Code LLM 事前学習時の評価データ混入への対策

金山 博　　大湖 卓也　　村岡 雅康　　吉田 一星
日本アイ・ビー・エム株式会社 東京基礎研究所
{hkana, ohkot, mmuraoka, issei}@jp.ibm.com

概要

本論文は、プログラムのソースコード用の言語モデルの性能を適切に評価する際に重要となる、モデルの事前学習における評価データの混入の問題について論じる。まず、コード用のモデル向けのベンチマークを網羅的に調査し、一部の評価データの内容が本質的に GitHub 由来の学習データ重複しうるという実態を示す。また、混入除去の妥当な方策として、評価データと学習データの間の文字列の一致の検出に加えて、由来となるレポジトリの情報を用いる手法を提案する。

1　はじめに

大規模言語モデル（LLM）を開発するにあたって、その適切かつ公正な性能評価のために、ベンチマークのテストデータと同じものが学習データに含まれること（混入）を防ぐ必要があり、そのための対策について議論されている [11, 25]。

一般の言語モデルに加えて、ソースコードの生成・補完・修正・解釈などに特化したモデル（以降では Code LLM と呼ぶ）の開発が盛んになっている [21, 6]。Code LLM の評価セットとしてさまざまなものが提案されているが、そのデータ混入についての議論は限定的である [24]。Code LLM の学習や評価においては、以下のような特徴がある。

- コード生成、翻訳、計算問題、コードの説明など、評価データの種類が多岐にわたる。
- コードは自然言語よりも構文の厳格さが求められる。自然言語の質問応答のように、評価データと学習データの類似性を意味的な近さで測ることは適切でない。
- 評価データの正解例の中には、“`return x+y`”のように、非常に短く一般的なコードがある。こうした文字列を含むものをすべて混入とみなすと、無害な学習データを失う可能性がある。

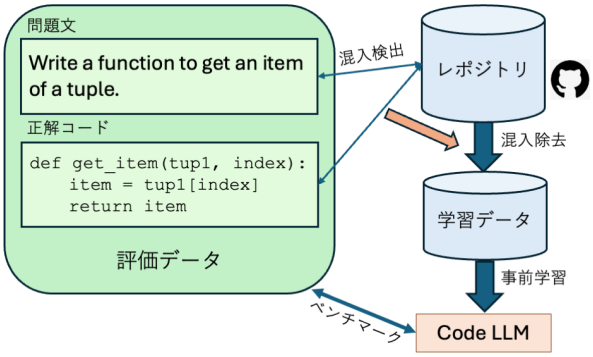

図 1　Code LLM のデータ混入除去の概念図。評価データの例は MBPP[22] より。

- 一般の LLM の学習データが web ページなど雑多なものであるのに対し、コードの学習データは GitHub のレポジトリのように比較的均質であり、レポジトリ名、プログラミング言語などのメタデータを活用しやすい。

本論文では、図 1 に示すような Code LLM におけるデータ混入の問題に取り組んだ結果として、評価に用いたデータの調査と、学習データとの一致の実態、混入の除去の方法について述べる。

2　関連研究

Riddell らは Code LLM の学習データと評価データの一致について調査を行った [24]。ソースコード生成の標準的な評価データである HumanEval[3] と MBPP[22] を、StarcoderData[15] などの事前学習データと比較し、それらの共通部分を検出した。その際に、正解となるソースコードの表層上の一致を見るだけでなく、Dolos[20] を用いて抽象構文木に変換して比較し、学習データとの一致度が高いテストケースほどスコアが高くなる傾向から、混入の影響を擬似的に示した。しかし、表 1 のように、極めて短い正解のコードや、構造が一致しただけで内容は異なるコードも例示されており、一致したものを一律に混入とみなして除去することには疑問が残る。

表 1 Riddell らの手法により、評価データの正解（左）と学習データ（右）が完全一致とみなされていた例。

def closest_num(N):
  return (N - 1)
	def percentage(x):
	  return (x - 1)

abs(a % 10) * abs(b % 10)	abs(fa - f0) < abs(fb - f0)

いくつかの Code LLM は、ベンチマークの結果とともに学習時の混入除去の状況も公開している。StarCoder[15] では、HumanEval や MBPP など主要な評価データと一致するデータを含むファイルを学習データから除外しており、Python のコードのうち 558 のファイルを除去したと報告している。次のバージョン [19] では、さらに空白除去などの正規化を施して除去するファイルを増やしている。

