Beijing Forest Studio

北京理工大学信息系统及安全对抗实验中心

代码摘要生成技术

硕士研究生 徐程柯

2024年09月22日

内容提要

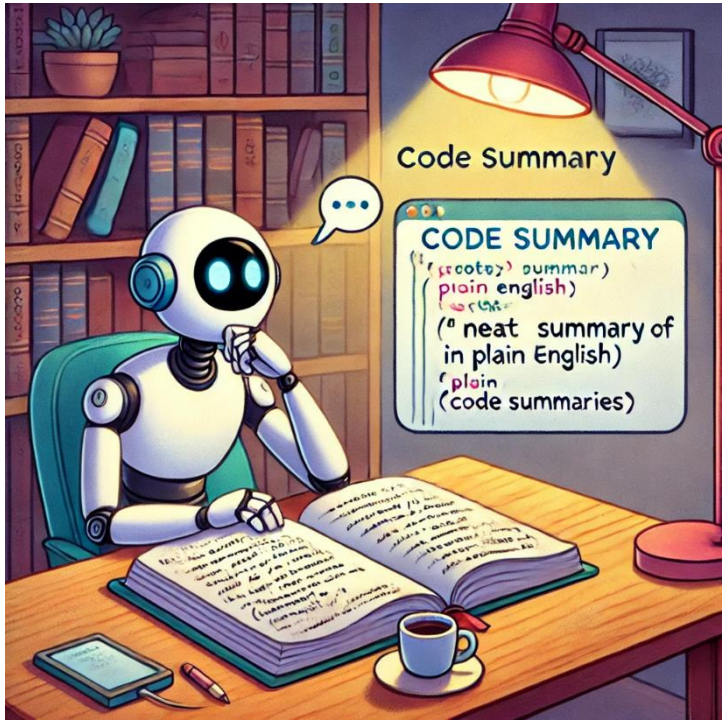
- 预期收获
- 内涵解析
- 背景简介
- 知识基础
- 算法原理
  - **CoSS**
  - **StructCodeSum**
- 特点总结与工作展望
- 参考文献

预期收获

- 预期收获
  - 掌握代码摘要的基本概念
  - 理解代码表示在代码摘要中的意义与作用
  - 理解代码摘要技术的基本原理
  - 了解代码摘要的应用领域和发展方向

内涵解析与研究目标 代码摘要

- 什么是代码摘要
  - 代码摘要（**Code Summarization**），是自动为目标代码生成富有信息性自然语言描述的技术
- 研究目标
  - 面向涵盖多种编程语言和应用场景的代码库，以及与代码相匹配的自然语言描述文档
  - 利用代码的语法和语义特征，结合自然语言处理（**NLP**）、图网络等理论，捕获代码功能
  - 实现代码摘要的自动化生成，提高软件开发和维护的效率



- 研究背景
  - 软件工程的复杂性增加：随着软件系统的规模和复杂性日益增加，理解和维护这些系统的难度也相应提高
  - 代码重用需求：在现代软件开发中，代码重用是提高开发效率的关键因素之一，若缺乏足够的文档或者代码描述，代码的重用率会大大降低
  - 自然语言处理技术的进步：大型预训练语言模型（**Large Language Model, LLM**）在大规模文本数据上进行预训练，学会了捕捉和理解复杂的语义信息，可以用于各类型下游任务中

高端的程序员，往往采用最朴素的编程方式

研究意义 代码摘要

- 研究意义
  - 提高软件开发的效率和质量：自动生成的代码摘要可以帮助开发者更快地理解复杂代码结构和业务逻辑，从而加速开发流程并减少因误解代码功能而引起的错误
  - 代码审查和质量控制：代码摘要可以在代码审查过程中提供关键信息，帮助审查者快速把握代码实现目标，从而提高审查的效率和质量
  - 辅助自动文档生成：通过自动生成与代码直接关联的摘要 ，可以确保文档的实时更新和准确性，同时减轻开发者的文档编写负担

研究历史 代码摘要

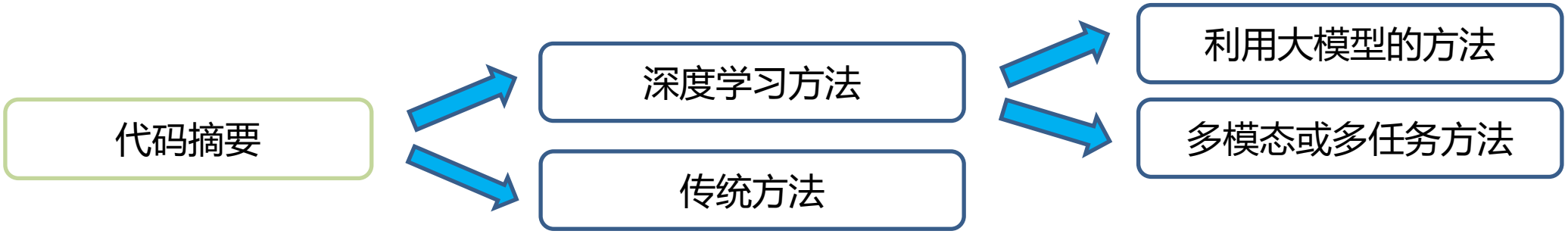
Haiduc等人首次尝试使用信息检索（IR）技术生成代码摘要，采用基于规则和基于IR的启发式方法，从代码中提取关键信息进行代码摘要合成

Y. Wan 等人引入抽象语法树（AST），将源代码的结构表示为语法树，其中每个节点表示语法级代码结构，来进行代码摘要生成

T. Ahmed等人利用LLM来理解代码功能，以零样本或通过上下文学习而无需任何参数更新的方式生成代码摘要

Konstas等提出了**Code-NN**方法，在机器翻译和文本摘要的基础上开发，将结构化的源代码扁平化为纯文本，并采用顺序模型来生成代码摘要

Xue 等人首次提出自动化代码摘要和代码生成的联合模型，通过将代码摘要和代码生成任务同时训练，提升模型理解自然语言和源代码的能力

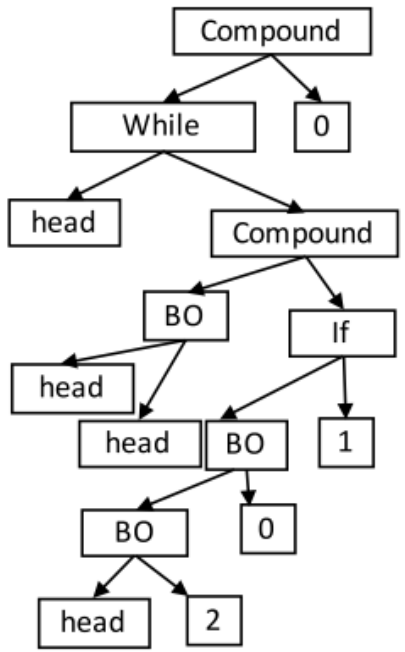


知识基础 AST&CFG表征

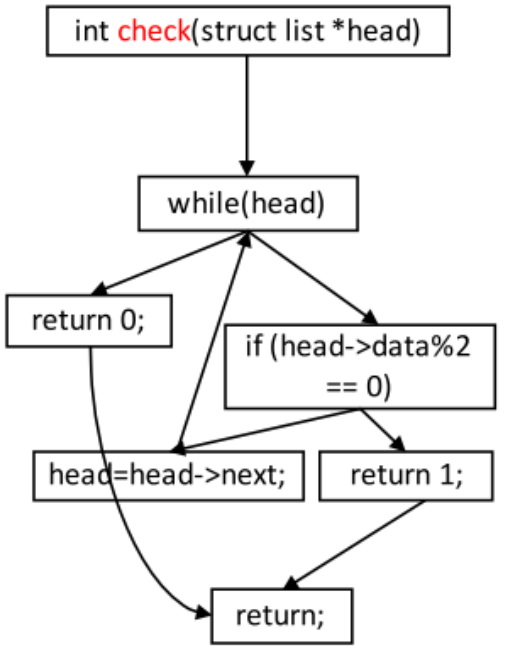
- 抽象语法树（**Abstract Syntax Tree，AST**）是一种用于表示源代码语法结构的树形数据结构
  - 节点：每个节点代表源代码中的一个语法元素，如变量、操作符、函数声明
- 控制流图（**Control Flow Graph，CFG**）是用于表示程序中基本块之间控制流关系的有向图
  - 基本块：包含一系列顺序执行的指令，不含跳转或分支
  - 边：基本块之间的控制流路径

```
1. // Verify whether an array of integers contains an even number.
2. int check(struct list *head){
3.    while(head){
4.        if (head->data%2==0)
5.            return 1;
6.        head = head->next;}
7.   return 0;}
```

