

Source Code Summarization with Structural Relative Position Guided Transformer

Zi Gong, Cuiyun Gao, Yasheng Wang, Wenchao Gu
Yun Peng, Zenglin Xu

In Proceedings of the 29th International Conference on Software Analysis, Evolution and Reengineering
(SANER' 22), pp13-24

背景：深層学習とコード要約

深層学習の開発・保守タスク支援への利用

- コード検索 [1][2][3]
- コードクローン検出 [4][5][6]
- **コード要約** [7][8][9]
- コード生成 [10][11][12]

コード要約タスクの有用性

- 開発者は開発の59%をプログラムの理解に時間を消費 [13]

高品質なコード要約はプログラム理解の効率化に重要 [14]

→ プログラムの挙動を自然言語で提供する

[13] X. Xia, L. Bao, D. Lo, Z. Xing, A. E. Hassan, and S. Li, "Measuring program comprehension: A large-scale field study with professionals," IEEE Transactions on Software Engineering, vol. 44, no. 10, pp. 951–976, 2017.

[14] H. He, "Understanding source code comments at large-scale," in Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2019, pp. 1217–1219.

既存研究

Information-Retrieval (IR) ベース^{[15][16][17]}

- ソースコード中から理解に重要な箇所を抽出
- プログラムの挙動の説明になっていない

Recurrent Neural Network (RNN) ベース^[18 - 25]

- ソースコードの時系列学習をおこない要約を生成
- 規模が大きいと十分な学習ができない

Transformerベース^{[26][27][28]}

- RNNベースの上位互換
- Positional EncodingとSelf-Attentionを利用することで入力データ全体の時系列学習が可能
- ソースコードの構造を考慮していない

研究目的

ソースコードの構造情報の利用による性能向上

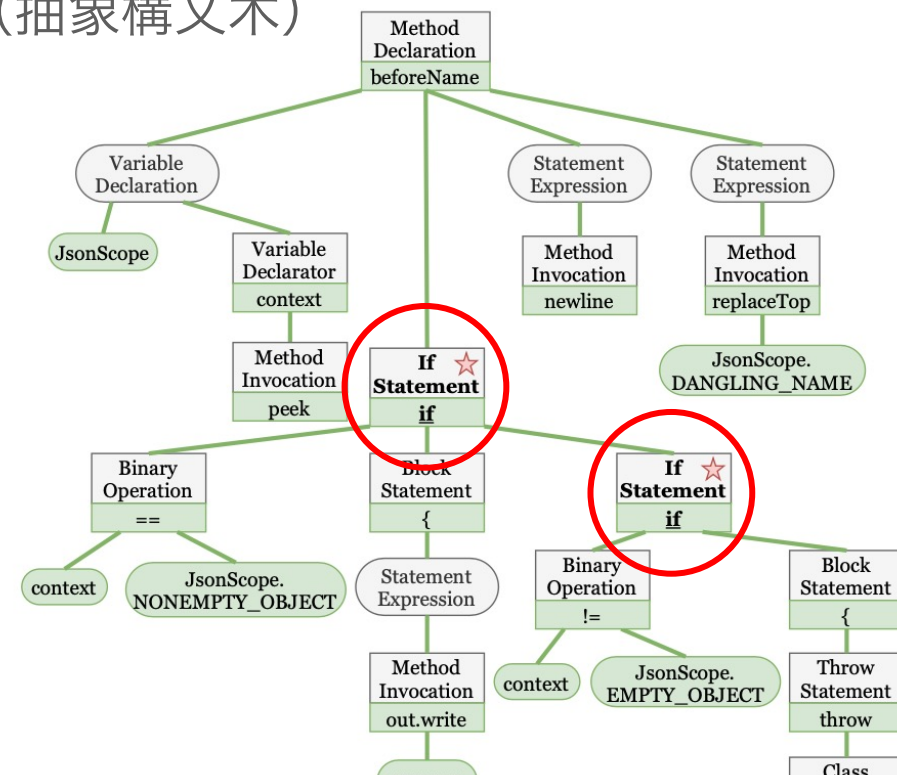
ソースコードの例

```
private void beforeName() throws IOException {  
    JsonScope context=peek();  
    if (context == JsonScope.NONEMPTY_OBJECT) {  
        out.write(',');  
    }  
    else if (context != JsonScope.EMPTY_OBJECT) {  
        throw new IllegalStateException("Nesting problem: " + stack);  
    }  
    newline();  
    replaceTop(JsonScope.DANGLING_NAME);  
}
```

