

On the Compression of Language Models for Code: An Empirical Study on CodeBERT

Giordano d'Aloisio
DISIM Department
University of L'Aquila
L'Aquila, Italy
giordano.daloisio@univaq.it

Luca Traini
DISIM Department
University of L'Aquila
L'Aquila, Italy
luca.traini@univaq.it

Federica Sarro
Department of Computer
Science
University College London
London, UK
f.sarro@ucl.ac.uk

Antinisca Di Marco
DISIM Department
University of L'Aquila
L'Aquila, Italy
antinisca.dimarco@univaq.it

発表者: 福井 遥陽

背景

言語モデルは幅広いソフトウェア工学タスクで利用されているが、大きな計算コストが実用化の障害となっている。

これに対処するために、圧縮手法を適用しモデルの効率性を改善しようとしている。

→これらの手法は推論時間の遅延やメモリ使用量の最適化を目指す
が、モデル性能の低下を伴う。（トレードオフ）

しかし、実際にこれらの手法がどのように影響するかは、
理解に大きなギャップが存在する。

目的 | アプローチ

本研究では、3種類のSEタスク-脆弱性検出、コード要約、コード検索-に対して異なるモデル圧縮手法が与える影響を調査することを目的とする。

CodeBERTを3種類のタスク向けにファインチューニングし、知識蒸留、量子化、プルーニングという3つの圧縮戦略が

- (1) タスク遂行における効果
- (2) 推論時間 (遅延)
- (3) モデルのメモリサイズ

にどんな影響を与えるかを評価する。

アプローチ

脆弱性検出

言語モデルにコード関数を入力し，その関数にセキュリティ上の脆弱性が含まれているかどうかを予測させる．モデルは，コードが脆弱か否かを支援す二値ラベルを出力．

今回は，Devinデータセット（「脆弱有り/なし」のラベル付きの27318個のC関数）を用いてCodeBERTをファインチューニングし，評価した．

アプローチ

コード要約

コードスニペットに対して自然言語による説明を自動生成するタスクである。言語モデルにはコード関数が与えられ、その要約文を生成する。

今回は、Seq2Seqモデルを採用した。このモデルはCodeBERTをエンコーダ層とし、6層のとらんすトランスフォーマをデコーダ層とする構成である。

学習と評価には、CodeSerchNetデータセット（javaの181061件のコード・コメントペア）を用いた。

アプローチ

| コード検索

自然言語文を入力として、意味的に関連するコードスニペットを検索する。まず、コメントとコードスニペットをそれぞれ埋め込み表現に変換し、そのない席に基づく類似どっでランキングを行う。

本研究では、CodeSerchNetデータセット（280634件のコード・コメントペア）を用いてCodeBERTをファインチューニングしている。

言語モデルの圧縮戦略

知識蒸留

大規模なモデルの挙動を小規模モデルに模倣させる技術。この戦略により、より少ないメモリで高速な推論が可能となる。

量子化

モデルの重みの精度を下げることで圧縮する手法。例として、32ビット浮動小数点を8ビット整数に変換するなど。

プルーニング

言語モデルの効果にあまり寄与しないと判断される重みを削除することで効率化する手法

評価指標

評価には効果と効率の両方で測定した。

1) 効果

- 脆弱性検出

- Accuracy: 正しく分類された関数の割合
- F1-score: 精度と再現率の調和平均
- MCC: 正例・負例のバランスを考慮した指標

- コード要約

- BLEU : 生成された要約と参照要約のn-gram一致度を測定
- BERTScore : 事前学習済みBERTを用いて、生成要約と参照要約の意味的類似度を測定
- SIDE : 意味的適切性を重視した要約品質評価指標

評価指標

- コード検索

- MRR : 正解コードスニペットがランキング上でどの位置に出現するかを評価
- MRR@1, MRR@5 : 上位1件および5件以内での正解出現率を測定

2) 効率

- 推論時間

- CPUとGPUの両方で測定
- 各モデルに対してバッチサイズ1でテストセットを処理し, 1サンプルあたりの平均推論時間を算出

- モデルサイズ

- モデルの保存ファイルのサイズを測定

実験結果

RQ1：圧縮戦略は脆弱性検出にどのような影響を与えるか？

知識蒸留	あり	なし
Accuracy	-2.2%	0.6
F1	+3.1%	0.541
MCC	-10.1%	0.247
推論時間(CPU)	-39.8%	15.902(sec)
モデルサイズ	-48.8%	499(MB)

量子化	あり	なし
Accuracy	-0.0%	0.63
F1	-1.1%	0.541
MCC	+0.4%	0.247
推論時間(CPU)	+41.9%	15.902(sec)
モデルサイズ	-51.4%	499(MB)

プルーニング	あり	なし
Accuracy	-4.4%	0.63
F1	-41.0%	0.541
MCC	-11.3%	0.247
推論時間(CPU)	+15.5%	15.902(sec)
モデルサイズ	0.0%	499(MB)

実験結果

RQ2：圧縮戦略はコード要約にどのような影響を与えるか？

知識蒸留	あり	なし	量子化	あり	なし
BLEU	-42.3%	18.791	BLEU	+0.4%	18.791
BERTScore	-6.1%	0.888	BERTScore	0.0%	0.888
SIDE	-70.6%	0.871	SIDE	0.1%	0.871
推論時間(CPU)	-39.8%	157.369(sec)	推論時間(CPU)	-20.3%	157.369(sec)
モデルサイズ	-33%	707(MB)	モデルサイズ	-42%	707(MB)

プルーニング	あり	なし
BLEU	-4.3%	18.791
BERTScore	-0.1%	0.888
SIDE	+0.4%	0.871
推論時間(CPU)	-45.3%	157.369(sec)
モデルサイズ	0.0%	707(MB)

実験結果

RQ3：圧縮戦略はコード検索にどのような影響を与えるか？

知識蒸留	あり	なし	量子化	あり	なし
MRR	-52.4%	0.329	MRR	-0.2%	0.329
MRR@1	-57.7%	0.242	MRR@1	-0.0%	0.242
MRR@5	-54.3%	0.310	MRR@5	-0.3%	0.310
推論時間(CPU)	-84.7%	6.047(sec)	推論時間(CPU)	+34.2%	6.047(sec)
モデルサイズ	-48.7%	499(MB)	モデルサイズ	-51.4%	499(MB)

プルーニング	あり	なし
MRR	-3.2%	0.329
MRR@1	-3.4%	0.242
MRR@5	-3.4%	0.310
推論時間(CPU)	+5.5%	6.047(sec)
モデルサイズ	0.0%	499(MB)

考察

本研究によって量子化が最も堅牢な圧縮手法であることが確認された。

全てのタスクにおいて効果をほぼ維持しながら効率性を大幅に改善した。

- ・ 知識蒸留は推論時間やモデルサイズの削減に効果的だが、特にコード検索タスクにおいては効果がわずかに低下した。
- ・ プルーニングは削減率が高い場合にせいのうていかを招くため、適用範囲を慎重に見極める必要がある。