大規模言語モデルの
ファインチューニング技術と評価
(FT-LLM)
成果報告

2026/03/13

チーム・ビクトリー

目次

メンバー紹介

- チームリーダー
  - 本郷 颯人　（東京大学）
- メンバー
  - 加地 翔太　（独立研究者）
  - 小谷 真士　（大阪公立大学）
  - 嶋中 雄大　（芝浦工業大学）
  - 宮川 裕貴　（中京大学）
  - 中島 悠樹　（東京大学）
  - 安孫子 リク（東京大学）
  - 松田 公慶　（筑波大学）

メンバー紹介

- チームリーダー
  - 本郷 颯人　（東京大学）
- メンバー
  - 加地 翔太　（独立研究者）
  - 小谷 真士　（大阪公立大学）
  - 嶋中 雄大　（芝浦工業大学）
  - 宮川 裕貴　（中京大学）
  - 中島 悠樹　（東京大学）
  - 安孫子 リク（東京大学）
  - 松田 公慶　（筑波大学）

解法: データの用意

- 学習データに関する制約事項
  - モデルの学習には著作権的にクリーンなデータのみ利用可
  - 近年は強力なLLMによる生成データを利用した学習が主流だが、OpenAI等の利用規約では、「競合モデルの開発への出力の利用」が明示的に禁じられている場合がある
  - 上記2点を満たす大規模なデータは少ない
    - 探した限りでは1Mを超える量のデータは見つからなかった

解法: データの用意

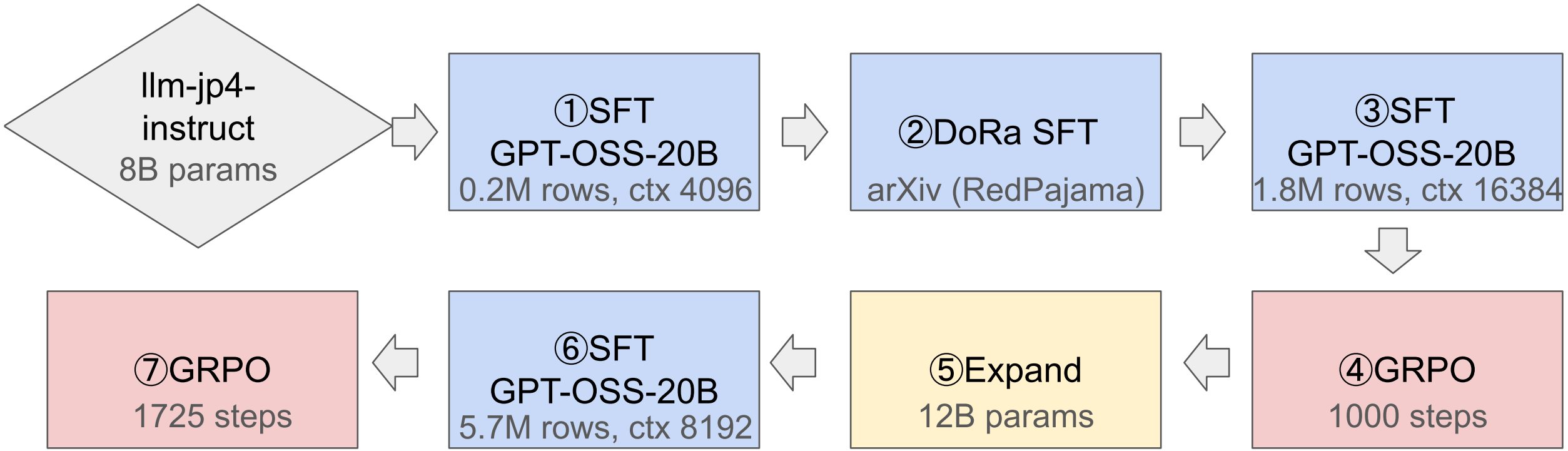
- GPT-OSS-20Bの利用
  - GPT-OSS-20Bを用いて、学習用の問題とその解答のデータを生成
  - Apache 2.0 ライセンスのため提出用のモデルに対する蒸留への利用が可能
  - 外部データを利用することなく、モデルだけから取得するため、著作権的にもクリーンである

