Beijing Forest Studio

北京理工大学信息系统及安全对抗实验中心

源代码变更表征

博士研究生 段学明

2026年03月22日

问题回溯

- 总结反思
  - 基础知识可适当增加

- 相关内容
  - 2023.07.16 张钊：《代码变更表示学习及其应用研究》
  - 2023.10.07 赵智洋：《代码变更表示学习技术》
  - 2024.09.17 段学明：《基于视觉直觉的源代码表征》

内容提要

- 预期收获
- 内涵解析与研究目标
- 研究背景与研究意义
- 研究历史与现状
- 知识基础
- 算法原理
  - Patcherizer
- 特点总结与工作展望
- 参考文献

预期收获

- 预期收获
  - 1. 了解源代码变更表征的主要实现方法
  - 2. 理解意图建模在源代码变更表征的重要性
  - 3. 了解源代码变更表征的前沿方法和未来发展

内涵解析与研究目标

- 研究目标
  - 将代码变更表征为数值向量
- 内涵解析
  - 补丁：指对源代码进行修改以修复软件缺陷的代码片段，包括代码变更、修复描述
  - 变更：对源代码具体修改，如增加新代码行、删除错误代码行、修改现有代码

```
@@ -421,7 +421,7 @@ public class TestFormAuthenticator
extends TomcatBaseTest {
-   private class FormAuthClientBase extends
SimpleHttpClient {
+   private abstract class FormAuthClientBase extends
SimpleHttpClient {
        protected static final String LOGIN_PARAM_TAG =
        "action=";
        protected static final String LOGIN_RESOURCE =
        "j_security_check";
```

Commit Message:
Make base class **abstract**

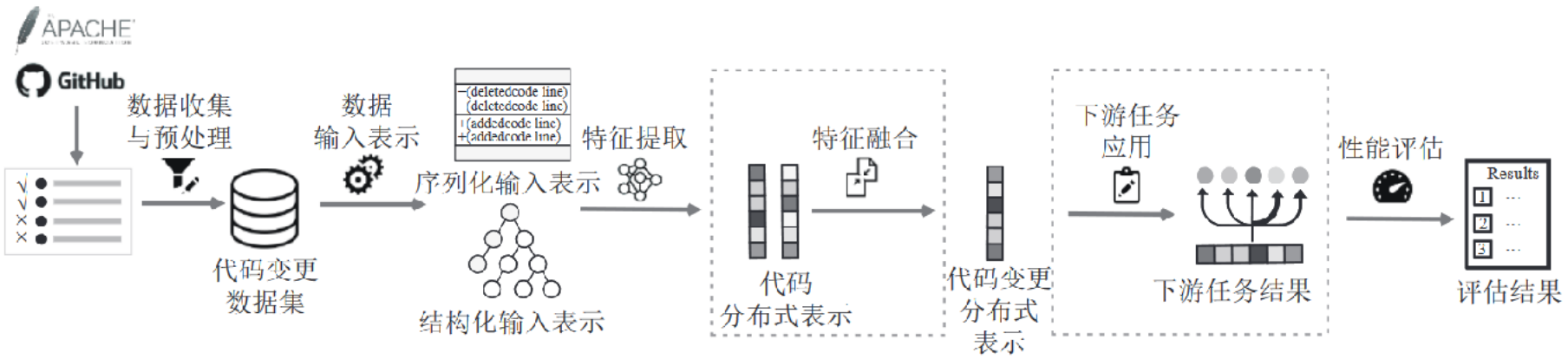
背景意义

- 研究背景
  - 随着开源社区和工业软件持续演化，代码仓库中每天都会产生大量补丁、提交和版本更新，单纯分析静态源代码已难以全面刻画软件演化过程中的真实语义
  - 传统源代码表征方法主要面向完整代码片段，侧重语法结构和语义信息建模，但对“代码是如何被修改的、为什么这样修改”关注不足
  - 近年来预训练模型、图神经网络和上下文交互建模的发展，为从序列和结构两个层面联合学习代码变更表示提供了新的技术基础
- 研究意义
  - 研究源代码变更表征有助于从“静态理解代码”进一步走向“动态理解软件演化”，提升模型对真实开发行为的刻画能力
  - 更准确的代码变更表征能够提升提交消息生成、补丁质量评估、缺陷引入检测等下游任务效果，增强软件工程智能化分析能力

