競技プログラミングを対象とした ファインチューニングによる オープンソースLLMの性能向上の可能性検討

5421001 斉藤 響　　5421058 三谷 涼輔

日本大学文理学部情報科学科　谷 聖一　研究室

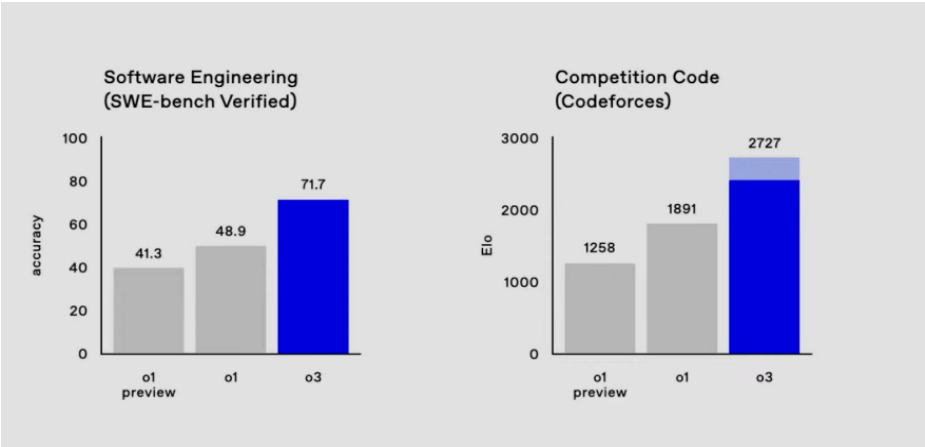
目次

1．導入

1-1. AIの進化

- ChatGPTやGoogle Gemini, など数多くのAIサービスが存在

- 最新AIモデル
  例）OpenAIの次世代AIモデル「o3」



Gemini

1-2. LLMとは

・LLM（Large Language Model, 大規模言語モデル）

機械学習における自然言語処理（NLP）の分野で扱われるモデル

膨大な量のテキストデータを学習し,

人間のように自然な文章を生成したり, 理解したりすることが可能

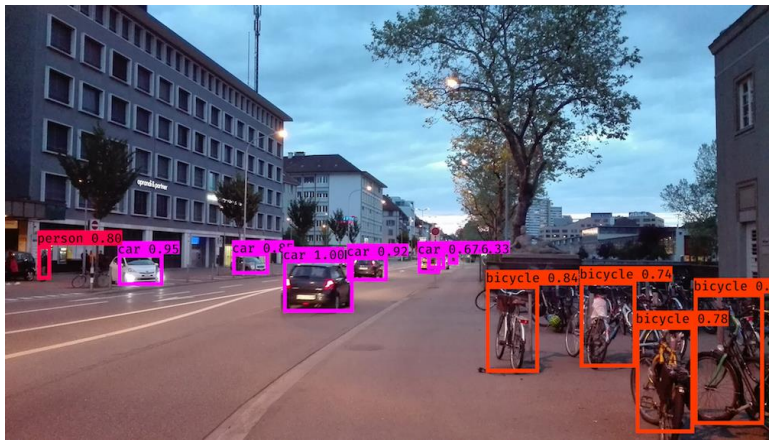
1-3. LLMの学習方法

- 事前学習（基礎）

自然言語における一般的な知識やパターンをあらかじめ学習

- ファインチューニング（応用）

特定のタスクに適応させるなどのため, 追加で学習させる



一般的なファインチューニングの例

特定タスク：メールのスパムフィルタリング
（メールが「迷惑メール」か「通常メール」かを判断する）

事前学習済モデルの選定	学習データの準備	学習	モデル評価
目的に応じたモデルの選定	**目的に応じた学習データの準備**	**モデルに適した学習の設定**	**モデルの性能を評価する評価基準を設定**
↓	↓	↓	↓
・日本語を学習した言語モデルを選定	**・メール本文 ・正解ラベル（スパムか否か）**	**・エポック数の設定（学習反復回数） ・学習率の設定（重み更新の割合） ・ ・ ・**	**• Accuracy（正解率） • Precision（適合率） • Recall（再現率） • F1スコア**

