

CODEPROMPTZIP: Code-specific Prompt Compression for Retrieval-Augmented Generation in Coding Tasks with LMs

Pengfei He, Shaowei Wang, Tse-Hsun Chen

In Findings of the Association for Computational Linguistics: ACL 2026, pages 27811–27825, San Diego, California, United States. Association for Computational Linguistics.

S2310157 谷口

背景

LLMに渡すプロンプト長のトレードオフ

- LLMを用いたコーディングでは、プロンプトにコード例を組み込むことで性能が向上する
 - クエリに関連するコード例を検索によって取得(RAG, 検索拡張生成)
- しかし、コード例は長くなりやすい
 - 数万トークン規模にまでなるとコストが高く問題になる
- プロンプトを圧縮する手法はあるが
既存のものは自然言語向けの手法

⇒ コード専用のプロンプト圧縮手法（CODEPROMPTZIP）の提案

圧縮する部分

指示やクエリはそのまま、コード例のみを圧縮

指示部

以下の入出力例を
参考に生成せよ

コード生成やバグ修正など
LLMが行うタスク内容を指示

入出力例1

入力コード例
⇒正解コード例

入出力例2

入力コード例
⇒正解コード例

プロンプトに組み込まれた
過去の問題解決例のみを削る

クエリ

入力コード
⇒ ?

今回解決する問題を示す

先行研究

自然言語タスクのプロンプト圧縮(LLMLingua)

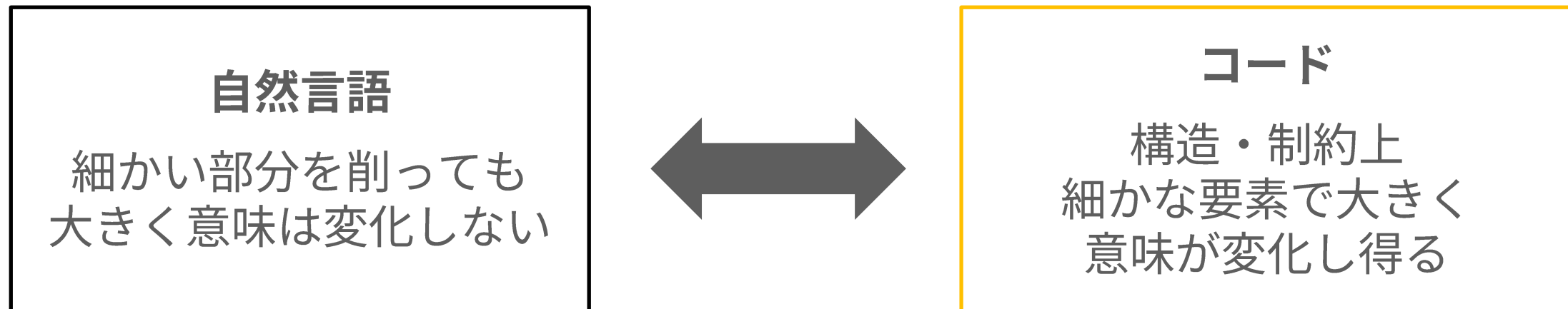
- 小型モデルで困惑度(PPL)を計算
重要なトークンだけ残す
 - 予測しづらいトークンほど重要
- 指示、例、クエリで
圧縮率を調整
 - 冗長な例を重点的に圧縮
- 自然言語タスクでは強く圧縮可能
 - 20倍まで圧縮して
完全一致率(EM)が1.5pt低下

GSM8K	トークン	圧縮	EM
Full-shot	2,366	-	78.85
LLMLingua	446	5x	79.08
LLMLingua	171	14x	77.41
LLMLingua	117	20x	77.33

出典: Jiang et al., “LLMLingua: Compressing Prompts for Accelerated Inference of Large Language Models,” EMNLP 2023, Fig. 1

コードの圧縮

自然言語の圧縮よりも慎重に



- 「予測しやすいトークンを削る」では不十分
- 実際に削ってみて問題ないトークンを削るようにする

各トークンの分類

抽象構文木(AST)を用いて各トークンを分類

分類	意味	例
Symbol	演算子・区切り記号	=, {, ;
Signature	宣言・引数など	calculate(int x)
Invocation	関数・メソッド呼び出し	foo.bar()
Identifier	変数名・名前	count, result
Structure	制御構造など	if, for, return