(a) Code snippet and description



(b) Abstract syntax tree



(c) Control-flow graph

CoSS: Leveraging Statement Semantics for Code Summarization

CoSS TIPO

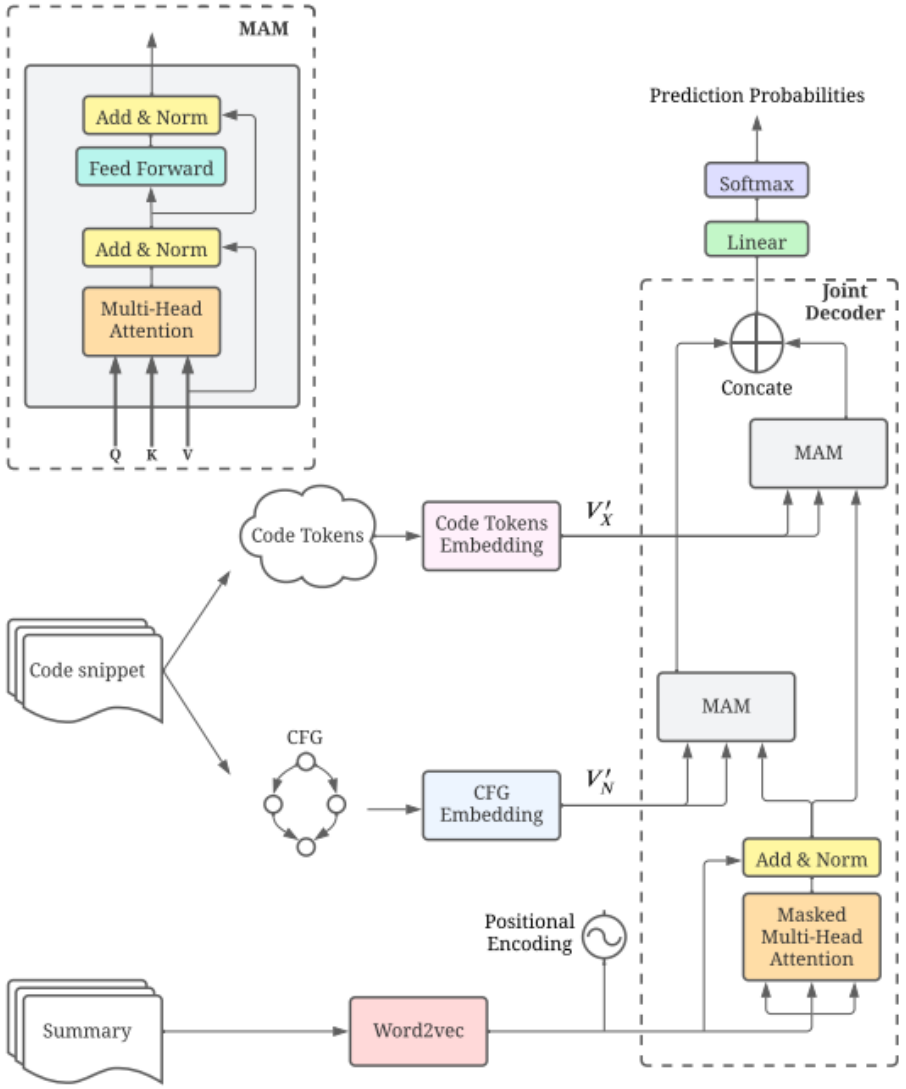
T	目标	分析代码语义和控制流特征，生成代码摘要
I	输入	代码数据集*3 (**Java, Python, Solidity**)
P	处理	**1.** 利用两个并行编码器从不同角度提取代码语义 **2.** 联合解码器生成代码摘要
O	输出	代码功能的自然语言描述

P	问题	**AST**对代码的基本单元分解过于细粒度
C	条件	每个代码片段可以解析为控制流图（**CFG**）
D	难点	如何有效提取代码语句的文本语义与控制流特征
L	水平	**IEEE TRANSACTIONS ON SOFTWARE ENGINEERING** （**2023 SCI**一区）

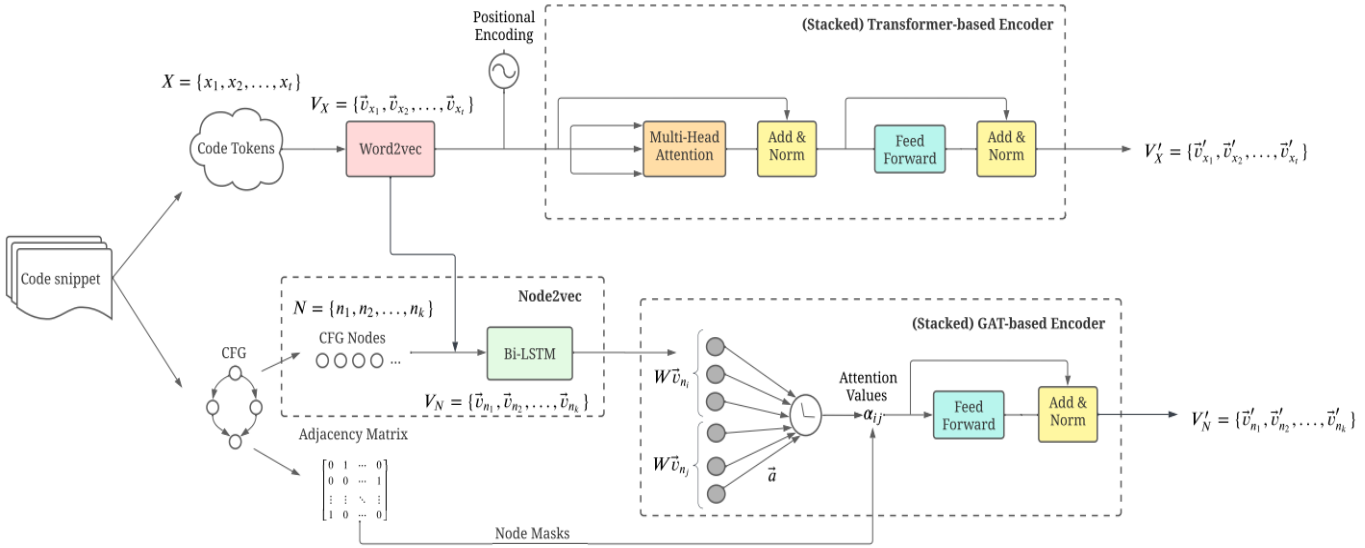
CoSS 算法核心

- 代码嵌入和编码器
  - 代码token嵌入与编码：从代码token中提取代码文本语义信息
  - CFG嵌入与编码：从CFG中提取代码结构语义信息

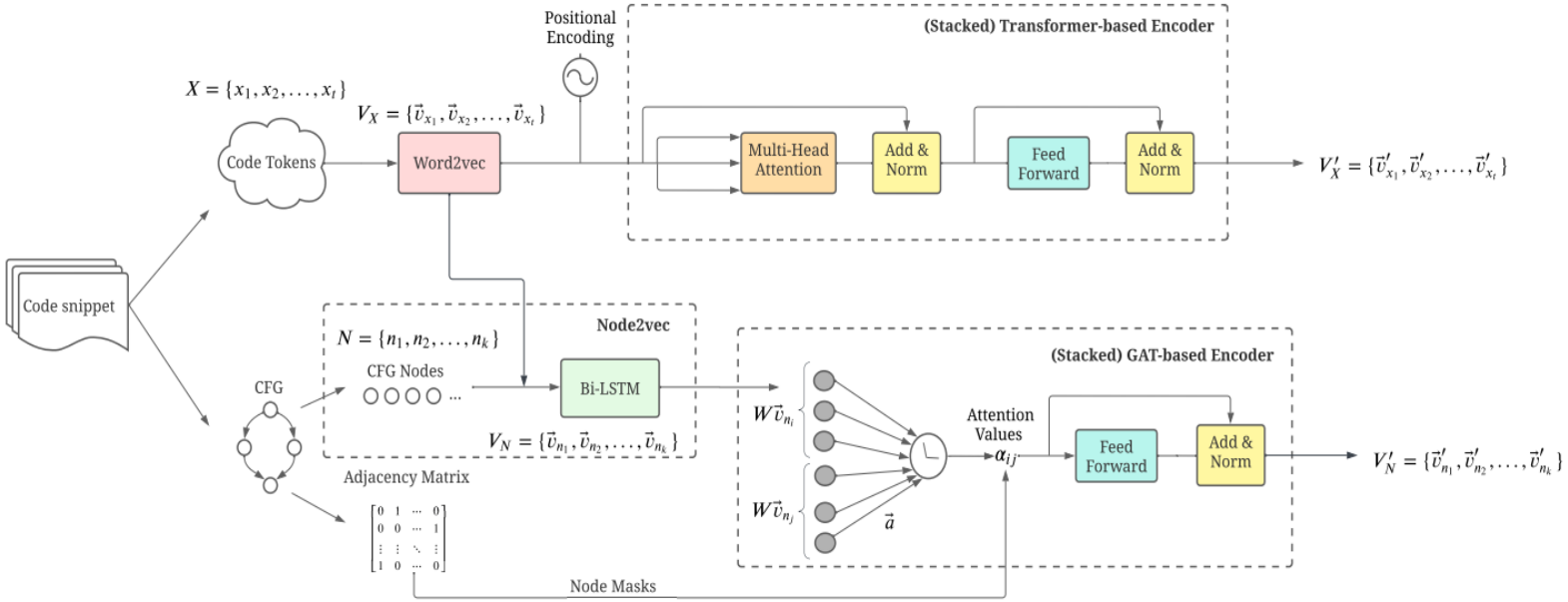
- 联合嵌入解码
  - 利用两个单独的多头注意力模块（MAM）捕获不同模态信息
  - 不同模态表征拼接，进行代码摘要生成



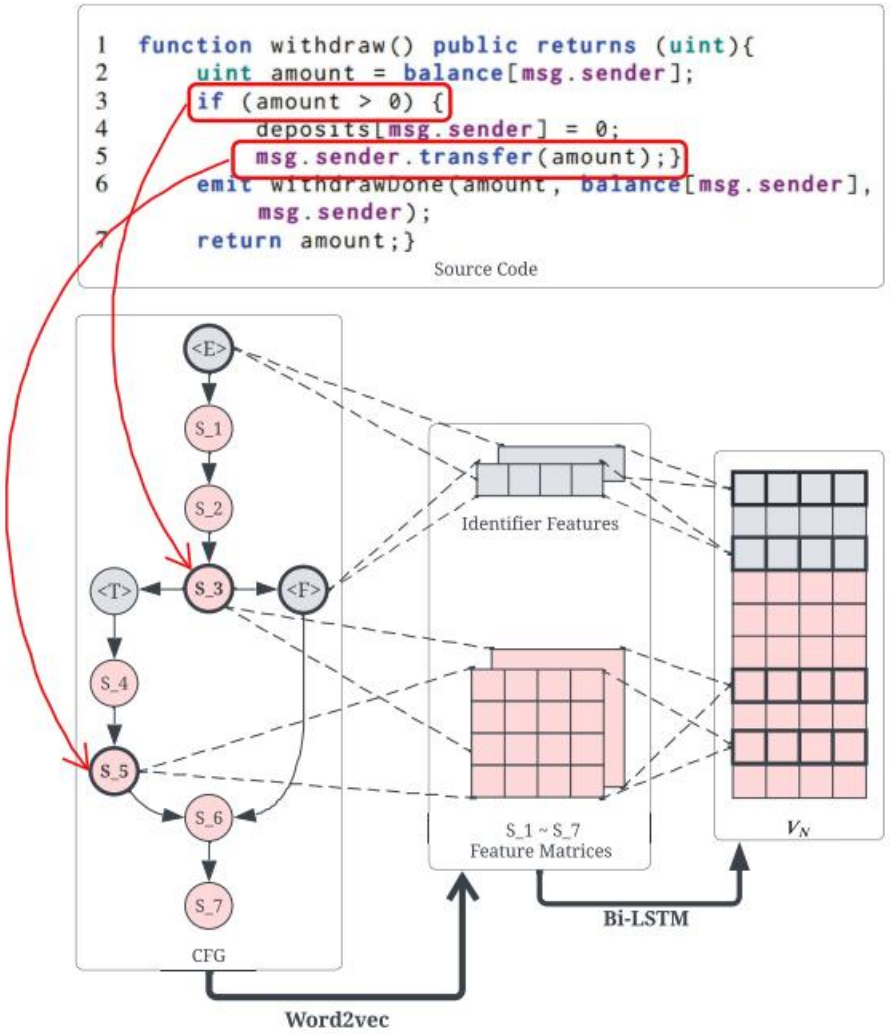
- 代码token嵌入与编码
  - 代码分词：以{.,’ ”: () ! - (space)}等符号为分隔符，将代码片段转化为token序列，即 $X = \{x_1, x_2, \ldots, x_t\}$
  - 向量初始化：利用Word2Vec将token序列$X = \{x_1, x_2, \ldots, x_t\}$表示为向量 $V_x = \{v_{x_1}, v_{x_2}, \ldots, v_{x_t}\}$，并将相对顺序信息注入标记向量
  - Transformer编码：利用多头注意力机制学习重要token向量，输出嵌入向量集合$V'_x = \{v'_{x_1}, v'_{x_2}, \ldots, v'_{x_t}\}$



