

Adapting Knowledge Prompt Tuning for Enhanced Automated Program Repair

Xuemeng Cai
Singapore Management
University
Singapore
xuemengcai@smu.edu.sg

Lingxiao Jiang
Singapore
Management
University
Singapore
lxjiang@smu.edu.sg

発表者: 福井 遥陽

背景 | 自動プログラム修復(APR)

| 自動プログラム修復

ソフトウェア開発のバグ修正を自動化

| 最新の研究

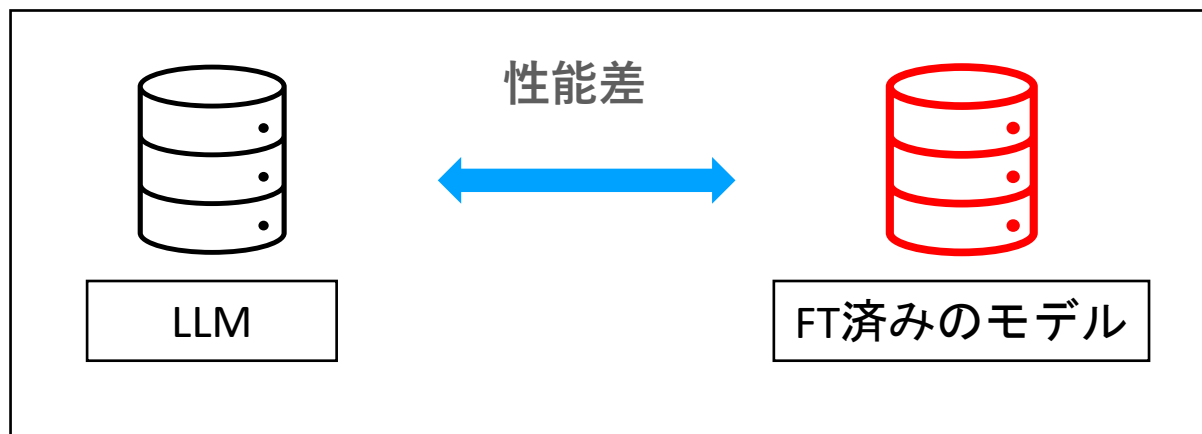
- ・ 代表的なAPRの評価指標を用いて最先端モデルのAPRタスクに対する性能を検証

→LLMが従来の手法やAPRツールと同等の性能を発揮

APRタスクにおいてLLM利用の可能性が注目

背景 | 自動プログラム修復(APR)

APRタスクにおいて、事前学習済みモデルとタスク用にファインチューニングされたモデルの間に**大きな性能のギャップ**



背景 | 自動プログラム修復(APR)

APRタスクにおいて、事前学習済みモデルとタスク用にファインチューニングされたモデルの間に大きな性能のギャップ

FT用のデータ量の不足が原因で、モデルの性能に制限

LLM

済みのモデル

FTしたモデルの使用が一般化

性能向上のためにプロンプトチューニングを導入

目的 | アプローチ

目的：訓練データ不足によるFT済みモデルの性能制限を改善

アプローチ：プロンプトチューニングを導入

➤ プロンプトチューニングとは...



プロンプトの例)
ハードプロンプト：自然言語
 $f(X, \text{mask}) = \text{Fix the bug in } [X][\text{mask}]$
ソフトプロンプト：連続ベクトル
 $f(X, \text{mask}) = [X] [\text{SOFT}] [\text{SOFT}] [\text{SOFT}] [\text{mask}]$

アプローチ

大まかに以下の三つの要素で構成される

A : 基本プロンプトチューニング

B : 知識プロンプトチューニング

C : プロンプトチューニングとファインチューニングの実装

アプローチ | 基本プロンプトチューニング

バグのあるコード(X)とモデルの出力スロット(mask)以外に、**ドメイン知識を含まないプロンプト**を用いて、プロンプトチューニングを行う。

	Basic Hard prompt	Basic Soft prompt
1	[X] [mask] is fixed program	[X] [mask] [SOFT] * 3
2	[X] fixed program is [mask]	[X] [SOFT] * 3 [mask]
3	Fix bug in [X] [mask]	[SOFT] * 3 [X] [mask]
4	Fix [X] fixed program is [mask]	[SOFT] [X] [SOFT] * 3 [mask]
5	Fix [X] [mask] is fixed program	[SOFT] [X] [mask] [SOFT] * 3
6	[X] is buggy program [mask] is fixed program	[X] [SOFT] * 3 [mask] [SOFT] * 3
7	Fix [X] is buggy program [mask] is fixed program	[SOFT] [X] [SOFT] * 3 [mask] [SOFT] * 3

アプローチ | 知識プロンプトチューニング

前述したプロンプトに加えて、各バグにと犯したコードやバグに関するドメイン知識を含むプロンプトを用いる。

以下のデータセットからドメイン知識を抽出

Dataset	Domain knowledge
Defects4j [37]	Repair action, Repair pattern
ManySStuBs4J-SStuBs [38]	Bug type, AST of buggy nodes
TFix [31]	Bug type, Error message
xCodeEval [39]	Error message, Tags of potential algorithmic techniques in buggy code

プロンプト例) Please fix a buggy program *XX*. The bug type is *bugType**bugType*. *maskmask* is the fixed version.

