DEIM Forum 2023 1b-5-5

大規模言語モデルへの追加事前学習による誤り訂正モデルのコードへの適用

相馬　菜生† 　梶浦　照乃†† 　高橋　舞衣† 　倉光　君郎†

† 日本女子大学理学部数物情報科学科　〒 112–8681 東京都文京区目白台 2-8-1
†† 日本女子大学大学院理学研究科数理・物性構造科学専攻　〒 112–8681 東京都文京区目白台 2-8-1
E-mail: †{m1916045sn,m1816023kt,m1916046tm}@ug.jwu.ac.jp, ††kuramitsuk@fc.jwu.ac.jp

あらまし　誤り訂正モデルとは，誤りのあるテキストと誤りのないテキストを用意し，誤りの特徴を学ばせることで正解を予測するためのモデルである．このモデルは自然言語からソースコードまで様々な対象に応用可能である．本研究は，初学者の書いたコードを主な対象として，誤り訂正を行うことを目指す．誤り訂正モデルの精度向上のため，大規模な自然言語コーパスで構築された言語モデルに加え，ソースコードの追加事前学習を加える点が特徴である．本発表では，multilingual T5 にコードの構文やタイプミスなどの誤りの特徴を追加で学習させることでモデルを構築し，精度の向上を定量的に評価する．

キーワード　機械学習，誤り訂正，大規模言語モデル，事前学習

1　はじめに

近年，大規模言語モデルは文章生成，翻訳，誤り訂正など幅広い自然言語処理タスクに活用されている．2018 年に Google の Jacob Devlin らは様々なタスクに対し，柔軟な対応が可能な自然言語処理モデル BERT を発表した [1]．さらに，2020 年には 101 種類の自然言語で事前学習されたマルチタスクに適用するモデル mT5 が発表され，自然言語処理タスクへ大きな影響を与えている [2]．コードにおいても例外ではなく，ソフトウェア工学分野のコード修正やコード翻訳など様々なタスクへの応用が盛んである．コード修正の原理は自然言語処理タスクである誤り訂正と本質的には同一であり，誤りを含むテキストから正しい答えを予測するものとなっている．また近年では，AI ベースのコード修正技術も盛んに研究が行われている [3]．本研究は，我々が現在開発中のプログラミング学習支援システム KOGI [4] にコード修正の機能を取り入れることを目標としている．KOGI は，Google Colaboratotory 上で動くシステムで．現在コード翻訳，対話などさまざまな機能が搭載されているが，更にコード修正が可能になれば初学者の学習の効率化が期待できる．図 1 は，本研究のコード修正の成果を取り入れた動作例を表している．まず，エラーが発生すると KOGI が出現する．そこで「直してみて」と KOGI にお願いすることで正しいコードが出力される．初学者の中にはエラーをどう直せば良いのかわからず諦めてしまう人も多く見受けられるため，このような訂正機能をつけることで学習意欲を持続させることができるはずである．本研究では，KOGI にコード修正を取り入れるため，KOGI で用いられている大規模言語モデル mT5 をベースにコード修正を試みる．mT5 とは Google が開発した多言語版の Transformer ベースの大規模言語モデルであり，多言語テキストデータ mC4 で事前学習されている．しかしながら，

図 1　プログラミング学習支援システム KOGI のコード修正の動作例

mC4 は英語や日本語などの自然言語のデータであり Python は含まれていないため，コードが対象となるコード修正には適さないと考えられる．そこで，本研究では，mT5 に python コードを追加で事前学習しモデルを構築する．このモデルのことを今後，mPyT5(mT5+Python) と呼ぶこととする．また，本研究では，コードにノイズを加えて破壊し得られたデータを使って追加事前学習を行う方法を提案する．