解法: データの用意

- データ生成の流れ

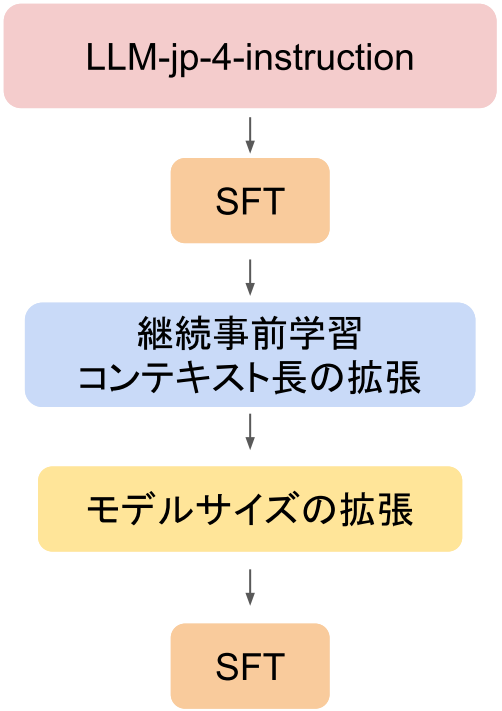
1. データ生成条件を設定
    - 数学の分野、難易度（1~10の10段階）、ツールの使用可否
2. 問題の生成
    - 1で設定した条件をプロンプトで指定し、GPT-OSS-20Bで問題と解答、その思考過程を生成(9M)
3. フィルタリング
    - LLM-as-Judgeを主としたフィルタリング
    - 問題と解答の整合性をGPT-OSSで評価

