

To What Extent do Deep Learning-based Code Recommenders Generate Predictions by Cloning Code from the Training Set?

Matteo Ciniselli, Luca Pascarella, Gabriele Bavota

19th International Conference on Mining Software Repositories(MSR2022)

発表者：松尾 和輝

背景

- コード補完ツール

- コードの一部やコメントから、続くコードを補完してくれるツール

例) GitHub Copilot^[10]

- 開発者に次に書くべきコードを推奨することにより、実装作業を大幅にスピードアップすることが可能^[17, 18]
- GitHubに含まれる大量のコードを用いて学習
 - コードの使用はライセンスによって規制されている

[10] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374 (2021).

[17] Sangmok Han, David R Wallace, and Robert C Miller. 2009. Code completion from abbreviated input. In 2009 IEEE/ACM International Conference on Automated Software Engineering. IEEE, 332–343.

[18] Sangmok Han, David R Wallace, and Robert C Miller. 2011. Code completion of multiple keywords from abbreviated input. Automated Software Engineering 18, 3-4 (2011), 363–398.

コード補完ツールの問題

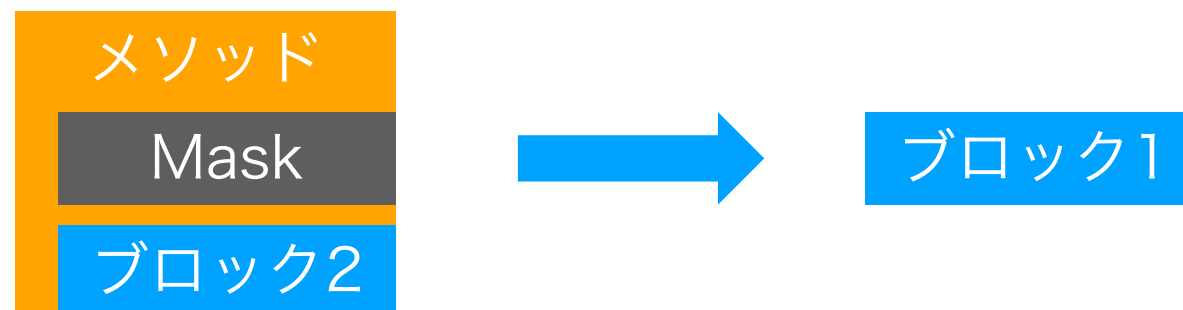
- ① 学習データに、ライセンスされたコードを使用できるのか
- ② 学習したモデルで生成されたコードは、学習データのクローンなのか、そうでないのか
が不明
- ②に関して、クローンを生成する場合、ライセンス違反になる可能性がある

目的とアプローチ

- 目的
 - コード補完ツールが、学習に使用したデータからコードのクローンを生成する傾向がどの程度あるのかを調査する
- アプローチ
 - コード補完ツールが生成するコードと、学習に使用したコードの一致率を調査

機械学習モデル

- GitHub Copilotは学習データが公開されていないため使用不可
 - 新しいモデルを用意する必要がある
- Ciniselliら^[12]が使用した事前学習済みT5モデルを使用
 - 新しいデータセットでファインチューニング
 - 一部がマスクされたメソッドから、マスクされた部分を予測するモデルを構築



[12] Matteo Ciniselli, Nathan Cooper, Luca Pascarella, Antonio Mastropaolo, Emad Aghajani, Denys Poshyvanyk, Massimiliano Di Penta, and Gabriele Bavota. 2022. An Empirical Study on the Usage of Transformer Models for Code Completion. IEEE Transactions on Software Engineering (2022).

データセット (1/3)

- GitHubリポジトリ, GitHub Androidリポジトリから, 約3.7MのJavaメソッドを収集
- 以下の条件に合うメソッドを除外
 - 3行未満のコードを持つメソッド
 - テストメソッド
 - 非ASCII文字 (e.g. 中国語テキスト) を含むメソッド
 - Ciniselliらによって使用された事前学習データセットで重複するメソッド

841,318件のメソッドを収集

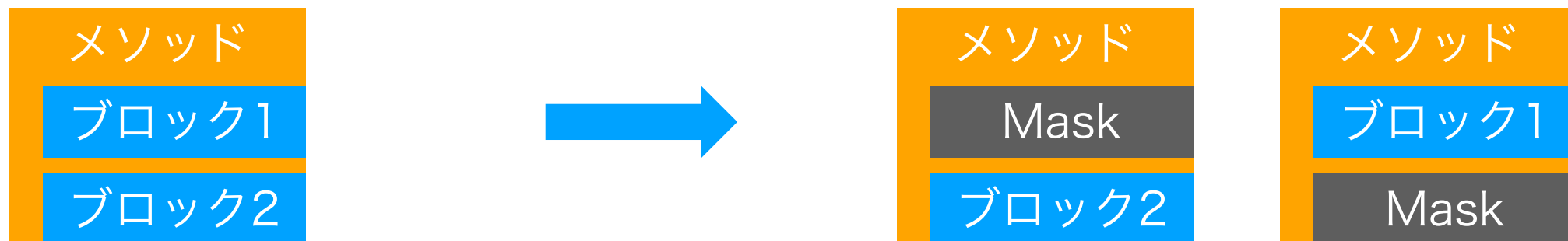
データセット (2/3)

- データセット内の重複するメソッドを除外
 - 2つのメソッド間の重複したトークンの割合を計算、割合が閾値を超えた場合にメソッドを除外（両方？片方？）
 - 4種類の閾値で計算し、データセットを作成（下表）
 - 重複するものを多く含むデータセットは、殆ど重複しないデータセットに比べて、モデルが学習用データからより多く「コピー」するようになるのでは
 - Set Th. : トークンの重複なし
 - Multiset Th. : トークンの重複あり