アプローチ | プロンプト・ファインチューニングの実装

- | プロンプトチューニングにOpenPromptフレームワークを使用

- | PyTorchをベースに実験を実行

実験設計

- | RQ1...データ不足の状況で，APRタスクの完了にプロンプトチューニングはどれほど効果的か？
- | RQ2...異なる種類のプロンプトはプロンプトチューニングの性能にどう影響するか？
- | RQ3...ドメイン知識の統合は，知識プロンプトにどれぐらい有益か？
- | RQ4...トレーニングデータセットのサイズはプロンプトチューニングの性能にどう影響するか？

実験設計

使用するモデル：

- CodeT5+
- GPT-Neo 1.3B

- 1.コードコーパスで事前学習されている
- 2.パラメータ数が15億未満
- 3.APRにそのまま適用可能

実験設計

用いる評価データセット

Dataset	Language	Training Set	Val. Set	Test Set
BugsInPy [44]	Python	932	112	122
Code Refinement [45]	Java	523	65	65
Defects4J	Java	502	63	64
ManySStuBs4J-SStuBs	Java	260	32	32
TFix	JavaScript	816	102	102
xCodeEval-APR-C	C	1,082	135	135

全てのデータセットは連続したコード断片になるように前処理

実験設計

評価指標

1 Exact Match(EM)

- ・ 生成された修正パッチが参照修正と完全に一致した割合

2 Syntactically Correct Patch(SC)

- ・ 修正パッチが参照パッチと構文的に同等である割合

3 CodeBLEU

- ・ 修正パッチと参照パッチの類似度を測定する指標

実験結果 | RQ1

Model	Tuning Methods	BugsInPy			Code Refinement			Defects4J		
		EM	SC	CodeBLEU	EM	SC	CodeBLEU	EM	SC	CodeBLEU
Naive Copy	–	0	0	76.94	0	0	75.82	0	0	75.94
CodeT5+ 220M	Fine-tune prompt tuning	13.93	14.75	77.66	0.51	0.51	75.45	5.21	5.21	76.22
		15.57	16.39	76.65	1.54	1.54	86.42	14.06	14.06	75.41
CodeT5+ 770M	Fine-tune prompt tuning	13.52	13.93	76.95	0	0	75.41	4.17	4.17	75.98
		18.03	18.85	77.17	2.56	2.56	86.27	12.50	12.50	75.45
GPT-Neo 1.3B	Fine-tune prompt tuning	0	0	75.79	0	0	75.82	0	0	73.7
		13.11	14.75	74.38	1.03	1.54	84.16	12.50	12.50	73.87
Model	Tuning Methods	ManySSuBs4J			TFix			xCodeEval		
		EM	SC	CodeBLEU	EM	SC	CodeBLEU	EM	SC	CodeBLEU
Naive Copy	–	0	0	91.10	0	0	57.19	0	0	75.45
CodeT5+ 220M	Fine-tune prompt tuning	49.31	49.31	94.92	20.92	21.90	63.63	1.60	1.73	73.53
		57.29	57.29	94.13	23.53	24.18	64.53	3.46	3.46	75.21
CodeT5+ 770M	Fine-tune prompt tuning	48.61	48.61	94.17	21.13	22.22	64.16	2.35	2.35	73.56
		57.29	57.29	93.69	25.82	26.14	66.08	4.44	4.69	74.92
GPT-Neo 1.3B	Fine-tune prompt tuning	0	0	90.36	0	0	56.40	0	0	74.89
		54.17	54.17	93.53	17.97	17.97	62.47	2.22	2.22	74.98

