

Beyond Functional Correctness: Investigating Coding Style Inconsistencies in Large Language Models

Yanlin Wang, Tianyue Jiang, Mingwei Liu*, Jiachi Chen, Zibin Zheng
School of Software Engineering, Sun Yat-sen University

Proceedings of the ACM on Software Engineering, Volume 1, Issue FSE2025
Article Pages ??? - ???

発表者：野口

背景：コード生成

与えられた要件に従ったコードを自動で生成

- Ex) CodeLlama, StarCoder, Codex

CodeLLMと呼ばれる

- コード生成の正確性向上に焦点
- コーディングスタイルについては十分な研究がされていない

コーディングスタイルに関連する未解決の課題

- コーディングスタイルの分類と定義の十分な詳細
- Code LLMと人間のスタイルの違いに関する研究
- 複数のCode LLM間のスタイルの比較研究

目的・リサーチクエスチョン

目的

- Code LLMが生成したコードと人間の開発者によるコードのコーディングスタイルの不整合を検証する

リサーチクエスチョン (RQ)

- RQ1：コーディングスタイルの不整合はどんな種類がある？
- RQ2：コーディングスタイルの不整合の分布は？
- RQ3：Code LLMと人間はどちらのコーディングスタイルが優れている？
- RQ4：プロンプトの工夫でCode LLMのコーディングスタイルは改善される？

実験設定

Code LLM

- CodeLlama-7B, StarCoder2-7B, DeepSeekCoder-1.3B, DeepSeekCoder-6.7B
- コード生成で高い性能を示し, 主流のOSS Code LLM

ベンチマーク

- Coder Eval
 - 実用的なコード生成タスクにおける性能を評価するためのベンチマーク
 - 関数シグネチャ
 - タスクの説明
 - 正解のコード
 - 生成コードの正しさを検証するテスト

これら4つを含む

RQ1：コーディングスタイルの不整合の特定

データ収集

- Coder Evalの230件のPythonコード生成タスクを対象
 - 4つのCode LLMに各タスクごとに10件のコードを生成

フィルタリング

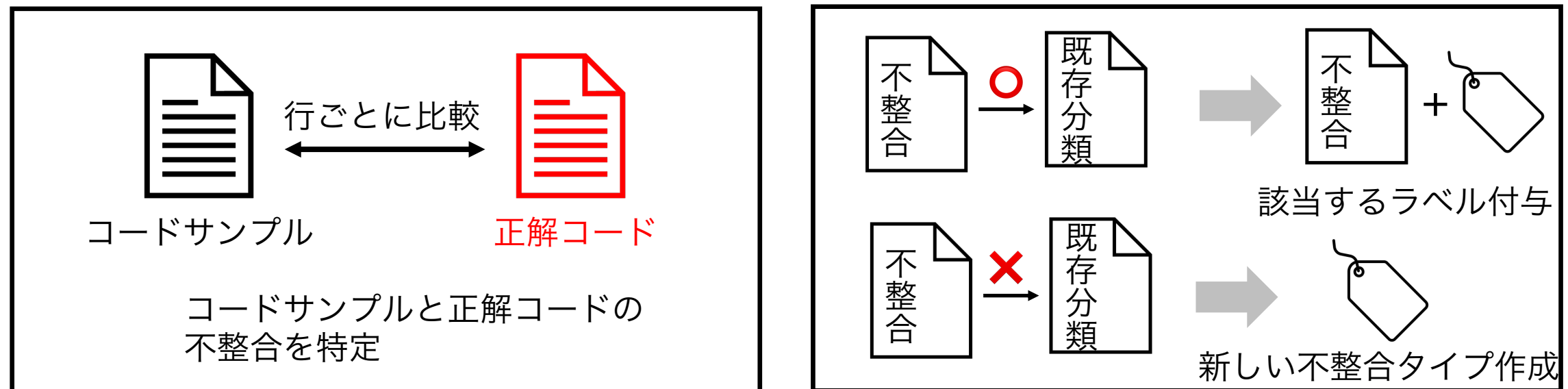
- テストに不合格になったもの
 - コーディングスタイルの不整合が見られないもの
 - テストに合格しても、タスクの要求と誤っているもの
 - 関数の実装に不要な冗長コードを含むもの
- 最終的に、820件のコードサンプルを収集

