

AST-Trans: Code Summarization with Efficient Tree-Structured Attention

Ze Tang

State Key Laboratory for Novel
Software Technology
Nanjing University
Nanjing, China
2228291607@qq.com

Xiaoyu Shen*

Alexa AI
Amazon
Berlin, Germany
gyouu@amazon.com

Chuanyi Li, Jidong Ge

State Key Laboratory for Novel
Software Technology
Nanjing University
Nanjing, China
lcy,gjd@nju.edu.cn

Liguo Huang

Department of Computer Science
Southern Methodist University
Dallas, Texas, USA
lghuang@lyle.smu.edu

Zhelin Zhu, Bin Luo

State Key Laboratory for Novel
Software Technology
Nanjing University
Nanjing, China
zzl,luobin@nju.edu.cn

背景

ソースコード要約

- コードの機能や目的を簡単な自然言語での説明
- 開発者がコードを素早く理解するために必要なもの [\[43\]](#)

問題点

- 高品質な要約を提供するにはかなりの労力が必要
- 間違っていたり古くなっている記述が多い [\[18\]](#)

自動ソースコード要約

- 開発者の労力などを削減するための自動化技術
 - ルールベース
 - IR (Information Retrieval) ベース
 - Transformerベース (state-of-the-art)

Transformerにおける構造情報付加

Transformerの特徴として、入力されたトークン列の構造情報を埋め込むことができる

- ソースコードを木構造（Abstract Syntax Tree）に変換し、ノード間の距離を構造情報として行列に保存

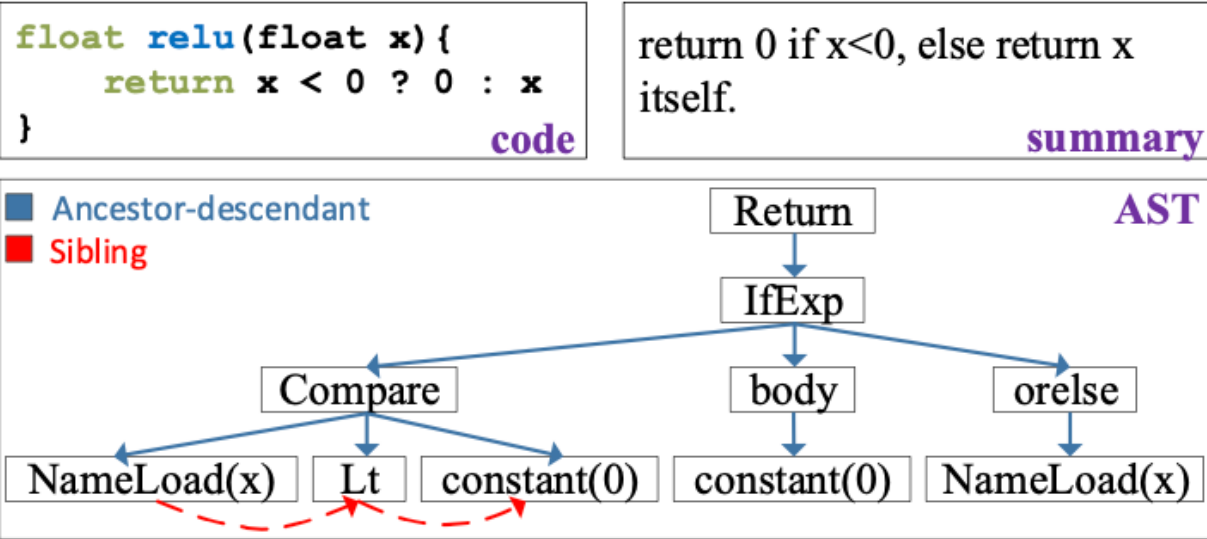
既存研究が用いた探索方法

- Pre-order Traversal [\[21\]](#)
- Structure-based Traversal [\[18\]](#)
- Path Decomposition [\[4\]](#)