Summary
Reference inserts any necessary separators and whitespace before a name.
also adjusts the stack to expect the name's value.
SIT: inserts necessary separators and whitespace before a name.
SCRIPT: inserts the necessary separators and whitespace before a name. or
adjusts the stack to the given value.

テキストのみでは“if”は構造的に
遠い距離に存在するため関連性が低い

AST (抽象構文木)

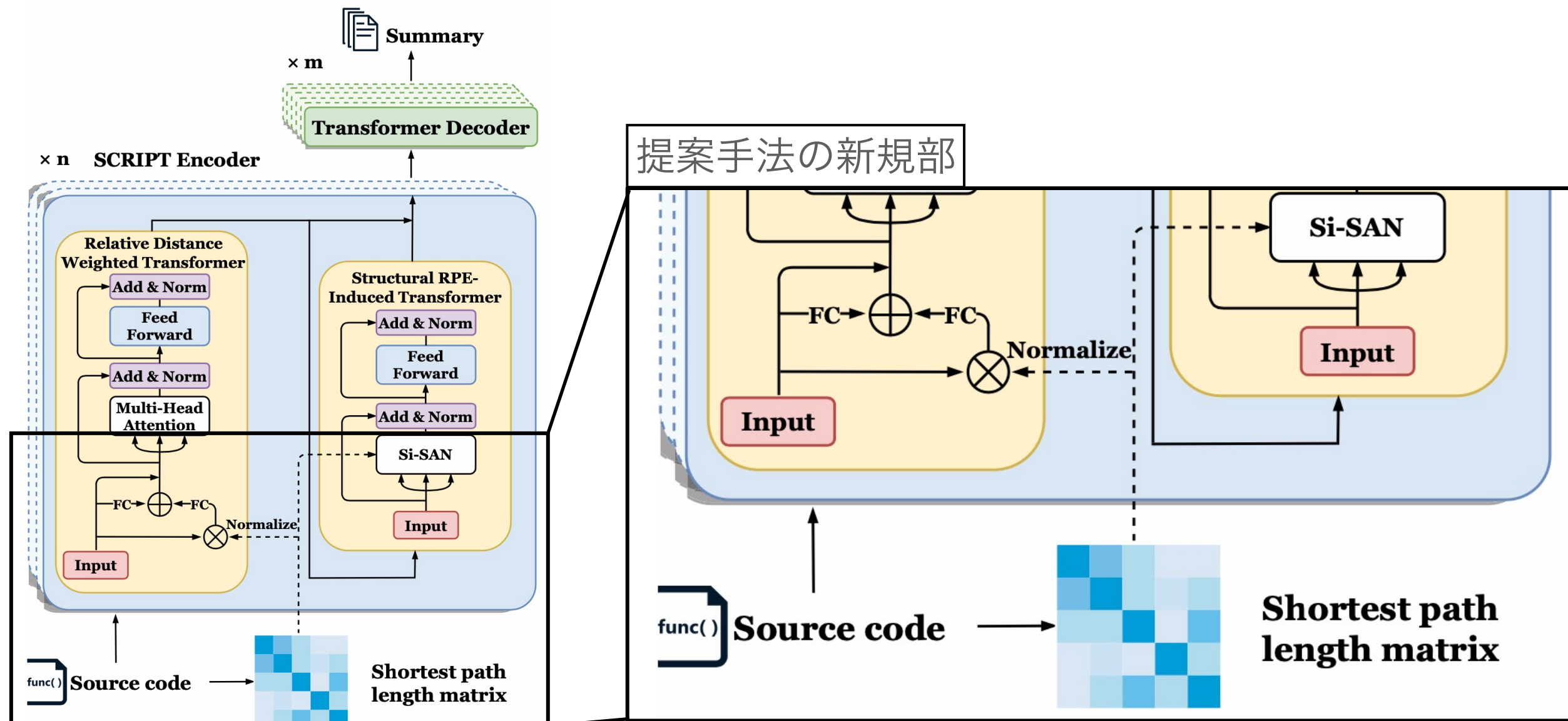


ASTの構造で考えると、
2つの“if”は近い距離に存在する

SCRIPT

Transformerモデルを利用した要約手法

- 基本的には, Transformerを利用したエンコーダ・デコーダ
- エンコーダ部に2種類のTransformerエンコーダを連結