本論文の残りの構成は以下の通りである．2 節では，本研究のベースとなっている大規模言語モデルについて説明する．3 節では，自然言語の誤り訂正とそのコード修正モデルへの応用について説明し，コード修正モデルの構築する際の問題点を提示する． 4 節では，提案手法である追加事前学習について述べる．追加事前学習としてこれまで使われてきた MLM と MSP について触れた後，今回新たに提案する BI 法について説明する．5 節では，従来手法も含めて評価実験を行い，BI 法により事前学習を行う mPyT5 の性能を評価する．6 節で本論文をまとめる．

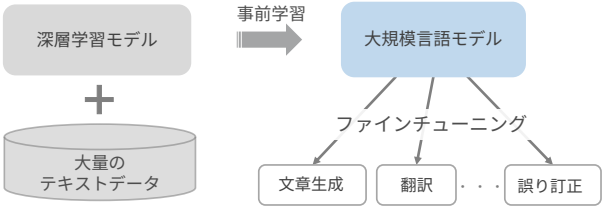

図 2　大規模言語モデルの構築とタスクへの適用

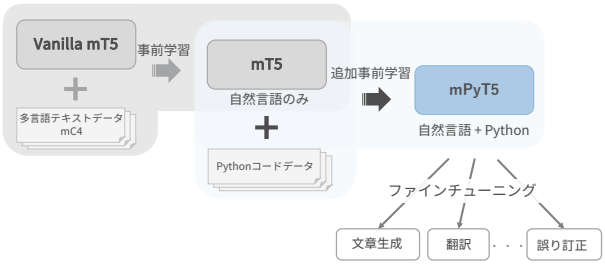

図 3　mPyT5 の構築方法

2　大規模言語モデル

本研究の提案の根幹となる大規模言語モデルについて概説する．大規模言語モデルは，大量のテキストデータを学習し，言語を理解できるようになるよう訓練されたモデルである．文章生成，翻訳，誤り訂正など幅広い自然言語処理タスクに活用されている．図 2 は事前学習による大規模言語モデルの構築とタスクへのファインチューニングを示したものである．大規模言語モデルは，深層学習モデルに大量のテキストデータを自己教師あり学習させることで構築された汎用言語モデルである．この言語モデルを，文章生成や翻訳といった自然言語処理タスクにファインチューニングすることで，それぞれのタスクに適したモデルが構築できる．事前に文脈的に自然な単語の並びを理解する目的で学習をしているため，少量のデータでも高い精度でタスクを解くことができる．大規模言語モデルとしては，mT5，CodeT5 [5] などが挙げられる．

mT5(multilingual T5) は， Google が開発した多言語版 T5 の事前学習済み言語モデルである．mT5 では，英語，日本語を含む 101 言語の多言語テキストデータセット mC4(multilingual Colossal Clean Crawled Corpus) を学習しているが，プログラミング言語に関しては学習していない．CodeT5 は，T5 をベースとした事前学習済みプログラミング言語モデルである．自然言語で学習されていた mT5 に対し，CodeT5 はソースコードで事前学習されているが日本語などの自然言語は学習していない．ソースコードのデータセットは，8 つのプログラミング言語 (Python， Java，JavaScript，PHP，Ruby，Go，C および C #) から成る．

なお，mT5，CodeT5 ともにモデルのアーキテクチャは全て Transformer ベースの Encoder-Decoder モデルとなっている．

3　誤り訂正モデルとコード修正モデル

自然言語の誤り訂正は，様々な分野で積極的に活用されており，現代社会にとって重要なものとなっている．本節では，誤り訂正モデルとそのコード修正への応用について現状と課題をまとめる．