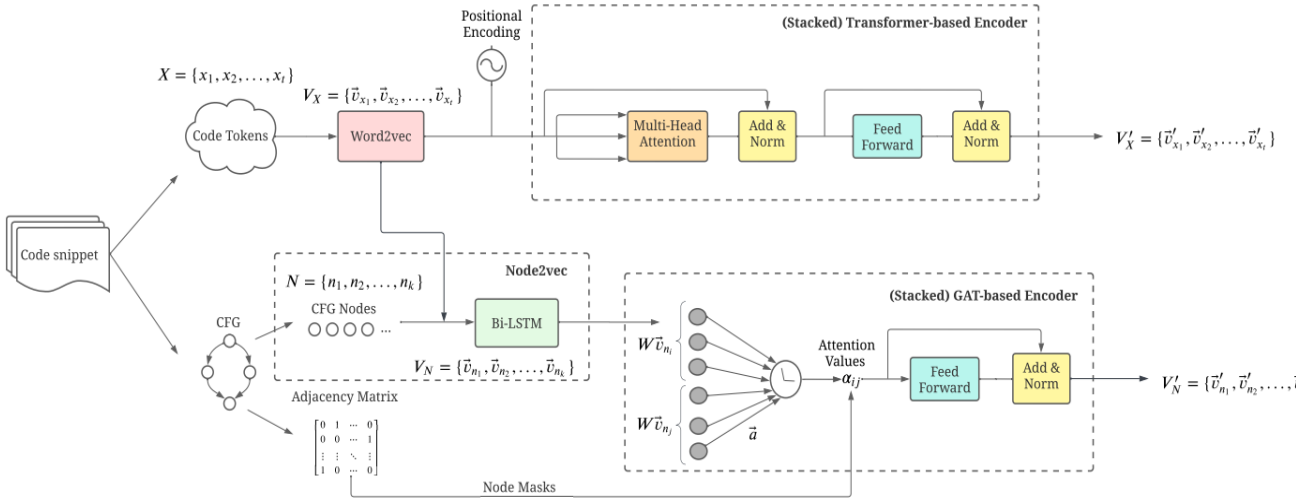
- **CFG嵌入与编码**
  - 目的：在聚合语句的控制流特征时学习语义信息
  - **node2vec** 模块：利用**Word2Vec**并添加**Bi-LSTM**层生成节点的初始化嵌入，学习每个**CFG**节点的语义特征
  - 基于图注意力网络（**GAT**）的编码器：将**CFG**转化为单类型边的有向图，利用**GAT**提取**CFG**的结构特征



- **node2vec** 模块
  - 对于长度为k的节点集合$N = \{n_1, n_2, \dots, n_k\}$，**Word2vec** 模块会滑动 m 个标记并生成 $m \times F$的向量矩阵$V_N = \{v_{n_1}, v_{n_2}, \dots, v_{n_k}\}$
  - 向量矩阵$V_N = \{v_{n_1}, v_{n_2}, \dots, v_{n_k}\}$输入 **Bi-LSTM** 层以捕获语句中单词之间的顺序依赖关系，并生成 $1 \times F$ 向量作为节点的初始嵌入



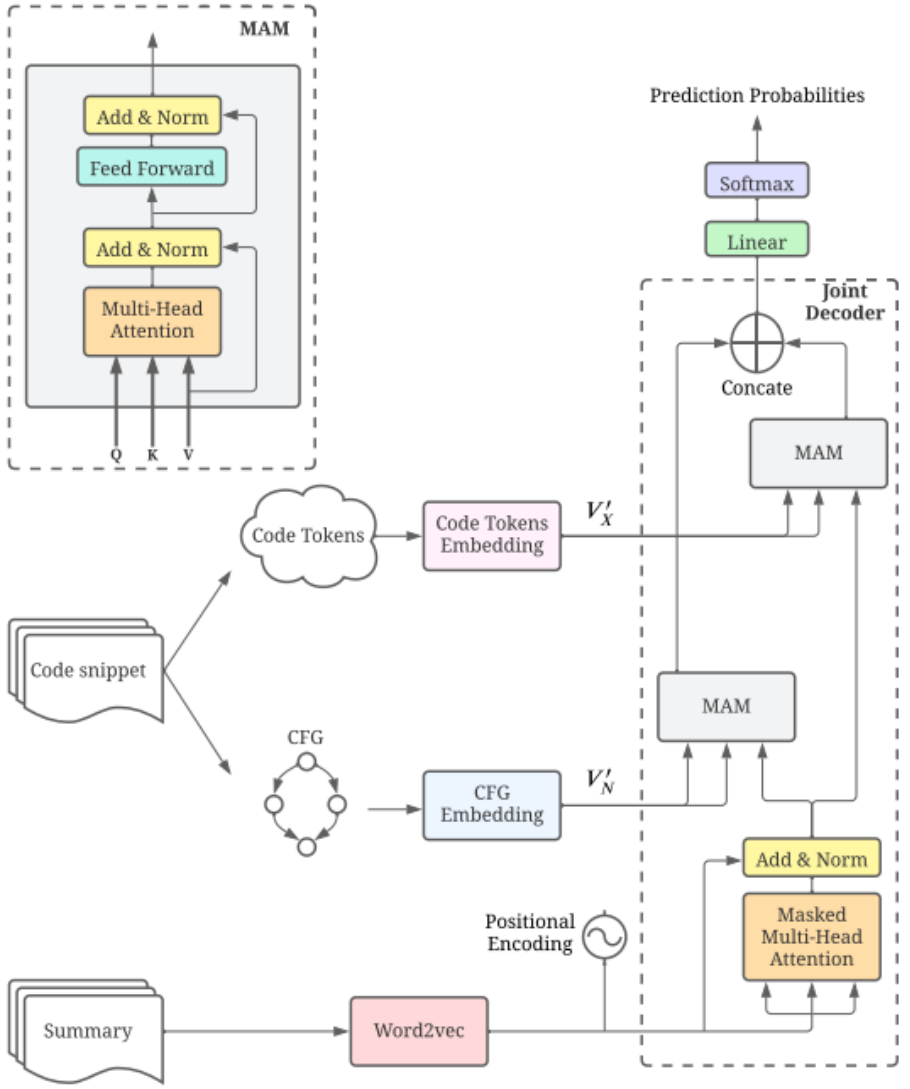
- 基于图注意力网络（GAT）的编码器
  - 将CFG转化为单类型边的有向图： True 和 False 边分别转换为标识符为 <T> 和 <F> 的节点，标识符 <E> 表示程序入口
  - 利用注意力机制，通过聚合邻居的特征来学习新的节点表示$V'_N = \{v'_{n_1}, v'_{n_2}, \dots, v'_{n_k}\}$



- $\alpha_{ij} = \frac{LeakyReLU(a^T[Wv_{n_i}||Wv_{n_j}])}{\sum_{k\in N_i} LeakyReLU(a^T[Wv_{n_i}||Wv_{n_k}])}$
- $v"_{n_i} = ||_{k=1}^{K}\sigma\left(\sum_{k\in N_i}\alpha_{ij}^k W^k v_{n_j}\right)$
- $v'_{n_i} = LayerNorm(v"_{n_i} + FFN(v"_{n_i}))$

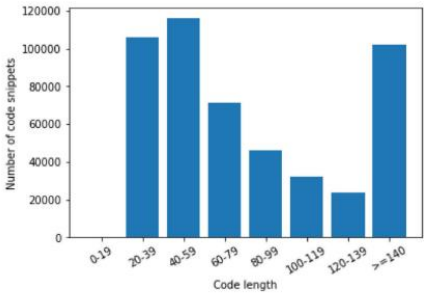
CoSS CFG嵌入和编码

- 联合嵌入解码
  - 从标签<SOS>开始生成，将V'_X和V'_N分别输入单独的多头注意力模块（MAM）来捕获注意力信息
  - 损失函数$Loss = -\frac{1}{|y|}\sum_{t=1}^{|y|}\log(P(y_t))$
  - 训练目标对下一个token预测最准确

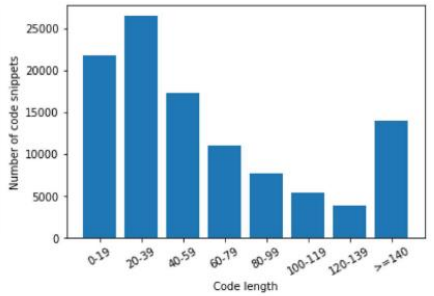


CoSS 实验设计

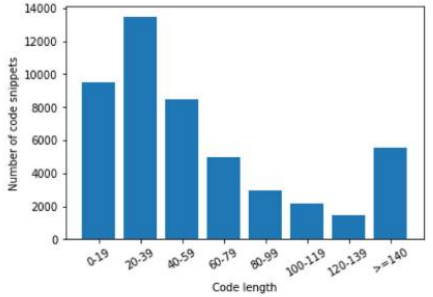
- 数据资源
  - 数据集：代码数据集*3（Java、Python、Solidity）
- 对比方法
  - 与9种方法进行比较，其中CAST将AST拆分为子树进行学习，CodeBERT 是一个基于Transformer的模型，CPG是AST、CFG与PDG的合并图
- 实验设置
  - 评价指标：BLEU-4 , ROUGE-L , METEOR , CIDER