1-4. オープンソースLLMの利用について ～研究における利便性～

- **クローズドモデル**

  モデルの内部構造が非公開の場合が多く，
  モデル動作原理の理解が困難

- **オープンソースモデル**

  モデルの内部構造が公開されており，
  LLMの透明性や信頼性の確保に向けた研究に寄与

1-4. オープンソースLLMの利用について ～自己管理する場合～

- **クローズドモデルを用いたファインチューニング**

  →モデルのプロバイダーに学習データを提供する必要がある

- **オープンソースモデルを用いたファインチューニング**

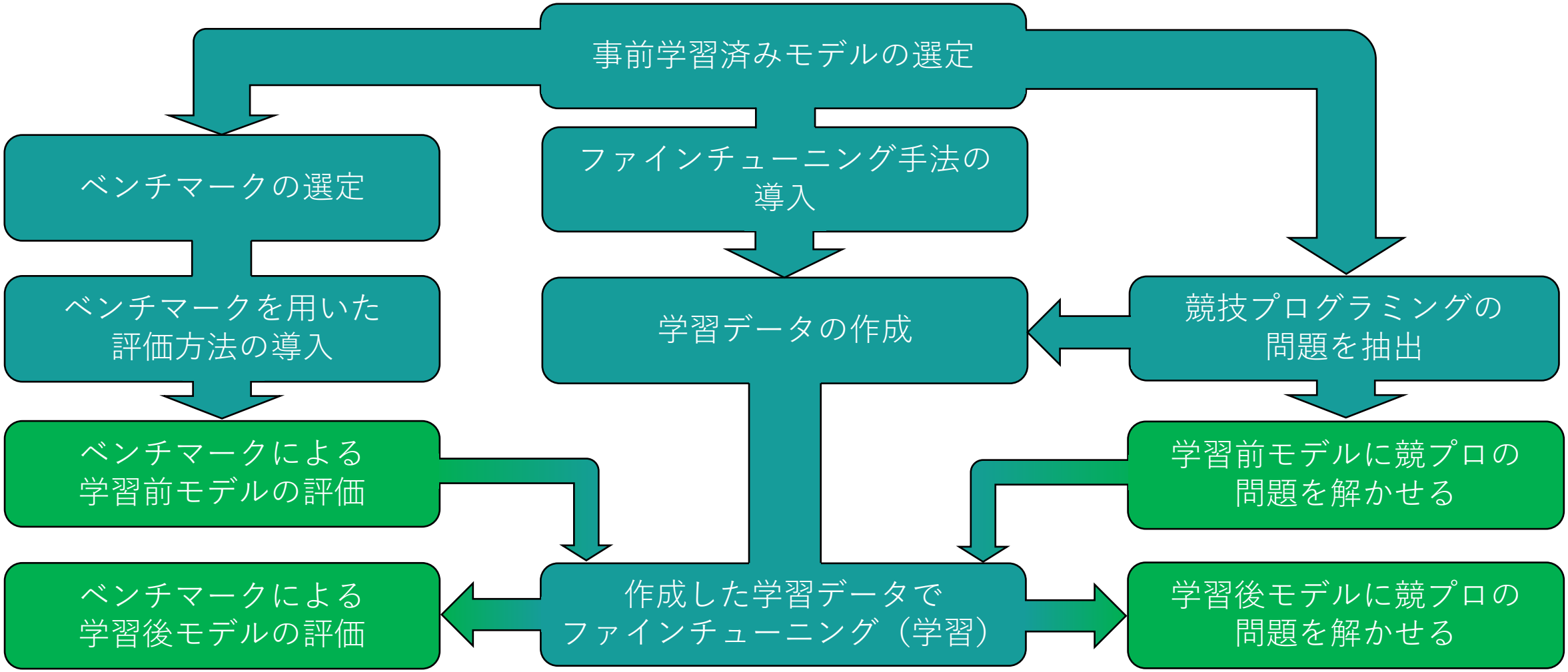
  →クローズドな自己環境を構築すると,
  学習データを第三者に提供する必要はない