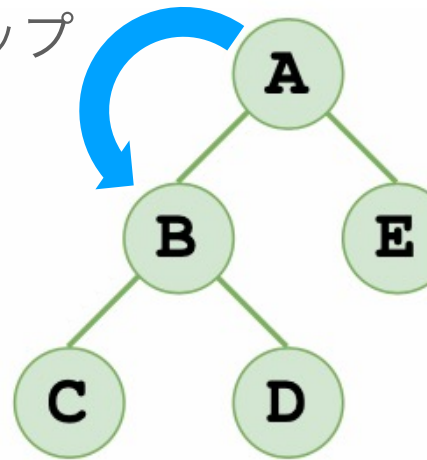


SCRIPT : 提案部 (1)

Shortest Path Length Matrix

- ASTの各ノード間の最短距離を記録した対称行列

AからBまで1ホップ



(a) AST.

	A	B	C	D	E
A	0	1	2	2	1
B	1	0	1	1	2
C	2	1	0	2	3
D	2	1	2	0	3
E	1	2	3	3	0

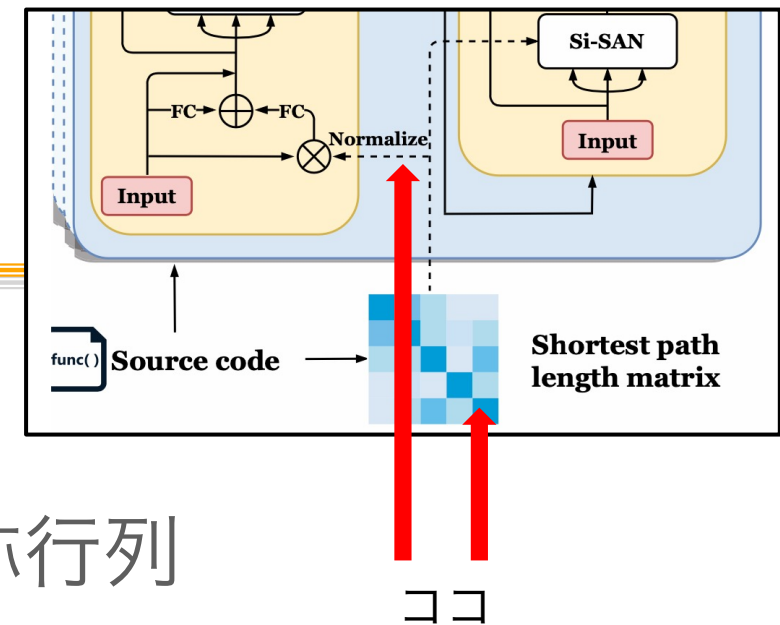
(b) Shortest path length matrix.

(A, B) に1を記録

- 正規化 (Normalization)

- 対称となるノード間距離の逆数と全てのノード間距離の逆数の総和との比

$$\bar{M}_{i,j} = \begin{cases} \frac{1/M_{i,j}}{\sum_{z \in \{y | M_{i,y} \neq 0\}} (1/M_{i,z})}, & \text{if } M_{i,j} \neq 0, \\ 0, & \text{otherwise.} \end{cases}$$

