



Learning Program Representations with a Tree-Structured Transformer

Wenhan Wang, Kechi Zhang, Ge Li, Shangqing Liu, Anran Li, Zhi Jin, Yang Liu†



ソフトウェア工学のための深層学習(DL for SE)

- ソフトウェア工学アプリケーションの急速な発展

⇒膨大に増加したコード関連データを用いたプログラム理解

ソフトウェア工学のための深層学習

- コードクローン検出
- 脆弱性検出
- コード要約

などに応用, ソフトウェア開発を向上

既存研究

- プログラムの理解のための構造化ベースの技術

既存のDL for SEの種類	木構造ニューラルネットワーク 	グラフニューラルネットワーク 
特徴	ソースコードの構文木を学習モデルへ入力	プログラムをグラフへ畳み込み学習モデルへ入力
課題	コンテキスト情報を含んだノード表現を学習できない	ソースコードの長期的な依存関係を捉えられない
	一部の木構造, グラフベースで - プログラムの実行順序情報をとらえられない - 構文木を追加処理する必要がある場合が存在し, 木構造がオーバーヘッドすることで本来の構造情報を失ってしまう	

提案手法：Tree-Transformer

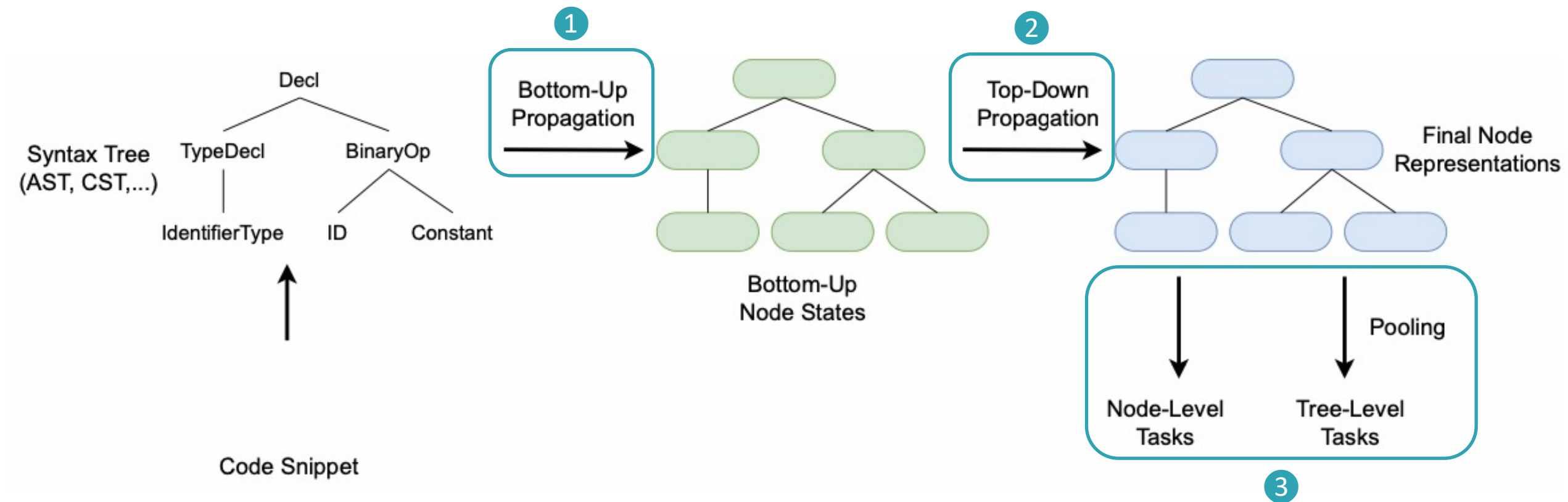


Fig. 1. The overall pipeline of Tree-Transformer.