全てのデータセットでプロンプトチューニングのほうがファインチューニングより精度が高い。

平均でCodeT5で37.1%， GPT-Neoで71.2%の向上

実験結果 | RQ2

プロンプトの種類は性能に影響を与えるか？

Basic Prompt Template	BugsInPy			ManySSuBs4J			TFix			xCodeEval		
	CodeT5+ 220M	CodeT5+ 770M	GPT-Neo 1.3B	CodeT5+ 220M	CodeT5+ 770M	GPT-Neo 1.3B	CodeT5+ 220M	CodeT5+ 770M	GPT-Neo 1.3B	CodeT5+ 220M	CodeT5+ 770M	GPT-Neo 1.3B
1	13.93	15.57	4.10	57.29	56.25	1.04	22.88	22.22	6.54	3.21	3.70	0.74
2	13.11	12.30	12.30	55.21	54.17	54.17	23.20	21.90	17.97	2.72	4.20	1.98
3	13.93	13.93	13.11	56.25	56.25	51.04	21.57	22.22	16.67	2.96	3.95	1.98
4	13.93	18.03	13.93	55.21	57.29	52.08	22.22	23.86	16.99	2.47	3.70	1.23
5	13.93	16.39	3.28	57.29	55.21	8.33	22.22	21.90	6.54	3.46	2.96	0.74
6	15.57	15.57	4.10	55.21	55.21	13.54	21.90	20.92	11.76	3.21	4.44	0.74
7	13.93	14.75	4.92	55.21	56.25	22.92	21.90	21.90	11.44	3.46	3.46	1.48

テンプレートごとに性能が微量に変わることが示された

実験結果 | RQ3

ドメイン知識の統合によって性能はどう変わるか？

データセット	知識なし	知識有り	改善率
BungsInPy	0.153	0.179	+17.0%
Defects4J	0.090	0.111	+23.3%
ManySStuBs4J-SStuBs	0.128	0.139	+8.6%
Fix	0.158	0.188	+19.0%
平均	0.133	0.153	+15.1%

全てのデータセットにおいて知識有りのプロンプトチューニングの方が精度が高い

実験結果 | RQ4

トレーニングデータサイズが与える影響は？

Language	Methods	CodeT5+ 220M								CodeT5+ 770M							
		1 shot		8 shot		16 shots		32 shots		1 shot		8 shot		16 shots		32 shots	
		EM	C.BLEU	EM	C.BLEU	EM	C.BLEU	EM	C.BLEU	EM	C.BLEU	EM	C.BLEU	EM	C.BLEU	EM	C.BLEU
Python	Fine-tune	0	12.47	0	56.64	0	72.95	0	73.83	0	21.56	0	63.58	0.67	73.76	0.67	75.11
	Prompt tuning	0	43.35	0	66.81	0	75.45	0.67	75.84	0	57.56	0.67	70.24	0.67	75.50	1.67	75.80
Java	Fine-tune	0	6.06	70.33	83.54	90.00	87.56	174.33	91.52	0	18.46	102.67	84.60	113.33	87.32	203.00	92.08
	Prompt tuning	1.67	30.98	75.00	85.21	111.67	91.34	191.00	93.25	3.33	44.28	104.33	88.43	145.00	91.76	218.67	93.24
JavaScript	Fine-tune	0	9.19	4.00	47.93	8.00	53.28	15.00	56.62	0	15.18	7.67	50.99	12.33	54.66	20.00	56.50
	Prompt tuning	1.33	25.22	4.67	51.01	8.50	54.82	20.33	57.47	1.34	31.52	11.33	52.35	19.50	55.38	24.67	57.50
C	Fine-tune	0	15.10	2.00	70.86	2.33	73.39	3.67	73.96	0	5.72	6.00	72.39	4.00	73.78	4.00	73.92
	Prompt tuning	0	61.61	3.33	72.81	3.00	73.25	3.33	73.59	1.00	61.30	4.67	72.43	2.33	49.55	4.33	73.76

データ量が多いほどモデルの性能が向上

考察

知識の効果に関する分析

- ・全てのドメイン知識が有効ではない
- ・今後は知識線タックや重み付けの戦略が重要

ソフトvsハードの安定性と効果

- ・ソフトの方が安定して性能が高い
- ・モデルが大きいほどソフトの優位性が顕著に

小規模データ下での優位性

- ・FTはデータ不足では性能劣化のリスクが高い
- ・PTは小規模なデータでも高精度を達成可能

考察

モデルサイズの影響

- ・モデルサイズが大きいほどPTの効果が顕著

実践的な応用と課題

- ・PTは計算資源の節約にもつながる。（モデル全体の再学習不要）
- ・少量のデータしか得られない現場での活用が期待できる。

今後の課題

1. 知識抽出の自動化：プロンプト埋め込みは手作業なため自動化が推奨される。
2. タスク適応型プロンプト生成：手動設計のため、タスクごとに最適化できない。

所感

この論文では、学習用のデータの不足をプロンプトチューニングという手法を用いて改善した.

似たような手法は他のタスクの分野でも見受けられていたので、データ不足を改善させるには有効な手法だと感じた.