

Do Large Language Models Pay Similar Attention Like Human Programmers When Generating Code?

BONAN KOU, Purdue University, USA

SHENGMAI CHEN*, Brown University, USA

ZHIJIE WANG, University of Alberta, Canada

LEI MA, The University of Tokyo, Japan and University of Alberta, Canada

TIANYI ZHANG, Purdue University, USA

Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering(FSE '24)

背景・問題

近年、LLMがコード生成において大きな進歩を遂げている

- GPT-4は、HumanEvalのPythonプログラミングタスクの84%を正しく解決

一方、LLMがなぜ・どのように自然言語の説明から正しいコードを生成できるかはよくわかっていない

- 原因：LLMの複雑さ，不透明性

➤ 人間のプログラマーとLLMとのギャップが不明

- タスクの深い意味を人間同様に理解しているのか，それとも表面的なパターンを学習しているだけなのか

モデルの注意分析

モデルが予測や生成を行う際、入力のどの部分に、どの程度の”注意度”を割り当てているかを分析

例) コード生成

タスク「Write a **function** to calculate the **average** of a **list** of numbers.」

赤：注意度 高 青：注意度 低

→**赤色**の単語に重点を置いてコードを生成

注意分析により、以下のメリットが得られる

- モデルがどのように機能するかを理解・説明になりうる
- モデルの注意を人間の注意に合わせることで、パフォーマンスが向上
(例：画像分類, 自動運転, コード要約)

目的・アプローチ

コード生成における，LLMと人間とのギャップの調査

- タスク記述の**注意する部分**に違いはあるかを調査
- コード生成におけるLLMの解釈可能性・信頼性の提供
- パフォーマンス向上に向けた知見の提供

アプローチ

- **注意単語**を元に，LLMと人間の注意部分の違いを分析

注意単語：

タスク解決において重要と認識される，タスク記述内のフレーズ・単語・トークン

- LLMと人間の注意単語を比較することで調査

アプローチの概要

①人間の注意単語の分析

- HumanEval+MBPPの1138のタスクを利用
- タスクを解決するために重要だと考えられる単語を手動でラベル付け

②LLMの注意単語の分析

- コード生成時，タスク記述のどの単語を重要視しているかを分析
- 自己注意ベース，勾配ベース，摂動ベースの3種類12手法により分析

③LLMと人間の注意単語の一致度の測定

- ①②で得た，LLM・人間が重要視している単語がどの程度一致するか
- 2種類のトークン重複指標により算出

①人間の注意単語の分析

プログラマーの注意単語が示された、注意ベンチマークをデータセットとして作成

- 方法：Python経験者(5年以上)2人で、重要だと考えられる単語・フレーズを手動でラベル付け

HumanEval Prompt #2
Given a positive floating point number, it can be decomposed into integer parts and decimals. Return the decimal part of the number.
HumanEval Prompt #12
Out of list of strings , return the longest one. Return the first one in case of multiple strings of the same length . Return None in case the input list is empty .

■ Data type ■ Operator ■ Conditional ■ Property

- 4つのキーワードタイプに分けてラベル付け
- 1138のラベル付けされたタスク記述が作成
- 平均単語数：29.6個，平均ラベル付け単語数：7個

②LLMの注意単語の分析

各LLMで1138のタスクに対してコードを生成, 3種類12手法の各注意計算手法でトークンごとの注意スコアを算出

例)

タスク: "Write a function to add two numbers."

トークン: ["Write", "a", "function", "to", "add", "two", "numbers"]

注意スコア: [0.15, 0.1, 0.35, 0.2, 0.55, 0.35, 0.3]

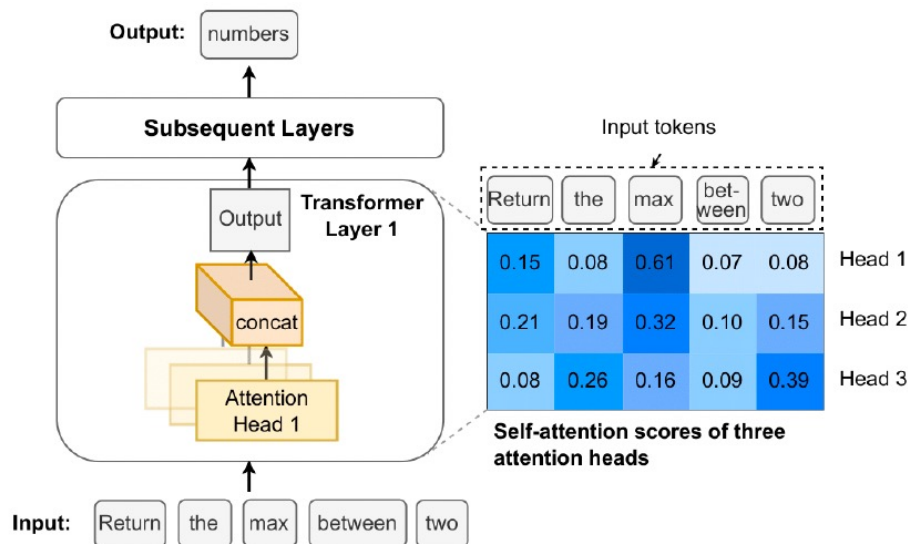
注意単語: "add", "function", "two", ..., "a"

使用モデル: 6個

GPT-4, InCoder-1.3B, CoderGen-2.7B, PolyCoder-2.7B,
CodeParrot-1.5B, GPT-J-6B

②-1 自己注意ベース

TransformerモデルのSelf-Attention機構を直接利用して
トークンの**注意スコア**を算出



- multi-headのスコアは合計で算出
例) "max"の注意度=0.61+0.32+0.16=1.09
- 層について, 3種類の集計方法を採用
例) FIRST: 最初のレイヤーのみの注意スコア
LAST: 最後のレイヤーのみの注意スコア
ALL: すべての層からの注意スコアの合計
- 生成ステップについて, 2種類の集計方法を採用
例) READING: 入力を読み取る際の注意スコア
CODING: コード生成の際の注意スコア

