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

Ze Tang[1], Xiaoyu Shen[2], Chuanyi Li[1], Jidong Ge[1], Liguo Huang[3], Zhelin Zhu[1], Bin Luo[1]

[1]State Key Laboratory for Novel Software Technology, Nanjing University

[2]Alexa AI, Amazon

[3]Department of Computer Science, Southern Methodist University Dallas

2. Background

Code Summarization/Code Comment Generation

```
float relu(float x){
    return x < 0 ? 0 : x
}
```
code

source code

return 0 if x<0, else return x itself.

summary

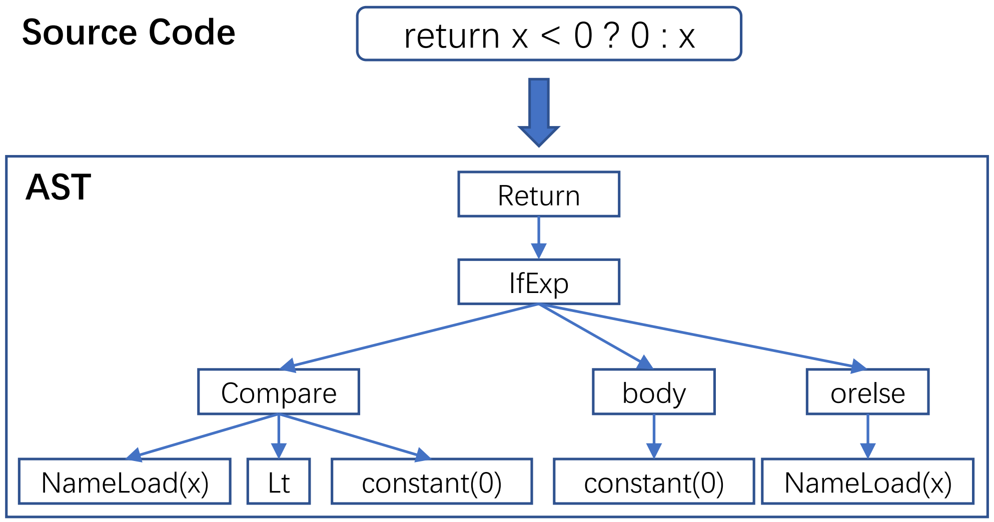
natural language description

Input	Method	Model
Code	split code into tokens	RNN[1] , Transformer[2]
Abstract Syntax Tree(AST)	use tree-based model or GNN	Tree-LSTM[3]
Linearized AST	convert AST to sequence • Pre-order Traversal(POT) • Structure-based Traversal(SBT) • Path Decomposition(PD)	LSTM[4] , Code2Seq[5]