OpenCoder[10] では、教師ありファインチューニング（SFT）の際に、10-gram が一致するものを除去している。実際に除去されたファイルの数や、事前学習データについては記述されていない。

3 Code LLM の学習と評価

ここでは、Code LLM の事前学習と評価に使われるデータや関連する処理について解説する。

3.1 学習データ

事前学習に用いられるデータとして、StarCoderData[1)]、github-code-clean[2)] などが知られており、いずれも GitHub のレポジトリから収集したコードやそれに付随するデータが格納されている。事前学習に使う際には、一般的に、重複除去、ライセンスに応じたフィルタリング、プログラミング言語ごとの重みづけといった前処理が行われる。評価データの混入除去もその一つである。

通常、ソースコードは元のファイルの単位で格納されており、レポジトリ名、ファイル名、プログラム言語[3)] などのメタデータが付与されている。

3.2 評価データ

Code LLM のベンチマークに使われる主なデータセットを表 2 に列挙する。HumanEval、MBPP などコード生成の評価データは、自然言語で与えられた課題 (prompt) と、正解となるコード (canonical solution) から構成される。予めプログラムの断片が与えられていて、その続きを補完するタスク設定もある。その場合、正解となるコードは非常に短いコードの断片となりうる。一方で、非常に長いコードを対象とするベンチマークもあり、LCC (Long Code Competition)[8] の各データは 2 千文字から 10 万文字の間に分布している。

HumanEval などは人手で構築された評価データであるのに対し、LCC などは既存のレポジトリのコードから題材が抽出されている。RepoBench[18]、RepoQA[17] は、レポジトリ内の複数のファイルの情報をもとにタスクを解く設定になっており、いずれも既存の Git レポジトリがテストデータとして使われている。

MBPP、CanItEdit、GSM8k[4]、DS1000[13] など、対象のプログラミング言語を Python に限定しているベンチマークが多いが、HumanEval は 6 言語[4)]、CodeNet[23] は 5 言語、CrossCodeEval[5] は 4 言語、LCC は 3 言語を扱っている。

4 データ混入の検出

4.1 混入の検出における課題

今回は表 2 に列挙した評価データについて、事前学習データへの混入を防ぐことを目的とする。その際に課題となるのが以下の点である。

検出漏れ 評価データと学習データの内容に重複があり、正当な評価に支障をきたす可能性があるものの、両者のフォーマットや表現の違いにより検出ができずに、学習データに残ってしまう場合。

過検出 実際には無関係なコードの断片が重複とみなされたり、極めて短く一般的なコードを含むために多くのファイルが混入とみなされてしまう場合。著しい場合には学習データの量が減ったり、データの偏りが生じたりと、モデルの品質に影響を及ぼしうる。

計算量 事前学習のデータ量は膨大であり、評価データの数が増えると全てのペアの比較のための計算量が大きくなる。

4.2 混入検出の手法

GitHub から取得した 3,736,205,248 ファイル（約 21TB）の学習データ[5)] と、表 2 にある評価データ（テストに用いる部分）を比較した。評価データご

1) https://huggingface.co/datasets/bigcode/starcoderdata
2) https://huggingface.co/datasets/codeparrot/github-code-clean
3) 通常はファイルの拡張子で判定される [12]。
4) C++、Go、Java、JavaScript、Python、Rust。
5) プログラミング言語の比率を付録の表 6 に示す。

表 2 Code LLM のベンチマークに使われる主な評価データ。データ数はテストデータの事例の数。

データセット	評価タスク	HuggingFace 上でのデータセット名	データ数
MBPP [22]	コード生成	google-research-datasets/mbpp	600
HumanEval(Pack) [3]	コード生成その他	bigcode/humanevalpack	164 × 6
GSM8k [4]	数学・推論	openai/gsm8k	1319
DS1000 [13]	データサイエンスのコード生成	xlangai/DS-1000	1000
RepoBench [18]	レポジトリ単位のコード生成	tianyang/repobench_python_v1.1	23561
RepoQA [17]	レポジトリ単位のコード理解	evalplus/repoqa(git)	50
CanItEdit [2]	コード修正	nuprl/CanItEdit	105
MATH [9]	数学	https://people.eecs.berkeley.edu/ hendrycks/MATH.tar	5000
OCW [14]	数学	zhangirazerbayev/ocwcourses	272
BFCL [26]	機能呼び出し	gorilla-llm/Berkeley-Function-Calling-Leaderboard	5227
CodeNet [23]	コード翻訳	iidai/codenet	200 × 5
CruxEval [7]	コード推論	xcruxeval-org/cruxeval	800
LCC [8]	長いコード	microsoft/LCC_{csharp,python,java}	10000 × 3
CrossCodeEval [5]	複数ファイルからの生成	Vincentvmt/CrossCodeEval	9923
CodeContests [16]	キー獲得	deepmind/code_contests	165
SAT [1]	標準試験問題	mcaleste/sat_multiple_choice_math_may_23	32
BigCodeBench [27]	コード補完・生成	bigcode/bigcodebench	1140