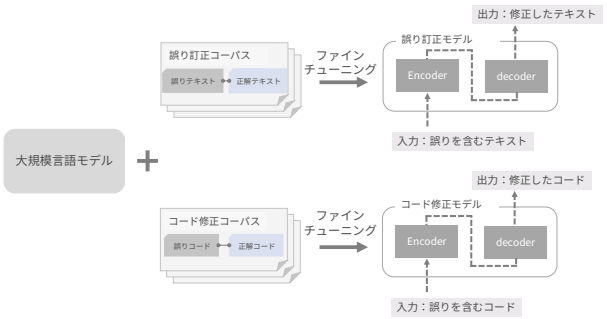

図 4　誤り訂正モデルとコード修正モデル

誤り訂正モデルとは，誤りのあるテキストと誤りのない正解テキストを用意し，テキストの文法の間違いやスペルミスなどの誤りの特徴を学ばせることで，正解を予測するモデルである．最近では，誤り訂正に関して，BERT などの深層学習モデルを用いた研究も盛んに行われている．一方，コード修正モデルは，誤り訂正モデルの枠組みを活用し，テキストの代わりにコードを用いる．コードの構文やタイポなどの誤りの特徴を学習させることで，誤りを含むコードを修正し，正しいコードを生成するモデルである．図 4 に示すように，誤り訂正モデルの構築方法とコード修正モデルの構築方法の基本的な原理は同じである．コード修正の場合には，深層学習モデルに誤りを含むコードと正解コードが対となったコード修正コーパスを学習させることでコード修正タスクに対応したモデルを構築する．この際，ファインチューニングで用いるコード修正コーパスは，コードの誤り部分を人手で修正し作成する必要がある．よって，コード修正モデルをファインチューニングのみを用いて構築する場合の課題として，作成可能なコーパスの量に限りがあることが挙げられる．

4　追加事前学習

近年，BERT や mT5，CodeT5 な大規模言語モデルに関する研究は積極的に進められ，新しいモデルが発表されている．事前学習の手法である自己教師あり学習は，人手によるアノテーションの必要がない．そのため大量のデータの学習が可能である．我々は，自然言語が事前学習済みのモデルに，追加で事前にコードを学習させることで mPyT5 を構築したのち，ファインチューニングでコード修正モデルを構築し，精度向上を目指す．図 3 は mPyT5 の構築方法を示している．mPyT5 は，mT5 に Python 言語モデルを追加で学習しており，日本語と Python の両方に対応した事前学習済み言語モデルである [6].

本研究では mPyT5 構築時に，事前学習済みモデル mT5 に自己教師あり学習を用いてコード修正向けの Python の言語モデルを追加で事前に学習する手法を新たに提案する．今回，学習させる Python コードデータとしては，以下のデータセットを使用した．

- CodeSearchNet [7]：6 つのプログラミング言語で書かれた関数のコードが含まれているデータセット．本研究では Python コードのみ抽出し扱った．
- Kaggle：データサイエンスのコンペティションプラットフォームである Kaggle が提供するデータサイエンスのデータセット．
- CoNaLa [8]: カーネギーメロン大学が公開する StackOverflow から機械的に収集された Python のデータセット．

追加事前学習に用いる事前学習方法を以下にまとめる．Masked Language Modeling(4.1 節) と Masked Span Prediction(4.2 節) は従来から知られている方法である．今回，我々は新たに BI 法 (4.3 節) を提案する．

4.1 Masked Language Modeling

Masked Language Modeling(MLM) とは，一部のトークンをマスクに置き換え，元の文を復元することで，トークン間の関係やテキスト全体を理解する手法である．BERT の学習方法としても知られており，様々なモデルの事前学習の方法として使用されている．評価実験では，文献 [9] と同様の方法で，mT5 に Python 言語を学習させる． 学習させる Python コードの 15%のトークンをランダムにマスクした入力テキストから，元の完全な文である出力テキストを予測するように学習する．入力と出力の例は以下の通りである．以下，<extra_id_0>, <extra_id_1>... はマスクの箇所を表す特殊トークンである．

元のテキスト `train.head()`
入力テキスト `<extra_id_0>.head<extra_id_1>`
出力テキスト `train.head()`

4.2 Masked Span Prediction

Masked Span prediction(MSP) とは，MLM ではトークンをマスクに置き換えたのに対し，任意の長さの連続した複数トークンをひとかたまりとしてランダムにマスクし，元の入力テキストを復元する方法である．CodeT5 の事前学習方法の一つでもある．MLM では出力テキストが元の完全な文であるのに対し，MSP では入力でマスクされたトークンを特殊トークンで繋いだテキストである．入力と出力の例は以下の通りである．MLM と同様に<extra_id_0>, <extra_id_1>... はマスクの箇所を表す特殊トークンである．また，改行は<nl>に置き換えた．