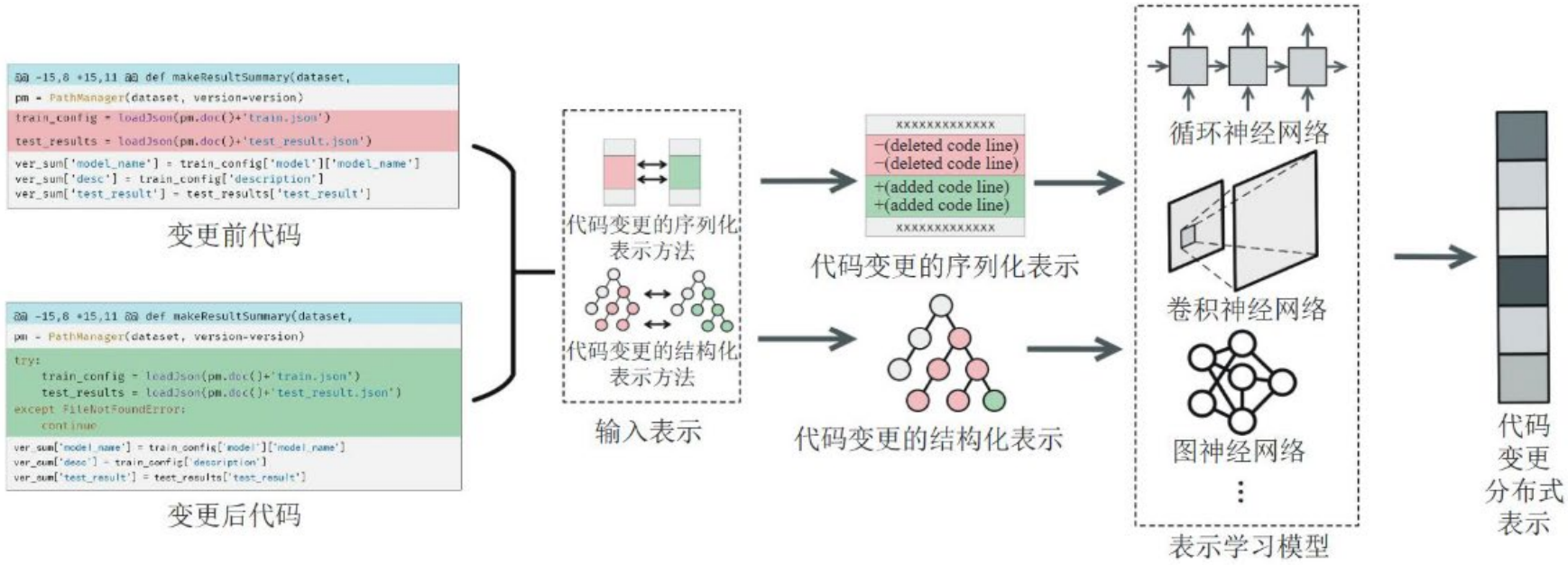
研究历史

- 代码变更表征技术发展
  - 传统：特征工程
    - 做法：通过人工设计的特征或特征提取规则将代码变更表示为特征向量
    - 局限：
      - 依赖手工分析
      - 只能提取到显式的、浅层的特征
  - 如今：表示学习
    - 基于显式信息交互的代码变更表示学习
    - 基于隐式信息交互的代码变更表示学习