比較するベースライン

- 木構造ニューラルネットワークベースモデル

Tree-LSTM, TBCNN, TreeCaps(variable-to-static Ver.), **Child-Sum Tree-LSTM**

- グラフニューラルネットワークベースモデル

グラフ畳み込みネットワーク(GCN), グラフ同型ネットワーク(GIN),
ゲートグラフニューラルネットワーク(GGNN)

- Transformerベースモデル

Tree-PE, **GRAT**, 逐次的なTransforemer, CodeBERT, C-BERT

Research Question

RQ1 : Tree-Transformerは構文木レベルの予測タスクでどのような性能を示すか

- プログラムの抽象構文木を入力し，実装されている機能で分類

	POJ	Java250	Python800	C++1000	C++1400
Train samples	36,400	45,000	144,000	300,000	252,000
Validation samples	5,200	15,000	48,000	100,000	84,000
Test samples	10,400	15,000	48,000	100,000	84,000
Avg. nodes	189.58	339.33	232.27	376.90	472.04
Avg. children per node	1.90	3.05	2.77	3.09	3.12
Avg. depth	13.32	17.26	14.48	15.46	16.43
Vocabulary	44	222	161	346	346

RQ1 : 構文木レベルの予測タスクでどのような性能を示すか

TABLE III
PROGRAM CLASSIFICATION ACCURACY(%) ON CODENET AND POJ DATASETS.

	Java250	Python800	C++1000	C++1400	Overall	POJ
GCN	89.06	91.81	93.54	92.89	91.83	93.93
GIN	90.76	93.17	95.54	94.50	93.49	95.76
GGNN	88.46	89.92	89.75	88.01	89.04	93.22
Tree-LSTM (root)	93.19	93.95	95.79	95.20	94.53	94.70
Tree-LSTM (attention)	93.71	93.83	95.79	95.24	94.64	94.95
TBCNN	90.32	91.10	93.17	93.03	91.91	94.15
TreeCaps	91.42	90.26	93.55	93.24	92.12	95.88
Tree-PE [19]	91.65	91.11	92.30	88.97	91.01	94.19
GREAT	93.15	93.30	93.46	93.72	93.41	92.64
GCN+Graph [29]	92.70	93.82	95.76	95.26	94.39	95.57
GIN+Graph [29]	93.26	94.17	96.34	95.95	94.93	95.96
GGNN+Graph	93.61	92.23	91.72	92.48	92.51	94.80
HPG+HGT [21] ³	93.95	94.99	-	-	-	-
Tree-Transformer (Ours)	95.32	95.30	97.11	96.98	96.18	96.12
Transformer	93.49	93.99	89.93	67.87	86.32	88.13
C-BERT [29]	97.40	97.09	93.79	91.83	95.03	-
CodeBERT	96.47	97.41	86.13	83.05	90.77	98.40

- 全てのデータセットにおいて
グラフ構造ベースライン
木構造ベースラインを上回る

⇒ 大規模な事前学習済み
Transformerと比較しても、
軽量な上競争力がある

Research Question

RQ2 : ノードレベルの予測タスクにおいてTree-Transformerはどのように機能するか

- 誤用された二項演算子の位置特定と修復

	Wrong Operator
Train samples	155,628
Validation samples	16,868
Test samples	86,231
Avg. nodes	222.39
Avg. children per node	2.84
Avg. depth	13.28
Vocabulary	286,456

RQ2 : ノードレベルの予測タスクでどのように機能するか

TABLE V

THE ACCURACY OF WRONG OPERATOR LOCALIZATION AND REPAIR.