問題点

- 全てのノードの関係をエンコードするため計算が複雑になる
- 不要な依存関係まで表現する

線形での表現例：Pre-order Traversal



探索によって線形配列に変換



構造情報		Return	IfExp	Compare	...	NameLoad(x)
	Return	0	-1	-2	...	-9
	IfExp	1	0	-1	...	-8
	Compare	2	1	0	...	-7
	:	:	:	:	⋮	:
	NameLoad(x)	9	8	7	...	0

AST-Trans (提案手法)

不要な依存関係の情報を排除し，正しい依存関係の検出と計算効率の向上を図る符号化手法

- ノードの親子関係と兄弟関係に注目

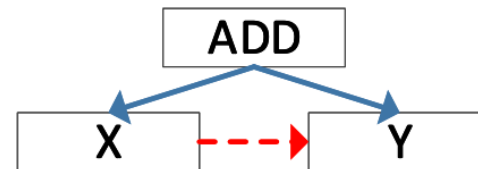
Linear Relationship
ADD → X → Y



	ADD	X	Y
ADD	0	-1	-2
X	1	0	-1
Y	2	1	0

Relationship Matrices

Tree Structure Relationship



■ A
■ S

A

	ADD	X	Y
ADD	0	-1	-1
X	1	0	∞
Y	1	∞	0

S

	ADD	X	Y
ADD	0	∞	∞
X	∞	0	-1
Y	∞	1	0

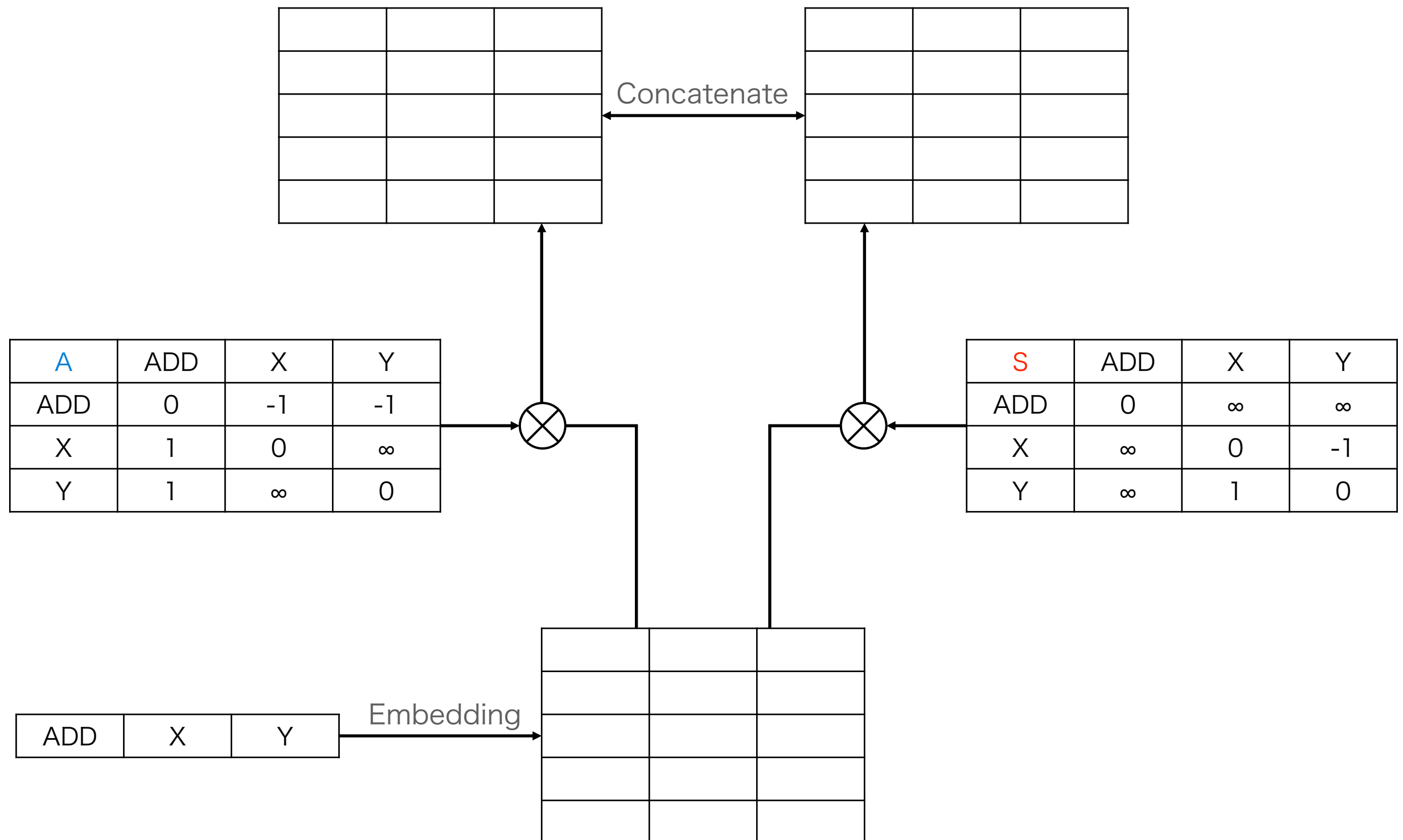
A : Ancestor-Descendent
→ 異なる深さのノードとの関係

S : Sibling
→ 同じ深さのノードとの関係
ただし，同じ親ノードを持つものに限る

線形構造では，ADDと変数Yの距離は2

木構造では，ADDと変数Yの距離は1

入力データの形



評価実験

データセット

Perspectives	Java	Python
# of Train instances	69,708	55,538
# of Validation instances	8,714	18,505
# of Test instances	8,714	18,502
Avg. # of tokens in code	120	48
Avg. # of nodes in AST	158	100
Avg. # of tokens in SBT	632	402
Avg. # of tokens in summary	18	9

- Java^[19], Python^[51]の公開されているベンチマーク
 - 要約文が4単語以下のメソッド, setter/getterメソッド, testerメソッドは対象から除外する^[41]