表 3 部分文字列およびレポジトリの検査によって検出された混入の数と、評価データの異なり数。

検出方法	評価データ	頻度	異なり数
文字列	MATH	9,245	2,096
	CrossCodeEval	3,550	3,409
	BFCL	2,789	25
	HumanEval	2,233	328
	DS1000	1,496	35
	LCC	1,492	1,115
	MBPP	258	62
	CodeNet	75	31
	GSM8k	30	21
	BigCodeBench	5	2
	CruxEval	4	2
レポジトリ	RepoBench	36,376	951
	RepoQA	6,951	29

との検出対象とするフィールドは付録の表 7 に示す。学習データの各ファイルが、評価データの文字列を含む場合に、その ID をアノテーションとして付与し、後に学習データから除去できるようにする。その際に、空白や改行の除去、大文字→小文字の変換の正規化を行った。また、検出漏れや過検出を防ぐために、以下のような処理を行った。

短い文字列の無視 HumanEval や DS1000 の正解のコードや BFCL[26] の指示の中には、非常に短く、学習データに含まれていてもほぼ混入とはみなせないものが含まれており、これは Riddell らの調査 [24] でも問題となっていた。事前に学習データの一部との文字列マッチングをして多くの一致が見られた評価データのうち、短く一般的な文字列は混入検出の対象から除外することにした。例外と判断した文字列は補足の表 8 に列挙する。

コメントの除去 HumanEval の指示文 (prompt) の中には途中までのコードがコメントアウトされた状態で含まれている。混入を調べる際には、コメントを除外した後にマッチングさせた。

レポジトリ単位の検出 既存のレポジトリの複数のファイルを対象としており、かつレポジトリが明示されているデータセット（RepoBench、RepoQA）[6] については、内容の文字列マッチングではなく、同一のレポジトリを由来とするデータを混入とみなすこととした。

学習データのファイル数は非常に多いため、マッチングの計算量を減らすために次の工夫を行った。

プログラミング言語の限定 評価データが扱う言語と一致しない学習データは検査対象としない。これにより、87.3% のマッチングの試行が削減できた。

学習データのクリーニング 学習データ側にはあらかじめ重複除去やその他のフィルタリングを適用したのちに混入の検出を行った。

重複 評価データの中で重複している文字列がある場合には、該当する複数の ID を出力できるようにして、文字列マッチングの回数を減らした。これにより、7.6% の試行が削減できた。

6） CrossCodeEval も同様に複数のファイルを扱うベンチマークであるが、評価データ側にレポジトリ名の情報が無いため、通常の文字列マッチングを行った。

表 4 多くの混入が検出されたレポジトリ。混入の頻度と、学習データ内の当該レポジトリ由来のファイル数を示す。

	レポジトリ名	混入件数	ファイル数	混入データ
(a)	STEAM-Center-NYC/ 10030-Implementing-Artificial-Intelligence-in-Arithmetic-Classes	2,087	12,485	MATH
(b)	esslushy/ChatGPT-Python-Programs	370	781	HumanEval
(c)	openai/code-align-evals-data	333	403	HumanEval
(d)	roozbehid/WasmWinforms	221	18,489	LCC-csharp
(e)	facebookresearch/miniF2F	142	1,182	MATH
(f)	CodedLadiesInnovateTech/python-challenge-solutions	51	2,144	MBPP

表 5 RepoBench の評価。学習データ中に含まれるテストデータと同一のレポジトリ由来のファイルの数ごとの、EM (Exact Match) と ES(Edit Similarity) のスコアを示す。