元のテキスト `y_head=sigmoid(0)<nl>y_head`
入力テキスト `y_head=sigmoid(<extra_id_0><nl>y_head`
出力テキスト `<extra_id_0>0)<extra_id_1>`

4.3 BI 法

データにノイズを加えて取り除くタスクで学習させる方法であり，Break-It-Fix-It [10] の方法や DAE(Denoising Auto-Encoder) を元に，コードにノイズを加えることで破壊し，元のコードを復元する方法でタイポなど誤りの特徴を事前に追加的に学習させる方法である．表 1 はコードの破壊方法の例である．本研究では，学生のエラーをもとに初学者が間違えやすい括弧の不足やタイポなどを破壊の方法として取り入れた．また，コードの破壊する割合を変え，BI25，BI50，BI75，BI90 の 4 つのパターンのモデルを構築した．BI50 では，50%の頻度で変数名を破壊し，5%の頻度で文字の削除，5%で”,”と”.”の入れ替えを行う．それに対し，BI90 では，90%の頻度で変数名を破壊する．実際破壊したコードの例は表 2 である．BI90 では元の正しい文が予測できないほど破壊している．

表 1　BI 法での破壊例

破壊の方法	正しいコード	破壊したコード
, . の入れ替え	pd.read_csv(A)	pd,read_csv(A)
: の削除	for i in range(n):	for i in range(n)
()[]{}<tab>などの文字欠け	print(”A”)	print(”A”
typo：打ち忘れ	Hrello World!	Hllo World!
typo：入れ替え	Hello World!	Ehllo World!
typo：打ち間違え (近い位置)	Hello World!	Hsllo World!
typo：文字過多	Hello World!	Hrello World!

5 実　　験

2 節では，mT5，mPyT5，CodeT5 という 3 つの深層学習モデルについて説明した．更に，4 節では mPyT5 における事前学習方法として BI 法を含む 3 つの手法について説明した．本節では，これらの深層学習モデルにそれぞれファインチューニングによりコード修正コーパスを学習させコード修正モデルを構築し，評価実験を行う．mPyT5 の構築は，エポック数を 5 とし，学習を行った．

5.1 コード修正コーパスと前処理

コード修正コーパスとは，誤りコードと正しいコードが対となったコーパスである．本実験で使用するコーパスは以下の 2 つが挙げられる．

- ErrorFix：実際に発生したエラーから取得した誤りコードと，人手で誤りを修正した正しいコードが対となる自作のコーパス．誤りコードは競技プログラミングのサイト Atcoder を用いたプログラミング入門講座やデータサイエンスの講義のログから約 9000 件のエラーを取得し，SyntaxError などの軽微な誤りを含むコードを抽出した．抽出した誤りコードの誤り部分を人手で修正し，正しいコードを生成することでコード修正コーパスを作成した．
- FixEval [11]： CodeNet [12] から提出された Java と Python のプログラムであり，競技プログラマーが評価のために異なるオンライン審査員に提出した難易度の異なるプログラムの集合体．本研究では，Python コードのみ抽出し誤りと正しいコードが対となるコーパスを作成した．

どちらも前処理として，改行コードやインデントは，特殊トークン<nl>や<tab> に置き換え，train データ，test データ，valid データを用意した．それぞれ準備したデータ件数は表 3

表 2　実際の破壊コード BI50/BI90 比較

正しいコード	破壊したコード (BI50)	破壊したコード (BI90)
Red<nl>Blue	ed<nl>Blue	Re<nl>NBlue
import re<nl>RawPurchaseAmount	import re<nl>RawPurchaseAmount	import de<nl>RawPutchaseAmount
y_head=sigmoid(0)<nl>y_head	g_head=sigmoid(0)<nl>_head	y_had=isgmoid(0)<nl>y_ead