RQ1：コーディングスタイルの不整合の特定

オープンコーディング

- データからパターンを見出す手法
- [8]を初期分類とし，定義や分類をより精密化・拡張する

反復的コーディング

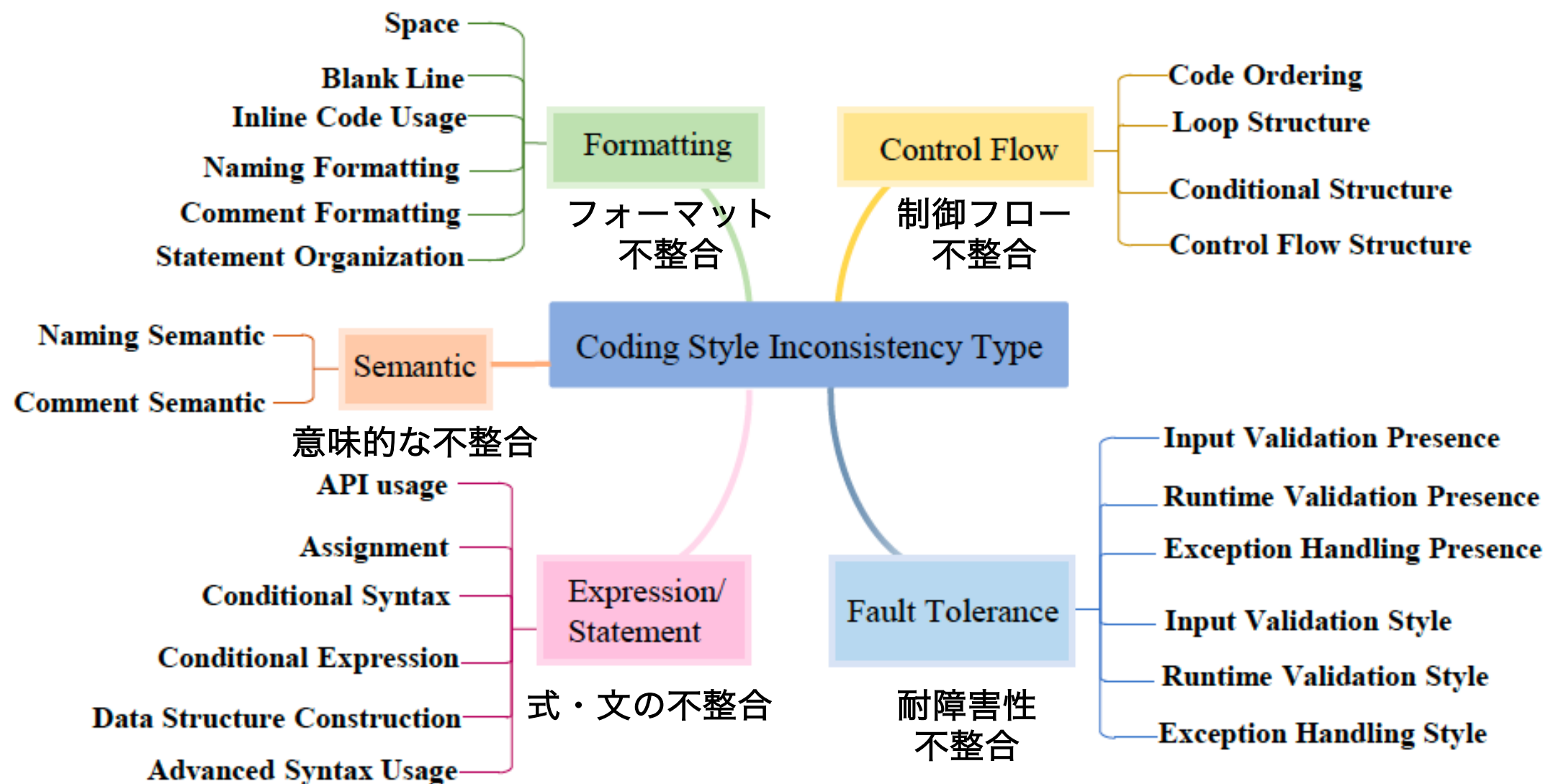


