

Calibration and Correctness of Language Models for Code

Claudio Spiess* UC Davis USA cvspiess@ucdavis.edu	David Gros* UC Davis USA dgros@ucdavis.edu	Kunal Suresh Pai UC Davis USA kunpai@ucdavis.edu	Michael Pradel Univ. of Stuttgart Germany michael@binaervarianz.de	Md Rafiqul Islam Rabin Univ. of Houston USA mrabin@central.uh.edu
Amin Alipour Univ. of Houston USA maalipou@central.uh.edu	Susmit Jha SRI USA susmit.jha@sri.com	Prem Devanbu UC Davis USA ptdevanbu@ucdavis.edu	Toufique Ahmed UC Davis USA tfahmed@ucdavis.edu	

2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)
✧arXiv Preprint

背景 | コード生成と意思決定

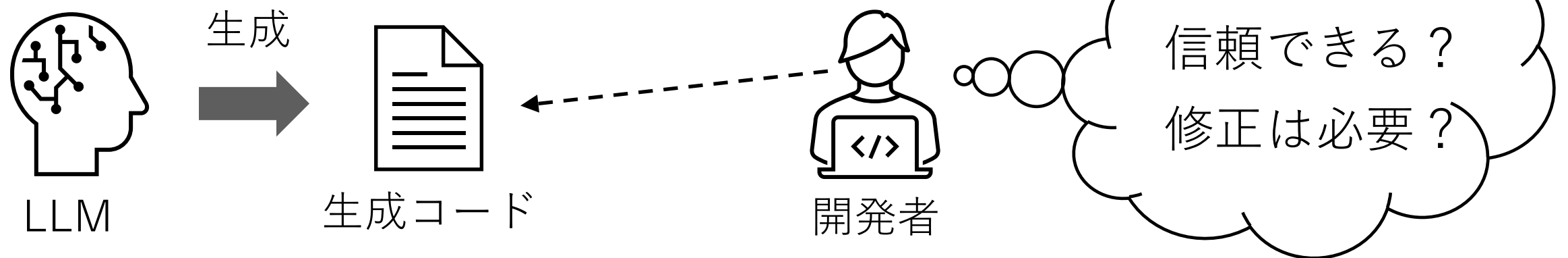
LLMがソフトウェアタスクに用いられる

- 例：関数生成やコード補完、バグ修正

正しくないコードが生成されることもある

- 品質，安全性に課題

そのため，LLM利用の際には**意思決定**が不可欠

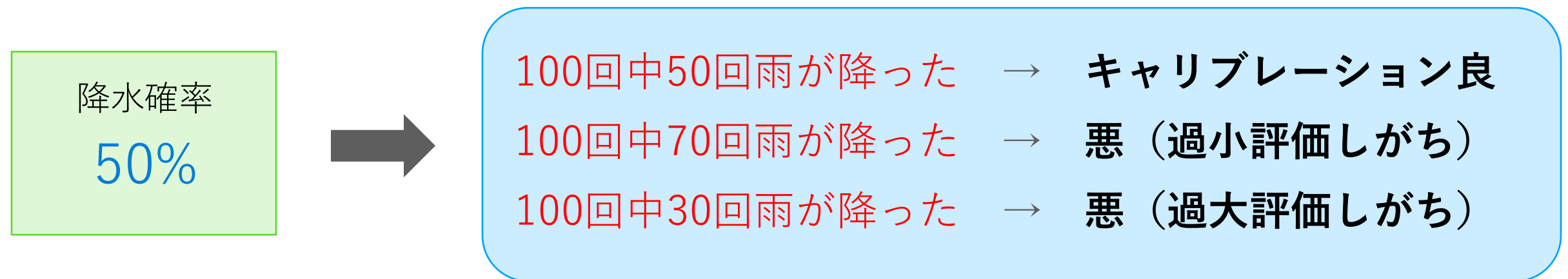


- 生成コードを利用すべきか，破棄すべきか
- どのくらいレビューをするべきか

背景 | キャリブレーション

出力の信頼度と実際の結果との整合性を示す指標

- 例：天気予報（雨予想）



- 意思決定や，予測の補正に用いられる
 - この天気予報を信じるべきか，信じないべきか？
 - 過大・過小評価をどのくらい補正すれば良いか？
- 利用の際に意思決定が重要なLLMにおいて，キャリブレーションを利用できないだろうか？