(a) Length distribution of Java code



(b) Length distribution of Python code



(c) Length distribution of Solidity code

Approaches	Primary Technique	Modality
CODE-NN [7]	LSTM	Code Tokens
Code2seq [45]	LSTM	AST (Paths)
Hybrid-DRL [10]	Tree-LSTM, RL	Code Tokens, AST
RL-Guided [25]	HAN, RL	Code Tokens, AST, CFG
CoCoSum [46]	GRU, MRGNN	Code Tokens, AST, UML
CAST [13]	Transformer, RvNN	Code Tokens, AST (Splitted)
SG-Trans [41]	Transformer	Code Tokens, DFG
SmartDoc [43]	Transformer, Pointer, TL	Code Tokens
CodeBERT [15]	pre-trained Transformer	Code Tokens
HGNN [44]	Hybird GNN, IR	CPG

CoSS 实验设计

- **BLEU-4**

$$BLEU = BP \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

$$BP = \begin{cases} 1 & c > r \\ \exp\left(1 - \frac{r}{c}\right) & c \leq r \end{cases}$$

- **ROUGE-L**

$$P_{lcs} = \frac{LCS(X,Y)}{c}$$

$$R_{lcs} = \frac{LCS(X,Y)}{r}$$

$$F_{lcs} = \frac{(1+\beta^2) \cdot P_{lcs} \cdot R_{lcs}}{R_{lcs} + \beta^2 \cdot P_{lcs}}$$

- **METEOR**

$$METEOR = (1 - Pen) \cdot F_{mean}$$

$$F_{mean} = \frac{PR}{\alpha P + (1-\alpha)R}$$

$$Pen = \gamma \cdot \left(\frac{ch}{m}\right)^{\beta}$$

- **CIDER**
  - 计算**TF-IDF**权重
  - 构建**n-gram**
  - 加权**n-gram**的匹配
  - 计算相似度

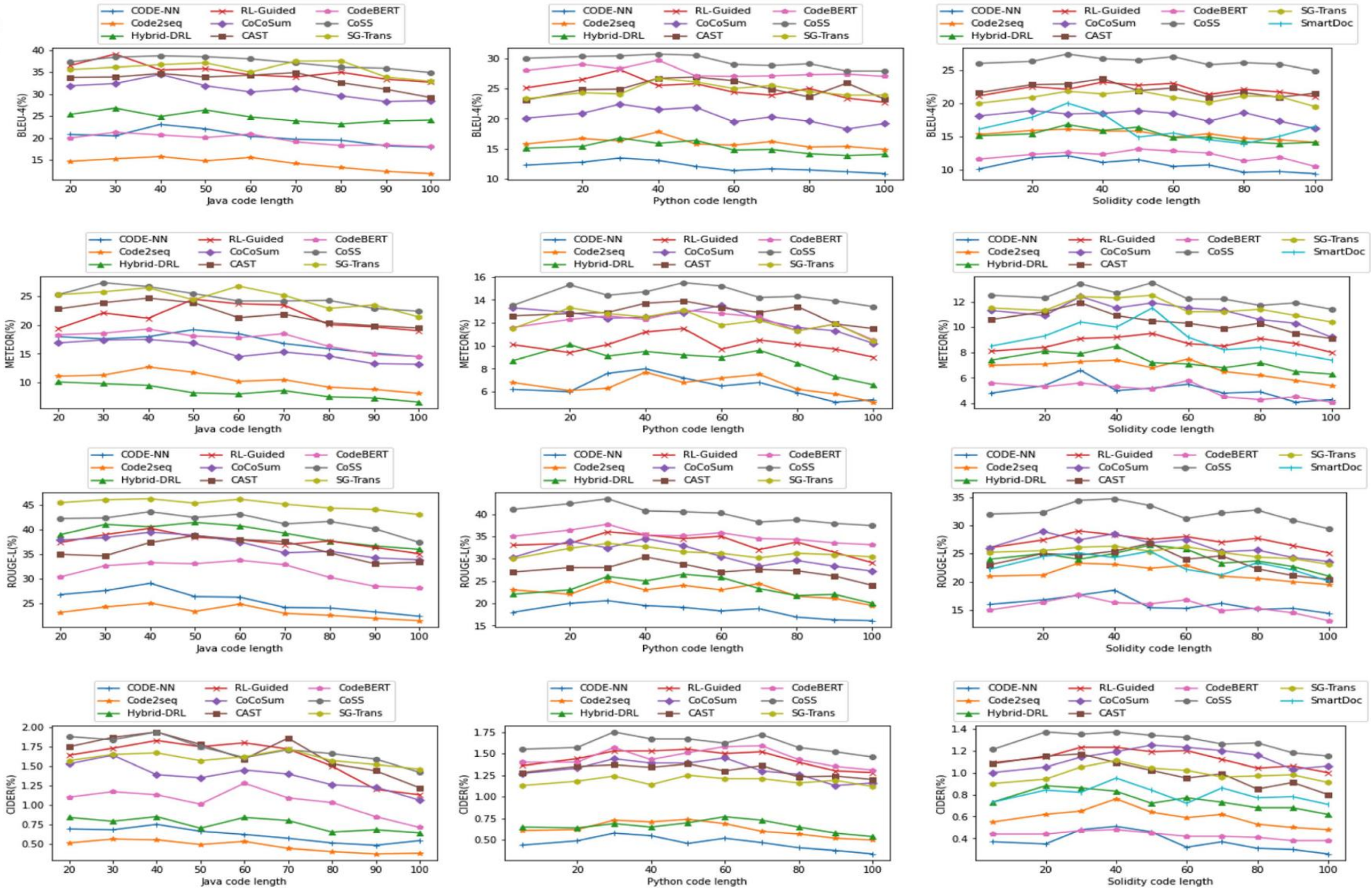
CoSS 对比实验

- 实验结果
  - **Code-NN仅利用代码token，难以捕获深层次的代码特征**
  - **CodeBERT 仅在 Python 中表现出良好的性能**
  - **CoSS与HGNN在HGNN作者构建的C语言上进行比较**
    - 性能相当，但由于**CPG**的复杂，导致训练时间和硬件资源的消耗较高
    - 不支持在**Solidity**等编程语言

Model	C		
	B	M	R
HGNN	**14.01**	14.50	30.89
CoSS	13.84	**14.63**	**32.25**

Model	Java				Python				Solidity			
	B	M	R	C	B	M	R	C	B	M	R	C
CODE-NN	19.33	18.04	26.77	0.58	12.95	5.85	18.47	0.44	10.42	4.57	16.65	0.37
Code2seq	13.25	10.07	23.56	0.45	16.32	6.73	23.35	0.61	15.16	6.98	21.00	0.55
Hybrid-DRL	25.11	9.29	39.13	0.75	14.95	8.37	22.01	0.65	14.39	7.87	24.76	0.73
RL-Guided	35.10	20.01	37.19	1.68	26.18	10.43	34.12	1.36	22.08	8.55	26.30	1.09
CoCoSum	30.01	16.04	37.83	1.46	21.13	13.06	30.48	1.27	18.64	10.88	26.65	0.97
CAST	32.15	22.78	35.59	1.72	24.89	12.62	27.31	1.28	21.78	10.85	23.94	0.98
SG-Trans	35.89	23.67	**44.18**	1.54	25.06	12.86	31.13	1.22	21.91	11.01	25.87	1.02
CodeBERT	19.89	17.33	29.48	1.04	28.58	11.54	35.65	**1.40**	11.12	5.50	15.48	0.43
SmartDoc	-	-	-	-	-	-	-	-	16.31	8.57	22.65	0.88
CoSS	**36.70**	**25.74**	40.41	**1.79**	**30.26**	**14.64**	**41.01**	**1.40**	**26.59**	**12.07**	**32.34**	**1.21**

CoSS 对比实验


CODE-NN
Code2seq
Hybrid-DRL
RL-Guided
CoCoSum
CAST
CodeBERT
CoSS
SG-Trans
SmartDoc
BLEU-4(%)
METEOR(%)
ROUGE-L(%)
CIDER(%)
Java code length
Python code length
Solidity code length

CoSS 消融实验

- 消融实验
  - 验证三种不同输入模式（**token**、**CFG**和**AST**）下的各组合的模型性能
  - 验证各核心组件的有效性
    - **CoSS-T**：使用**LSTM**替换**Transform**作为**token**的编码器
    - **CoSS-B**：利用平均池化替换**Bi-LSTM**，获取**CFG**节点嵌入
    - **CoSS-G**：将**CFG**遍历为序列获取节点嵌入
    - **CoSS-J**：采用**LSTM**作为解码器

Modality/Variant	Java				Python				Solidity			
	B	M	R	C	B	M	R	C	B	M	R	C
Code tokens	29.27	20.16	31.45	1.34	20.44	9.18	23.78	0.95	18.62	8.80	21.77	0.79
AST	22.13	13.52	26.80	0.92	14.32	5.53	19.35	0.53	12.11	4.98	17.04	0.49
CFG	24.12	12.57	30.11	1.05	18.83	7.79	22.55	0.80	17.41	7.08	20.56	0.70
Code tokens + AST	33.45	22.82	36.99	1.51	23.48	11.10	26.81	1.17	21.08	9.52	23.15	0.95
Code tokens + CFG (CoSS)	36.40	**25.74**	40.41	**1.79**	30.26	**14.64**	41.01	**1.40**	**26.59**	12.07	32.34	**1.21**
AST + CFG	31.05	23.73	33.00	1.22	22.81	10.45	23.99	1.08	19.78	9.85	22.94	0.88
Code tokens + AST+ CFG	**36.91**	25.07	**42.06**	1.72	**30.65**	14.17	39.83	1.37	25.88	**12.97**	**34.23**	1.15
CoSS-T	29.12	18.79	29.08	1.21	21.33	8.97	24.95	1.02	19.80	7.45	20.04	0.80
CoSS-B	34.44	23.26	37.14	1.66	27.35	13.57	33.13	1.22	23.42	10.01	29.25	1.26
CoSS-G	30.97	22.98	33.39	1.40	24.20	11.02	25.34	1.18	20.76	9.95	24.94	1.02
CoSS-J	28.09	15.45	27.61	0.97	18.68	8.59	20.17	0.64	16.61	8.10	19.04	0.60