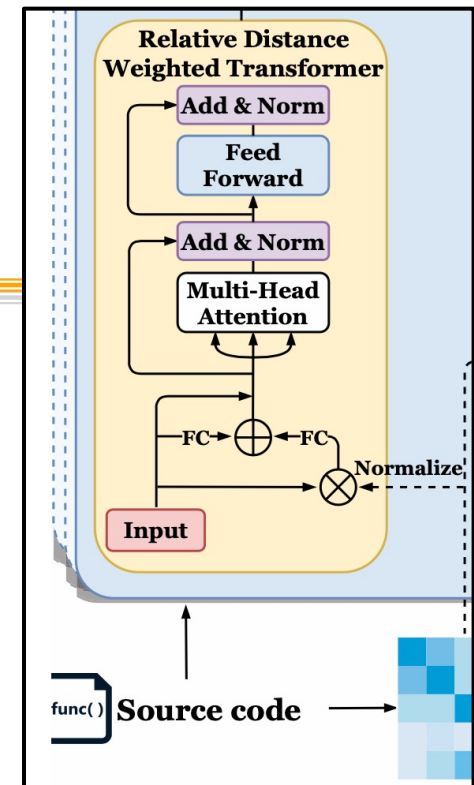


SCRIPT : 提案部 (2)

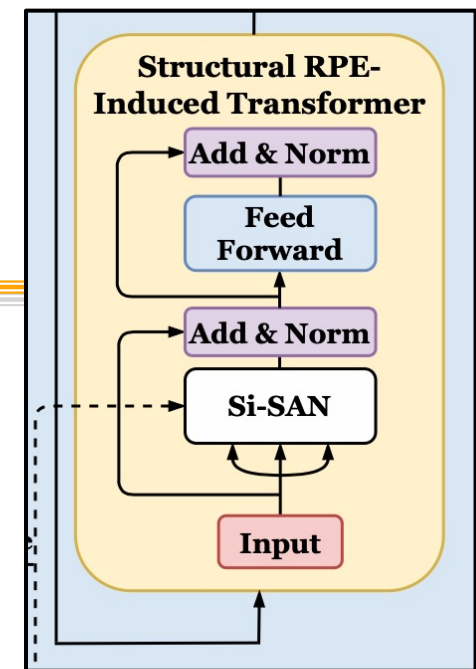
Relative Distance Weighted Transformer

目的 : コードトークン間の位置関係情報を埋め込む

- ノード間距離による重み付け
 - 入力されるコードトークンに重み (SPLM) を掛ける
- 重み付け有/無のベクトルの加重和
 - 重み付け有/無それぞれのベクトルを全結合層 (FC) に入力し, 出力されたベクトルを足し合わせる
 - FC : Fully Connected Layer



SCRIPT : 提案部 (3)



Structual RPE-Induced Transformer

目的 : コードの詳細な情報の埋め込む

- RWDTの出力ベクトルにコードの情報を付加
 - 抽象構文構造, 制御フロー, データ依存関係の行列

$$A_{mv} = \alpha A_{ast} + \beta A_{fl} + \gamma A_{dp}$$

通常のTransformerモデルで使用されている
Multi Head Attention

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V,$$

提案手法のTransformerモデルで使用されている
Structual RPE-Induced Self-Attention

$$\text{SRPEi-SAN}(X) = \text{Softmax} \left(\frac{A_{mv} \cdot QK^T}{\sqrt{d_k}} \right) V,$$

評価実験

データセット（メソッド数）^{[31][32]}

Datasets	Java	Python
Train	69,708	55,538
Validation	8,714	18,505
Test	8,714	18,502
Avg. tokens in code	73.76	49.42
Avg. tokens in summary	17.73	9.48

Research Question

- ベースライン（既存手法）と比較してSCRIPTの性能は？
- SCRIPTの各コンポーネントの有効性は？
- どのコンポーネントでASTの構造情報を付加すると有効か？

[31] Y. Wan, Z. Zhao, M. Yang, G. Xu, H. Ying, J. Wu, and P. S. Yu, “Improving automatic source code summarization via deep reinforcement learning,” in Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, 2018, pp. 397–407.

[32] X. Hu, G. Li, X. Xia, D. Lo, S. Lu, and Z. Jin, “Summarizing source code with transferred api knowledge,” in Proceedings of the 27th International Joint Conference on Artificial Intelligence, 2018, pp. 2269–2275.