の通りである．

表 3　データ件数 (件)

	train	test	valid
FixEval	7000	1500	1500
ErrorFix	1080	241	233

コード修正モデルの学習は，train データをバッチサイズ 16 で学習を行い，テストデータの検証ロスが最小になる epoch 数で止めた．

5.2 評価尺度

今回，比較に用いた評価尺度は次の通りである．

- 正解率 (EM)：正解テキストと予測テキストの完全一致率．
- 構文パス率：Python の構文解析器をパスした率．
- CodeBLEU [13]：自然言語の評価に適する BLEU に対し，コードの重要な構文的・意味的特徴を捉えた評価方法．
- Edit Sim [14]：レーベンシュタイン距離に基づく正解テキストと予測テキストの類似度を表す評価尺度．Python ライブラリ python-Levenshtein を使用．

なお，EM，BLEU，Edit Sim では，空白などのコードレイアウトの違いによる影響を取り除くため，Python コード整形器 Black を通してから評価を行っている．

5.3 実験結果と考察

表 4 は FixEval コーパスでの結果を，表 5 は ErrorFix コーパスでの結果をまとめたものである．

まず，mT5 と追加事前学習した 6 つのモデルを比較すると，精度の向上がみられた．ErrorFix コーパスの EditSim に着目した際，mPyT5 は，有意水準 α =0.05 にて mT5 を統計的に有意に上回っている．特に，ErrorFix コーパスの EM に着目すると，最も精度の高い mPyT5(BI75) は mT5 より 29.46 高い．他の評価尺度の結果に関しても，比較的 BI 法の精度が高いという結果が得られた．このことにより，軽微な誤りのコード修正タスクにおいて，3 つの事前学習方法の中で，コードを破壊して学習させる BI 法の有効性が確認できた．また，ErrorFix コーパスでは mPyT5(BI75) が EM の値が最も高かったことからも，誤りを含む率が高ければ高いほど精度が高くなるとは限らないことが明らかとなった．

追加事前学習した 6 つのモデルを比較した場合，どちらのコーパスでも MSP の精度が低い．これは，事前学習の際の入出力テキストが要因であると考えられる．BI 法と MLM に関しては，出力がマスクの埋められた完全なコードだが，MSP に関してはマスクが入った形となっているため，精度が思うように上がらないと考えられる．表 6 は ErrorFix コーパスにおいて追加事前学習を行なったモデルの中で EM が最も高い mPyT5(BI75) と最も低い mPyT5(MSP) の実際の予測結果を一部抜粋したものである．mPyT5(MSP) の実際の予測例から，入力がそのまま予測として出てしまっているものや，不要な文字が追加されコードの構文として間違いであるものが見受けられ，Python 言語モデルの学習が足りていないと考えられる．本研究での 5epoch の追加事前学習では不十分だった可能性もあり，今後学習の epoch 数を増やし十分に学習することで，MSP による Python モデルの学習が行き渡り精度が向上すると考えられる．

表 4　FixEval

	EM	構文パス率	CodeBLEU	Edit Sim
mT5	5.07	84.80	58.36	79.11
CodeT5	**6.13**	81.40	59.44	78.60
mPyT5(MLM)	5.60	85.40	**59.85**	79.18
mPyT5(MSP)	**6.13**	85.80	59.36	78.91
mPyT5(BI25)	5.20	85.93	59.78	79.25
mPyT5(BI50)	5.53	85.93	58.67	79.29
mPyT5(BI75)	5.53	85.73	59.44	**79.31**
mPyT5(BI90)	5.73	**86.33**	59.01	79.18

表 5　ErrorFix

	EM	構文パス率	CodeBLEU	Edit Sim
mT5	17.84	52.28	81.62	92.83
CodeT5	36.10	66.39	82.29	92.75
mPyT5(MLM)	42.74	**74.27**	85.19	94.43
mPyT5(MSP)	18.26	56.43	65.01	90.16
mPyT5(BI25)	43.98	73.86	85.46	94.25
mPyT5(BI50)	46.06	72.61	85.23	94.09
mPyT5(BI75)	**47.30**	72.61	**85.59**	94.39
mPyT5(BI90)	45.23	73.44	85.34	**94.52**