CoSS 训练效率

- 训练效率
  - **CodeBERT 和 SmartDoc 是仅需微调的模型，并且是单模态输入，速度最快**
  - **CoSS-T 花费的时间是 CoSS 的两倍，基于 Transformer 的编码器比其他组件对 CoSS 的训练效率贡献更大**
  - **使用 CFG 的模型在训练中明显比使用 AST 的模型效率更高**

THE AVERAGE TIME CONSUMPTION IN TRAINING (MINS/EPOCH)

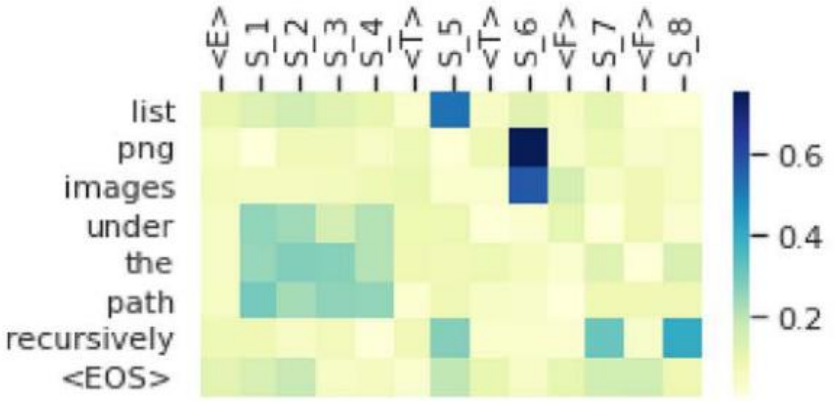
Model/Modality/Variant	Java	Python	Solidity
CODE-NN	95	24	14
Code2seq	101	27	15
Hybrid-DRL	178	44	24
RL-Guided	194	52	30
CoCoSum	171	48	27
CAST	110	31	18
SG-Trans	103	26	15
CodeBERT	71	19	11
SmartDoc	-	-	10
CoSS	83	21	13
Code Tokens	56	16	8
AST	94	21	11
CFG	45	11	6
Code tokens + AST	108	28	14
AST + CFG	99	26	13
Code tokens + AST + CFG	152	39	21
CoSS-T	40	95	23
CoSS-B	68	17	8
CoSS-G	80	22	13
CoSS-J	91	24	13

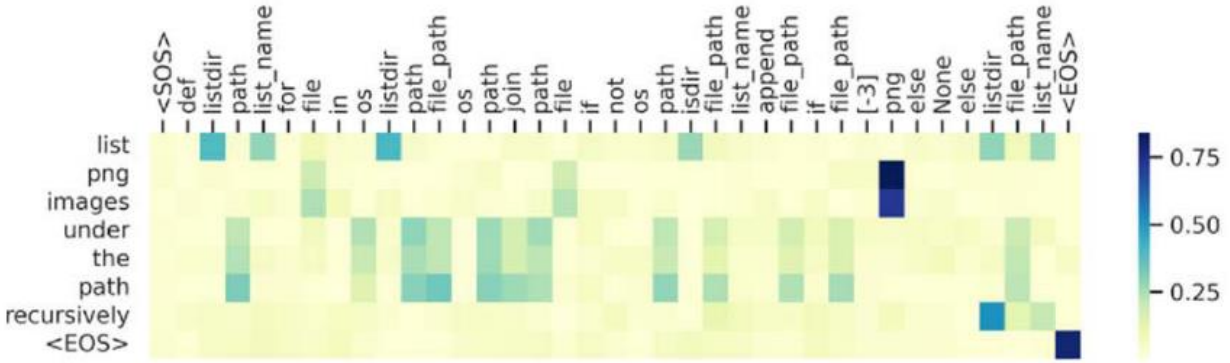
CoSS 案例分析

- 案例分析——**Python**

```python
def listdir(path, list_name):
    for file in os.listdir(path):
        file_path = os.path.join(path, file)
        if not os.path.isdir(file_path):
            list_name.append(file_path)
            if file_path[-3:] == "png"
            else None
        else: listdir(file_path, list_name)
```

Ground Truth	Find all png images under the path recursively.
CODE-NN	append png file name to path.
Code2seq	append png files path to the list.
Hybrid-DRL	list all the png files within the path.
RL-Guided	list all png images in the folder and add to list.
CoCoSum	list file path end with png.
CAST	add png images to the list.
SG-Trans	list all png files under file path.
CodeBERT	find png images under the path.
CoSS	list png images under the path recursively





Learning to Generate Structured Code Summaries from Hybrid Code Context

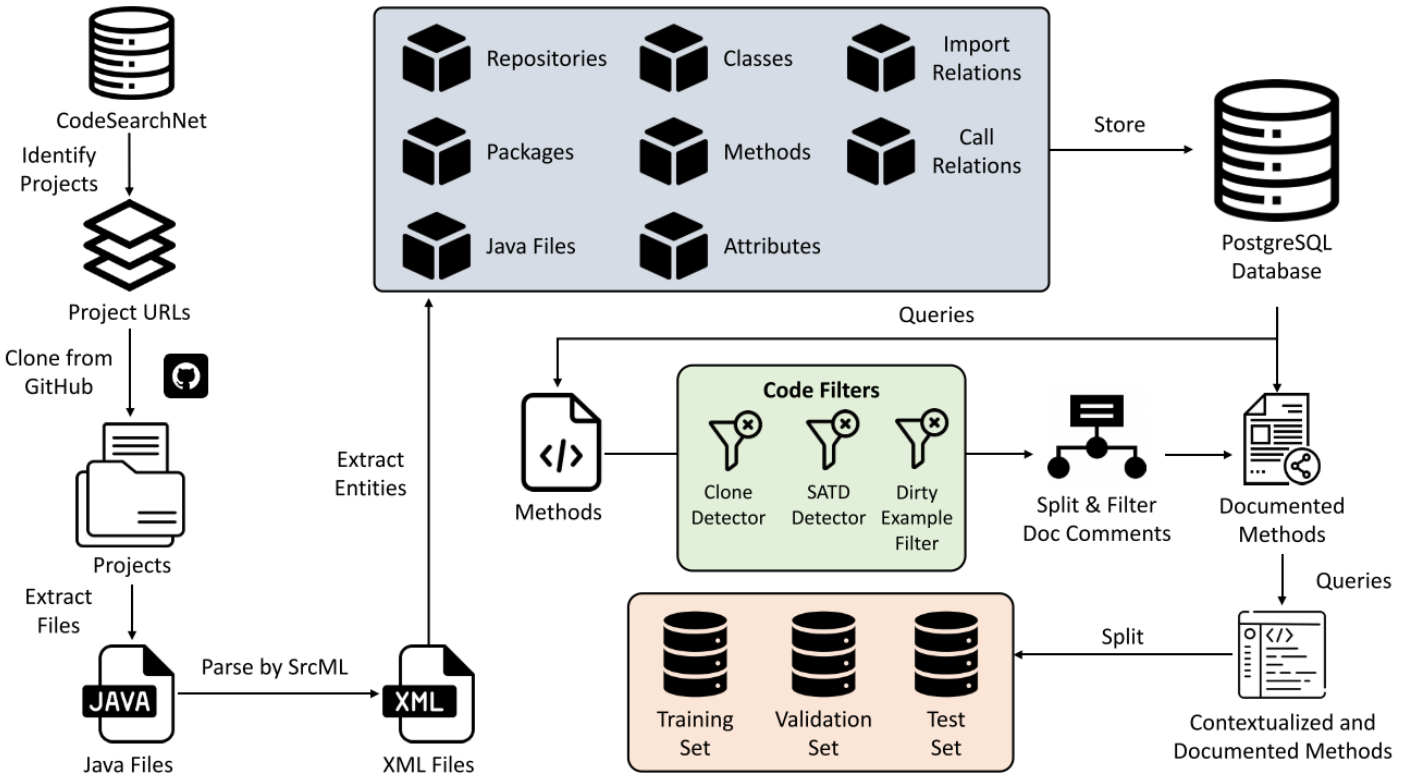
StructCodeSum TIPO

T	目标	建立基于**LLM**的方法和适合不同场景的轻量级方法
I	输入	代码数据集、**code Llama**
P	处理	**1.** 代码摘要数据集构建 **2.** 结构化代码摘要（包括**LLM**与轻量级）
O	输出	详细的代码功能以及输入输出等相关信息的自然语言描述

P	问题	现有的基于深度学习的模型通常旨在为代码生成简短的功能描述
C	条件	代码库具有相应的使用说明文档
D	难点	缺乏对不同类型代码上下文的利用和分析
L	水平	**IEEE TRANSACTIONS ON SOFTWARE ENGINEERING** **（2024 SCI一区）**

