

Towards Understanding the Characteristics of Code Generation Errors Made by Large Language Models

Zhijie Wang, Zijie Zhou, Da Song, Yuheng Huang, Shengmai
Chen, Lei Ma, Tianyi Zhang
ICSE2025

背景 | LLMによるコード生成

自然言語からコードの自動生成

LLMの進歩⇒大幅に改善

LLMの課題

- ・多くのタスクにおいて信頼性の高い正しいコードを生成できない

⇒LLMが失敗するケースについての理解が不足

- ・通常どのような種類のコード生成エラーを起こすのか
- ・異なるLLMが類似したエラーをおこすのか

概要 | エラーの二つの次元による分析

意味的特性

- ・ 条件の欠落
- ・ 誤った方向性
- ・ 不正確な条件

⇒ **正しいプログラムの生成におけるLLMの限界の理解**

構文的特性

- ・ 不正確なコードブロック
- ・ 不正確な関数引数

⇒ **コード構造を生成する現在のLLMの能力を評価**

研究方法 | RQ

RQ1

—異なるLLMはどのようなコード生成エラーを起こすか。

RQ2

—コード生成エラーを修正するにはどの程度の労力が必要か。

RQ3

—タスクの複雑さはLLMのコード生成にどのように影響するか。

RQ4

—部分的に失敗したコードは、完全に失敗したコードと異なる特性を示すか。

研究方法 | 対象LLM

6つの代表的なコード生成LLM

- CodeGen-16B
- InCoder-1.3B
- GPT-3.5
- GPT-4
- SantaCoder
- StarCoder

⇒ 幅広いモデルサイズと性能レベルをカバー

研究方法 | 不正確なコードの収集

HumanEvalベンチマーク

- ・各モデルに対し、再現性を確保 (temperature=0)
- ・テストケースの実行⇒テストケースを通過しても完全に正しくないコードを特定 (手動チェック)

⇒6モデル全体で合計557個の不正確なコード解を特定

研究方法 | 不正確なコードの分析

オープンコーディング

- ①557個の誤解からランダムに160個をサンプリング
- ②著者間でコードのデバッグと手動修正を行う
- ③エラーの場所と根本原因を記録・評価

⇒全体で約328人時を費やしてラベル付け

研究方法 | 修復労力の分析 (RQ2)

3つの指標

- ①Levenshtein距離：2つの文字列間の編集の手間
 - ②Jaccard類似度：2つの集合における構成要素の重複度
 - ③CodeBERTScore：意味的な近さを測る指標
- ⇒不正解コードと正解コードの間のテキスト的・意味的類似性を算出

結果 | RQ1 (コード生成エラーの特性)