➤ 著者3名で実施し，議論を通じて決定

RQ1：コーディングスタイルの不整合の特定

オープンコーディングで24種類の不整合タイプが定義

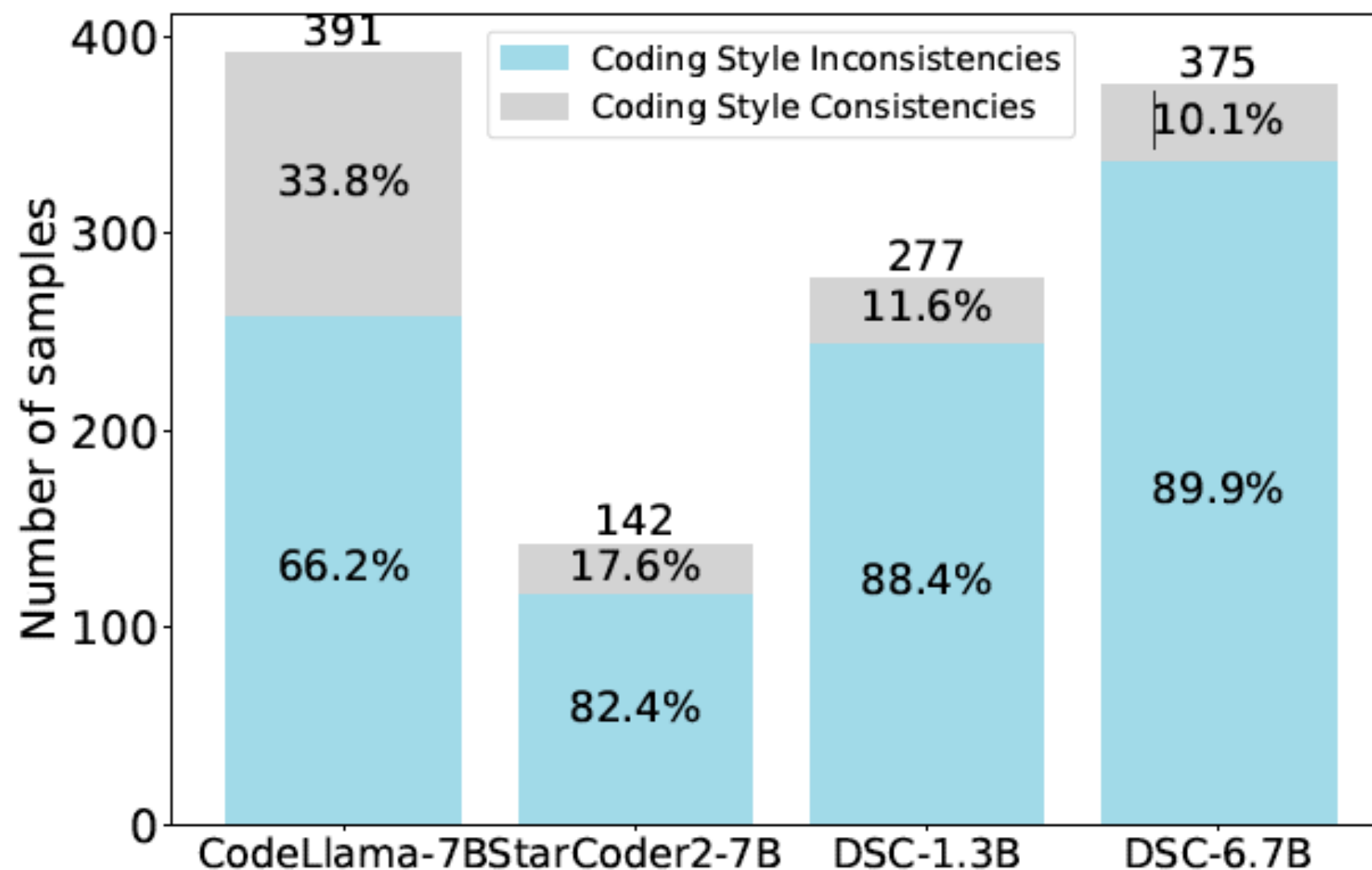
- 主要な焦点に基づいて，5つの側面に分類