問題 | LLM生成の信頼度

LLMは出力の信頼度として、**トークン確率**を出力

生成コード例

“result = x + y”



トークン列：

["result", " =", " x", " +", " y"]

↑ ↓ ↑ ↓ ↑ ↓ ↑ ↓ ↑ ↓

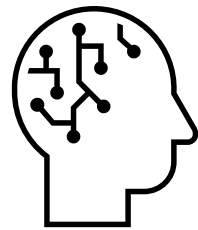
トークン確率：

[0.8 , 0.9, 0.7, 0.6, 0.8]

問題：**実際の正しさ**と一致していないことが多い

- キャリブレーションが悪い（過大評価が多い）
- 開発者が誤ったコードを信じて採用してしまうリスク

自信度は
80% !



LLM

生成



"result"

実際は50/100 “result”が正解



キャリブレーション悪（過大）

一方、LLMのキャリブレーションに関する研究は少ない

目的・アプローチ

コード生成におけるLLMのキャリブレーションの評価と改善

- LLMの信頼度として使えそうな指標は？
- キャリブレーションの改善方法の検討

評価：2分類・計5種類の信頼度指標を比較

- 内在的信頼度
 - モデルが出力時に自然と持っている確率に基づく
- 反省的信頼度
 - モデル自身に「このコードは正しいか？」と後から問う

改善：分類ごとに改善手法を提案

- Plattスケーリング（=ロジスティック回帰による補正）
- Few-shot promptingによる補正

信頼度指標① | 内在的信頼度

LLMが生成時に内部的に保持している確率情報

- 出力中のトークンごとの信頼度をもとに算出

① 平均トークン確率

- 各部分に対して、モデルが平均的にどのくらい確信を持っているか

ex)

トークン列： ["result", "=", "x", "+", "y"]

トークン確率： [0.8, 0.9, 0.7, 0.6, 0.8]

$$\text{信頼度①} = (0.8 + 0.9 + 0.7 + 0.6 + 0.8) \div 5 = \mathbf{0.76}$$

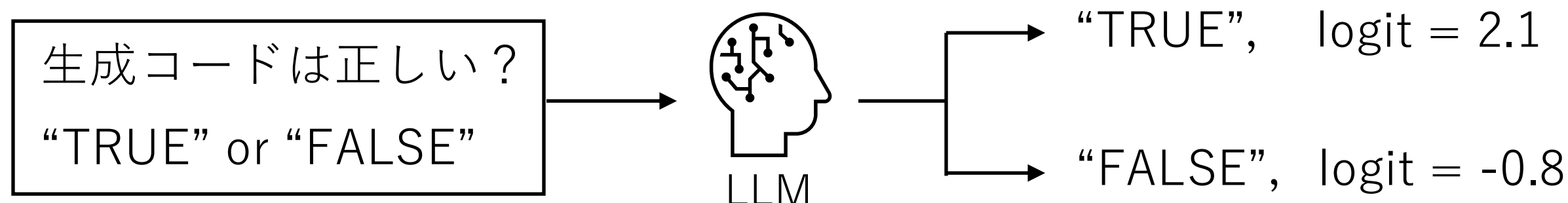
② トークン確率の積

- そのコード全体が生成される確率

ex) $\text{信頼度②} = 0.8 \times 0.9 \times 0.7 \times 0.6 \times 0.8 = \mathbf{0.24192}$

信頼度指標② | 反省的信頼度

出力後に、「このコードは正しいか？」をモデル自身に問う



③ TRUEのロジット値（確率変換前の生の値）

ex) 信頼度③ = $\text{sigmoid}(2.1) = 0.89$

④ TRUE/FALSEのsoftmax（TRUE/FALSEのみの確率）

ex) 信頼度④ = $e^{2.1} / (e^{2.1} + e^{-0.8}) = 0.95$

⑤ モデル自身が出力した確率（数値回答）

ex) 信頼度⑤ = 正しい確率は? →  85% ! = 0.85

信頼度指標② | 反省的信頼度

出力後に, 「このコードは正しいか？」をモデル自身に問う

生成コード
“TRUE”