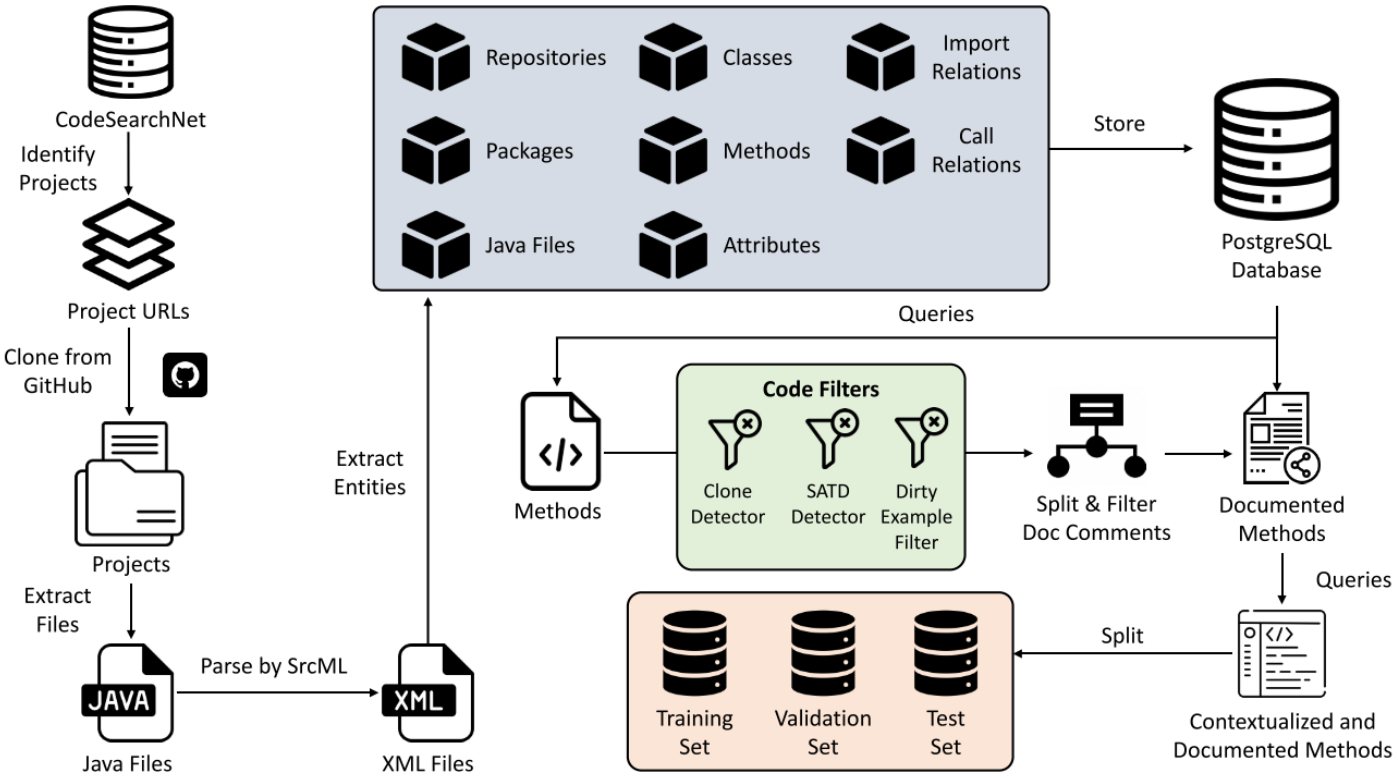
StructCodeSum 数据集构建

- 数据集构建
  - 广泛使用的代码摘要数据集不包含程序上下文以及每个目标代码的不同类型的摘要内容
  - 质量良莠不齐：没有按规范的注释文档超过25%
  - 构建过程
    - 数据收集
    - 文档拆分
    - 数据预处理
    - 数据集统计



StructCodeSum 数据集构建

- 数据收集与过滤
  - 空文档删除
  - 相近算法删除
  - 自认技术债务（SATD）方法删除
  - 测试代码删除
- 文档拆分
  - 代码示例与SATD删除
  - 返回描述
  - 异常描述
  - 使用说明描述
  - 概括性描述



StructCodeSum 数据集构建

- 数据预处理
  - 文档选择（函数、参数、使用描述、异常描述、返回描述）
  - 符号替换
- 数据集统计
  - 总计620,225<方法、上下文、带注释的文档>三元组的数据集
  - 异常和使用描述由于其数据量和内容不适合模型训练和评估

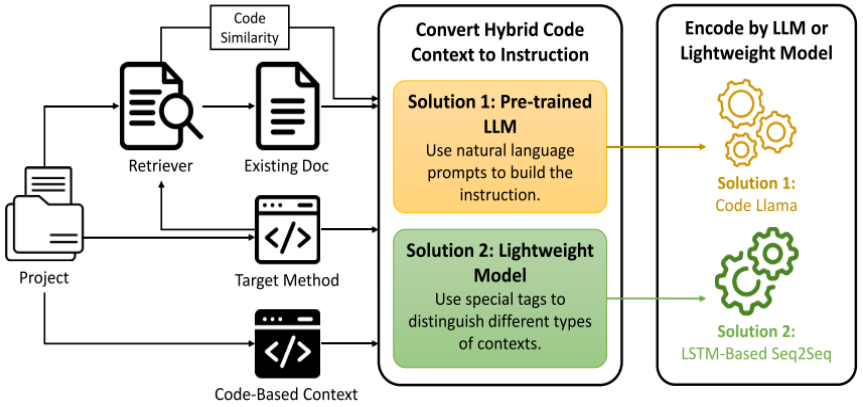
Name	Count
# Documentation comments	620,225
# Function descriptions	464,143
# Parameter descriptions	388,952
# Methods with parameter descriptions	258,313
# Return descriptions (identified by rules)	334,300 (35,156)
# Exception descriptions (identified by rules)	61,359 (1,755)
# Methods with exception descriptions	44,749
# Usage descriptions	21,237
# Total words	11,197,594

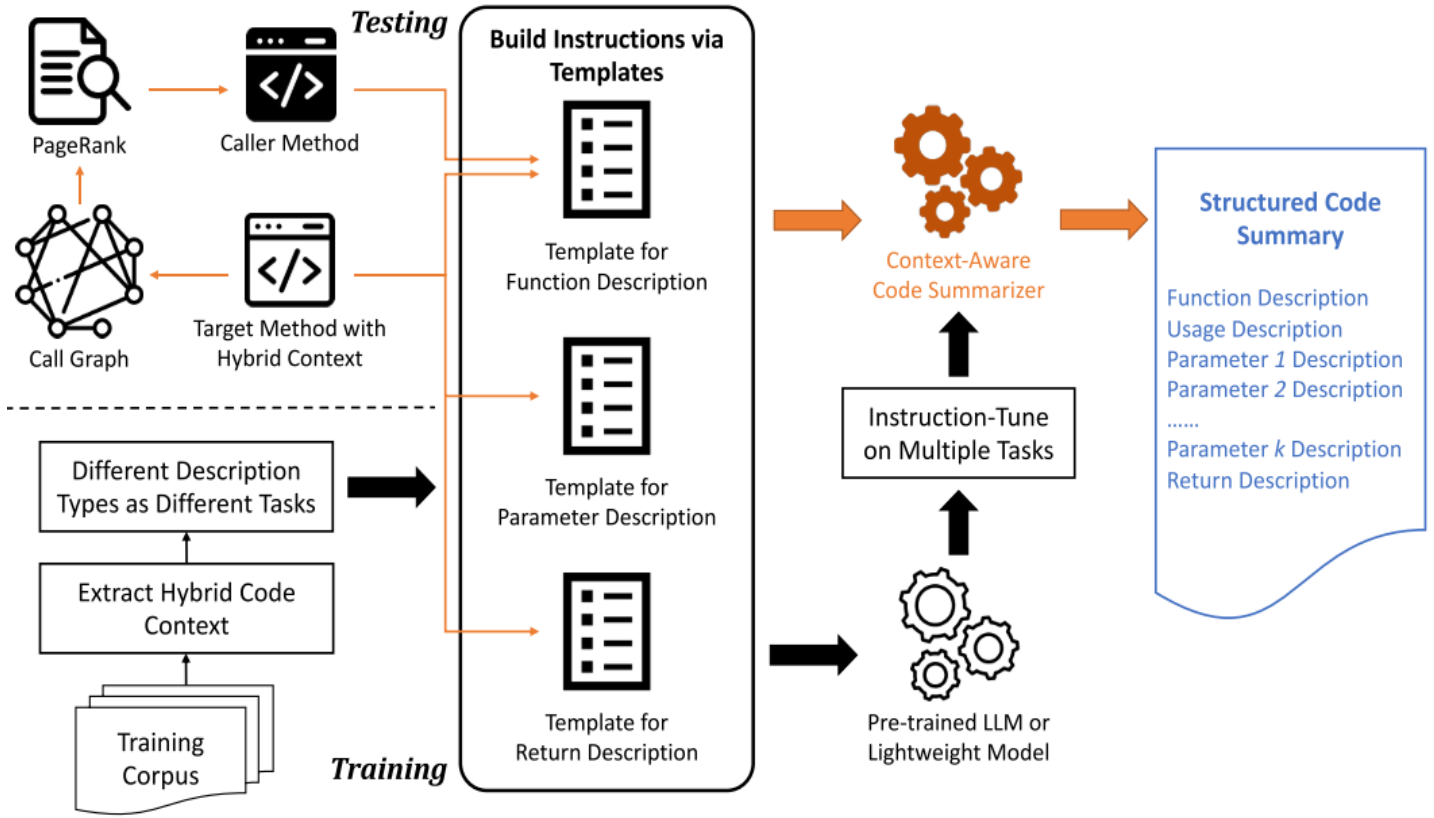
Target Methods			Contexts of the Target Methods				
# Methods	# Tokens	# Lines	# Projects	# Classes	# Attributes	# Class Methods	# Calls
620,225	40,123,925	6,226,876	4,244	153,659	823,503	2,067,575	1,308,471

Splitting Strategies		# Target Methods	# Function Descriptions	# Parameter Descriptions	# Return Descriptions
Mixed-project	Training	496,180	371,212	310,610	267,530
	Validation	62,022	46,307	39,312	33,351
	Testing	62,023	46,624	39,030	33,419
Cross-project	Training	496,467	371,305	306,813	271,831
	Validation	62,068	46,483	41,547	31,883
	Testing	61,690	46,355	40,592	30,586