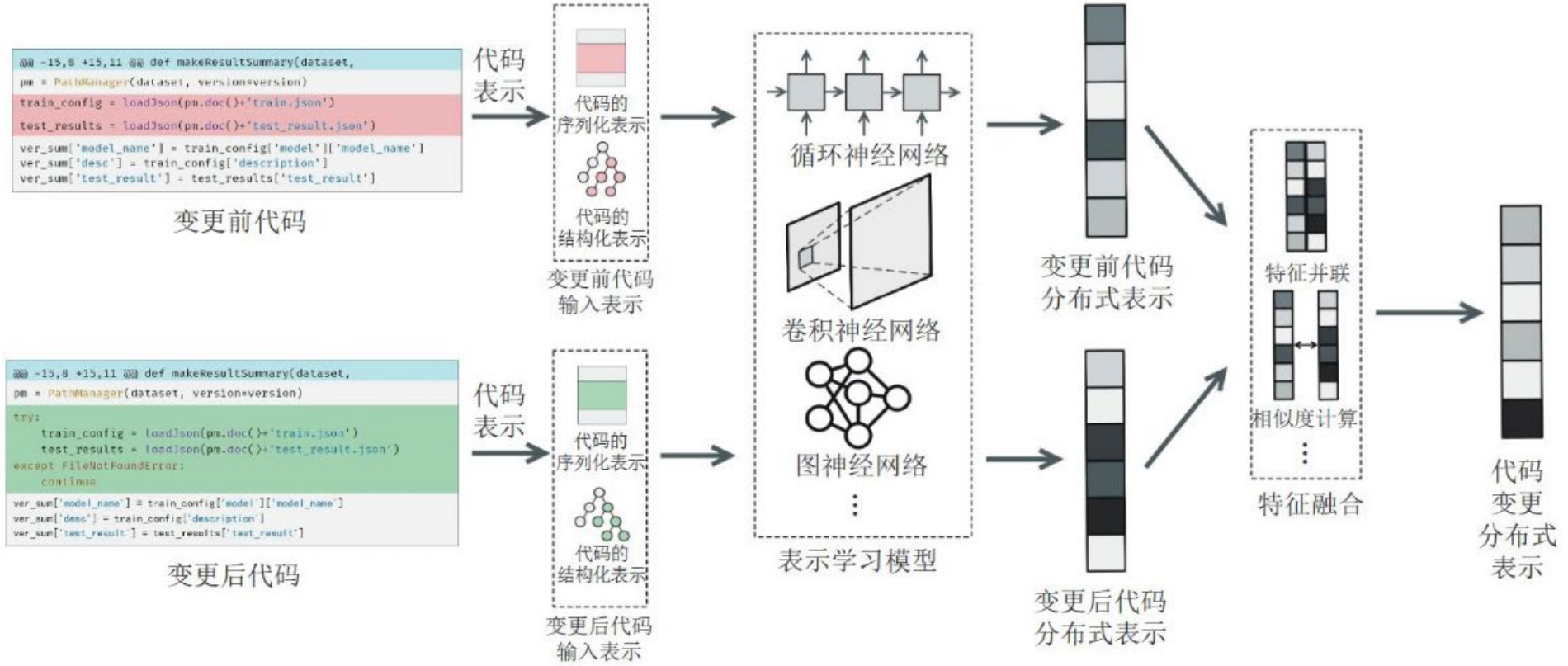
- 代码变更表示学习应用的一般框架
  - 数据收集和预处理
    - 目的：过滤噪声数据或不符合要求的数据，提高数据质量
  - 数据输入表示
    - 目的：将经过预处理的代码变更数据转换为表示学习模型能够处理的表示形式
      - 表示形式：序列化输入表示、结构化输入表示