意味的特性

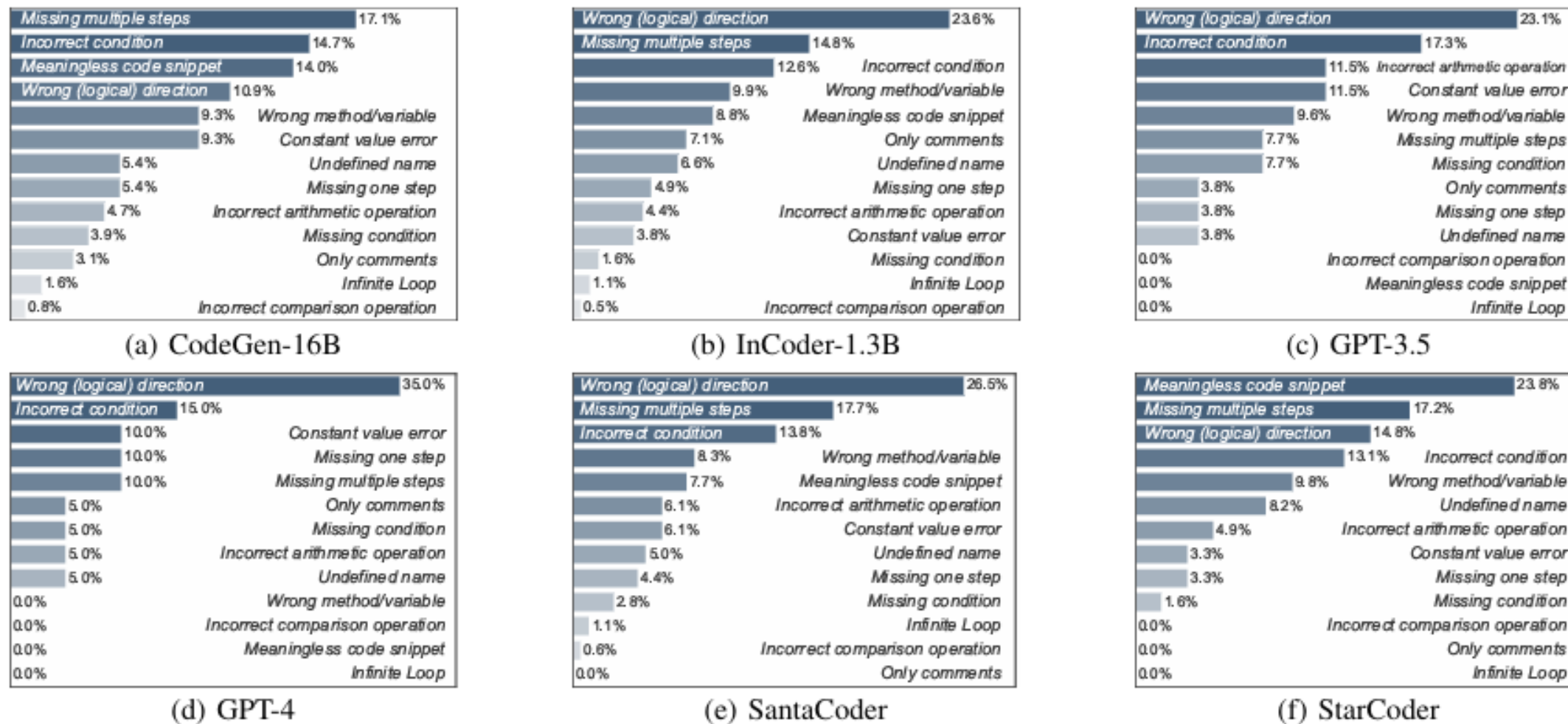


Fig. 1: Distribution of *semantic characteristics* of code generation errors made by six LLMs.

- ・最も共通するのは「誤った方向性」「不正確な条件」
⇒複雑なタスク要件の解釈や正しい論理条件の生成に苦戦

結果 | RQ1 (コード生成エラーの特性)