表 6　予測結果一部抜粋 (ErrorFix)

入力	出力	予測 (BI75)	予測 (MSP)
df,head(10)	df.head(10)	df.head(10)	df.head(10)
prnt(\"Hello World\")	print(\"Hello World\")	print(\"Hello World\")	#prnt(\"Hello World\")
ptint('0')	print('0')	print('0')	ptint('0')

6 むすびに

本論文では，誤りを含む Python コードを入力することで，誤りを修正し，正しい Python コードを生成するコード修正モデルを構築した．コード修正の精度向上に向け，追加事前学習方法として，自己教師あり学習を用いて精度の比較・評価を行なった．その結果，自己教師あり学習の有意性が示された．また，コードを機械的に破壊し学習を行う事前学習方法である BI

法は軽微な誤りであるコード修正に適していることが明らかとなった．特に ErrorFix コーパスでの実験結果より，初学者のエラーを修正する能力が高いことが確認できた．今後，学習方法を工夫し，自己教師あり学習の方法を掛け合わせることで，コード修正の精度向上を目指したい．

文　献

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[2] Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. mT5: A massively multilingual pre-trained text-to-text transformer. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 483–498, Online, June 2021. Association for Computational Linguistics.

[3] A. Kanade R. Gupta, S. Pal and S. Shevade. DeepFix: fixing common c language errors by deep learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[4] Momoka Obara, Yuka Akinobu, Teruno Kajiura, Shiho Takano, and Kimio Kuramitsu. A preliminary report on novice programming with natural language translation. In *IFIP WCCE 2022: World Conference on Computers in Education*, 2022.

[5] Yue Wang, Weishi Wang, Shafiq Joty, and Steven C. H. Hoi. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation, 2021.

[6] Miyu Sato Mai Takahashi Kimio Kuramitsu Teruno Kajiura, Nao Souma. An additional approach to pre-trained code model with multilingual natural languages. In *29th Asia-Pacific Software Engineering Conference (APSEC 2022)*, 2022.

[7] Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. CodeSearchNet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436*, 2019.

[8] Pengcheng Yin, Bowen Deng, Edgar Chen, Bogdan Vasilescu, and Graham Neubig. Learning to mine aligned code and natural language pairs from stack overflow. In *International Conference on Mining Software Repositories*, MSR, pages 476–486. ACM, 2018.

[9] 梶浦 照乃, 小原 百々雅, 秋信 有花, and 倉光 君郎. 多言語 t5 への追加事前学習による python 言語モデルの構築. In *The 6th cross-disciplinary Workshop on Computing Systems, Infrastructures, and Programming (xSIG2022)*, 2022.

[10] Michihiro Yasunaga and Percy Liang. Break-It-Fix-It: unsupervised learning for program repair. *arXiv preprint arXiv:2106.06600*, 2021.

[11] Ahmad Wasi U. Lourentzou Ismini Haque, Md M. and Chris Brown. FixEval: execution-based evaluation of program fixes for programming problems. *arXiv preprint arXiv:2206.07796*, 2022.

[12] Ruchir Puri, David S. Kung, Geert Janssen, Wei Zhang, Giacomo Domeniconi, Vladimir Zolotov, Julian Dolby, Mihir Choudhury Jie Chen, Lindsey Decker, Veronika Thost, Luca Buratti, Saurabh Pujar, Shyam Ramji, Ulrich Finkler, Susan Malaika, and Frederick Reiss. CodeNet: a large-scale ai for code dataset for learning a diversity of coding tasks. *arXiv preprint arXiv:2105.12655*, 2021.

[13] Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. Codebleu: a method for automatic evaluation of code synthesis. *CoRR*, abs/2009.10297, 2020.

[14] Li Yujian and Liu Bo. A normalized levenshtein distance metric. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29:1091–1095, 2007.