評価指標

BLEU-4

- N-gramフレーズの平均マッチ率 (Precision)
 - 同じフレーズがどれくらい生成できているか

ROUGE-L

- 共通している最長フレーズ長
 - 同じ文をどれくらい再現できているか

METEOR

- 正解単語の使用率 (Recall)
 - 正解文で使用されている単語がどれくらい含まれているか

RQ1・2：結果

Model	Java			Python		
	BLEU	ROUGE-L	METEOR	BLEU	ROUGE-L	METEOR
CODE-NN (Iyer et al., 2016)	27.60	41.10	12.61	17.36	37.81	09.29
Tree2Seq (Eriguchi et al., 2016)	37.88	51.50	22.55	20.07	35.64	08.96
RL+Hybrid2Seq (Wan et al., 2018)	38.22	51.91	22.75	19.28	39.34	09.75
DeepCom (Hu et al., 2018a)	39.75	52.67	23.06	20.78	37.35	09.98
API + Code (Hu et al., 2018b)	41.31	52.25	23.73	15.36	33.65	08.57
Dual Model (Wei et al., 2019)	42.39	53.61	25.77	21.80	39.45	11.14
Transformer (Ahmad et al., 2020)	44.58	54.76	26.43	32.52	46.73	19.77
SiT* (Wu et al., 2021)	45.70	55.54	27.55	33.46	47.50	20.28
SCRIPT	46.89	56.69	28.48	34.00	48.15	20.84
SCRIPT w/o RDW-Transformer	46.72	56.50	28.32	33.75	47.66	20.65
SCRIPT w/o SRPEi-Transformer	46.42	56.29	28.15	33.60	47.61	20.43

全ての指標において既存手法を上回る性能

RDW-Transformerに比べて，SRPEi-Transformerを組み込まない方が性能が下がる

RQ3 : 結果

Model	Java			Python		
	BLEU	ROUGE-L	METEOR	BLEU	ROUGE-L	METEOR
Equip in RDW-Transformer	46.53	56.36	28.25	33.68	47.70	20.52
Equip in SiT	46.89	56.69	28.48	34.00	48.15	20.84
Equip in all encoder layers	46.66	56.45	28.33	33.70	47.81	20.64

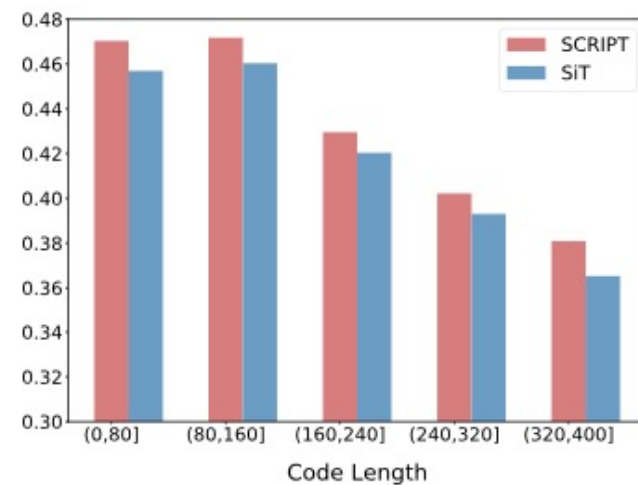
低下している

SiT (SRPEi-Transformer) にASTの構造情報を付加した場合が最も性能が良い (提案手法)

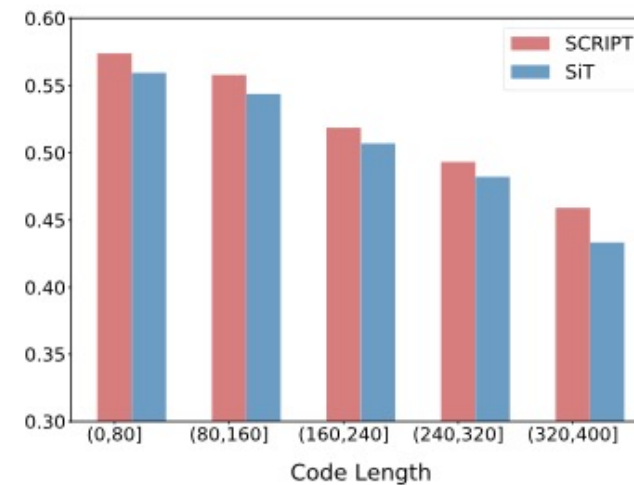
- RDW-TransformerにASTの構造情報を付加した場合, 性能向上しない / 低下させる