- ・それぞれの分類に属するトークンを消していき
その性能低下を参考に各分類の削除優先順位を付けていく

研究で扱うタスク

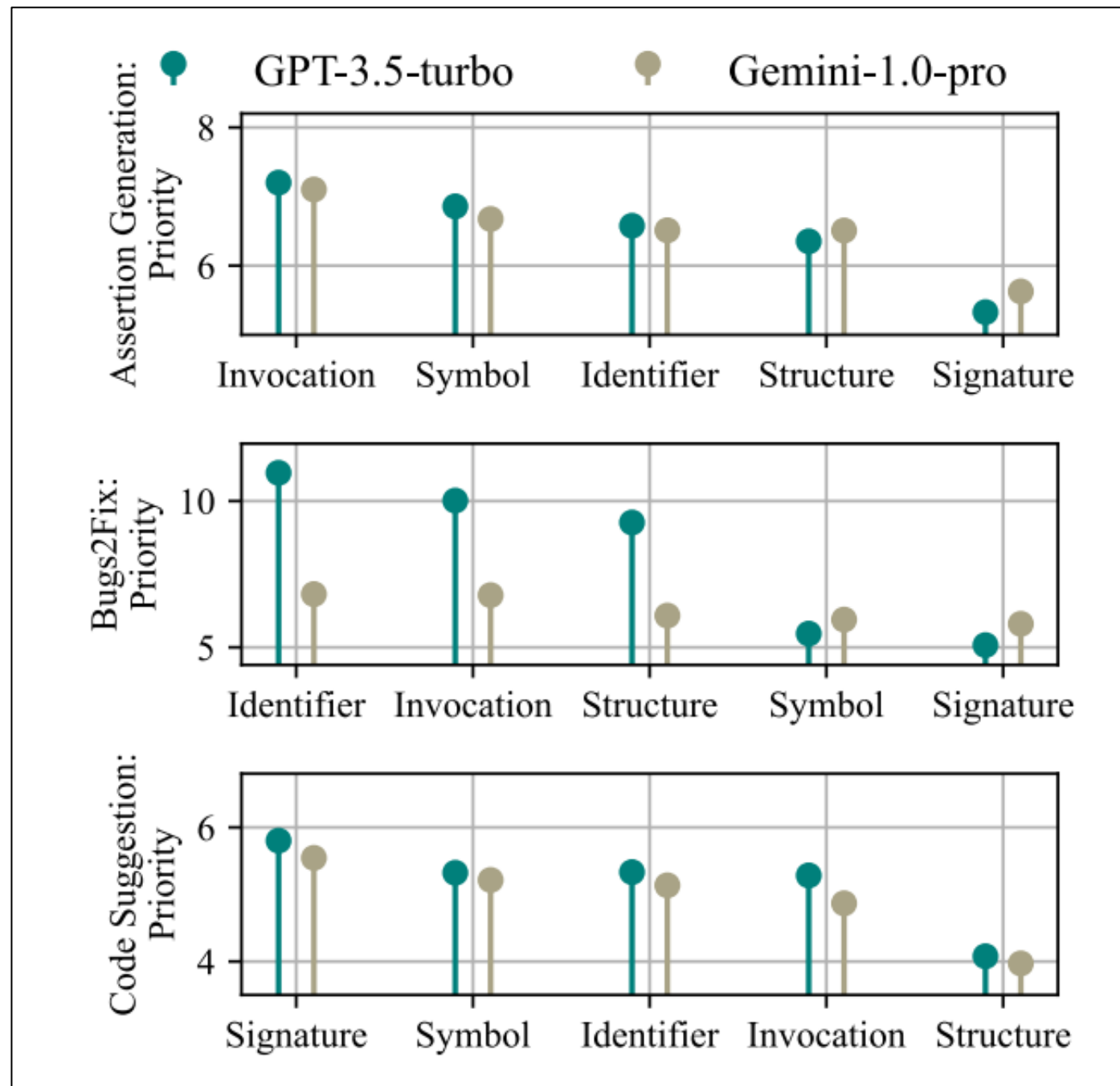
3種のコード生成系Javaデータセット

タスク (データセット)	クエリ内容	出力内容	評価指標
テスト生成 (Assertion Generation)	テスト対象メソッド +途中までのテスト	assert文	Exact Match
バグ修正 (Bugs2Fix)	バグを含むメソッド	修正版コード	CodeBLEU
コード提案 (Code Suggestion)	メソッドヘッダ または関数概要	関数本体コード	CodeBLEU