学習データ件数	EM	ES
10〜	7.33	48.1
1〜9	6.83	45.1
0	6.47	43.8

4.3 検出の結果

上記の方法により、表 2 の評価データに対して、学習データと比較した結果をした。表 3 は評価データごとの検出された学習データの数と、評価データ側の異なり数を示す。11 種の評価データについて、正規化された文字列の一致が観測され、これは学習データ全体の 0.00056% であり、それらを除去することにより失われる知識量はごく僅かである。評価データのうち、CanItEdit、OCW、CodeContests、SAT については一致は検出されなかった。一方で、LCC の長い文字列でも多くの一致が見られており、その長さから偶然の一致ではないことは明らかである。

表 3 の末尾の RepoBench と RepoQA は、レポジトリが一致することにより検出したものである。内容の一致は問わないので、評価への影響が無いデータもあるだろうが、高々全体の 0.0012% であり、安全のためにすべてを除去しても事前学習への影響は小さいと考えられる。

文字列の一致が 50 件以上見つかったレポジトリを表 4 に示す。これらはいずれも単一の評価データの複数の問題や答えを含んでいた。(d) は「評価データの作成元となったレポジトリ」であり、それ以外は「既存の評価データを解くためのレポジトリ」であることがわかった。表 4 にある 6 つのレポジトリを由来とするファイルは全体の 0.00095% であり、レポジトリの性質上、学習データからすべて除去するのが妥当といえる。

5 データ混入の影響

データ混入がある状態と無い状態の学習データを作成し、モデルの事前学習を行うことは、直接的にデータ混入の影響を測る手段ではあるが、多くのモデルの事前学習を行ったりその評価をしたりすることにはコストがかかる。そこで、既存のベンチマークのスコアと、学習データとの一致の度合いの関連を見ることによって、データ混入の影響を推し量る。HumanEval、MBPP の内容の一致による影響は Riddell らにより既に示されている [24] ため、ここではレポジトリ単位の調査結果を示す。

レポジトリ単位の混入除去を意識しないデータ事前学習した 1.4B パラメータの Code LLM で RepoBench[18] のテストケースを実行した。単一ないし複数のファイルのコードを見て次の行を予測するタスクであり、データには由来となるレポジトリの情報が含まれている。そこで、学習データの中にテストケースのレポジトリ由来のファイルが含まれている数に応じてテストケースを分割してスコアを算出した結果が表 5 である。学習データの中に直接の文字列の混入が無くても、同一のレポジトリのデータが存在する場合に精度が高くなる傾向が見られた。レポジトリ単位で学習データを除去することによってベンチマークへの影響は防止できる。

6 まとめ

本論文では、Code LLM のベンチマークに使われるデータについて網羅的に、学習データとの一致を調査した。その結果、一部のレポジトリにおいて公正な評価に影響を及ぼしうる混入が見つかった。コードの内容（表層や論理構造）を考慮したマッチングに加えて、レポジトリの情報を考慮することがより本質的であり、評価の妨げとなる混入を簡便かつ安全に防止できることを示唆した。

参考文献

[1] Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Albert Q Jiang, Jia Deng, Stella Biderman, and Sean Welleck. Llemma: An open language model for mathematics. *arXiv preprint arXiv:2310.10631*, 2023.

[2] Federico Cassano, Luisa Li, Akul Sethi, Noah Shinn, Abby Brennan-Jones, Jacob Ginesin, Edward Berman, George Chakhnashvili, Anton Lozhkov, Carolyn Jane Anderson, et al. Can it edit? evaluating the ability of large language models to follow code editing instructions. *arXiv preprint arXiv:2312.12450*, 2023.

[3] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé, Jared Kaplan, Harrison Edwards, Yura Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, David W. Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William H. Guss, Alex Nichol, Igor Babuschkin, Suchir Balaji, Shantanu Jain, Andrew Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew M. Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *ArXiv*, Vol. abs/2107.03374, , 2021.

[4] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[5] Yangruibo Ding, Zijian Wang, Wasi Ahmad, Hantian Ding, Ming Tan, Nihal Jain, Murali Krishna Ramanathan, Ramesh Nallapati, Parminder Bhatia, Dan Roth, et al. CrossCodeEval: A diverse and multilingual benchmark for cross-file code completion. *Advances in Neural Information Processing Systems*, Vol. 36, , 2024.

[6] Shahriar Golchin and Mihai Surdeanu. Time travel in LLMs: Tracing data contamination in large language models. In *The Twelfth International Conference on Learning Representations*, 2024.