本演習では,
クローズドモデルではなく, オープンソースモデルを研究利用

1-5. 演習の概要

目的

モデル評価結果と競プロの解答結果から
作成した学習データを用いたファインチューニング後のモデル性能を検証



2．準備

2-0.実験環境

venvで仮想環境構築

OS : Ubuntu 22.04.5 LTS

CPU：AWD EPYC 7V12 64-Core processor

CPUメモリ：54GB

GPU：Tesla T4 （Cuda 12.6）

GPUメモリ：16GB

ストレージ：126GB

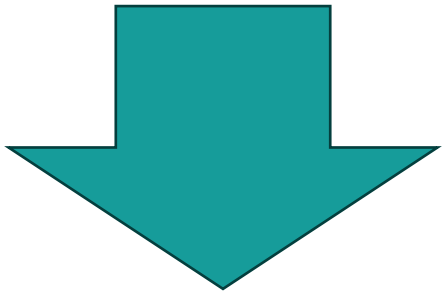
Python : 3.10.7

2-1. ファインチューニング

- **従来のファインチューニング（フル・ファインチューニング）**

数億〜数十億のパラメータを持つ大規模モデルでは, すべてのパラメータを更新するために多くのGPUが必要

このような環境を用意する必要がある



PEFT を用いることで解決！

2-1. ファインチューニング

- **PEFT (Parameter-Efficient Fine-Tuning)**

モデル全体ではなく, 一部のパラメータのみ再学習することで, コストを大幅に削減することができる効率的な手法

パラメータ数が多大なモデルでも, 単一GPUでのファインチューニングが可能

→PEFTとして実用化されている手法の内,
LoRA（Low-Rank Adaptation）を採用

2-1. ファインチューニング

- **LoRA（Low-Rank Adaptation）**

Edward Huらによって発表されたファインチューニングの効率化技術

モデルの重み行列の更新に低ランク行列を適用することで,
従来のファインチューニング手法と同程度の精度を保ちながらも
ファインチューニングに必要なパラメータ数を大幅に削減できる

2-1. ファインチューニング

重み行列の更新量を２つの低ランク行列で表現

$$W = W_{\text{base}} + \Delta W$$

$$\Delta W = A \cdot B$$

２つの低ランク行列を更新していくことで, 計算量を削減

2-1. ファインチューニング

[LoRAの効果]
2020年に発表されたGPT-3(175B) をフル・ファインチューニングする場合と比較すると,

「学習に必要なパラメータ数は 1/10000
使用するGPUメモリ数は 1/3 」

※論文（"**LoRA: Low-Rank Adaptation of Large LanguageModels**"）より

2-2. モデルの選定

NinedayWang/PolyCoder-160M

NinedayWang/PolyCoder-2.7B

EleutherAI/gpt-neox-1.3b

EleutherAI/gpt-neox-20b

EleutherAI/gpt-neo-1.3b

rinna/gpt-neox-japanese-3.6b

bigcode/santacoder (1.1B)

bigcode/gpt_bigcode-santacoder (1.1B)

bigcode/starcoder (15.5B)

bigcode/starcoder2-7b

モデルのパラメータ数や
コード生成能力などを考慮して
断念したものが多い

2-2. モデルの選定

・パラメータ数がそれほど大きくない（実験環境で動作）

・ある程度性能が高い

・PEFTを動かすことができる

ElutherAI/gpt-neox-20bの場合,

パラメータ数が非常に大きく, その分PEFTやHumanEvalによる評価の処理時間がかなり長くなってしまう

→実際に半日以上かかるケースがあった

2-2. モデルの選定

・パラメータ数が極端に大きくない（実験環境で動作）

・ある程度性能が高い

・PEFTを動かすことができる

NinedayWang/PolyCoder-160Mの場合,

パラメータ数が小さく, PEFTやHumanEvalは安定した時間で利用できるが, HumanEvalによる評価があまり良くなかった

2-2. モデルの選定

・パラメータ数が極端に大きくない（実験環境で動作）

・ある程度性能が高い

・PEFTを動かすことができる

bigcode/starcoder2-7b：PEFTを動かすことができず断念

starcoder2のアーキテクチャをPEFTがサポートしていなかった

使用モデル１：stabilityai/stable-code-3b（2024.1.17）

開発元：Stability AI

主に生成AI（画像・テキスト・コード生成）モデルの開発を手掛け, 世界中の開発者に向けてAIリソースを公開

事前学習データセット

・ウェブのテキストデータ

・プログラムコードやGitHubの問題に関するデータ

・プログラミングコードに特化したデータセット

・数学的な問題や内容に特化したデータ

使用モデル1：stabilityai/stable-code-3b（2024.1.17）

ベースモデル：stabilityai/stablelm-3b-4e1t (2023-8.25)

ウェブのテキストデータやプログラムコード,

GitHubの問題に関するデータなどを学習した汎用的な言語モデル

ファインチューニングするのが前提の基盤モデルとされ,

従来の最先端の3Bパラメータ言語モデルや, 7Bパラメータ規模のオープンソース言語モデルでさえも凌駕している(公式ページより)