RQ2：コーディングスタイル不整合の分析①

不整合なコーディングスタイルの割合

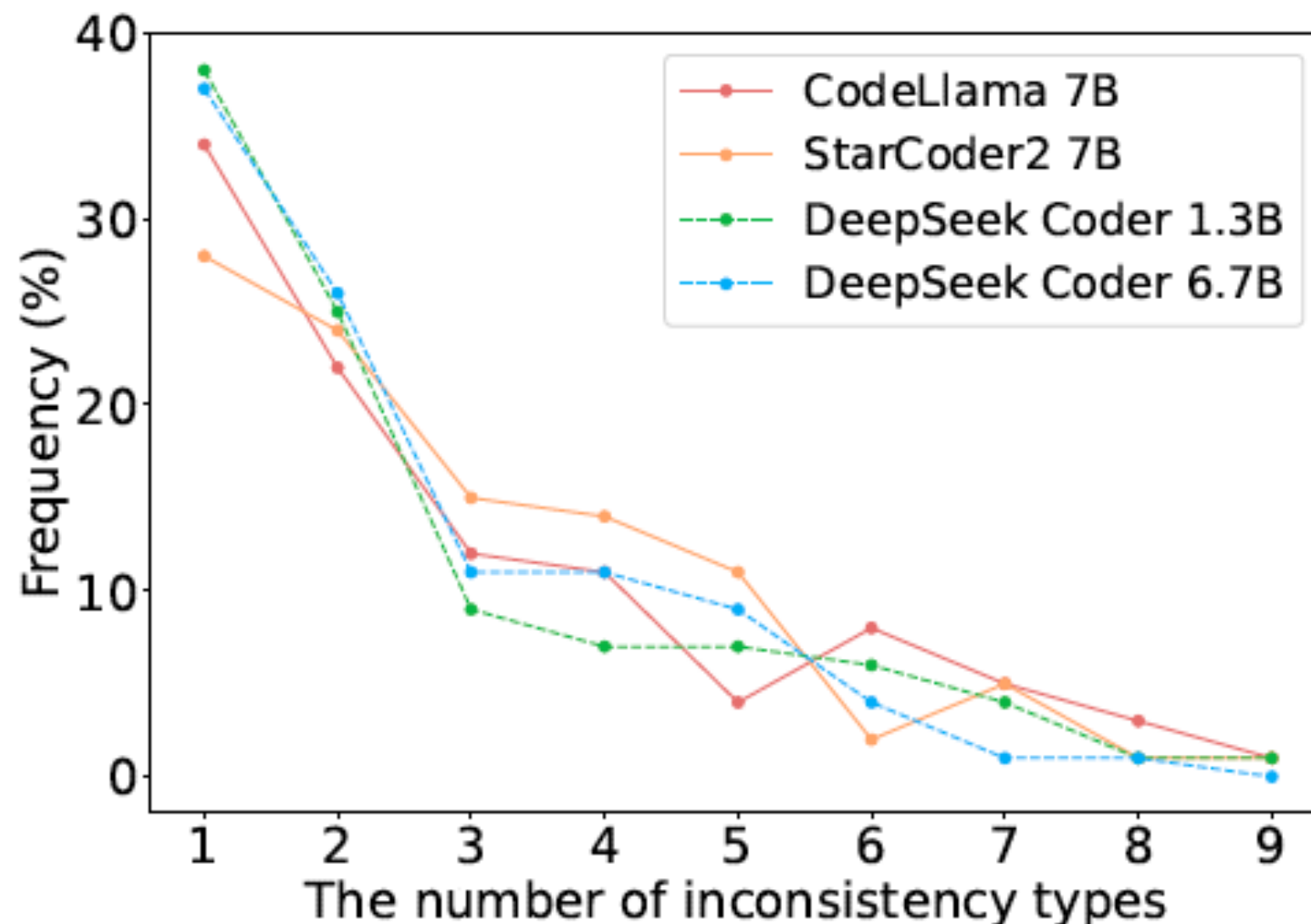
- 4つのCode LLMでの不整合なコーディングスタイルを持つファイル数を計測