[7] Alex Gu, Baptiste Roziere, Hugh James Leather, Armando Solar-Lezama, Gabriel Synnaeve, and Sida Wang. CRUXEval: A benchmark for code reasoning, understanding and execution. In *ICLR 2024 Workshop on Navigating and Addressing Data Problems for Foundation Models*.

[8] Daya Guo, Canwen Xu, Nan Duan, Jian Yin, and Julian McAuley. Longcoder: A long-range pre-trained language model for code completion. In *International Conference on Machine Learning*, pp. 12098–12107. PMLR, 2023.

[9] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

[10] Siming Huang, Tianhao Cheng, Jason Klein Liu, Jiaran Hao, Liuyihan Song, Yang Xu, J Yang, JH Liu, Chenchen Zhang, Linzheng Chai, Ruifeng Yuan, Zhaoxiang Zhang, Jie Fu, Qian Liu, Ge Zhang, Zili Wang, Yuan Qi, Yinghui Xu, and Wei Chu. OpenCoder: The open cookbook for top-tier code large language models. *ArXiv*, Vol. abs/2411.04905, , 2024.

[11] Alon Jacovi, Avi Caciularu, Omer Goldman, and Yoav Goldberg. Stop uploading test data in plain text: Practical strategies for mitigating data contamination by evaluation benchmarks. In *Conference on Empirical Methods in Natural Language Processing*, 2023.

[12] Denis Kocetkov, Raymond Li, Loubna Ben Allal, Jia Li, Chenghao Mou, Carlos Muñoz Ferrandis, Yacine Jernite, Margaret Mitchell, Sean Hughes, Thomas Wolf, Dzmitry Bahdanau, Leandro von Werra, and Harm de Vries. The Stack: 3 TB of permissively licensed source code. *arXiv preprint arXiv:2211.15533*, 2022.

[13] Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-tau Yih, Daniel Fried, Sida Wang, and Tao Yu. DS-1000: A natural and reliable benchmark for data science code generation. In *International Conference on Machine Learning*, pp. 18319–18345. PMLR, 2023.

[14] Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *Advances in Neural Information Processing Systems*, Vol. 35, pp. 3843–3857, 2022.

[15] Raymond Li, Loubna Ben allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia LI, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Joel Lamy-Poirier, Joao Monteiro, Nicolas Gontier, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Ben Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy, Jason T Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Urvashi Bhattacharyya, Wenhao Yu, Sasha Luccioni, Paulo Villegas, Fedor Zhdanov, Tony Lee, Nadav Timor, Jennifer Ding, Claire S Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro Von Werra, and Harm de Vries. Starcoder: may the source be with you! *Transactions on Machine Learning Research*, 2023. Reproducibility Certification.

[16] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, Vol. 378, No. 6624, pp. 1092–1097, 2022.

[17] Jiawei Liu, Jia Le Tian, Vijay Daita, Yuxiang Wei, Yifeng Ding, Yuhan Katherine Wang, Jun Yang, and Lingming Zhang. RepoQA: Evaluating long context code understanding. *arXiv preprint arXiv:2406.06025*, 2024.

[18] Tianyang Liu, Canwen Xu, and Julian McAuley. Repobench: Benchmarking repository-level code auto-completion systems. *arXiv preprint arXiv:2306.03091*, 2023.

[19] Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, et al. Starcoder 2 and the stack v2: The next generation. *arXiv preprint arXiv:2402.19173*, 2024.

[20] Rien Maertens, Charlotte Van Petegem, Niko Strijbol, Toon Baeyens, Arne Carla Jacobs, Peter Dawyndt, and Bart Mesuere. Dolos: Language-agnostic plagiarism detection in source code. *Journal of Computer Assisted Learning*, Vol. 38, No. 4, pp. 1046–1061, 2022.

[21] Mayank Mishra, Matt Stallone, Gaoyuan Zhang, Yikang Shen, Aditya Prasad, Adriana Meza Soria, Michele Merler, Parameswaran Selvam, Saptha Surendran, Shivdeep Singh, Manish Sethi, Xuan-Hong Dang, Pengyuan Li, Kun-Lung Wu, Syed Zawad, Andrew Coleman, Matthew White, Mark Lewis, Raju Pavuluri, Yan Koyfman, Boris Lublinsky, Maximilien de Bayser, Ibrahim Abdelaziz, Kinjal Basu, Mayank Agarwal, Yi Zhou, Chris Johnson, Aanchal Goyal, Hima Patel, Yousaf Shah, Petros Zerfos, Heiko Ludwig, Asim Munawar, Maxwell Crouse, Pavan Kapanipathi, Shweta Salaria, Bob Calio, Sophia Wen, Seetharami Seelam, Brian Belgodere, Carlos Fonseca, Amith Singhee, Nirmit Desai, David D. Cox, Ruchir Puri, and Rameswar Panda. Granite code models: A family of open foundation models for code intelligence. *arXiv*, 2024.

