受験番号	※記入不要
氏　名	門谷　拓能
現在の専門	情報通信
希望研究室	自然言語処理学

小　論　文
(情報科学区分)

取り組みたい研究テーマ： 大規模言語モデルによるユーザーが期待するソースコードの生成とソースコードの変換の検討

１．これまでの修学内容

私は大学で大規模言語モデルを用いた逆コンパイル手法に関する検討を行っている。

近年の大規模言語モデルは、自然言語処理やプログラミングコード生成の分野で注目を集めており、膨大なデータセットから文脈を理解し、高い柔軟性を持つコード生成を可能にしている。逆コンパイル技術は、ソフトウェア解析やリバースエンジニアリングにおいて重要な役割を果たすが、従来の逆コンパイルツールは静的解析やルールベースの手法に依存しており、特に複雑な制御構造や最適化されたバイナリコードに対しては精度や可読性の面で限界がある。また、生成されたコードが並列処理に適さないことが多く、高性能コンピューティングを要する現代のソフトウェア開発の要件を十分に満たせていない。

ここで、大規模言語モデルである GPT-4o を用いて逆コンパイル手法を検討する。入力を 16 進数バイナリコードとし、出力をアセンブリコードとするモデルと出力されたアセンブリコードを入力とし、出力を C コードとするモデルを用意した。データは PolyBench を用いた。この状態では期待される出力が得られなかった。そのため、それぞれのモデルに対して、データセットは PolyBench を用いてファインチューニングを行った。その結果は、16 進数バイナリコードからアセンブリコードを生成する段階で困難を示した。そのため、アセンブリコードから C コードを生成するモデルを用いて、実験を行う。テストするデータは PolyBench である。結果は表 1 に示した。これより、大規模言語モデルは従来手法に比べ、処理内容の正確性に加え、並列化が容易なコードを生成する能力において優位性を示した。一方で、モデルのブラックボックス性、データセットの偏りや最大トークン数といった課題も確認された。今後の取り組みとしては、バイナリからアセンブリへの変換精度を向上させることが不可欠である。データセットの拡充だけでなく、トランスフォーマーの最適化や、新たなアプローチの探索により、性能向上を目指す必要がある。

表１　実験結果

	コンパイル	元のコードとの結果比較
2mm	×	×
3mm	○	○
adi	○	×
atax	×	×
bicg	×	×
correlation	×	×
covariance	×	×
deriche	×	×
doitgen	○	○
durbin	○	×
fdtd-2d	○	○
gemm	×	×
gemver	○	○
gesummv	○	○
gramschmidt	○	×
heat-3d	○	×
jacobi-1d	○	×
jacobi-2d	○	×
ludcmp	×	×
mvt	×	×
symm	○	×
syr2k	×	×
syrk	×	×
trmm	○	×

２．取り組みたい内容

従来、ソースコードの生成は手動で行うことが主流であり、開発者がプログラムを一行一行書き進めていた。しかし、ソフトウェア開発においてコードの量が増加し、複雑さも増してきたため、手作業では効率的に対応しきれないという問題が生じている。このため、ソースコードを自動的に生成できる技術が必要とされている。近年、大規模言語モデル（LLM）は膨大なデータを学習することにより、自然言語処理のみならず、プログラミング関連の言語処理ができ、自然言語からコードの要約・生成・修正・

受験番号	※記入不要
氏　名	門谷　拓能
現在の専門	情報通信
希望研究室	自然言語処理学

小　論　文
(情報科学区分)

変換といったコード関連タスクを実行可能である[1]。

研究目的は、大規模言語モデルによるユーザーが期待するソースコードの生成とソースコードの変換である。ソースコードの生成において、ユーザーは特定のタスクに合った正確で効率的なコードを期待する。また、既存のコードを新しい形式に変換したり、異なるプログラミング言語間でコードを移行したりする必要性も高まっている。このため、ユーザーが期待するコードの生成と変換を、より精度高く、柔軟に対応できるシステムの開発が求められる。