要するに,

内在的信頼度：出力**時**の内部情報

反省的信頼度：出力**後**の自省情報
または出力そのもの

= 2.1

= -0.8

③ T

ex) 信

④ T

ex) 信頼度④ = $e^{2.1} / (e^{2.1} + e^{-0.8}) = 0.95$

⑤ モデル自身が出力した確率（数値回答）

ex) 信頼度⑤ = 正しい確率は？ →  85% ! = 0.85

実験設定①

3種類のコード生成タスクを対象

- 関数生成 (HumanEval, MBPP)
- コード補完 (DyPyBench)
- プログラム修正 (Defect4J, SSTUBS)

➤ それぞれの出力に対して信頼度を算出

出力の正確性は、テスト通過 or 完全一致 (0,1) で評価

- 関数生成についてはテスト通過で,
- コード補完についてはテスト通過or完全一致で,
- プログラム修正は完全一致で評価

ここで得た信頼度と正確性からキャリブレーションを算出

実験設定②

キャリブレーション算出方法

- ECE : 信頼度と正解率がどのくらいズレているか
 - 範囲 : 0 ~ 1, 小さいほどキャリブレーションが◎
 - Skill Score : ナイーブ予測(全体平均正解率が信頼度)との比較
 - 範囲 : $-\infty \sim 1$, 負の数なら×, 正の数ならキャリブレーションが◎
- 信頼度と実正解とのズレや, ベースラインとの比較を結果とする

使用モデル

- GPT-3.5
- Codex
- CodeGen2

結果① | 内在的信頼度

内在的信頼度は、いずれも不良

- GPT3.5での結果（最良モデル，一部抜粋）

信頼度指標	コード補完		関数生成		プログラム修正	
	ECE	SS	ECE	SS	ECE	SS
①平均トークン確率	0.46	-0.87	0.23	-0.18	0.73	-3.39
②トークン確率積	0.15	-0.03	0.63	-1.70	0.28	-0.63

※ECEは低いほどよく，SSは高い方がよい

- いずれもSSが負
 - キャリブレーションとしては役立たず
- 平均トークン確率は大幅に過大評価傾向

結果② | 反省的信頼度

反省的信頼度は、内在に比べると比較的マシ

- こちらもGPT3.5での結果

信頼度指標	コード補完		関数生成		プログラム修正	
	ECE	SS	ECE	SS	ECE	SS
③ TRUEロジック	0.16	-0.13	0.37	-0.47	0.04	+0.05
④ TRUE/FALSE	0.15	-0.15	0.19	+0.01	0.24	-0.30
⑤ 数値回答	0.42	-0.92	0.22	-0.24	0.60	-2.72