3. Motivation

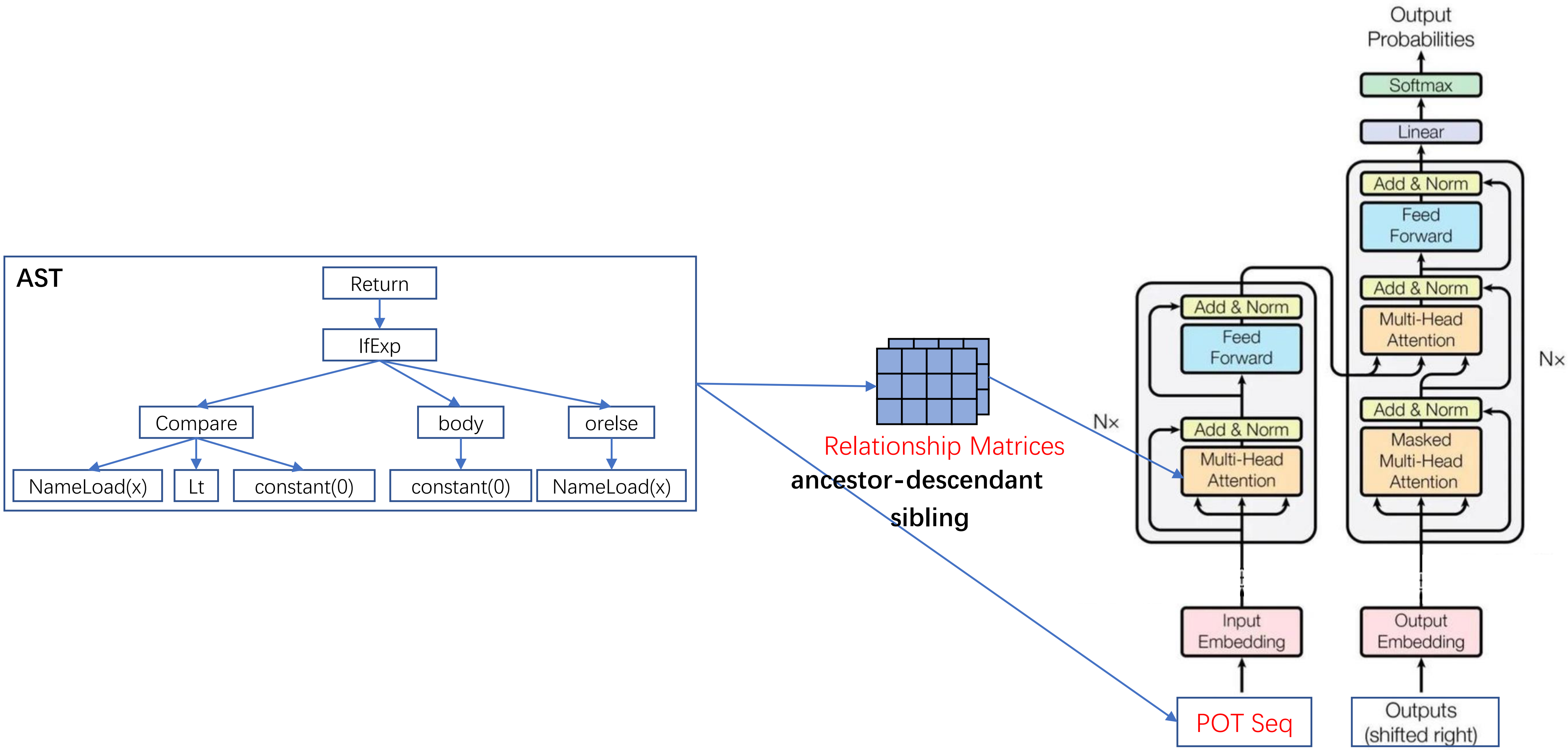
The length of **linearized AST** is much longer than **source code**

- **Hard to learn :** encoding SBT underperforms encoding source code when using Transformer[2]
- **Significant computational overhead :** quadratically with the sequence length in Transformer

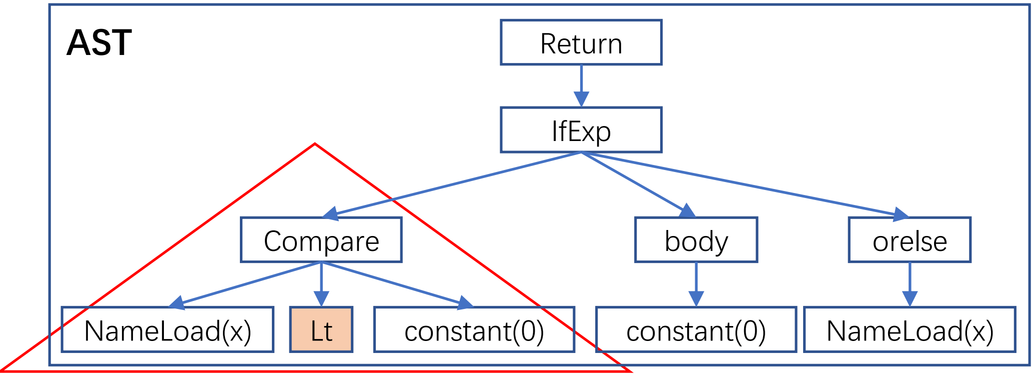


Methods	Linearized AST sequence
POT	Return IfExp Compare NameLoad(x) Lt constant(0) body constant(0) orelse NameLoad(x)
SBT	(Return (IfExp (Compare (constant(0)) constant(0) (Lt) Lt (NameLoad(x)) NameLoad(x)) Compare (body (constant(0)) constant(0)) body (orelse (NameLoad(x)) NameLoad(x)) orelse) IfExp)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

