheng, M. Qiu, and H. Chen. Investigating code generation performance of chatgpt with crowdsourcing social data. In *2023*

- IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*, pp. 876–885, 2023. DOI:10.1109/COMPSAC57700.2023.00117.
- [10] GitHub. Github copilot · your ai pair programmer. <https://github.com/features/copilot>. Accessed: Feb, 2024.
- [11] D. Hendrycks, C. Burns, S. Basart, A. Critch, J. Li, D. Song, and J. Steinhardt. Aligning ai with shared human values. In *The International Conference on Learning Representations*, Virtual Event, May 2021.
- [12] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt. Measuring massive multitask language understanding. In *The International Conference on Learning Representations*, Virtual Event, May 2021.
- [13] A. Hindle, E. T. Barr, Z. Su, M. Gabel, and P. Devanbu. On the naturalness of software. In *Proceedings of the 34th International Conference on Software Engineering, ICSE '12*, p. 837–847. IEEE Press, 2012.
- [14] A. Irrera. Banks scramble to fix old systems as it "cowboys" ride into sunset — reuters. <https://web.archive.org/web/20230620040913/https://www.reuters.com/article/us-usa-banks-cobol-idUSKBN17COD8>, Apr. 2017. Accessed: Jan. 2024.
- [15] M. Jiao, T. Yu, Xuan Li, G. J. Qiu, X. Gu, and B. Shen. On the evaluation of neural code translation: Taxonomy and benchmark. In *The International Conference on Software Engineering*, pp. 1597–1608, Pittsburgh, USA, May 2022. DOI:10.1145/3510003.3510060.
- [16] W. Jiao, W. Wang, J. Huang, X. Wang, S. Shi, and Z. Tu. Is chatgpt a good translator? a preliminary study. *eprint arXiv:2301.08745*, Jan. 2023. DOI:10.48550/arXiv.2301.08745.
- [17] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. Scaling laws for neural language models. *eprint arXiv:2001.08361*, Jan. 2020. DOI:10.48550/arXiv.2001.08361.
- [18] G. Kim, P. Baldi, and S. McAleer. Language models can solve computer tasks. *eprint arXiv:2303.17491*, Mar. 2023. DOI:10.48550/arXiv.2303.17491.

- [19] R. J. Kizior, D. Carr, and P. Halpern. Does cobol have a future? In *Proc. Information Systems Education Conf*, Vol. 17, 2000.
- [20] T. Kojima, S. (Shane) Gu, M. Reid, Y. Matsuo, and Y. Iwasawa. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems*, Vol. 35, pp. 22199–22213, 2022.
- [21] V. D. Lai, N. T. Ngo, A. P. B. Veyseh, H. Man, F. Derroncourt, T. Bui, and T. H. Nguyen. Chatgpt beyond english: Towards a comprehensive evaluation of large language models in multilingual learning. *eprint arXiv:2304.05613*, Apr. 2023. DOI:10.48550/arXiv.2304.05613.
- [22] M. Lee, P. Liang, and Q. Yang. Coauthor: Designing a human-ai collaborative writing dataset for exploring language model capabilities. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, CHI '22, New York, NY, USA, 2022. Association for Computing Machinery. DOI:10.1145/3491102.3502030.
- [23] R. Li, L. B. Allal, Y. Zi, N. Muennighoff, D. Kocetkov, C. Mou, M. Marone, C. Akiki, J. Li, J. Chim, Q. Liu, E. Zheltonozhskii, T. Y. Zhuo, T. Wang, O. Dehaene, M. Davaadorj, J. Lamy-Poirier, J. Monteiro, O. Shliazhko, N. Gontier, N. Meade, A. Zebaze, M. Yee, L. K. Umapathi, J. Zhu, B. Lipkin, M. Oblokulov, Z. Wang, R. Murthy, J. Stillerman, S. Sankalp Patel, D. Abulkhanov, M. Zocca, M. Dey, Z. Zhang, N. Fahmy, U. Bhattacharyya, W. Yu, S. Singh, S. Luccioni, P. Villegas, M. Kunakov, F. Zhdanov, M. Romero, T. Lee, N. Timor, J. Ding, C. Schlesinger, H. Schoelkopf, J. Ebert, T. Dao, M. Mishra, A. Gu, J. Robinson, C. J. Anderson, B. Dolan-Gavitt, D. Contractor, S. Reddy, D. Fried, D. Bahdanau, Y. Jernite, C. M. Ferrandis, S. Hughes, T. Wolf, A. Guha, L. von Werra, and H. de Vries. Starcoder: May the source be with you! *eprint arXiv:2305.06161*, May 2023. DOI:10.48550/arXiv.2305.06161.
- [24] Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. Dal Lago, T. Hubert, P. Choy, C.d’Autume, I. Babuschkin, X. Chen, P. S. Huang, J. Welbl, S. Goyal, A. Cherepanov, J. Molloy, D. J. Mankowitz, E. S. Robson, P. Kohli, N. D. Freitas, K. Kavukcuoglu, O. Vinyals. Competition-level code generation with alphacode. *Science*, Vol. 378, No. 6624, pp. 1092–1097, 2022. DOI:10.1126/science.abq1158.