Model	Loc Acc(%)	Loc & Rep Acc (%)
GCN	84.97	60.44
GIN	85.69	61.61
GGNN	84.77	58.71
Tree-PE [19]	78.65	53.99
GREAT	85.43	59.32
GCN+Graph	86.37	65.05
GIN+Graph	86.48	64.66
GGNN+Graph	84.11	61.01
Tree-Transformer (Ours)	88.26	68.58
Transformer	73.13	45.22
CodeBERT	94.61	83.89

- 特に, LocとRepの結合精度で優れた性能を達成
- CodeBERTは事前学習に800万以上のデータサンプルが必要
⇒大量のデータから自然さを捉えることに優れている

Research Question

RQ3 : Tree-Transformerの各部分はパフォーマンスにどのように寄与しているのか

- TypeScriptの識別子の型推論

	Type Inference
Train samples	608,156
Validation samples	24,424
Test samples	27,870
Avg. nodes	652.53
Avg. children per node	3.03
Avg. depth	17.09
Vocabulary	100,128 (manually set)

RQ3 : 各部分はパフォーマンスにどのように寄与しているのか

TABLE VII

ABLATION STUDY ON PROGRAM CLASSIFICATION (JAVA250) AND WRONG OPERATOR LOCALIZATION AND REPAIR.

Model	Java250	WrongOp	
		Loc	Loc & Rep
Tree-Transformer	95.32	88.26	68.58
-position encoding	94.99	85.78	63.32
-fraternal attention	94.66	86.26	63.91
-fraternal attention +position encoding	94.95	87.18	65.64
-top-down propagation	94.01	N/A	N/A

- トップダウン/ボトムアップ伝播の部分がどのタスクでも完結で強力
- 位置エンコーディングの部分は、異なるクラスのプログラムの区別に必須ではないが、演算子の誤用の特定には重要で、兄弟ノードattentionと併用すると効果的

考察

Tree-Transformerの長所

- 既存の逐次的な手法よりも占有するメモリを抑え、より大きな入力が可能

Tree-Transformerの短所

- 再起的に処理を行うため、実行時間は逐次的な手法より長くなる

妥当性への脅威

- コードを構文木へ変換する際のパース処理
 - ⇒ 広く利用されるオープンソースのパーサを選択
- RQ3でデータセットをフィルタリングしたこと
 - ⇒ データセットを公開

まとめ

- プログラム表現学習の新しい木構造ネットワーク手法
Tree-Transformerを提案
 - 木構造の兄弟/親子ノードの依存関係を捉えるために,
トップダウン/ボトムアップの再起型の双方向attention機構を活用
- ⇒ プログラム分類, 誤用演算子の特定と修復, 型推論の
3つのタスクにより有効性を示す

所管

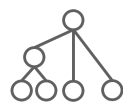
- RQ3でボトムアップのみを除いて実験していない
 - ボトムアップのみを除いた場合か，ほかの3つを除いた場合で実験するべき
- 型推論のタスクがTypeScriptだけ
 - 他の動的型付けについても考察した方がいいのでは？

• 木構造/グラフベース手法の比較

TABLE I

A COMPARISON OF EXISTING TREE/GRAPH-BASED PROGRAM REPRESENTATION LEARNING TECHNIQUES.

Approach	Global	Nodes	Order	Original
GNN+AST	×	✓	×	✓
Child-Sum Tree-LSTM [16]	✓	×	×	✓
N-ary Tree-LSTM [16]	✓	×	✓	×
TBCNN [6]	×	×	✓	✓
TreeCaps [9]	✓	×	✓	✓
Code2Vec [17]	×	✓	×	×
Code2Seq [18]	×	✓	×	×
ASTNN [7]	✓	×	✓	×
Tree-PE [19]	✓	✓	✓	×
Code Transformer [5]	✓	✓	✓	×
TreeBERT [20]	✓	✓	✓	×
Tree-Transformer(Ours)	✓	✓	✓	✓
GNN+Augmented AST [10]	×	✓	✓	×
Devign [3]	×	✓	✓	×
GREAT [13]	✓	✓	✓	×
HPG+HGT [21]	×	✓	✓	×



Global : 長期的な依存関係を捉えられるか

Nodes : コンテキスト情報を含んだ
ノード表現の学習

Order : プログラムの実行順序情報を
捉えられるか

Original: 構文木を追加処理しても本来の
構造情報を保てるか