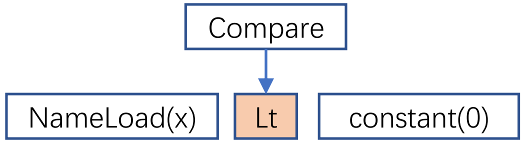
4. AST-Trans —— Overall Architecture



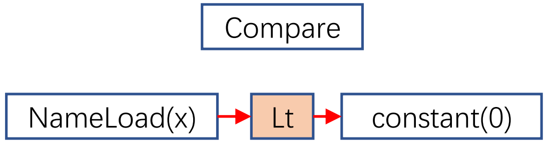
4. AST-Trans—— Relationship Matrices



windowed



Ancestor-descendant relationship



Sibling relationship

In AST, children nodes are orderd

$$A_{ij} = \begin{cases} \mathbf{SPD}(i,j) & \text{if } |\mathbf{SPD}(i,j)| \le P \\ \infty & \text{otherwise} \end{cases}$$

$$S_{ij} = \begin{cases} \mathbf{SID}(i,j) & \text{if } |\mathbf{SID}(i,j)| \le P \\ \infty & \text{otherwise} \end{cases}$$

P: max distance

SPD(i,j) : Shorted Path Distance between node i and j

SID(i,j) : horizontal SIbling Distance between node i and j

$A_{ij} = -A_{ji}$ and $S_{ij} = -Sji$

	Compare	NameLoad(x)	Lt	constant(0)
Compare	0	1	1	1
NameLoad(x)	-1	0	∞	∞
Lt	-1	∞	0	∞
constant(0)	-1	∞	∞	0

$\{A_{ij}\}$

	Compare	NameLoad(x)	Lt	constant(0)
Compare	0	∞	∞	∞
NameLoad(x)	∞	0	1	∞
Lt	∞	-1	0	1
constant(0)	∞	∞	-1	0

$\{S_{ij}\}$

P=1

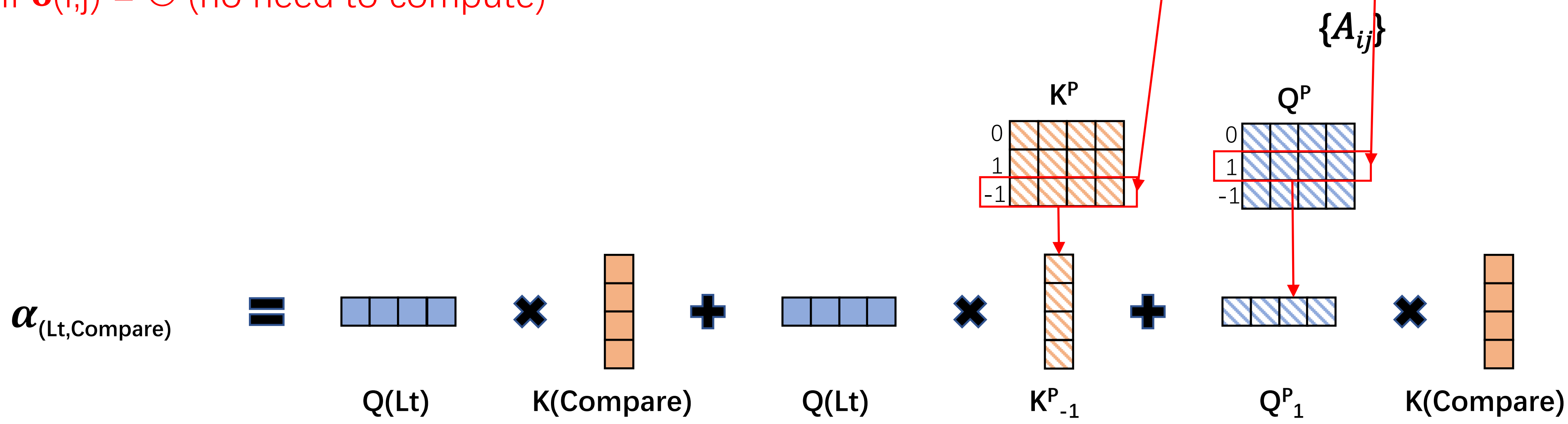
4. AST-Trans—— Tree-Structured Attention