大規模言語モデルとして GPT を選定する。GPT だけでシステム開発を行えるくらいの高いパフォーマンスと柔軟性、広範な知識の学習、コンテキスト理解が可能になっているためである[2]。現時点では、ユーザーが期待するコードはC コードで出力するものとし、この生成されたコードを LLVM を用いて中間表現である IR コードに変換する。そこから対応する言語への変換をIRCoderというモデルを用いて行う。C コードの選定理由は、C コードはメモリ管理やシステムコールなど低レベルな操作に強みを持っており、ハードウェア寄りの処理を行うプログラムで使用されることが多い。C コードは多くの学術的な教材やプログラミング理論で使用されており、コンピュータサイエンスの基本を学ぶために広く利用されている。C コードは比較的シンプルな構造を持ちながらも強力な機能を持つため、他の言語に変換する際に良いテストケースとなる。IR コードの選定理由は、IR コードはコンパイラの汎用最適化で扱うコードであり、高水準言語とプラットフォームに依存しない表現となっている。IRCoder は、多言語コード生成タスクにおいて、即時の堅牢性やコード理解の向上を示し、多様なコード生成タスクにおいて一貫した改善をもたらす[3]。

具体的な手法として、プロンプトを用いた実験を行う。まず、C コードの生成である。ここでは、ユーザーから与えられた要求(問題の定義)をもとに、プロンプトを設計し、GPT を使用してC コードを生成する。プロンプト設計に関して、ユーザーが期待する具体的なコードの機能や目的を明確に伝えることが重要である。また、ユーザーが求めるコードの前提条件や環境(例：特定のライブラリや関数を使用すること、バージョンなど)を適切に提供する。さらに、複雑なタスクの場合、プロンプトを段階的に分けて実行させることも有効である。最初にアルゴリズムの概要を説明し、その後に実装の詳細を求めるといった方法である。次に生成されたC コードを IR コードに変換する。ここでは、LLVM を用いて生成された C コードを IR コードに変換する。これより IRCoder というモデルを用いて、対応する言語への変換を行う。

評価方法に関しては、コンパイル可能かどうか、結果が期待されるものになっているか、生成されたC コードは可読性が高いかで評価する。コードのコンパイル可能性は、生成されたソースコードが正しい構文、適切なライブラリのインクルード、変数や関数の宣言、制御フローなどに誤りがないかを評価する。コードが期待通りの動作をするかどうかを確認するためには、与えられた問題に対してコードが正しい結果を返すかどうかを評価する。ここで、C コードと変換後のコードでの同じ入力を与え、結果比較も行う。さらに、もしコンパイルエラーや実行結果が期待通りでない場合、プロンプトの微調整が必要である。例えば、コードが正常に動作しない場合、追加の詳細をプロンプトに加えて再度生成を試みる。このように、「コンパイル可能かどうか」と「結果が期待されるものになっているか」を評価基準として、コード生成とコード変換の精度を高めることができる。プロンプトを改善し、必要に応じてフィードバックを行うことで、最終的にユーザーが期待するソースコードを提供することが可能になると思われる。

参考文献

[1] 伊東和香, et al. ″コード生成タスクにおけるプロンプトの指示形式の差異が与える性能分析.″ *人工知能学会全国大会論文集 第 38 回 (2024)*. 一般社団法人 人工知能学会, 2024.

[2] “【生成 AI】知らないと後悔する、GPT-4o だけでシステム開発を 300%効率化するハック【CodeAGI】”.Qiita.https://qiita.com/nqdior/items/1bef77d46e199f8ec97c,(2025 年 1 月 11 日)

[3] Paul, Indraneil, Goran Glavaš, and Iryna Gurevych. ″Ircoder: Intermediate representations make language models robust multilingual code generators.″ *arXivpreprint arXiv:2403.03894* (2024).