- 基于显式信息交互的代码变更表示学习
  - 在数据输入表示阶段进行显式地比对与融合



- 基于隐式信息交互的代码变更表示学习
  - 在特征融合阶段进行隐式地比对和融合



算法原理 Patcherizer

Learning to represent code changes

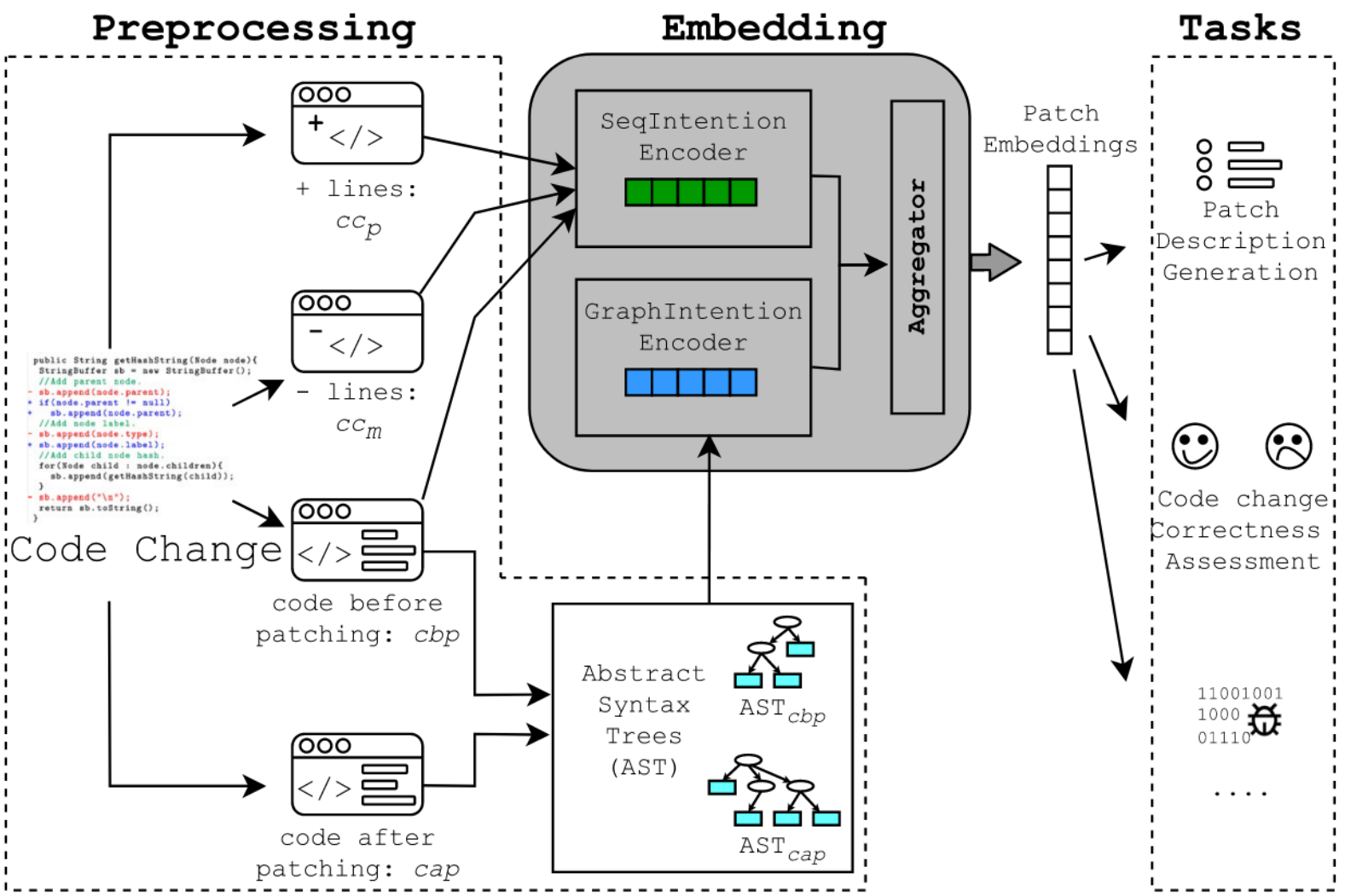
算法原理 TIPO

T	目标	生成任务无关、可迁移的源代码变更统一表征
I	输入	变更前代码、变更后代码
P	处理	1. 上下文构造 2. 序列意图编码 3. 图意图编码 4. 聚合为统一表征
O	输出	统一的代码变更表征