RQ2：コーディングスタイル不整合の分析②

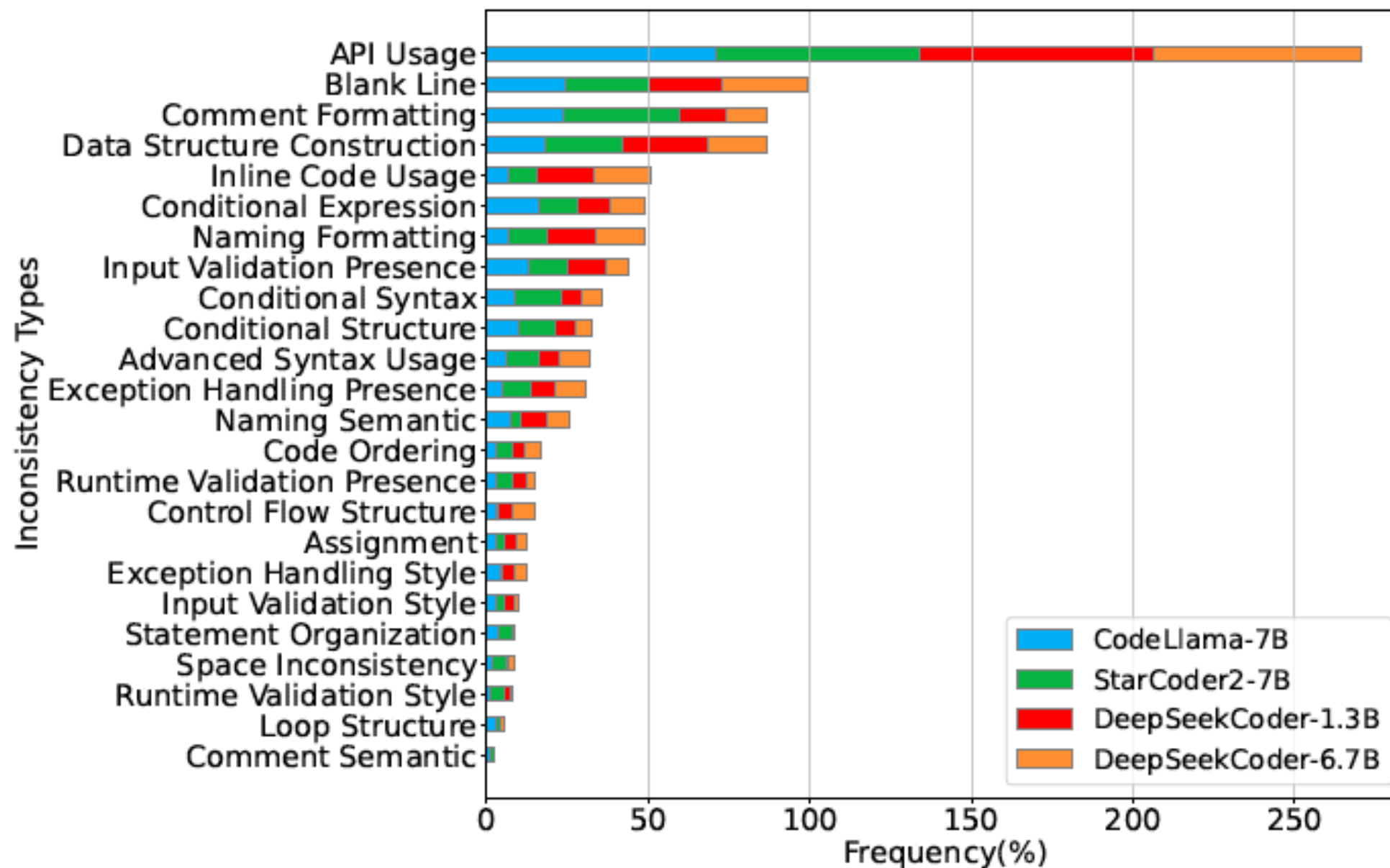
単一のコードサンプル内に存在する不整合の数

- 各モデルで1つのコードサンプルに現れる不整合タイプの数ごとに頻度を集計



RQ2：コーディングスタイル不整合の分析③

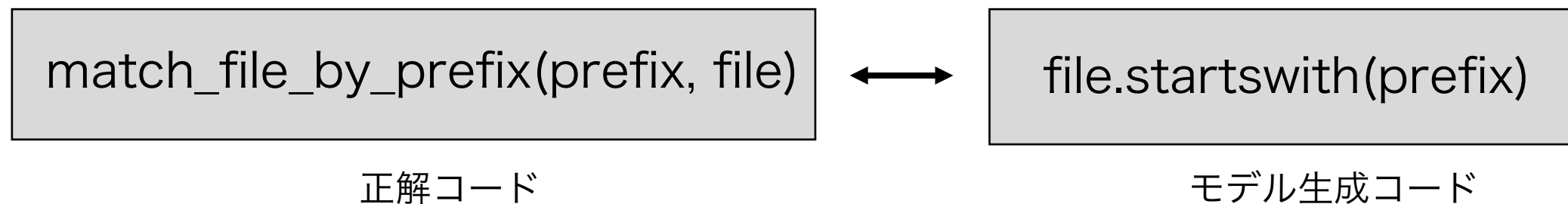
コーディングスタイルの不整合タイプの分布



RQ2：コーディングスタイル不整合の分析③

上位の不整合タイプに関して分析