構文的特性

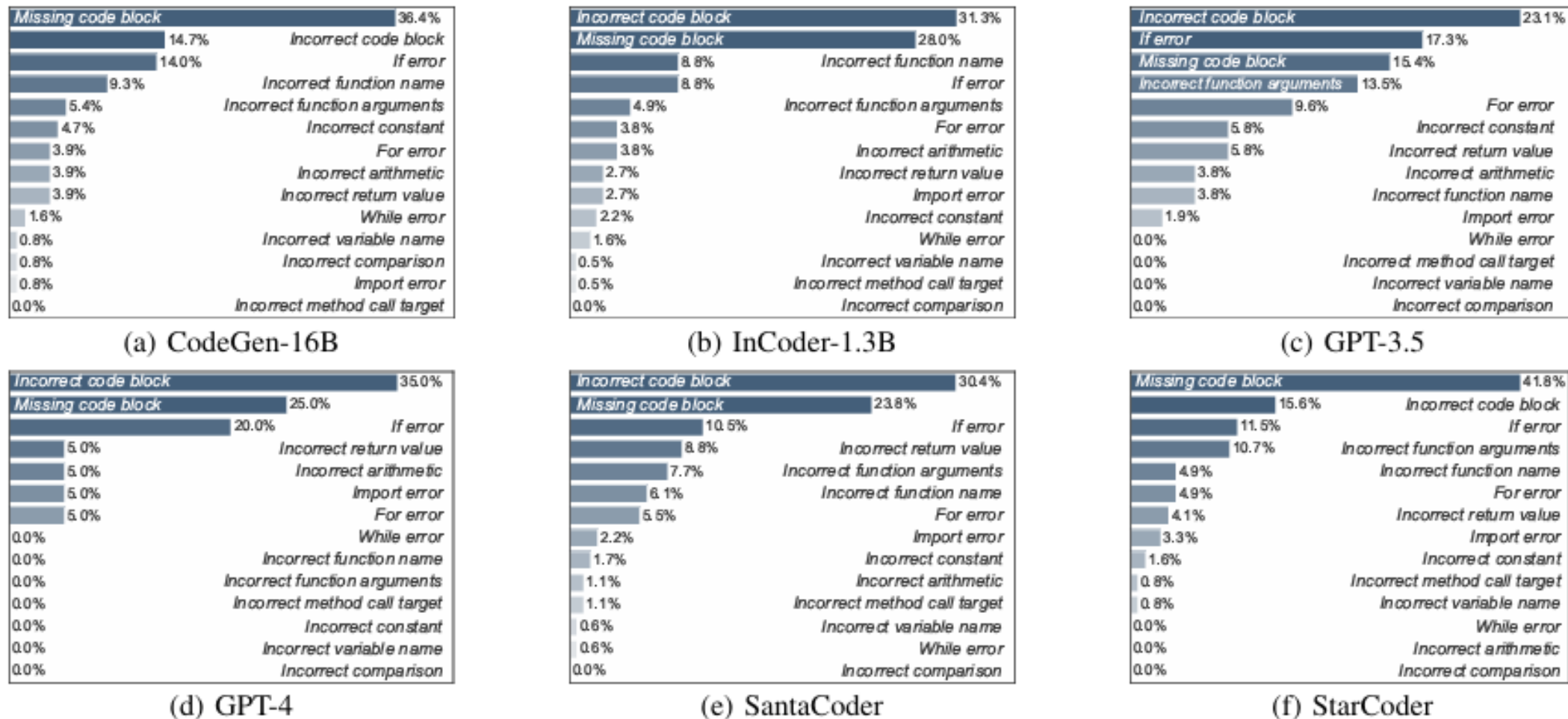


Fig. 2: Distribution of *syntactic characteristics* of code generation errors made by six LLMs.

- ・ 構文的特性の40%以上は「コードブロックの欠落・誤り」

結果 | RQ 2 (コード生成エラーの修復労力)

3つの指標

- Levenshtein距離

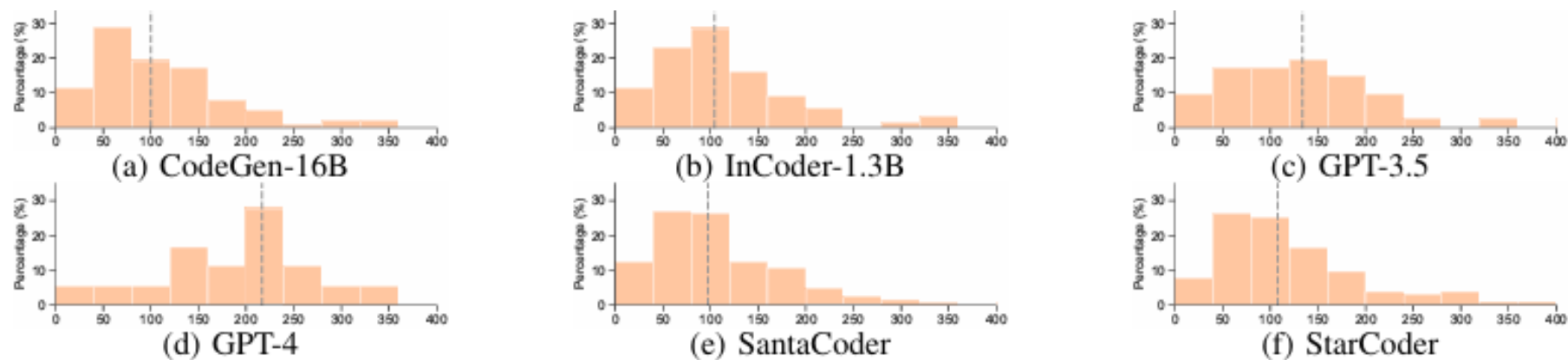


Fig. 4: Levenshtein distance between the incorrect code and correct code. The vertical dashed lines indicate the medians.