StructCodeSum 模型结构

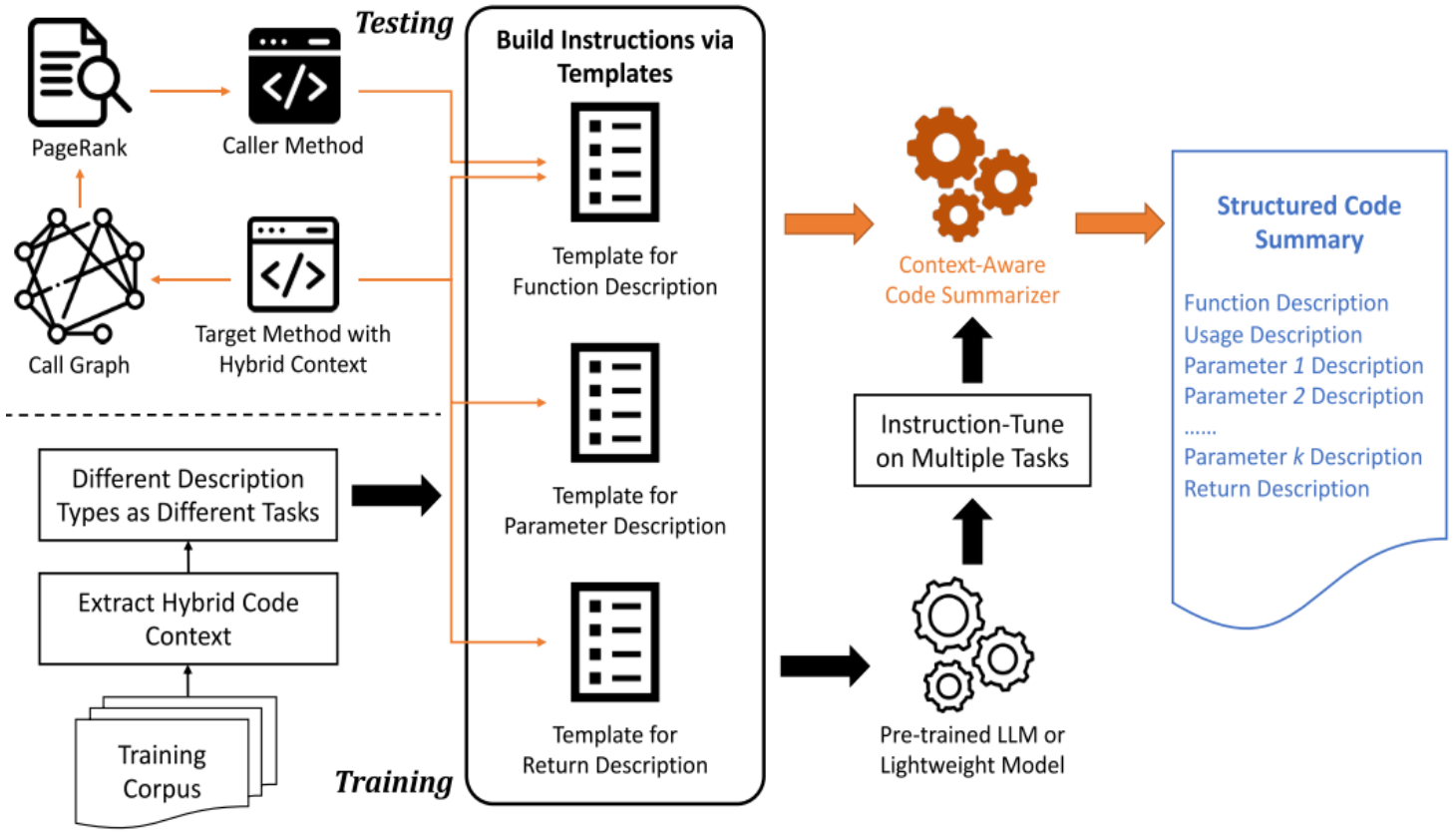
- 结构化代码摘要
  - 基于LLM的方法
  - 轻量级（基于LSTM）的方法
  - 输入模型的上下文
    - 基于代码的上下文
    - 基于文档的上下文





StructCodeSum 模型结构

- 基于代码的上下文
  - 路径上下文：包名称加上其封闭类的名称
  - 类上下文：类中属性和同级方法
- 基于文档的上下文
  - 项目中与目标方法最相似的记录方法
    - 词频-逆文档频率（TF-IDF）表示为向量
    - $\cos(r^i, r^j) = \frac{r^i \cdot r^j}{||r^i|| ||r^j||} \in [-1,1]$
    - $\text{sim}(x^i, x^j) = 1 - \frac{dis(x^i \cdot x^j)}{\max(|x^i|, |x^j|)}$



StructCodeSum 模型结构

- **StructCodeSum**
  - **LLM指令微调**
    - 损失函数$L(\theta) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T_i} logp_\theta(y_t^i|y_{<t}^i, x^i)$
    - 相关代码阈值分别为：**0.33**、**0.48**和 **0.63**
  - 轻量级微调
    - 分别针对函数、参数和返回描述分别微调，获得三个预测器
  - 多任务方式生成
    - 文档不完整或不具备返回语句等
    - 文档中的所有描述导致输入更长

System Prompt (Fixed)

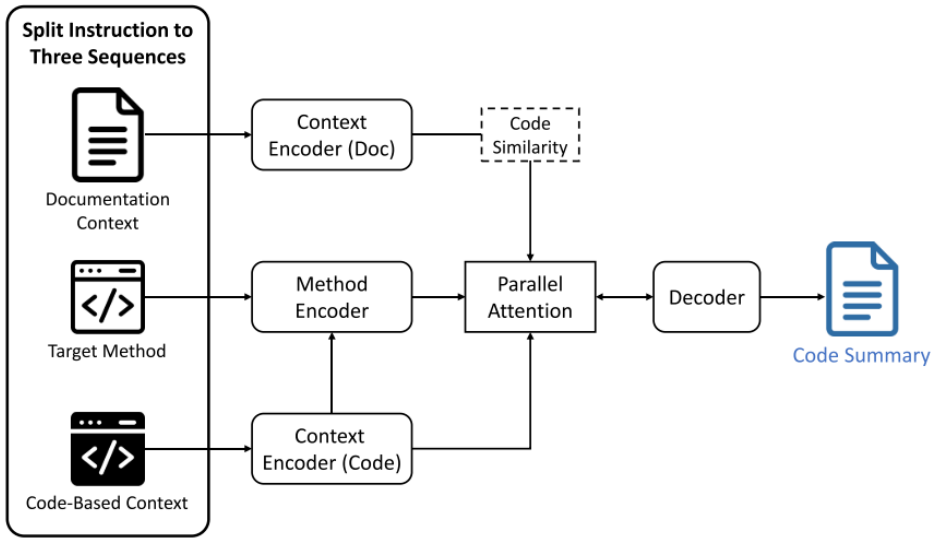
You are a helpful, precise, detailed, and concise artificial intelligence assistant with deep expertise in Java programming languages. You excel at understanding and summarizing code effectively. In this task, you are required to read the provided Java method and its context. Based on the given requirements, generate a specific type of summary. This summary can be a description of the method's functionality, a description of the return value, or a description of a parameter.
Requirements:
The generated summary should accurately reflect the object to be described.
The generated summary should be concise and easy to understand.
If context is provided, you need to understand the code based on the context and generate a description.
Below is a method and its context. Your task is to provide the summary based on its target and format.

User Prompt

Context:
Sibling methods signatures:
<sibling method signature - 1>
<sibling method signature - 2>
…
Class Attributes:
<attribute - 1>
<attribute - 2>
…
Path:
Project name : <project name>
Package name : <package name>
Class name : <class name>
Class Signature : <class signature>
Relevant documentation (Relevance level : {low, medium, high, very high})
<Relevant document>
Method:
<code>
Summary target:
{Functionality description, Return value description, Parameter description of <parameter name>}
Response (one sentence):

Output

<summary>



StructCodeSum 实验设计

- 数据资源
  - 数据集：代码数据集*1（Java）
- 对比方法
  - 与7种方法进行比较：**CodeNN、AST-AttendGRU、 AST-AttendGRU + FC 、 AST-AttendGRU + PC 、Rencos、Re^{2}Com、CodeBERT**
    - **AST-AttendGRU是多输入模型，包括token和扁平化的AST，即SBT**
    - **AST-AttendGRU + FC引入方法级上下文作为补充，随机截取前25个词**
    - **AST-AttendGRU + PC引入项目级上下文作为补充，随机选取最多10个**
    - **Rencos是第一个将基于检索和基于 NMT 的方法结合在代码摘要中的工作**
    - **Re^{2}Com从目标代码、SBT、相似代码片段、示例注释四个角度分析**
- 实验设置
  - 评价指标：**BLEU-4 , ROUGE-L , METEOR , SIDE**

StructCodeSum 代码上下文影响

- **RQ1：不同代码上下文的影响**
  - 返回和参数描述的得分远高于函数描述
  - 在混合项目分割下，**Hybrid**上下文效果最佳，**Docs**次之
  - **Path**的贡献比类上下文大，一个包中包含了一组具有相似功能、业务或代码组件的程序
  - 数据集中删除了相似代码，不同的项目很难具备相似的代码模式和词汇，跨项目性能下降
  - 轻量级模型没有在大规模项目上进行预训练，难以应对不同的编码风格、结构和上下文，除了返回描述之外，基于代码的上下文几乎无法提高性能，混合显示的性能略低于文档

Description	Context	Mixed-project			Cross-project		
		BLEU	METEOR	ROUGE-L	BLEU	METEOR	ROUGE-L
Function	None	20.73	24.15	45.81	13.36	20.58	40.83
	Path	24.67	26.35	49.61	14.21	21.26	41.92
	Attributes	21.30	24.68	46.68	13.38	20.66	40.97
	Siblings	22.01	25.12	47.41	13.72	20.99	41.37
	Documents	41.17	34.11	62.62	23.20	23.76	45.40
	Hybrid	**42.29**	**35.01**	**63.80**	**24.65**	**24.94**	**46.87**
Return	None	31.58	28.51	51.01	16.99	22.16	43.45
	Path	35.37	31.46	55.93	18.82	23.79	45.82
	Attributes	31.96	29.29	51.97	17.01	22.39	43.59
	Siblings	32.93	30.16	53.49	17.68	23.03	44.70
	Documents	49.79	37.77	67.69	27.60	25.57	47.36
	Hybrid	**50.26**	**38.57**	**68.57**	**29.32**	**26.81**	**50.30**
Parameter	None	30.63	27.56	55.16	23.88	22.08	47.83
	Path	33.73	29.59	58.95	25.00	22.73	48.63
	Attributes	30.75	27.80	55.21	23.98	22.00	48.26
	Siblings	31.48	28.29	56.14	24.36	22.35	48.42
	Documents	49.18	37.18	67.38	35.66	28.17	53.87
	Hybrid	**49.71**	**37.59**	**67.44**	**36.98**	**29.01**	**55.11**