議論：SiPとの比較（1）

コードの長さに関係なくSCRIPTの方が性能が良い

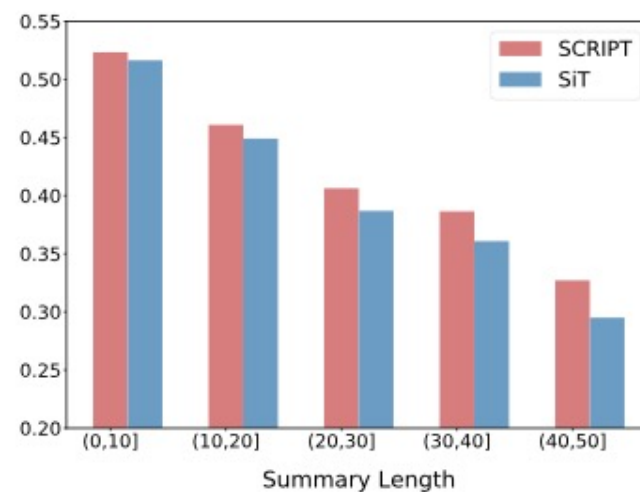


(a) BLEU

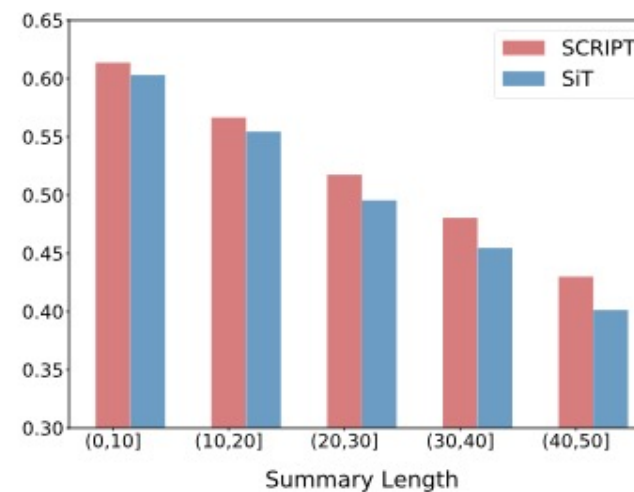


(b) ROUGE-L

要約文の長さに関係なくSCRIPTの方が性能が良い



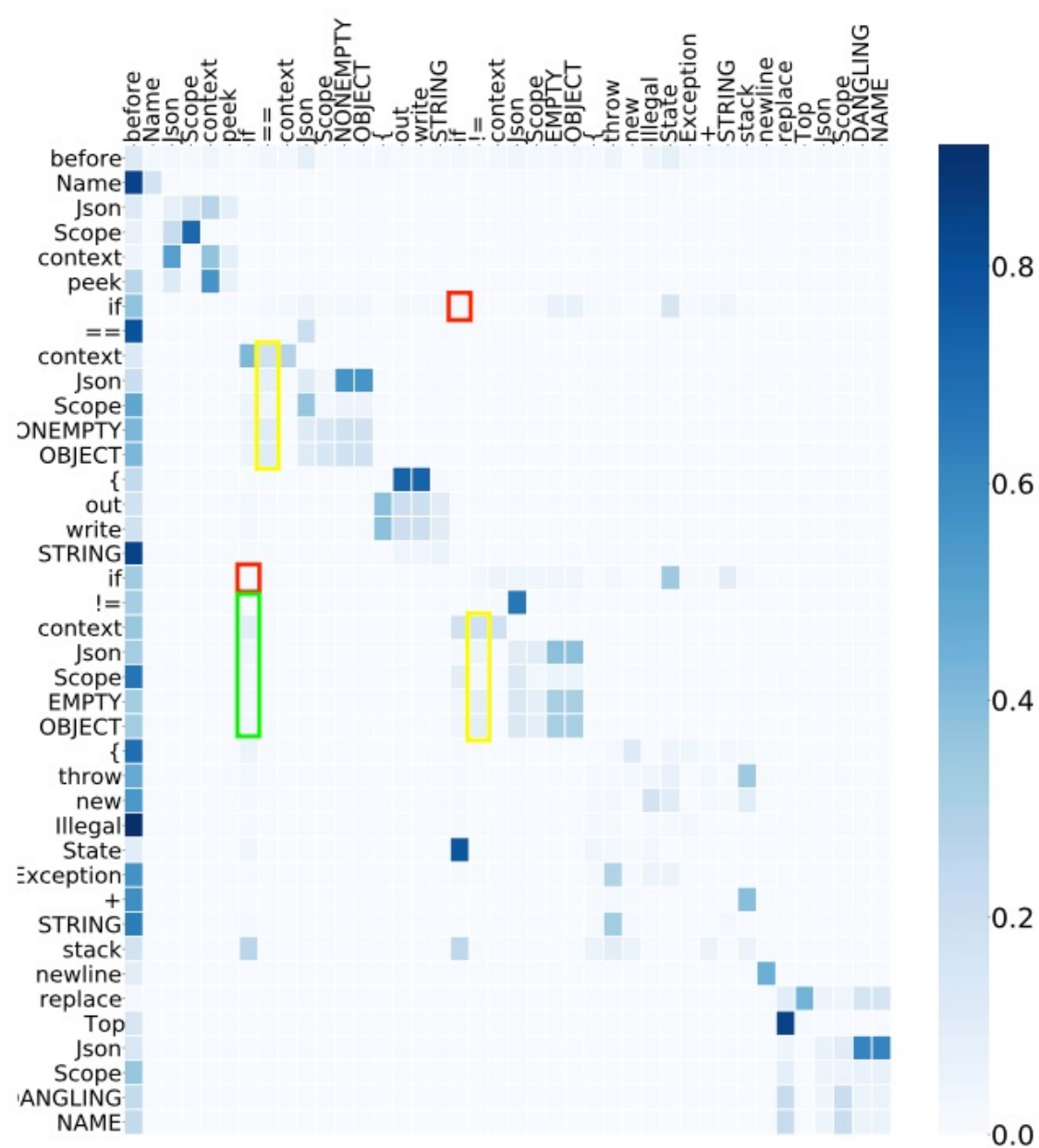
(a) BLEU



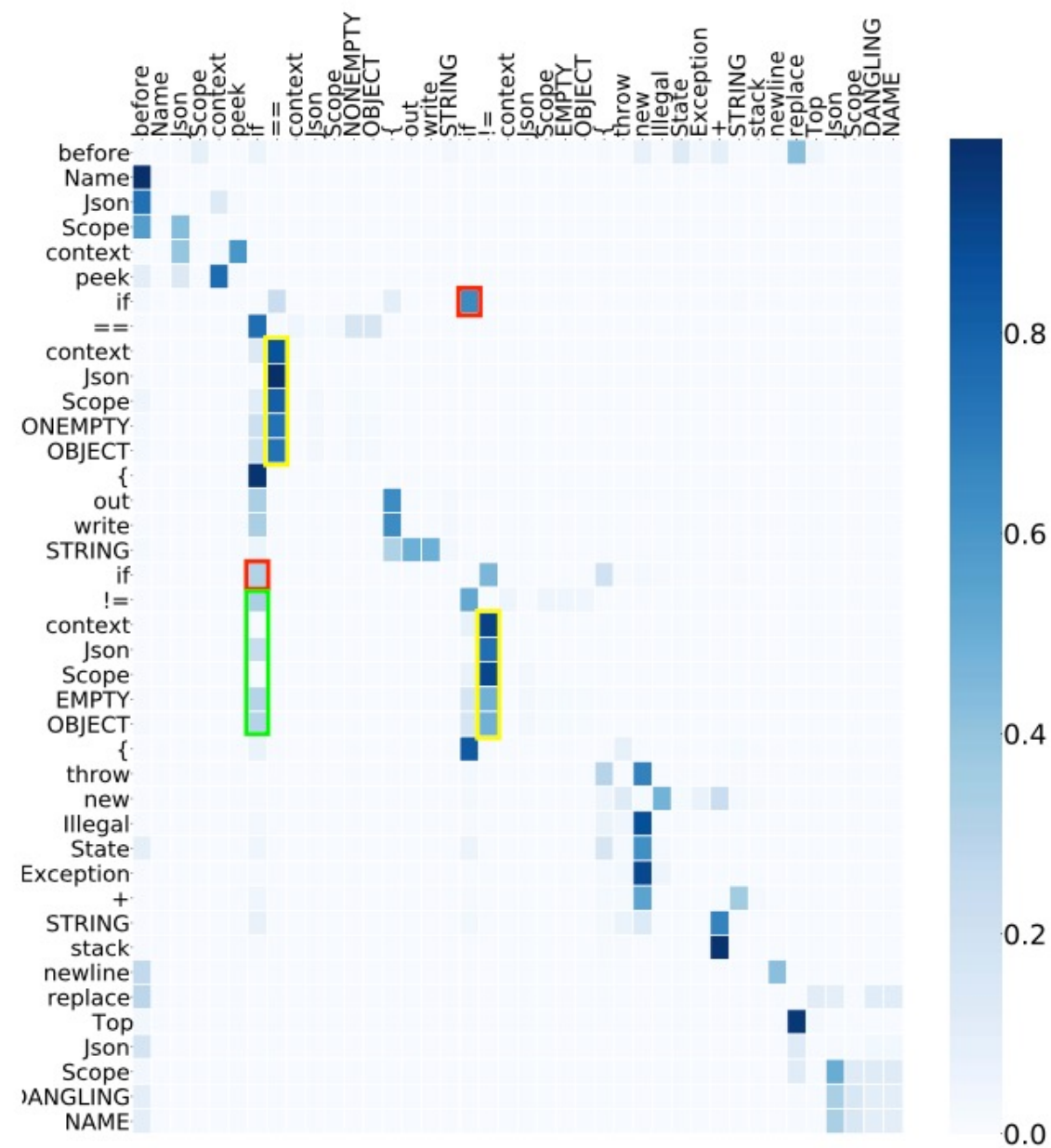
(b) ROUGE-L

議論：SiPとの比較 (2-1)

Attentionの重みについて



(a) SiT



(b) SCRIPT

議論：SiPとの比較（2-2）

赤枠

- “if”トークン同士の関連性

緑枠

- “if”トークンと条件文の関連性

黄枠

- 条件演算子と条件文の関連性