- これらのタスクでの性能低下で圧縮すべきトークンを決定
- 多くを削れた上で性能低下が少ないものが優先

各トークンの圧縮優先順位

タスクごとの圧縮優先順位



タスクごとにどのトークンの圧縮の優先度が高いかが異なる

ただし、同じタスクなら
モデルが違ってても
同じ優先順位に

モデルによる学習

圧縮手順を学習したモデル作成

- 求まった優先度順に指定の割合まで圧縮していったコードでデータセットを作成する
- 元コードと圧縮後コードのデータセットでCodeT5モデルを訓練

- ASTで解析⇒優先度順に圧縮という手順は完全な構文、文法のコードのみに適用できる
- 実際は不完全なコードが多くそれらに適用できないのは問題

⇒ 圧縮器として訓練することで

不完全なコードにも適用できるように

実験内容

3種のコード生成系タスク

タスク	クエリ内容	出力内容	評価指標
テスト生成 (Assertion Generation)	テスト対象メソッド +途中までのテスト	assert文	Exact Match
バグ修正 (Bugs2Fix)	バグを含むメソッド	修正版コード	CodeBLEU
コード提案 (Code Suggestion)	メソッドヘッダ または関数概要	関数本体コード	CodeBLEU

- ・ 圧縮なし、既存手法(LLMLingua, RECOMP)、提案手法で比較
- ・ 圧縮率及び各タスクの評価指標で評価
- ・ モデル:GPT-3.5-Turbo

実験結果

テスト生成(Assertion Generation)

手法	トークン数	圧縮率(%)	精度(%) (Exact Match)
LLMLingua	482	22.9	33.8
LongLLMLingua	474	24.2	34.1
RECOMP	465	25.6	23.4
CODEPROMPTZIP	<u>440</u>	<u>29.7</u>	42.1
圧縮なし	626	0	<u>50.5</u>

- ・ 圧縮による性能低下はあるが、最も効率的な圧縮

実験結果

バグ修正(Bugs2Fix)

手法	トークン数	圧縮率(%)	精度(%) (CodeBLEU)
LLMLingua	286	20.9	41.9
LongLLMLingua	287	20.6	42.1
RECOMP	268	25.9	45.3
CODEPROMPTZIP	<u>262</u>	<u>27.4</u>	61.9
圧縮なし	362	0	<u>81.4</u>

- ・ 効率的な圧縮はできているが、圧縮なしとの差が大きい

実験結果

コード提案(Code Suggestion)

手法	トークン数	圧縮率(%)	精度(%) (CodeBLEU)
LLMLingua	125	25.1	21.8
LongLLMLingua	126	24.1	21.2
RECOMP	132	20.9	21.0
CODEPROMPTZIP	<u>121</u>	<u>27.5</u>	23.7
圧縮なし	167	0	<u>24.7</u>

- 効率的な圧縮、性能低下も小さめ

結論と限界

CODEPROMPTZIPの性能

- プロンプトのコード例部分を圧縮するCODEPROMPTZIP
- 性能低下を抑えながら効率的に圧縮可能
- コード例を含むならどのプロンプトにも適用可能

CODEPROMPTZIPの限界

- Javaのメソッドレベルでのコード生成のみでの実験
- Pythonのような別言語での実験やリポジトリレベルの実験不足
- タスクごとに各トークン分類の圧縮優先度が異なるため
新しいタスクにおいては追加学習が必要

所感

自分の研究の実験に採用できそうな手法

- 自分の研究：コード検索のコスト削減手法の分析
- そのコスト削減手法の一つに取り入れられるかも
 - 「コード例を含むどのプロンプトにも適用可能」らしい
 - モデルの訓練ができるかどうか