- API使用の不整合



- 正解コード：リポジトリで定義された関数を呼び出している
- モデル生成コード：Pythonの組み込み関数を使って同様の機能を実装

- 空行とコメント整形の不整合

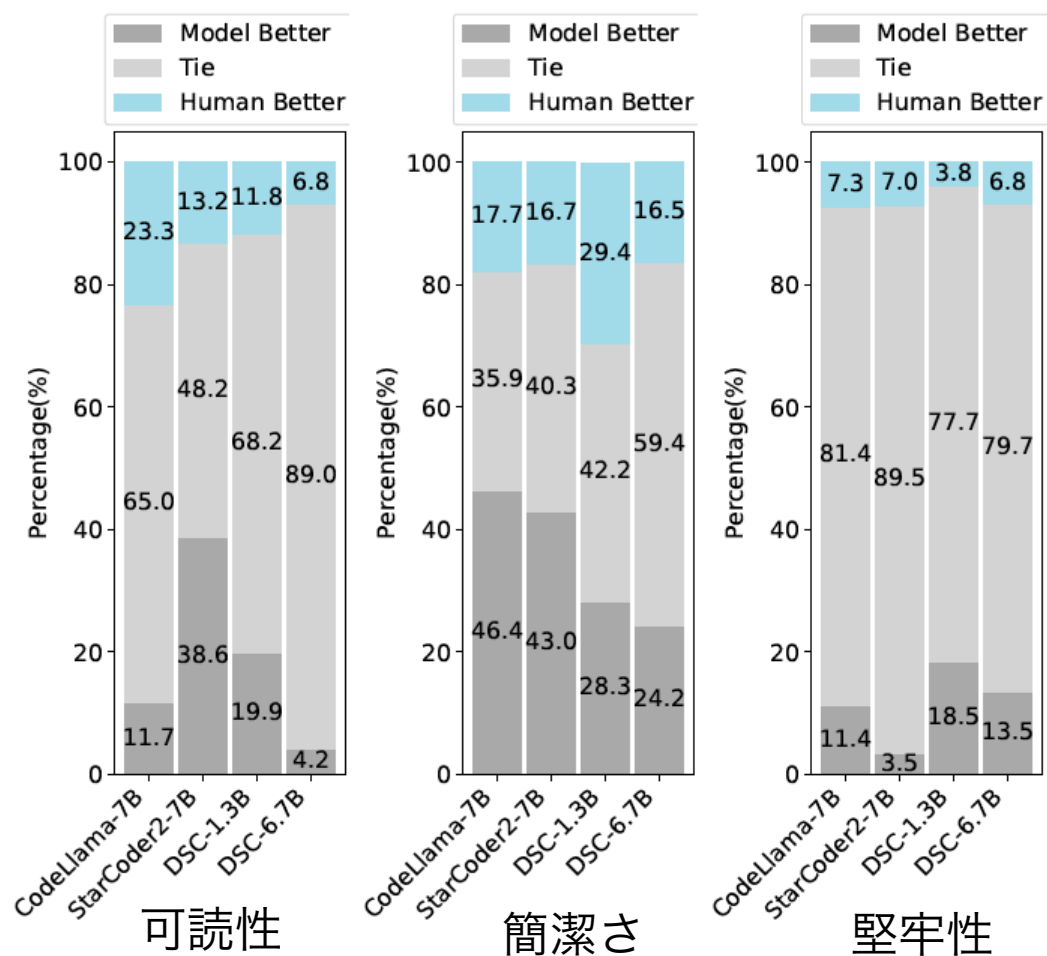
- 空行：モデルが生成したコードはコードブロックを分けるために空行を使用する傾向が少ない
- コメント整形：人間が書いたコードに比べて、モデル生成コードは意味のあるインラインコメントを生成する傾向が少ない