- CodeBERTScore：大部分のスコアが0.1未満

⇒生成されたコードが意味的にも正解から大きく乖離

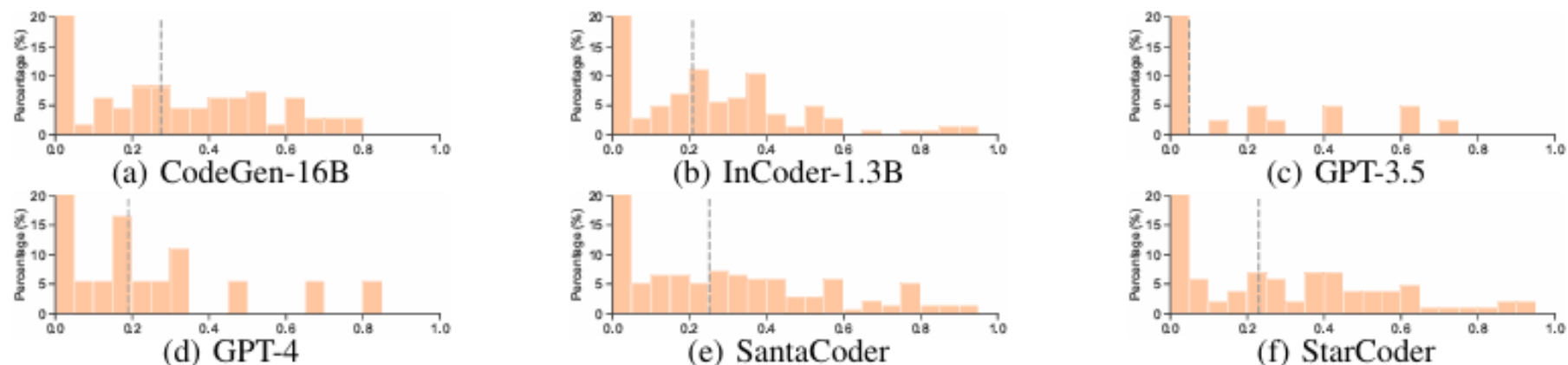


Fig. 5: CodeBERTScore between the incorrect code and correct code. The vertical dashed lines indicate the medians.

結果 | RQ 2 (コード生成エラーの修復労力)

3つの指標

- エラーの粒度分類：修復に必要なハンク数に応じて分類
 - 多くのエラーが「単一ハンク」か「複数ハンク」
- ⇒ 単一行の微修正ですむエラーは少ない

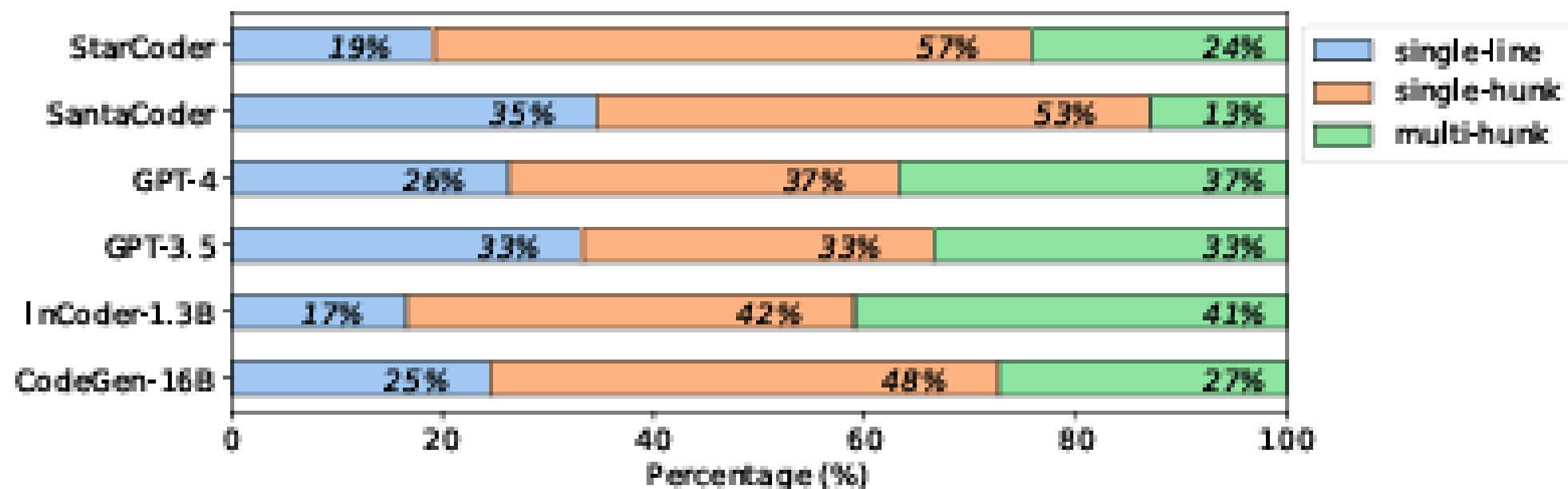


Fig. 6: Distribution of *single-line*, *single-hunk*, and *multi-hunk* faults in different LLMs

結果 | RQ 3 (タスクの複雑さの影響)

タスクの複雑さの指標

- ・ プロンプト長 (単語数)
- ・ 正解コードの行数 (LOC) /ASTノード数

Mann-Whitney U検定の結果

全てのモデルで成功タスクと失敗タスクの間に複雑さが影響

結果 | RQ 4 (テスト通過率の影響)

完全失敗した（全テストケース×）コードと部分失敗したコードの特性を比較

- ・ 部分失敗したコード：
「複数ステップの欠落」や「不正確な条件」多
- ・ 完全失敗したコード：
「無意味なコードスニペット」多

考察 | 議論と今後の研究

LLM生成コードのエラー修復

本論文の分類に基づく具体的なエラー種別をプロンプトに含める

⇒修正成功率が大幅に向上

LLM生成コードの正確性の推定

タスクの複雑さやプロンプトの特性から、モデルが対処できる限界やコードの正しさを事前予測・判定する技術が必要