解法: モデル学習のパイプライン



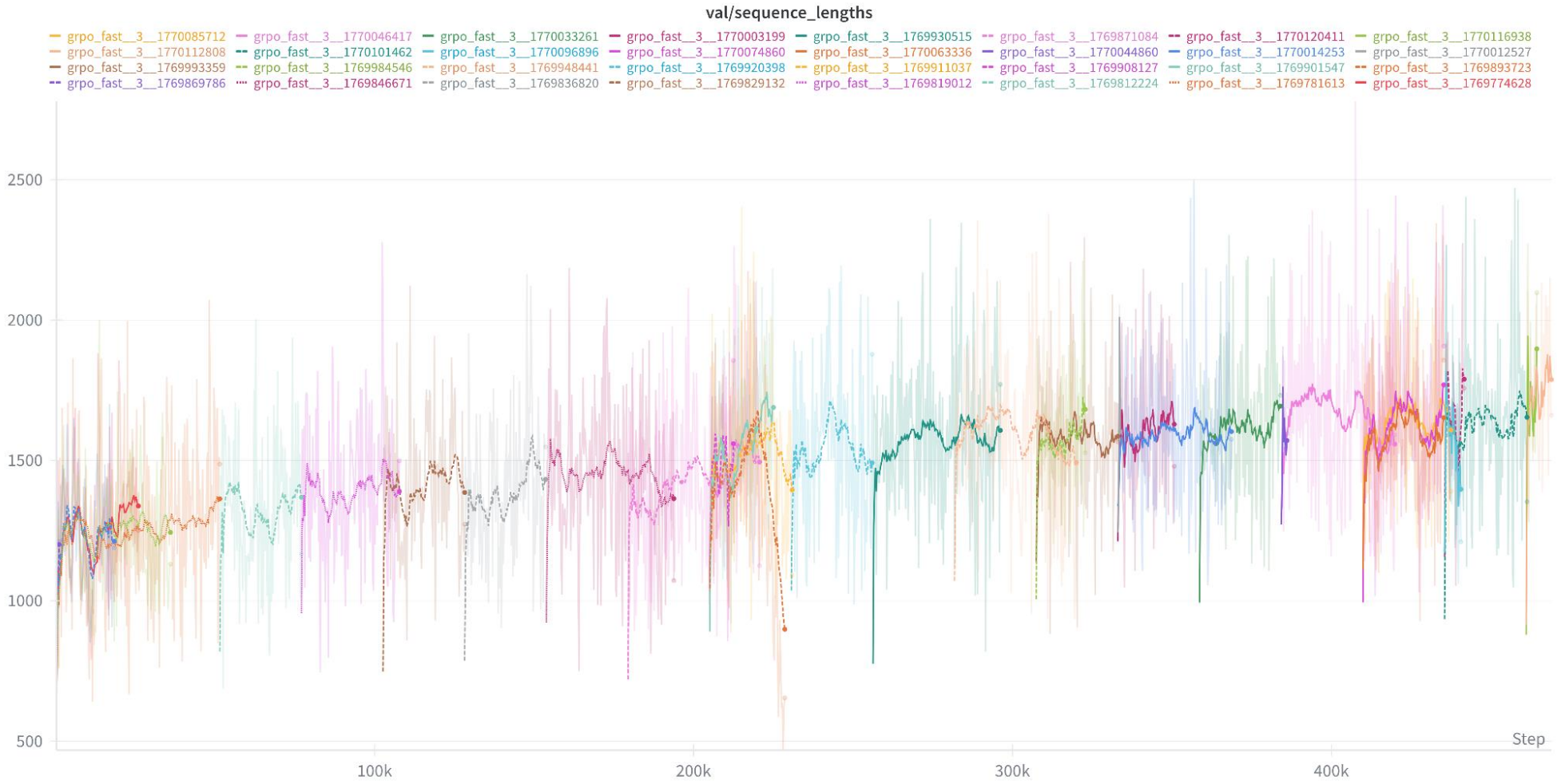
解法: 事後学習（SFT + モデルサイズ拡張）

- SFT
  - Open-R1 をベースとしたコードで学習
- 継続事前学習
  - arXivデータセットを利用した学習
  - コンテキスト長を16384まで拡張
  - DoRAを利用して破滅的忘却を防ぐ
- モデルサイズの拡張
  - OpT-DeUSを用いてモデルの層数を32層から48層へ



解法: 事後学習（open-instructによる非同期RL）

- バッチ内における平均生成長推移のグラフ



解法: 事後学習（open-instructによる非同期RL）

- Token-level loss 集約バイアスを代数的に定量化する
  - 問題意識：DAPOはtoken-level lossと呼ばれるが、各応答がloss集約の段階でどのように重みづけられるかは掴みにくい
  - そこで1応答単位の集約係数を定義し、GRPOとDAPOの違いを比較する

Sample-level loss

$$L_{GRPO} = \frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{t=1}^{|o_i|} l_{i,t}$$

Token-level loss

$$L_{DAPO} = \frac{1}{\sum_{j=1}^{G}|o_j|}\sum_{i=1}^{G}\sum_{t=1}^{|o_i|} l_{i,t}$$

解法: 事後学習（open-instructによる非同期RL）

- 1応答単位における集約係数 m_i の定義

$$m_i \coloneqq \sum_{t=1}^{L_i} w_{i,t}$$

- m_i：応答 i の票数
- L_i：応答 i の長さ
- G：グループサイズ（同一プロンプトから生成した応答本数）
- $w_{i,t}$：トークン(i,t)の集約係数
- $l_{i,t}$：トークン(i,t)の損失
  - 相対報酬に係数を乗じた形：-A × ratio × TIS（今回は着目しない）
- L_{tot}：グループ内層トークン数　$L_{tot} \coloneqq \sum_{j=1}^{G} L_j$
- $\overline{L_g}$：平均生成長　　$\overline{\mathrm{L}_g} \coloneqq \sum_{j=1}^{G} L_j$

解法: 事後学習（open-instructによる非同期RL）

- 応答単位の集約係数 m_i を定義すると何が言えるか
  - GRPOは各応答の票数が等しく $\frac{1}{G}$
  - DAPOは1応答当たりの票数が長さに比例する
    - 1応答あたり、生成長/平均生成長の重みが票数となる