RQ3：コーディングスタイルの比較

- 3つの観点で評価

- 可読性：コードの読みやすさ・理解しやすさ
- 簡潔さ：コードのシンプルさ・不要な要素の度合い
- 堅牢性：コードが潜在的なエラーをどれだけ処理できるか

➤ これらをモデル優位, 同等, 人間優位の3段階で評価



可読性

- DeepSeekCoder-6.7Bが最も高い
- CodeLlama-7Bが最も低い

簡潔さ

- DeepSeekCoder-1.3Bがあまり簡潔でなかった
- 簡潔さと可読性にはトレード関係にある

堅牢性

- 4つのモデルが比較的高い値を出している
- モデルがより堅牢なスタイルを学習したことを示唆

RQ4：プロンプト工夫によるスタイルの改善

- 使用モデル：DeepSeekCoder-6.7B
- タスク：Coder Evalからランダムに20タスク選択

Prompt-head-concise: # Complete the function below by following the provided function signature and docstring. Ensure the implementation adheres to good coding style practices, including readability, simplicity, and robustness. [Function Signature] [Docstring]

原則のルールなし
スタイルガイドラインを関数シグネチャと
ドックストリングの前に配置

Prompt-head-detailed: # Complete the function below by following the provided function signature and docstring. Ensure the implementation adheres to good coding style practices, including readability, simplicity, and robustness. For example: [Principle 1][Principle 2] [Principle 3] [Function Signature] [Docstring]

原則のルールあり
スタイルガイドラインを関数シグネチャと
ドックストリングの前に配置

Prompt-end-concise: [Function Signature] [The context of initial docstring, Style Guidelines: Complete the function below by following the provided function signature and docstring. Ensure the implementation adheres to good coding style practices, including readability, simplicity, and robustness.]

原則のルールなし
スタイルガイドラインを関数シグネチャと
ドックストリングの後に配置

Prompt-end-detailed: [Function Signature] [The context of initial docstring, Style Guidelines: Complete the function below by following the provided function signature and docstring. Ensure the implementation adheres to good coding style practices, including readability, simplicity, and robustness. For example: [Principle 1][Principle 2] [Principle 3]]