Disentangled Attention[6]

$$\tilde{\alpha}_{i,j} = \underbrace{Q(x_i)K(x_j)^\top}_{\text{content-to-content}} + \underbrace{Q(x_i){K^P_{\delta(i,j)}}^\top}_{\text{content-to-position}} + \underbrace{Q^P_{\delta(j,i)}K(x_j)^\top}_{\text{position-to-content}}$$

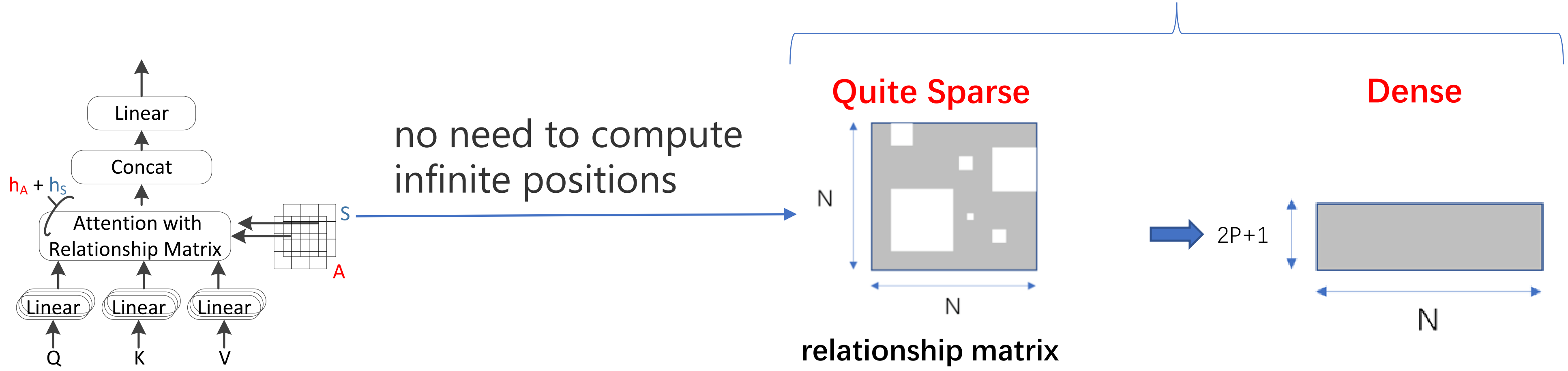
- K^P and Q^P are hype parameter matrices with shape (2P+1) X m
- $K^P_{\boldsymbol{\delta}(i,j)}$ is the $\delta(i, j)$-th row of K^P
- $\alpha_{i,j}$ = ∞ if $\boldsymbol{\delta}$(i,j) = ∞ (no need to compute)

	Compare	NameLoad(x)	Lt	constant(0)
Compare	0	1	1	1
NameLoad(x)	-1	0	∞	∞
Lt	-1	∞	0	∞
constant(0)	-1	∞	∞	0



5. Model Implementation

Gather with decomposed COO (GDC) Algroithm



GDC Algorithm:

1. Decompose the matrix
2. Reorder query and key context by col_index and row_index separately
3. Compute the attention scores

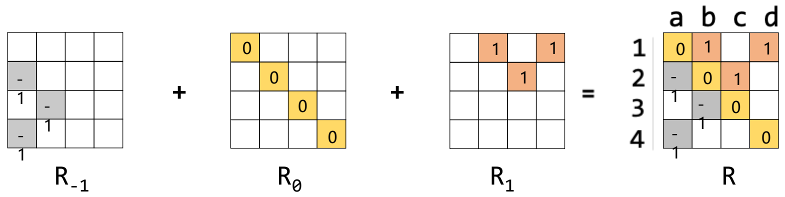
COO format

$$\begin{pmatrix} 5 & 0 & 0 & 0 \\ 0 & 8 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 6 & 0 & 0 \end{pmatrix}$$