使用モデル2：stabilityai/stable-code-instruct-3b（2024.3.26）

Stable Code 3Bを基盤として指示学習を行った大規模言語モデル

コード生成や対話的なタスク遂行能力が強化されている

プロンプトのデータセットと, それをDeepSeek Coder 34Bに与えた結果を合成データセットとして学習させた

指示学習：　指示→応答の形で行うファインチューニング

2-3. 学習データ

- AtCoderの正解プログラム（A～E問題）を680問程度抽出

※ IsHYuhi/AtCoder_Python/ABC (GitHub)

```python
H1 = int(input())
H2 = int(input())
print(H1-H2)
```

```python
li = [1, 2, 3, 4, 5, 6]
n = int(input())
n = n % 30
for i in range(n):
    m1 = i%5
    m2 = i%5 +1
    li[m1], li[m2] = li[m2], li[m1]
print("".join(map(str, li)))
```

2-4. 学習方法

2種類の方法で実験

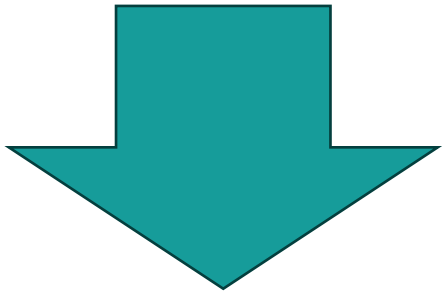
- コードすべてを予測させる学習方法
- コードの一部をランダムに隠しそれを予測させる学習方法
  （トークンマスキング）

DataCollatorForLanguageModeling()の引数を設定することで,
マスキングの有無や割合が変更可能

2-5. 評価

モデルのコード生成能力を評価するためには, 適切なベンチマーク（評価基準）が必要

本演習の学習データはPythonのソースコード



Pythonに特化したベンチマークを採用！

2-5. 評価

- **HumanEvalデータセット（2021-6.11）**

OpenAIが開発したPythonコードの生成能力を評価するためのベンチマーク

与えられた課題に対して正しいコードを生成できるかどうかを評価
評価は, 生成されたコードが与えられたテストケースを正しく通過するかどうかで行われる

2-5. 評価

"task_id": "test/0",

"prompt": "def return1():¥n",

"canonical_solution": "   return 1",

"test": "def check(candidate):¥n   assert candidate() == 1",

"entry_point": "return1"

---

task_id : データの識別子

prompt : モデルに与える入力データ

canonical_solution : プロンプトに対する正解ソリューション

test : 生成されたコードの正確性をテストするための関数

entry_point : テストが評価する関数名

2-5. 評価

・評価方法

pass@k：モデルが与えられたタスクに対して, k回答えを
生成したとき, 少なくとも一回は正確に答えられる割合

$$\text{pass@k} = 1 - \frac{C(n-c,k)}{C(n,k)}$$

n：生成されたサンプル数
c：正解したサンプル数
k：選択したサンプル数

※pass@1, pass@10,pass@100がサポートされている

例）pass@10 : 0.4999999999

→10回試行したときに, モデルが少なくとも1回はテストケースに
正しく答える割合が約50%であることを示す

2-5. 評価

- AtCoderを用いた評価データ

  問題文をプロンプトとして入力し, 生成されたコードから AtCoderにどれだけ対応できているかを評価

  学習データはABC001～ABC208まで.

  評価には, 学習データに含まれていない問題を使用

  ・ABC329　C-Count xxx (dif:205)

  ・AGC014　A-Cookie Exchanges (dif:296)　　など

3．演習

3-1. 演習結果（HumanEval）

stabilityai/stable-code-3b

HumanEval	学習前	マスキングなし	マスキングあり
Pass@1	0.0530	0.0859	0.1274
Pass@10	0.2560	0.2682	0.3536

評価値が約10%向上した

stabilityai/stable-code-instruct-3b

HumanEval	学習前	マスキングなし	マスキングあり
Pass@1	0.1749	0.1518	0.1568
Pass@10	0.5487	0.4085	0.5484

評価値は向上しなかった

3-1. 演習結果（AtCoder）

base,instruct共に

- 追加学習前:難易度150 以下の問題を正解
- 追加学習後:難易度150より高い問題は不正解

→「AtCoderの解答能力は向上しなかった」

※すべての問題で比較していないため,
正解できるようになっている問題はあるかもしれない

3-2. 今後の課題と考察

- **考察**

  マスキングの方が数値が高い理由として,
  マスクされた部分を前後のコードから意味を推測しながら学習を行うことによって,
  文脈能力が向上したからだと考えられる

  baseは, 純粋なコード補完の特性を持つため, 競技プログラミングデータで素直に向上

  instructは, 指示応答の形でファインチューニングを行わなかったため,
  優れていた指示応答の機能が薄れてしまったことにより,
  追加学習がうまく機能しなかった可能性

- **今後の課題**

  問題文を含めた学習, 学習データの増量, プロンプトの最適な与え方