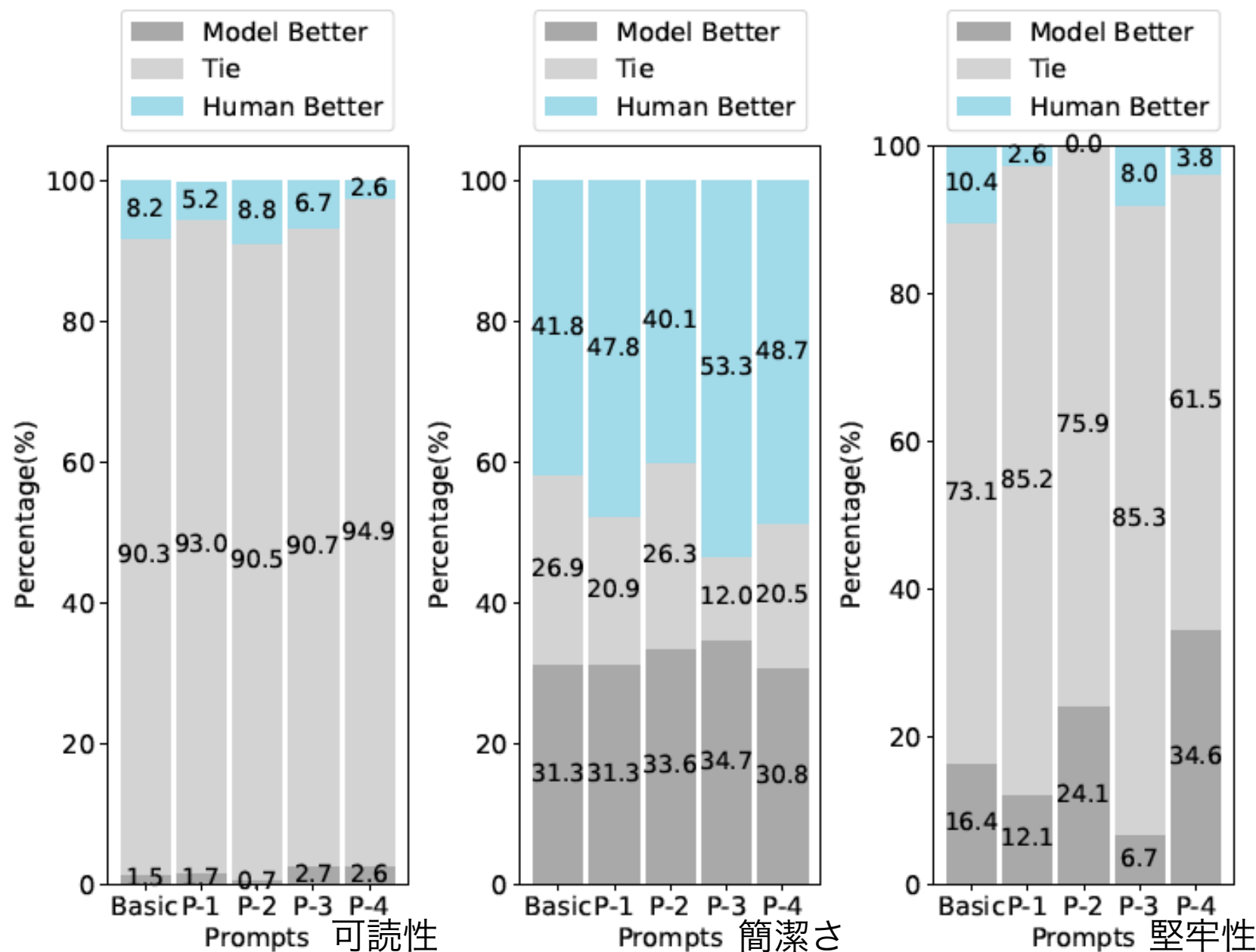
原則のルールあり
スタイルガイドラインを関数シグネチャと
ドックストリングの後に配置

Note:

- Principle 1: Include meaningful inline comments to enhance code readability
- Principle 2: Use if statements for input validation and try-except blocks for exception handling to improve code robustness
- Principle 3: Avoid excessive use of intermediate variables to keep the code simple
- “head” or “end”: The placement of the style guidelines
- “concise” or “detailed”: The level of detail of the style guidelines

読みやすさ：意味のあるインラインコメントを使う
簡潔さ：中間変数の使いすぎを避ける
堅牢性：入力のバリエーションやtry-exceptの利用

RQ4：プロンプト工夫によるスタイルの改善



- ・ 可読性が改善したものもあるが，簡潔さとはトレードオフの関係になってしまった
- ・ 堅牢性は全てのプロンプトで向上が見られた

総括

目的

- Code LLMが生成したコードと人間の開発者によるコードのコーディングスタイルの不整合を検証する

結果

- 24種類の不一致タイプを特定できた
- Code LLMと人間の間にコーディングスタイルの違いがあったが、Code LLM間ではあまり違いはなかった
- プロンプトエンジニアリングにより、スタイルの不整合をわずかに改善できた