計6パターンの自己注意ベース**注意スコア**算出方法

- レイヤー集計3種類×生成ステップ集計2種類=6手法

②-2 勾配ベース

入力特徴に関するモデルの出力の勾配を利用してモデルの**注意スコア**を算出

- 勾配は、入力の変化が出力に与える影響の大きさを示す

Saliency法

$$s_i = \frac{\partial \mathcal{F}(X)}{\partial x_i}$$

s_i : 入力トークン x_i に対する**注意スコア**

x_i : 入力トークン

$\mathcal{F}(X)$: モデル $\mathcal{F}$ に入力 X を与えた時の出力

Input×Gradient法

$$s_i = x_i \cdot \frac{\partial \mathcal{F}(X)}{\partial x_i}$$

勾配に入力トークンの埋め込み値を掛け合わせたバージョン

計4パターンの勾配ベース**注意スコア**算出方法

- 2手法×READING, CODING

②-3 摂動ベース

入力の一部を変更し、その後出力の差異に基づいてモデルの
注意スコアを算出

- モデルの内部情報が必要ない→GPT-4のような商用モデル向け

SHAP

- ゲーム理論のShapley値(協力ゲームにおける貢献度を表す値)を元に**注意スコア**を算出

BERT Masking

- 入力の各トークンを順番にマスクし、事前学習済みのBERTモデルで予測された置換トークンを用いて、モデルの出力への影響を測定
- 元の出力と新しい出力のBLEUスコアの差が**注意スコア**

③LLMと人間の注意単語の一致度の測定

人間が重要だとラベル付けした単語・フレーズと、LLMの注意スコアが高かった上位kトークンとの一致度を測定

- サブワードトークンについては、元の単語にマッピングして計算

1. San Martinoのトークン重複指標

- 人間の注意単語とLLMの注意単語の重複を、Precision, Recall, F1スコアで算出

2. Krippendorffのアルファ

- 「観測された不一致」が「偶然による期待される不一致」と比べてどれだけ小さいかを測定 $([-1, 1])$

k = 5, 10, 20で実験

- LLMの上位注意単語と人間の注意単語の一致度が高いほど、LLMと人間のギャップは小さいと判断

RQ1

モデルの注意は人間の注意とどの程度一致してるか？

- 最良の手法であったBERT_Masking(摂動ベース)の結果を掲載

Method	BERT_masking											
	Top 5				Top 10				Top 20			
	P	R	F1	KA	P	R	F1	KA	P	R	F1	KA
Incoder	48.8%	40.6%	44.3%	0.25	40.8%	67.4%	50.8%	0.25	36.6%	90.3%	52.1%	0.12
CodeGen	51.8%	43.2%	47.1%	0.29	41.8%	68.9%	52.0%	0.27	36.8%	90.7%	52.4%	0.13
CodeParrot	51.5%	43.0%	46.9%	0.29	42.1%	69.2%	52.3%	0.28	36.6%	90.2%	52.0%	0.12
GPT-J-6B	49.9%	41.3%	45.2%	0.26	41.0%	67.4%	51.0%	0.26	36.6%	90.3%	52.1%	0.12
PolyCoder	49.3%	41.3%	45.0%	0.26	40.9%	67.7%	51.0%	0.26	36.6%	90.4%	52.1%	0.12
GPT-4	32.7%	27.6%	30.0%	0.04	36.7%	61.7%	46.0%	0.17	36.3%	89.4%	51.7%	0.11

結果

- San Martinoのトークン重複指標：全ての設定でF1スコアが0.6未満
 - Krippendorffのアルファ：全ての設定でKAが0.3未満
- 全てのモデルにおいて，人間とLLMの間には一貫した注意の不一致が見られた

RQ2

注意単語はコード生成モデルのエラーを説明できるか？

- CodeGen-2.7BとGPT-4が生成したエラーコード(計211件)の原因を, BERT_Maskingで算出された注意単語で説明できるかを分析

結果

- 57/211件(27%)が説明可能
- 失敗原因の例：重要な条件への注意の欠如(6件)

```
1 # Write a function that matches a string that has an ``a`` followed by zero or more ``b`` s.  
2 def function(string):  
3     return string.count('ab') >= 1
```

条件に当たるフレーズに注意されていない

その他の例

- 操作やデータオブジェクトの重要な説明語への注意の欠如：13件
- 操作説明への注意の欠如：6件
- データ型への注意の欠如：7件
- NL単語とコード要素の間の不正確なマッピング：25件

その他RQ

RQ3：異なる注意計算方法が注意の一致にどのような影響を与えるか

- 3種類のベース方法のうち、摂動ベースが最も人間の注意と一致した
- READING時よりもCODING時の方が人間の注意と一致した

RQ4：どの注意計算方法が最も好まれるか

- ユーザー調査で、プログラミング経験者に注意計算方法を評価
- 摂動ベースが最も好まれた
- 一方、注意単語の情報があってもLLMへの信頼や確信は欠如
→注意単語だけではLLMの透明性は不十分

まとめ

LLMの注意単語を元に，コード生成時の人間とLLMの考えのギャップを調査

人間とLLMでは，注意単語の一致度は低かった

- LLMは現状注意すべき単語を取りこぼしている傾向

算出されたLLMの注目単語は，エラーの原因の一部を説明できる

- 一方，説明できないものが多数派．注意単語だけではLLMの理解には不十分