Description	Context	Mixed-project			Cross-project		
		BLEU	METEOR	ROUGE-L	BLEU	METEOR	ROUGE-L
Function	None	22.50	22.84	46.97	11.21	16.95	36.53
	Path	29.83	26.51	52.84	11.22	16.67	35.88
	Attributes	27.59	25.31	50.82	11.19	16.88	36.29
	Siblings	28.15	25.56	51.57	11.35	16.85	36.46
	Docs	37.31	30.37	56.44	**23.49**	**23.20**	45.96
	Hybrid	**41.00**	**32.37**	**60.42**	23.16	23.04	**46.08**
Return	None	35.29	28.30	54.98	16.13	17.66	38.94
	Path	41.84	32.37	61.54	17.34	18.56	40.95
	Attributes	39.40	30.88	59.01	17.02	18.07	40.19
	Siblings	41.11	31.81	60.38	16.63	17.84	39.83
	Docs	47.06	34.82	64.55	**28.65**	**24.94**	**51.14**
	Hybrid	**49.51**	**36.41**	**66.93**	28.11	24.64	50.43
Parameter	None	30.20	25.44	52.08	19.76	19.21	42.14
	Path	36.88	29.05	58.07	19.69	19.12	42.06
	Attributes	34.63	27.69	55.68	19.92	19.56	42.33
	Siblings	36.93	29.25	57.72	19.55	19.09	41.80
	Docs	44.49	32.76	62.04	**33.09**	**26.55**	**53.18**
	Hybrid	**47.10**	**34.47**	**64.81**	33.02	26.47	53.02

StructCodeSum 对比实验

- **RQ2：与最先进模型的比较**
  - **StructCodeSum (Lightweight) 与Code Llama不相上下，但仅包含44M参数量**
  - **AST-AttendGRU + FC由于随机选择上下文，导致上下文的不确定性**
  - **相比于Rencos 和Re^{2}Com， StructCodeSum从当前项目中查找相关文档，优于从训练数据中检索，并且跨项目设置下，难以检索相似代码或文档**

Model	Mixed-Project				Cross-Project			
	BLEU	METEOR	ROUGE-L	SIDE	BLEU	METEOR	ROUGE-L	SIDE
CodeNN	18.86	19.92	40.85	74.09	8.55	13.81	30.22	75.05
AST-AttendGRU	18.43	20.83	41.91	81.38	8.48	14.66	30.77	83.25
AST-AttendGRU + FC	20.86	22.03	43.49	80.09	9.74	16.10	33.67	82.53
AST-AttendGRU + PC	17.83	20.12	41.14	81.08	9.78	16.21	33.75	81.93
Rencos	23.57	23.73	48.13	81.92	11.56	17.37	37.25	83.42
Re^{2}Com	29.10	26.08	50.86	79.39	11.00	16.18	34.64	82.08
CodeBERT	27.84	25.82	52.25	72.40	12.73	18.03	38.67	82.57
Code Llama (Prompt)	21.83	23.16	42.40	83.22	23.20	23.76	45.40	84.83
StructCodeSum (Lightweight)	41.84	32.87	61.01	83.20	23.88	23.69	**46.90**	84.74
StructCodeSum (LLM)	**42.29**	**35.01**	**63.80**	**83.92**	**24.65**	**24.94**	46.87	**85.07**

StructCodeSum 人工评判

- **RQ3：人工评判**
  - 信息性：摘要所涵盖的代码中重要内容的比例
  - 自然性：摘要的语法准确性和流畅性
  - 从跨项目测试集中随机抽取**400**个方法，获得不同方法生成的摘要，并由**24**名计算机科学领域的开发人员或研究生进行评判，分数为从 **1** 到 **5** 的整数

Approach	Naturalness	Informativeness
Human-Written	4.78	4.20
Code Llama (Prompt)	4.55	3.38
StructCodeSum (LLM)	4.61	3.72

Example 1		

Source Code	`@Override`
	`public void shutdown() {`
	`runner.set(false);`
	`try {`
	`threadA.join();`
	`if (threadB != null)`
	`threadB.join();`
	`} catch (Exception e) {}`
	`CloseHelper.quietClose(driver);`
	`try {`
	`Thread.sleep(500);`
	`} catch (Exception e) {}`
	`}`	

Context	Package Name	`org.nd4j.parameterserver.distributed.transport`
	Class Name	`BaseTransport`
Summary	Human-Written	This method stops transport system.
	CodeBERT	Shutdown the client.
	Code Llama (No Context)	Shutdown the thread.
	StructCodeSum (LLM)	Shutdown the transport.

StructCodeSum 词汇分布讨论

- **Discussion**：代码摘要词汇分布
  - set是**token**序列的集合，x是目标方法，y是代码描述，q是代码上下文，d是文档上下文
  - 目标代码l_x的平均值约为**36%**，表明摘要中的大多数标记没有出现在相应的方法中
  - 文档上下文l_d与代码上下文l_{sib}占比高于l_x
  - 当l_x低于**25%**时，D_h超过**50%**，这意味着摘要中的一半标记只能在代码上下文中找到

Definition	Explanation
$I_x = \frac{\lvert set(y) \cap set(x) \rvert}{\lvert set(y) \rvert}$	% Summary tokens that appear in the target method.
$I_{path} = \frac{\lvert set(y) \cap set(q^{path}) \rvert}{\lvert set(y) \rvert}$	% Summary tokens that appear in the path context.
$I_{attr} = \frac{\lvert set(y) \cap set(q^{attr}) \rvert}{\lvert set(y) \rvert}$	% Summary tokens that appear in the class attributes.
$I_{sib} = \frac{\lvert set(y) \cap set(q^{sib}) \rvert}{\lvert set(y) \rvert}$	% Summary tokens that appear in the sibling method signatures.
$I_d = \frac{\lvert set(y) \cap set(d) \rvert}{\lvert set(y) \rvert}$	% Summary tokens that appear in the documentation context.
$I_h = \frac{\lvert set(y) \cap (set(q) \cup set(d)) \rvert}{\lvert set(y) \rvert}$	% Summary tokens that appear in the hybrid context.
$D_{path} = \frac{\lvert (set(y) - set(x)) \cap set(q^{path}) \rvert}{\lvert set(y) \rvert}$	% Summary tokens that appear in the path context but do not appear in the target method.
$D_{attr} = \frac{\lvert (set(y) - set(x)) \cap set(q^{attr}) \rvert}{\lvert set(y) \rvert}$	% Summary tokens that appear in the class attributes but do not appear in the target method.
$D_{sib} = \frac{\lvert (set(y) - set(x)) \cap set(q^{sib}) \rvert}{\lvert set(y) \rvert}$	% Summary tokens that appear in the sibling method signatures but do not appear in the target method.
$D_d = \frac{\lvert (set(y) - set(x)) \cap set(d) \rvert}{\lvert set(y) \rvert}$	% Summary tokens that appear in the documentation context but do not appear in the target method.
$D_h = \frac{\lvert (set(y) - set(x)) \cap (set(q) \cup set(d)) \rvert}{\lvert set(y) \rvert}$	% Summary tokens that appear in the hybrid context but do not appear in the target method.

Description	I_x Range	Count	I_x	$I_{path}(D_{path})$	$I_{attr}(D_{attr})$	$I_{sib}(D_{sib})$	$I_d(D_d)$	$I_h(D_h)$
Function	All	464143	36.15	8.87 (2.55)	15.21 (2.18)	39.53 (7.94)	53.90 (32.77)	**70.72 (38.11)**
	[0,25)	143478	13.32	6.25 (4.03)	8.85 (3.53)	23.55 (11.75)	52.53 (44.25)	**63.05 (50.48)**
	[25,50)	191172	34.33	8.89 (2.61)	14.69 (2.10)	38.14 (8.17)	53.61 (33.22)	**69.57 (37.48)**
	[50,75)	99091	56.61	11.46 (1.03)	20.52 (0.99)	54.19 (4.18)	56.15 (23.27)	**78.71 (25.42)**
	[75,100]	30402	88.65	12.70 (0.18)	31.17 (0.17)	75.91 (0.82)	54.82 (6.65)	**88.13 (13.43)**
Return	All	334300	35.43	9.85 (2.35)	14.41 (1.97)	38.41 (8.85)	55.12 (34.04)	**70.76 (39.69)**
	[0,25)	119947	11.27	4.63 (3.32)	6.61 (2.80)	22.08 (13.99)	51.33 (44.37)	**60.61 (51.22)**
	[25,50)	112469	33.81	9.19 (2.90)	13.31 (2.29)	36.92 (8.77)	54.72 (34.68)	**70.03 (39.20)**
	[50,75)	76164	58.62	13.68 (0.78)	21.83 (0.80)	53.34 (3.60)	59.97 (25.04)	**80.84 (26.89)**
	[75,100]	25720	86.48	25.73 (0.11)	33.59 (0.15)	76.87 (0.71)	60.16 (9.68)	**91.48 (16.78)**
Parameter	All	258313	38.52	8.10 (1.45)	14.64 (2.03)	42.85 (8.36)	61.22 (36.69)	**77.06 (41.07)**
	[0,25)	64791	12.99	5.29 (2.70)	9.16 (3.56)	24.29 (12.56)	61.53 (52.28)	**69.86 (57.41)**
	[25,50)	104606	33.87	7.40 (1.51)	13.04 (2.15)	39.12 (9.05)	59.84 (37.84)	**73.96 (41.86)**
	[50,75)	72345	57.66	10.44 (0.55)	20.18 (0.88)	57.54 (5.20)	62.65 (26.80)	**84.55 (29.35)**
	[75,100]	16571	84.15	13.26 (0.12)	21.92 (0.25)	74.79 (1.36)	62.59 (11.65)	**92.05 (16.83)**

特点总结与未来展望

特点总结与未来展望

- 特点总结

算法	CoSS	StructCodeSum
优势	**1.** 采用**GAT**嵌入**CFG**来获取代码的控制流特征 **2.** 采用多头注意力机制，使 **CoSS** 同时关注令牌级和语句级语义	**1.** 从混合代码上下文生成结构化代码摘要 **2.** 既提供了基于LLM的方法，也提供了轻量级的方法，适用于不同的场景
劣势	训练效率有待提升	依赖代码文档的质量

- 未来展望
  - 构建高质量代码数据集
  - 不同编程语言的文档风格有所不同，建立统一的处理方法

参考文献

[1] Shi C, Cai B, Zhao Y, et al. CoSS: Leveraging statement semantics for code summarization[J]. IEEE Transactions on Software Engineering, 2023, 49(6): 3472-3486.

[2] Zhou Z, Li M, Yu H, et al. Learning to Generate Structured Code Summaries from Hybrid Code Context[J]. IEEE Transactions on Software Engineering, 2024.

道德经

知人者智，自知者明。胜人者有力，自胜者强。知足者富。强行者有志。不失其所者久。死而不亡者，寿。

谢谢！