 - 複数文からなる要約文の場合, 最初の一文のみ対象とする

評価指標

- BLUE^[33], ROUGE-L^[27], METEOR^[6]

性能評価

従来の手法より優れた性能を発揮

- 入力データはPOT（深さ優先）が最も良い

Methods	Input	Java			Python		
		BLEU (%)	METEOR (%)	ROUGE-L (%)	BLEU (%)	METEOR (%)	ROUGE-L (%)
CODE-NN[20]	Code	27.6	12.61	41.10	17.36	09.29	37.81
API+CODE[19]		41.31	23.73	52.25	15.36	08.57	33.65
Dual Model[53]		42.39	25.77	53.61	21.80	11.14	39.45
BaseTrans*[1]		44.58	29.12	53.63	25.77	16.33	38.95
Code-Transformer*[57]		45.74	29.65	54.96	30.93	18.42	43.67
Tree2Seq[11]	AST(Tree)	37.88	22.55	51.50	20.07	08.96	35.64
RL+Hybrid2Seq[51]		38.22	22.75	51.91	19.28	09.75	39.34
GCN*[22]		43.94	28.92	55.45	32.31	19.54	39.67
GAT*[50]		44.63	29.19	55.84	32.16	19.30	39.12
Graph-Transformer*[40]		44.68	29.29	54.98	32.55	19.58	39.66
Code2Seq*[4]	AST(PD)	24.42	15.35	33.95	17.54	08.49	20.93
Code2Seq(Transformer)*		35.08	21.69	42.77	29.79	16.73	40.59
DeepCom[18]	AST(SBT)	39.75	23.06	52.67	20.78	09.98	37.35
Transformer(SBT)*		43.37	28.36	52.37	31.33	19.02	44.09
AST-Trans(SBT)*		44.15	29.58	54.73	32.86	19.89	45.92
Transformer(POT)*	AST(POT)	39.62	26.30	50.63	31.86	19.63	44.73
AST-Trans		48.29	30.94	55.85	34.72	20.71	47.77