SiPに比べてSCRIPTの方が関連性が強くなっている

→ コードの構造情報を付加することの有効性

まとめ

目的

- ソースコードの構造情報の利用による性能向上

提案手法 (SCRIPT)

- 2種類のTransformerエンコーダを連結させたモデル
 - Relative Distance Weighted Transformer
 - Structural RPE-Induced Transformer

結果

- Java
 - BLUE-4 : 46.89, ROUGE-L : 56.69, METEOR : 28.48
- Python
 - BLUE-4 : 34.00, ROUGE-L : 48.15, METEOR : 20.84

所感

最新のコード要約手法であるSiPに比べて性能が向上したことはわかったが、この数パーセントの凄さがいまいち理解できない

- コードの構造情報はそこまでクリティカルなのか？

Transformerの大まかな仕組みは理解できたが、この数式の理解が未だ及ばない. . .

- Self-Attentionに用いられているQuery, Key, Valueの概念的な理解はできるが、どのように算出されているのかはおそらく理解できない

この論文の提案部の数式も難しすぎる

参考

Transformerについて

- <https://tips-memo.com/translation-jayalmmar-transformer>
- <https://www.acceluniverse.com/blog/developers/2019/08/attention.html>

テキスト自動生成評価指標

- <https://qiita.com/amtsyh/items/a926b79b90dfabe895e9>

ROUGEについて

- <https://qiita.com/icoxfog417/items/65faecbbe27d3c53d212>