- [25] V. Liu and L. B. Chilton. Design guidelines for prompt engineering text-to-image generative models. In *CHI2022*, No. 384, pp. 1–23, New Orleans, USA, Apr. 2022. DOI:10.1145/3491102.3501825.
- [26] S. Lu, D. Guo, S. Ren, J. Huang, A. Svyatkovskiy, A. Blanco, C. B. Clement, D. Drain, D. Jiang, D. Tang, G. Li, L. Zhou, L. Shou, L. Zhou, M. Tufano, M. Gong, M. Zhou, N. Duan, N. Sundaresan, S. K. Deng, S. Fu, and S. Liu. Codexglue: A machine learning benchmark dataset for code understanding and generation. *eprint arXiv:2102.04664*, Feb. 2021. DOI:10.48550/arXiv.2102.04664.
- [27] OpenAI. Introducing chatgpt. <https://openai.com/blog/chatgpt>. Accessed: Feb, 2024.
- [28] OpenAI. Models. <https://platform.openai.com/docs/models/gpt-3-5>. Accessed: Dec. 2023.
- [29] OpenAI. Gpt-4 technical report. *eprint arXiv:2303.08774*, Mar. 2023. DOI:10.48550/arXiv.2303.08774.
- [30] S. Ouyang, J. M. Zhang, M. Harman, and M. Wang. Llm is like a box of chocolates: the non-determinism of chatgpt in code generation. *eprint arXiv:2308.02828*, Aug. 2023. DOI:10.48550/arXiv.2308.02828.
- [31] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. In M. Walker, H. Ji, and A. Stent, editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pp. 2227–2237, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. DOI:10.18653/v1/N18-1202.
- [32] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 2019.
- [33] Reuters. Cobol blues. <http://fingfx.thomsonreuters.com/gfx/rngs/USA-BANKS-COBOL/010040KH18J/index.html>, 2017. Accessed: Jan. 2024.
- [34] B. Roziere, M. Lachaux, L. Chausson, and G. Lample. Unsupervised translation of programming languages. In *The International Conference on Neural Information Processing Systems*, No. 1730, pp. 20601–20611, Virtual Event, Dec. 2020.

- [35] M. Shanahan. Talking about large language models. *eprint arXiv:2212.03551*, Dec. 2022. DOI:10.48550/arXiv.2212.03551.
- [36] M. Szafraniec, B. Roziere, H. Leather, F. Charton, P. Labatut, and G. Synnaeve. Code translation with compiler representations. In *The International Conference on Learning Representations*, May 2023.
- [37] Gemini Team. Gemini: A family of highly capable multimodal models. *eprint arXiv:2312.11805*, Dec. 2023. DOI:10.48550/arXiv.2312.11805.
- [38] H. Touvron, L. Martin, K. R. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. M. Kloumann, A. Korenev, P. S. Koura, M. A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom. Llama 2: Open foundation and fine-tuned chat models. *eprint arXiv:2307.09288*, Jul. 2023. DOI:10.48550/arXiv.2307.09288.
- [39] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*, p. 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc.
- [40] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, E. H. Chi, T. Hashimoto, O. Vinyals, P. Liang, J. Dean, and W. Fedus. Emergent abilities of large language models. *eprint arXiv:2206.07682*, June 2022. DOI:10.48550/arXiv.2206.07682.
- [41] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou. Chain-of-thought prompting elicits reasoning in large language models. *eprint arXiv:2201.11903*, Jan. 2022. DOI:10.48550/arXiv.2201.11903.
- [42] J. White, Q. Fu, S. Hays, M. Sandborn, C. Olea, H. Gilbert, A. Elnashar, J. Spencer-Smith, and D. C. Schmidt. A prompt pattern catalog to enhance prompt engineering

- with chatgpt. *eprint arXiv:2302.11382*, Feb. 2023. DOI:10.48550/arXiv.2302.11382.
- [43] ZDNET. Microsoft has over a million paying github copilot users: Ceo nadella — zdnet. <https://www.zdnet.com/article/microsoft-has-over-a-million-paying-github-copilot-users-ceo-nadella/>, Oct. 2023. Accessed: Feb, 2024.
- [44] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, Y. Du, C. Yang, Y. Chen, Z. Chen, J. Jiang, R. Ren, Y. Li, X. Tang, Z. Liu, P. Liu, J. Nie, and J. Wen. A survey of large language models. *eprint arXiv:2303.18223*, Mar. 2023. DOI:10.48550/arXiv.2303.18223.
- [45] 山岸伶, 笹晋也, 藤井翔太. 入力時の日本語と英語の差異が chatgpt で生成するコードの安全性に与える影響の考察. コンピュータセキュリティシンポジウム 2023 論文集, pp. 1544–1551, Oct. 2023.