

SelfCodeAlign: Self-Alignment for Code Generation

Y. Wei, F. Cassano, J. Liu, Y. Ding, N. Jain, Z. Mueller, et al.
NeurIPS 2024

背景

ソフトウェア開発へのLLMの応用

- 多様なプログラミングタスクで高い性能を達成
 - 自然言語の記述から関数やクラスを作るプログラム合成
 - バグを直すプログラム修正や実行速度を高めるコード最適化
 - コード補完・コード翻訳・テスト生成・ソフトウェアエージェント

背景

コード用LLMの学習法

- 事前学習
 - 数兆トークンのソースコードを用いた次トークン予測
 - コードは書けるが指示に従う能力は不十分
- 指示チューニング
 - 自然言語の指示と応答の組を用いた教師あり微調整
 - 指示に従い汎用的な開発タスクを解く能力が向上

指示データの収集における要件

- 質の高い指示データを大量に収集する必要がある
 - 多様で現実的な指示と正確に動作する応答の組が不可欠

先行研究

指示データの集め方と限界

集め方	データの作り方	限界
人手アノテーション	人が指示と応答を作成	膨大な作成費用
強いモデルからの蒸留	GPT の出力を収集	利用規約への抵触 より強いモデルが必要
公開データの流用	コミットを指示に転用	性能の低さ
自己生成	モデル自身が生成	コードでは性能が低下

- 人手や他のモデルや既存データに頼る集め方には問題がある
- 外部に頼らない自己生成は生成データの質が不十分

課題

自己生成で性能が低下する原因

- 指示の多様性を制御する仕組みがない
 - 少数の例から作るため似た指示に偏りやすい
- 応答の正しさを検証する仕組みがない
 - 誤ったコードがそのまま学習データに入る

➡ 多様性の制御と正しさの検証が課題

目的

自己生成のみによる指示チューニング

- 人手アノテーションや強い教師からの知識蒸留が不要
- ベースモデル自身の推論のみで高品質な指示データを構築
- 完全な透明性と寛容なライセンスを持つ学習パイプライン

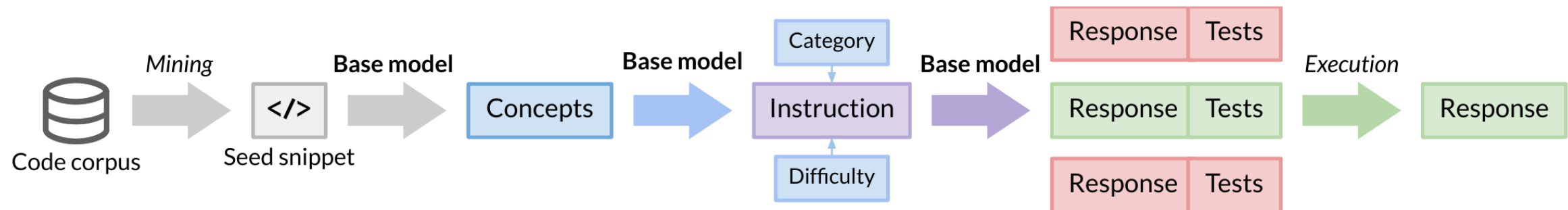
多様性と正しさの確保方法

- 概念抽出による指示の多様化
 - 公開コードから抽出したプログラミング概念から指示を生成
- 自作テストの実行による応答の選別
 - モデル自身が作成したテストを実行し通過した応答のみ採用

提案手法

SelfCodeAlign パイプラインの全体像

- データ生成から検証まですべて同一のベースモデルで実行



処理の流れ

- The Stack V1 から Python のシード関数25万件を収集
- シード関数からプログラミング概念を抽出
- 概念をもとに指示と応答とテストを生成
- テストを通過した組のみを学習データに採用

実験設定

主要実験の学習条件

- ベースモデルに CodeQwen1.5-7B-Base を使用
- 自己生成した約74,000件の指示データで微調整

比較対象モデル

- 学習データが透明なモデル
 - 同じベースモデルの Base・OctoPack と OctoCoder-16B
- 学習データが非公開のモデル
 - 同じベースモデルの Chat と CodeLlama-70B-Instruct