- ③と④は内在より改善
 - SSに正の値が出現, しかし依然として大差なし
- ⑤の数値回答は性能が悪い

結果② | 反省的信頼度

反省的信頼度は、内在に比べると比較的马シ

- こちらもGPT3.5での結果

- 内在的信頼度
- 反省的信頼度

いずれもキャリブレーションが悪い
→改善が必要

プログラム修正

SS

+0.05

-0.30

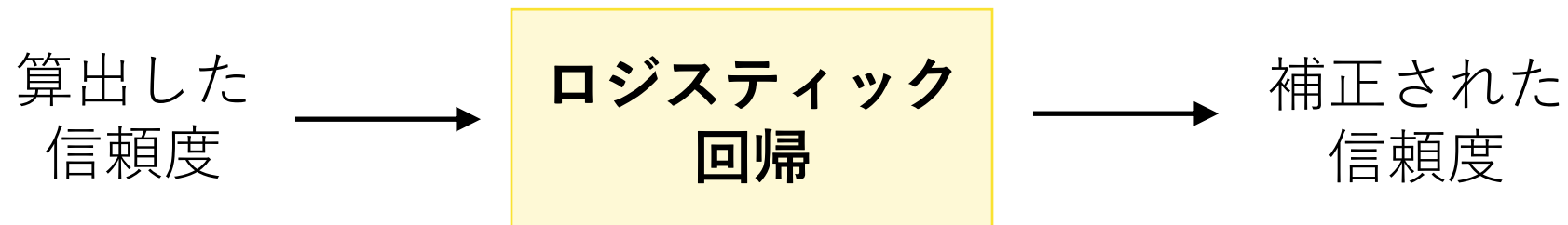
-2.72

- ③と④は内在より改善
 - SSに正の値が出現
- ⑤の数値回答は性能が悪い

改善手法 | 内在的信頼度

信頼度スコアをロジスティック回帰で補正

- Platt Scaling手法



- 一部の正解データで回帰パラメータを学習
- 残りのデータで補正効果を評価

補正イメージ

- 過大評価傾向→信頼度を下げる方向に補正
- 過小評価傾向→信頼度を上げる方向に補正

改善手法 | 反省的信頼度

Few-Shot promptingによる改善

- 正解例・不正解例を事前に示して学習させる

プロンプト例：

「以下の例を参考に、このコードは正しいですか？」

[類似コード1 → True/False]

[類似コード2 → True/False]

⋮

[類似コード5 → True/False]

[対象コード] → ?

類似したコード例を5つ提示

- データセット内からBM25で類似例を選択
- お手本を見せることでより正確な自己評価へ

改善結果 | 内在的信頼度

全体的に大幅改善

信頼度指標	コード補完		関数生成		プログラム修正	
	ECE	SS	ECE	SS	ECE	SS
①改善前	0.46	-0.87	0.23	-0.18	0.73	-3.39
①改善後	0.04	+0.08	0.20	-0.02	0.16	-0.03
②改善前	0.15	-0.03	0.63	-1.70	0.28	-0.63
②改善後	0.03	+0.07	0.09	+0.15	0.16	-0.05

- ECEが大幅改善，SSも正の値へ転換
- 全てのタスクで完全な改善をしたわけではない

改善結果 | 反省的信頼度

Few-shot + Platt Scalingが最も大きな改善

- タスクはコード補完のみ, 信頼度には④TRUE/FALSEを使用
- Zero-shot, 内在的補正の際に用いたPlatt Scalingとも比較

信頼度指標	コード補完	
	ECE	SS
④改善前	0.15	-0.15
④改善後(scaling)	0.22	0.00
④改善後(FS)	0.08	+0.10
④改善後(FS+scaling)	0.19	+0.15

Fes-ShotでSSが正へ転換

Scalingも併用でECE/SSともに改善

➤ 実用的な数値へ補正可能

議論

実用化への示唆

- 開発者の意思決定支援に貢献
 - 高信頼度→軽いレビューで採用
 - 中信頼度→慎重なレビュー後採用
 - 低信頼度→破棄または大幅修正

汎用性・一般化の課題

- 同じ手法でもタスクによってばらつきあり
- 補正には、タスク・ドメインごとにデータ収集・学習が必要
(学習時間・コストがかかる)

まとめ

LLMコード生成のキャリブレーションに問題あり

- 開発者の意思決定に影響

5種類の信頼度指標を評価

- 内在的よりも反省的の方がやや良好
- しかしいずれも校正不良

キャリブレーション改善手法を提案

- 内在的 : Platt Scaling
- 反省的 : Few-Shot

所感

LLMの活用に関する、新鮮な観点

- ハルシネーションとの付き合い方の一つ
- 生成AIの信頼性もAI自身が判断

直接的な貢献がわかりづらかった

- 回答の信頼性が高かかろうと、どのみち生成物の中身は確認するんだから、それで普通に判断すれば良いのでは（補正にはコストもかかるし．．．）
- 将来的には、安全性・確実性が求められる分野での活用や、AIのメタ認知的な要素の増強に役立つのかも？（人間の確認の必要性の低下とか？）