アブレーションスタディ

Sibling情報をより付加する方が性能が良い (Table5)

- AST-TransはAncestor-Descendentのhead数を1, Siblingのhead数を7に設定

Ancestor-DescendentやSiblingは距離を制限せず、使えるだけ使った方がいい (Table6)

Table 5: Ablation study on h_A and h_S on Java Dataset.

h_A	h_S	BLEU (%)	METEOR (%)	ROUGE-L (%)
0	8	47.74	30.21	54.56
1	7	48.29	30.94	55.85
2	6	48.28	30.94	55.64
3	5	48.25	30.92	55.66
4	4	48.23	30.96	55.68
5	3	48.11	30.93	55.46
6	2	48.1	30.74	55.22
7	1	48.24	30.91	55.57
8	0	48.07	30.62	55.29

Table 6: Ablation study on P_A and P_S on Java Dataset.

P_A	P_S	BLEU (%)	METEOR (%)	ROUGE-L (%)
0	0	36.34	23.83	45.58
1	1	46.95	30.33	54.24
5	1	47.45	30.11	54.28
5	3	47.82	30.29	54.62
5	5	48.14	30.77	55.45
10	5	48.29	30.94	55.85

まとめ

目的

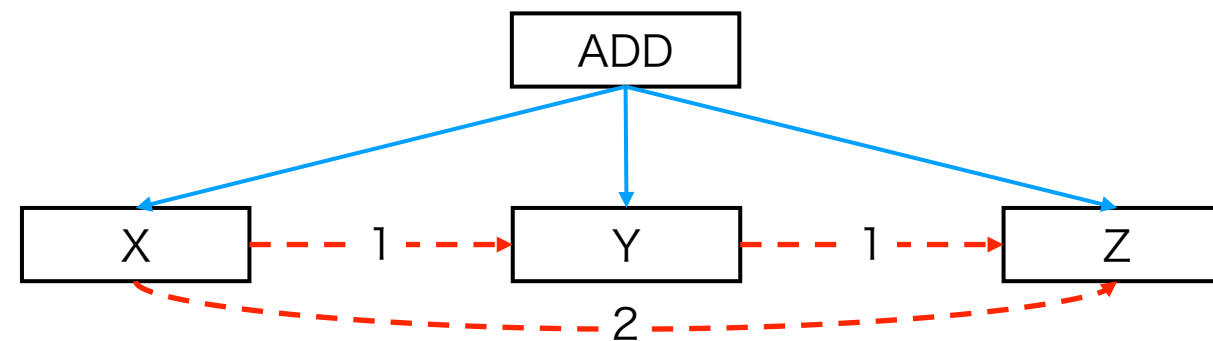
- 不要な依存関係の情報を排除し，正しい依存関係の検出と計算効率の向上を図る符号化手法の提案

結果

- ASTの親子関係，兄弟関係に着目することでソースコード要約の性能を向上させることが可能

所感

今までこの方法が試されていなかったことに驚いた
Siblingの計算が曖昧でモヤモヤする



- 公開されているスクリプトの処理が全く理解できない
 - この変数は何を表しているのかや, どのクラスやメソッドで何の処理を行っているのか読み解くのが困難

探索方法とデータ形式

Methods	Linearized AST sequence
POT	Return IfExp Compare NameLoad(x) Lt constant(0) body constant(0) orelse NameLoad(x)
SBT	(Return (IfExp (Compare (constant(0)) constant(0) (Lt) Lt (NameLoad(x)) NameLoad(x)) Compare (body (constant(0)) constant(0)) body (orelse (NameLoad(x)) NameLoad(x)) orelse) IfExp) Return
PD	Path1: Path1: Lt Compare constant(0) Path2: NameLoad(x) Compare constant(0) Path3: Path3: constant(0) Compare IfExp body constant(0) ...