[22] Augustus Odena, Charles Sutton, David Martin Dohan, Ellen Jiang, Henryk Michalewski, Jacob Austin, Maarten Paul Bosma, Maxwell Nye, Michael Terry, and Quoc V Le. Program synthesis with large language models. *arXiv*, 2021.

[23] Ruchir Puri, David S Kung, Geert Janssen, Wei Zhang, Giacomo Domeniconi, Vladimir Zolotov, Julian Dolby, Jie Chen, Mihir Choudhury, Lindsey Decker, et al. Codenet: A large-scale ai for code dataset for learning a diversity of coding tasks. *arXiv preprint arXiv:2105.12655*, 2021.

[24] Martin Riddell, Ansong Ni, and Arman Cohan. Quantifying contamination in evaluating code generation capabilities of language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14116–14137, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

[25] Cheng Xu, Shuhao Guan, Derek Greene, and M-Tahar Kechadi. Benchmark data contamination of large language models: A survey. *arXiv*, 2024.

[26] Fanjia Yan, Huanzhi Mao, Charlie Cheng-Jie Ji, Tianjun Zhang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Berkeley Function Calling Leaderboard. https://gorilla.cs.berkeley.edu/blogs/8_berkeley_function_calling_leaderboard.html, 2024.

[27] Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, et al. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. *arXiv preprint arXiv:2406.15877*, 2024.

補足資料

表 6 学習データのプログラム言語（一部はデータのタイプ）のファイル数の比率。

プログラミング言語	比率	プログラミング言語	比率	プログラミング言語	比率
JavaScript	26.3%	HTML	3.4%	XML	1.5%
PHP	7.3%	Python	2.6%	SCSS	1.0%
JSON	7.3%	Java	2.5%	Go	1.0%
C	5.1%	C++	2.2%	YAML	0.9%
Markdown	4.8%	CSS	1.8%	...	...
TypeScript	4.7%	C#	1.8%	Rust	0.2%

表 7 評価データ毎の検出対象とするフィールドと、扱うプログラミング言語 ('-' は特定の言語を指定しないもの)。原則として問題と解答の両方について検査するが、OCW や CruxEval の解答のように非常に短い文字列（例：数字のみ）となるフィールドは検出対象としない。

評価データ	フィールド名	言語
HumanEval	prompt, canonical_solution	C++, Go, Java, JavaScript, Python
MBPP	code, text	Python
GSM8k	question	Python
DS1000	prompt, reference	Python
CanItEdit	instruction_descriptive, instruction_lazy	Python
OCW	problem	-
BFCL	question	Python, Java, JavaScript, SQL
CodeNet	code	C++, C, Go, Java, Python
CruxEval	code_python	Python
LCC	content	C#, Java, Python
CrossCodeEval	prompt	C#, Java, Python, TypeScript
CodeContests	test_description	-
MATH	problem	-
SAT	Question	-
BigCodeBench	instruct_prompt, canonical_solution	Python

表 8 文字列マッチングの対象から除外した文字列。空白や改行はマッチングの際に除去され、大文字は小文字化されるため、それらの差異は考慮しなくてよい。

評価データ	文字列
HumanEval	return x+y
	return x+y}
	return x+y;}
	return x+y;}}
	return n**2
	return n*n
	return n*n}
	return n*n;}
	return n*n;}}
	n*(n+1)/2
	n*(n+1)/2}
	return len(str)}
	return len(string)
	return string.length();}}

評価データ	文字列
DS1000	a=a**power
	result=a.shape
	result=a.argmax()
	result=a.argmin()}}
	a_np=a.numpy()
	i=np.diag(i)
	plt.legend()
	x.assign(1)
	a=np.sign(a)
	plt.legend(loc=”lowerright”)
	ax.xaxis.tick_top()
	plt.xticks(rotation=45)
	plt.minorticks_on()
BFCL	say hi
	version?
	get version