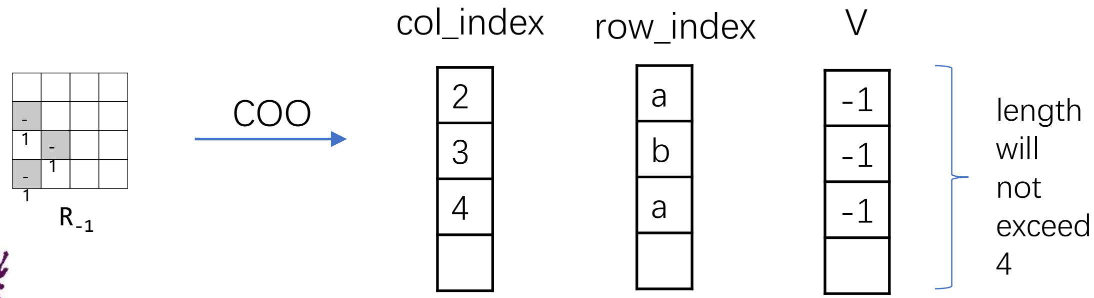
```
V         = [ 5 8 3 6 ]
COL_INDEX = [ 0 1 2 1 ]
ROW_INDEX = [ 0 1 2 3 ]
```

5. Model Implementation

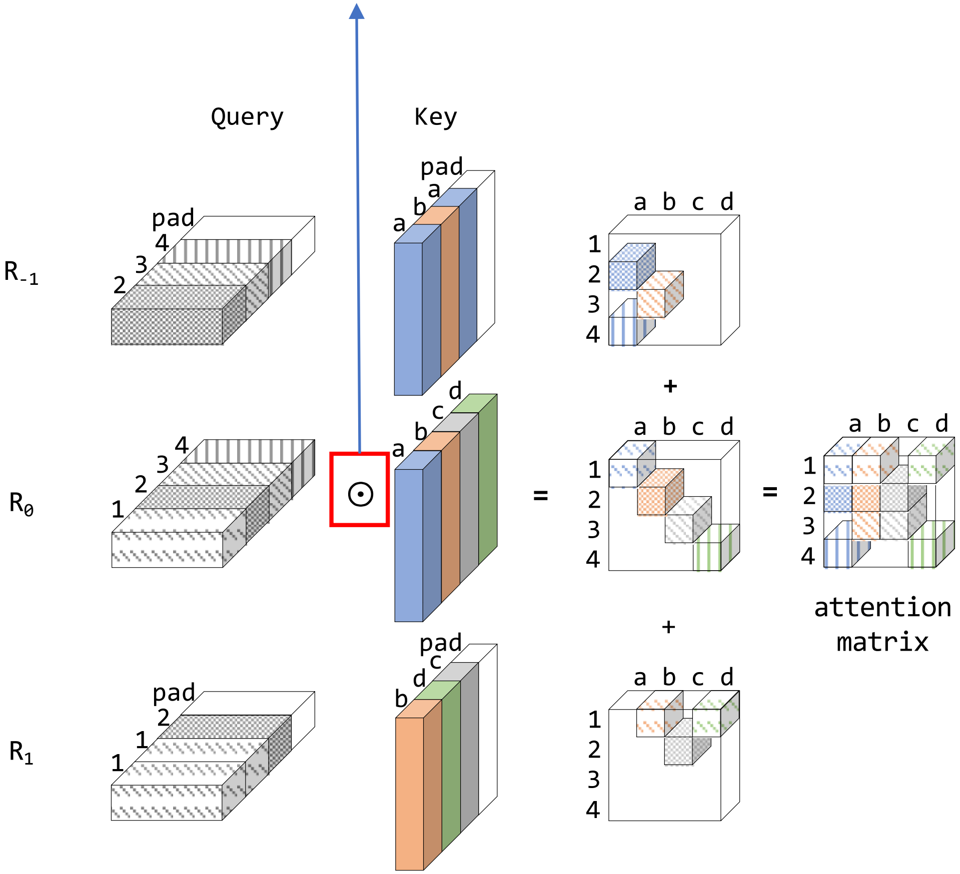
***Theorem**: the number of node pairs with the same distance length in relationship matrix will not exceed the size of the tree.*



1) **Decompose the matrix**: group node pairs with the same distance(value).



complexity: quadratically (matrix production) -> linearly(dot production)