評価

- HumanEval+ と MBPP+ を含む7種のベンチマーク
 - pass@1 で評価

実験結果

関数生成の性能比較

モデル	規模	指示データ	HumanEval+	MBPP+
Chat	7B	非公開	77.7	67.2
提案手法	7B	自己生成	67.1	65.2
CodeLlama-70B-Instruct	70B	非公開	65.2	61.7
OctoPack	7B	公開データ	50.6	63.2
Base	7B	なし	45.7	60.2
OctoCoder-16B	16B	公開データ	32.9	49.1

実験結果

他タスクでの性能比較

ベンチマーク	タスクの内容	Base	OctoPack	Chat	提案手法
LiveCodeBench	競技プログラミングのコード生成	19.3	19.3	23.2	22.4
EvoEval	関数単位のコード生成	36.2	42.2	48.0	43.6
ClassEval	クラス単位のコード生成	23.0	21.0	27.0	27.0
DS-1000	データ分析コードの補完	32.4	38.2	47.1	39.1
CanItEdit	指示に従う既存コードの編集	38.4	36.5	39.9	39.0

- 全タスクで OctoPack を上回り ClassEval では Chat 版と同点

実験結果

データ生成モデルの影響

学習対象モデル	教師モデル				
	StarCoder2 3B	Llama-3 8B	StarCoder2 15B	DeepSeek Coder-33B	CodeQwen 1.5-7B
StarCoder2-3B	35.4	34.1	39.0	42.1	40.2
Llama-3-8B	–	42.7	40.2	41.5	43.3
StarCoder2-15B	–	–	55.5	53.0	57.3
DeepSeek-Coder-33B	–	–	–	65.9	62.2
CodeQwen1.5-7B	48.8	54.9	56.1	59.1	65.2

- 性能差の小さい教師よりも自己生成データが有効

実験結果

選別の効果

- 指示を固定し応答の選び方のみ変更

応答の選び方	データ量	テスト通過率	pass@1
合否を見ずに選択	27.7k	24.1%	61.6
合否を見ずに選択	15.6k	24.2%	61.6
失敗した応答のみ	15.6k	0%	57.9
通過した応答のみ	15.6k	100%	65.2

- 件数によらず正しい応答だけの学習が有効

実験結果

シードと概念抽出の効果

設定	シード	指示の作り方	pass@1
提案手法	選別した関数	抽出した概念を基に生成	65.2
シード選別なし	無作為なコードスニペット	抽出した概念を基に生成	64.0
概念抽出なし	選別した関数	シードから直接生成	59.8

- シードを選別しないと概念の一貫性が低下する
- 直接生成するとシードの言い換えに近い指示になる

実験結果

蒸留との比較

- 学習対象の CodeQwen1.5-7B と件数は共通

データの作り方	教師モデル	実行で選別	pass@1
Evol-Instruct	GPT-3.5-Turbo	なし	59.1
OSS-Instruct	GPT-3.5-Turbo	なし	61.6
直接蒸留	GPT-4o	なし	65.9
提案手法	CodeQwen1.5-7B 自身	あり	67.1

- 実行による選別により GPT-4o の直接蒸留より高性能

今後の課題

データ生成と検証における制限

- 自動生成された単体テストに誤りが含まれる可能性
- 複数サンプリングと実行検証に多くの推論時間を要する

今後の展望

- 失敗した応答を強化学習に活用する余地
 - 現在は破棄している不通過の応答を負例として学習に利用可能
- 複雑なプログラム生成やエージェントによる開発への適用

まとめ

自己生成と自己検証による学習

- 概念抽出による多様化とテスト実行による応答選別を統合
- 人手データや強いモデルからの知識蒸留を使わずに学習

実験と分析の結果

- 70Bモデルや GPT-4o からの直接蒸留を上回る
- 差の小さい教師モデルより自身の生成データが有利

利用上の示唆

- 利用規約の制約なく独自モデルを強化したい場面で有効
- 教師との性能差が小さい場合は蒸留より自己生成を選択