$$m_i^{GRPO} = \sum_{t=1}^{L_i} \frac{1}{G L_i} = \frac{1}{G}$$

$$m_i^{DAPO} = \sum_{t=1}^{L_i} \frac{1}{L_{tot}} = \frac{L_i}{L_{tot}}$$

導出1：GRPOとの比を取る

$$\frac{m_i^{DAPO}}{m_i^{GRPO}} = \frac{L_i}{L_g}$$

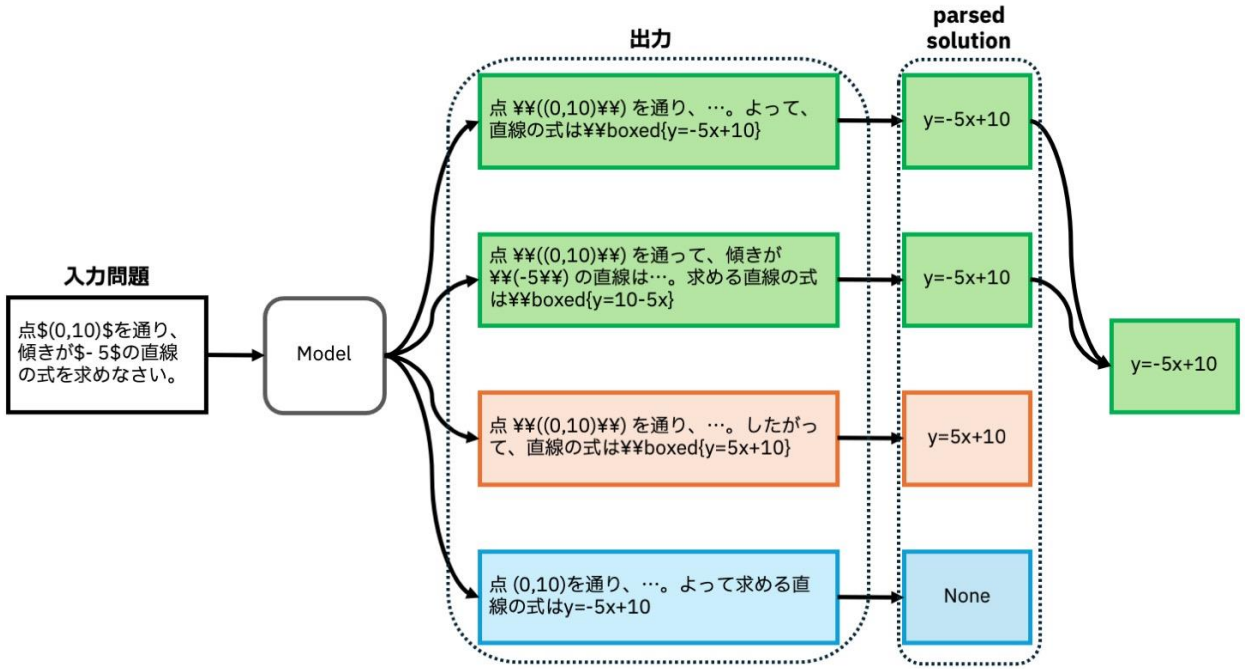
導出2：DAPOの平均応答票数との比を取る

$$\frac{m_i^{DAPO}}{\overline{m}^{DAPO}} = \frac{m_i^{DAPO}}{\frac{1}{G}} = \frac{\frac{L_i}{L_{tot}}}{\frac{1}{G}} = \frac{L_i}{L_g}$$

- DAPOのtoken-level loss集約の機能バイアスは”生成長/平均生成長”として定量化可能
- この倍率はGRPO基準で見たDAPOの相対重みでもある