Name	Set Th.	Multiset Th.	# of Methods
Very Weak	0.8	0.7	838k
Weak	0.6	0.5	711k
Strong	0.4	0.3	483k
Very Strong	0.3	0.2	273k

データセット (3/3)

- n個のブロックを持つメソッドmから, 1つのブロックをマスクしたメソッドをn個だけ生成



- それぞれで, 学習用データセットを作成
 - 評価, テスト用データセットは, 種類に関係なく作成

Filtering Level	Dataset	#Instances	#Distinct Methods
Very Weak	Training	1.2M	797k
Weak		1.02M	671k
Strong		703k	446k
Very Strong		390k	240k
All levels	Evaluation	15,783	8,560
	Test	15,742	8,506

データ分析

- 4つのモデルをトレーニングした後、4つのモデルすべてについて同じテストセットでテスト
 - 生成されたコードが、学習用データセットのクローンであるかを調査
 - クローン検出ツール, Simian clone detector^[19]を使用
 - クローン度合いを計算
 - Type 1 : 完全一致 (コメント, 空白等除く)
 - Type 2 : 型や識別子以外一致 (コメント, 空白等除く)
 - 未知トークンを生成した6,986個のメソッドを分析対象から除外し, のこった8,756個のメソッドについて分析

[19] Simon Harris. [n.d.]. Simian <https://www.harukizaemon.com/simian>.

結果

- クローン率
 - 生成されるコードの行数が多いほど，クローンは生成されにくい（データセットの種類に関わらず）
 - GitHub Copilotのようなコード補完ツールは，コードのクローンを生成する可能性は極めて低い

		>=2 Lines Clones		>=3 Lines Clones		>=4 Lines Clones		>=5 Lines Clones	
	Very Strong	8.2%	83.2%	0.9%	30.3%	0.2%	11.8%	0.0%	2.3%
	Strong	10.3%	84.6%	1.6%	33.9%	0.3%	12.5%	0.0%	2.6%
	Weak	12.5%	83.5%	2.3%	35.1%	0.6%	14.4%	0.1%	3.2%
	Very Weak	12.1%	83.5%	1.9%	31.9%	0.4%	12.0%	0.1%	2.5%
		Type 1	Type 2	Type 1	Type 2	Type 1	Type 2	Type 1	Type 2

Type-1：完全一致（コメント，空白等除く）
Type-2：型や識別子以外一致（コメント，空白等除く）

結果

- クローンの出所
 - Pretrain, Finetuningで使われた学習用データセットの両方をもとに生成されている（両方に類似するコードがある）
 - 両方のトレーニングフェーズで繰り返し見られるコードが、生成された予測に影響を与える可能性が高いことを示している

Very Strong	12.9%	3.4%	17.9%	2.2%	69.2%	94.4%
Strong	9.4%	2.4%	23.7%	3.1%	66.9%	94.5%
Weak	6.8%	2.3%	28.2%	4.0%	65.0%	93.7%
Very Weak	5.8%	2.3%	33.9%	4.1%	60.3%	93.6%
	Type 1 Pretrain	Type 2 Pretrain	Type 1 Finetuning	Type 2 Finetuning	Type 1 Both	Type 2 Both

結果

- 正しい予測と間違った予測
 - 正しい予測値には、トレーニングデータセットに存在するコードのクローンが含まれる可能性が高い
 - トレーニングデータセットとテストデータセットの重複を避けるために、適切にデータセットクリーニングを行う必要がある

Very Strong	12.3%	5.4%	87.7%	94.6%
Strong	11.1%	5.8%	88.9%	94.2%
Weak	12.0%	6.5%	88.0%	93.5%
Very Weak	13.3%	6.7%	86.7%	93.3%
	Type 1 CP	Type 2 CP	Type 1 WP	Type 2 WP

結果

- 出力例

```
/* cloned code */
errorFragment = new ErrorFragment();
Bundle msg = new Bundle();
msg.putString("msg", ex.getMessage());
errorFragment.setArguments(msg);
getSupportFragmentManager().beginTransaction()
    .replace(R.id.message, errorFragment).commit();

/* expected prediction */
{
errorFragment = new ErrorFragment();
Bundle msg = new Bundle();
msg.putString("msg", "No or poor internet connection.");
errorFragment.setArguments(msg);
getSupportFragmentManager().beginTransaction()
    .replace(R.id.message, errorFragment).commit();
}
```

```
/* cloned code */
if (s != null)
{
    try {
        return Integer.parseInt(s);
    } catch (NumberFormatException e) {
    }
}
return null;

/* expected prediction */
{
    Integer interval =
        OptionAdvanceParser.DEFAULT_INTERVAL;
    if (optionSet.hasArgument("interval")) {
        interval = (Integer) (optionSet.valueOf("interval"));
        if (interval < 1) {
            throw new IllegalArgumentException("Interval
                cannot be set below 1.0");
        }
    }
    return interval;
}
```

間違った予測の例

まとめ

- 目的
 - コード補完ツールが、学習に使用したデータからコードのクローンを生成する傾向がどの程度あるのかを調査する
- アプローチ
 - コード補完ツールが生成するコードと、学習に使用したコードの一致率を調査
- 結果
 - 生成されるコードの行数が多いほど、クローンは生成されにくい
 - GitHub Copilotのようなコード補完ツールは、コードのクローンを生成する可能性は極めて低い

所感

- 意味的に同じコードがどれだけの精度で生成できるのか気になった
 - クローンを生成しないならなんでもいいという問題ではない
- 色々説明が足りてない（気がする） . . .
 - ちゃんと読めてないだけ？