2) Reorder query and key context by col_index and row_index separately

3) Compute the attention scores

6. Results —— Compared with baselines

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**

6. Results —— Complexity

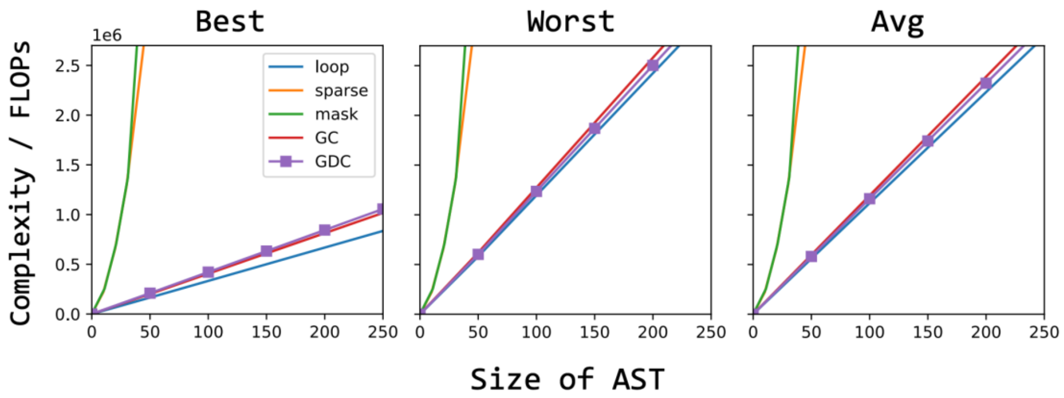


Figure 5: Theoretical complexity with $P = 5$, $m = 32$. loop has the lowest complexity but cannot be parallelized in practice.

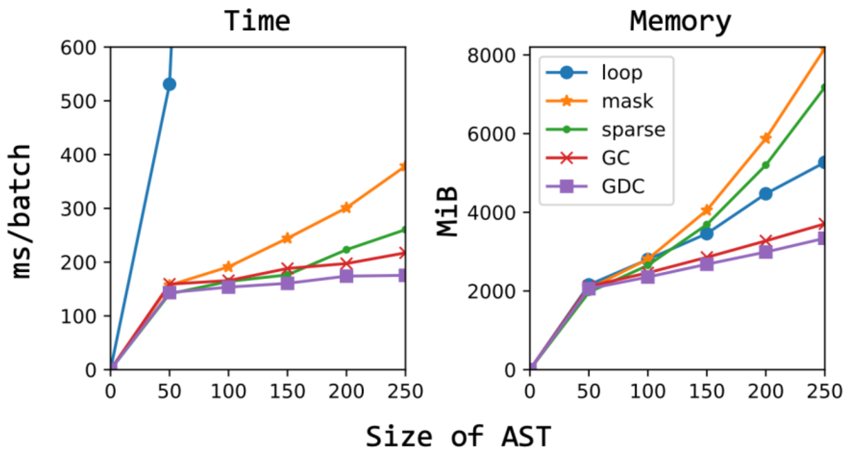


Figure 7: Runtime and memory cost of five implementations with batch size=16. The cost of the mask implementation is equal to the standard Transformer, which grows quadratically with the AST size.

Theoretical complexity

Runtime and memeory cost in GPU

Reference

1. Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2016. Summarizing Source Code using a Neural Attention Model. ACL 2016
2. Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2020. A Transformer-based Approach for Source Code Summarization. ACL 2020
3. Akiko Eriguchi, Kazuma Hashimoto, and Yoshimasa Tsuruoka. 2016. Tree-to- Sequence Attentional Neural Machine Translation. ACL 2016
4. Xing Hu, Ge Li, Xin Xia, David Lo, and Zhi Jin. 2018. Deep code comment generation. In Proceedings of the 26th Conference on Program Comprehension, ICPC 2018
5. Uri Alon, Shaked Brody, Omer Levy, and Eran Yahav. 2019. code2seq: Generating Sequences from Structured Representations of Code. ICLR 2019
6. Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2021. Deberta: decoding-Enhanced Bert with Disentangled Attention. In 9th International Con- ference on Learning Representations, ICLR 2021

THANKS