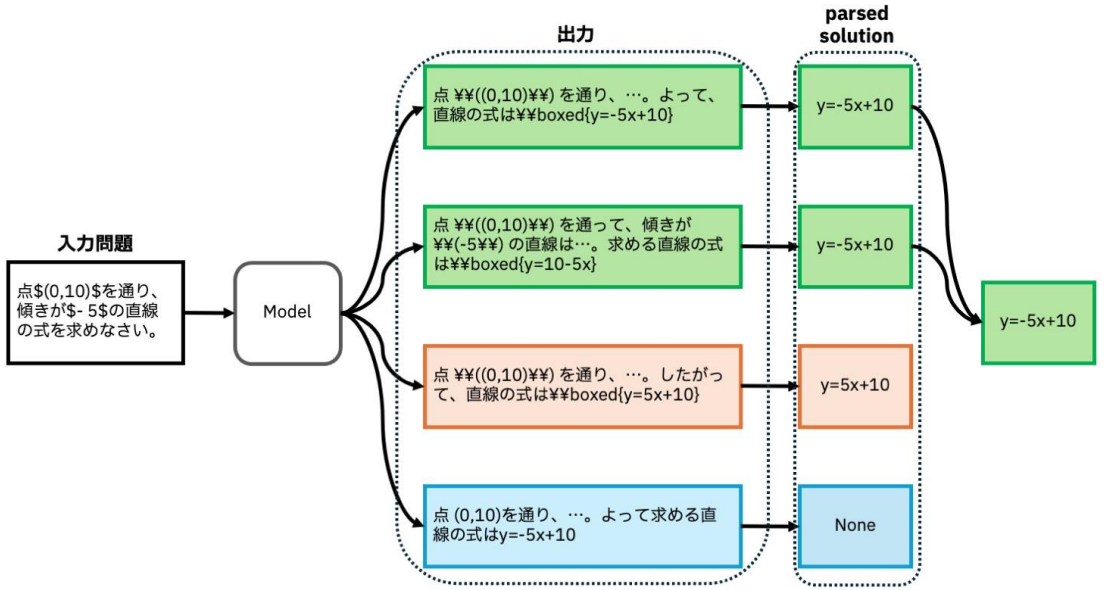
解法: 推論システム（Self-Consistency）

- Self-Consistencyの概要
  - 1つの問題に対して複数個の解答を取得
  - 得られた解答の内、最も出現頻度の高い解答をその問題に対する最終的な解答とする



解法: 推論システム（Self-Consistency）

- 多数決の実現
  1. LLMの出力をmath-verifyのparse関数で解答となる数式部分を抽出
     - [(出力中の該当箇所), (math-verify内の記法)]の形で取得可能
  2. math-verify内の記法で解答を集計し、多数決を適用
  3. 最多の解答を最終出力（”output”カラムへの記入内容）とする



解法: 評価結果

	cons@1	cons@40	cons@160
Llm-jp-4-instruct (baseline)	0.354	0.552	0.594
GRPO後(図1⑦)	0.818	0.878	**0.880**

- 事後学習（SFT + RL）とSelf-Consistencyにより、正解率が大幅に増加
  - 0.354（baseline・pass@1）→ 0.880（SFT + RL・cons@160）
- 事後学習・cons@1のみでも 0.818 を達成

その他の試行錯誤①: RSA

- 手法の概要
  1. N個の解答を生成、その中からK個をサンプリング
  2. 元の問題とサンプリングした解答をLLMに入力し、推論過程を集約
  3. 集約された内容を元に再度解答を生成
  4. 1 ~ 3 のループを繰り返す
- 採用しなかった理由
  - 膨大なトークン数が必要となり、モデルが処理できないケースがある
  - Aggregation（集約）に対応する学習が実施できなかった
    - フィルタリングが不十分で、誤答の排除が十分にできなかった

その他の試行錯誤②: s1

- 手法の概要
  - LLMの出力に”wait”を挿入して、問題と共に再入力することで精度が向上するという手法
- 採用しなかった理由
  - Self-Consistencyのみとs1との併用を比較した際に、推論時間の増加程度に対して精度の向上が見られなかった

その他の試行錯誤③: コードSFT

- 手法の概要
  - LLMにPythonコードを生成・実行させ、その結果を踏まえて後の推論を行う
  - Python経由で電卓等のツールの利用ができるため、複雑な計算における計算ミスの抑制を期待
- 採用しなかった理由
  - 推論時間が大幅に増加し、500分の制限時間に抵触する可能性が高い
  - CoT形式の方が精度が高かった

その他の試行錯誤④: RAG

- 手法の概要
  - 例題と入力問題を multilingual-e5-base でEmbeddingし、比較することで、参考となる問題を検索
  - 類似した問題を解答と合わせて、参考問題としてLLMに入力
  - LLMが解答作成時に指針のような形で参考する期待
- 採用しなかった理由
  - 高品質な参考問題を用意できなかった
  - 簡易実験で精度の向上が見られなかった