P	问题	现有表征方法仅能描述“改了什么”，难以显式建模代码变更背后的语义意图
C	条件	代码变更能够被重建为较完整的变更前后代码片段
D	难点	如何建模意图，从表面相似但真实语义目的不同的代码变更中识别意图
L	水平	2026 CCF-B

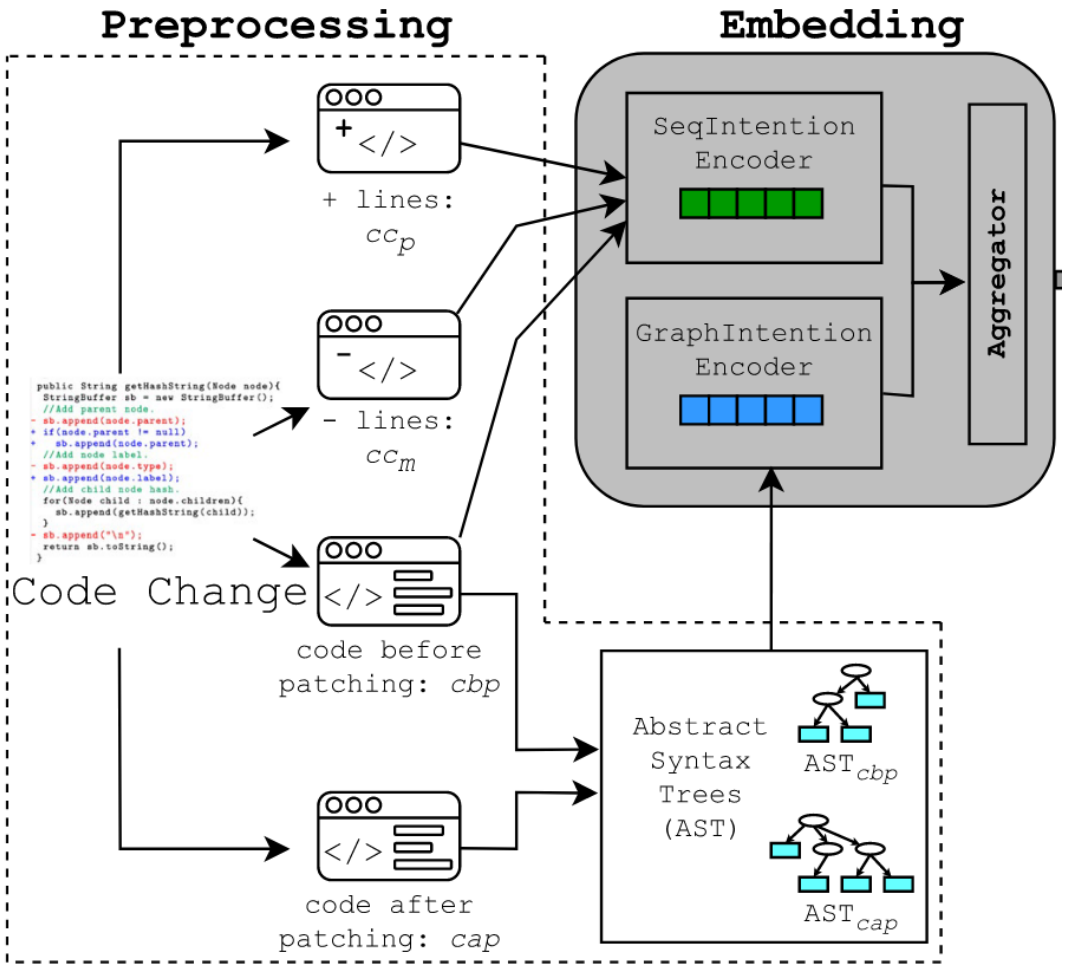
算法原理

- 算法原理图
  - 预处理
    - 上下文构造
  - 序列意图编码
  - 图意图编码
  - 多源嵌入聚合
  - 预训练
  - 下游任务微调



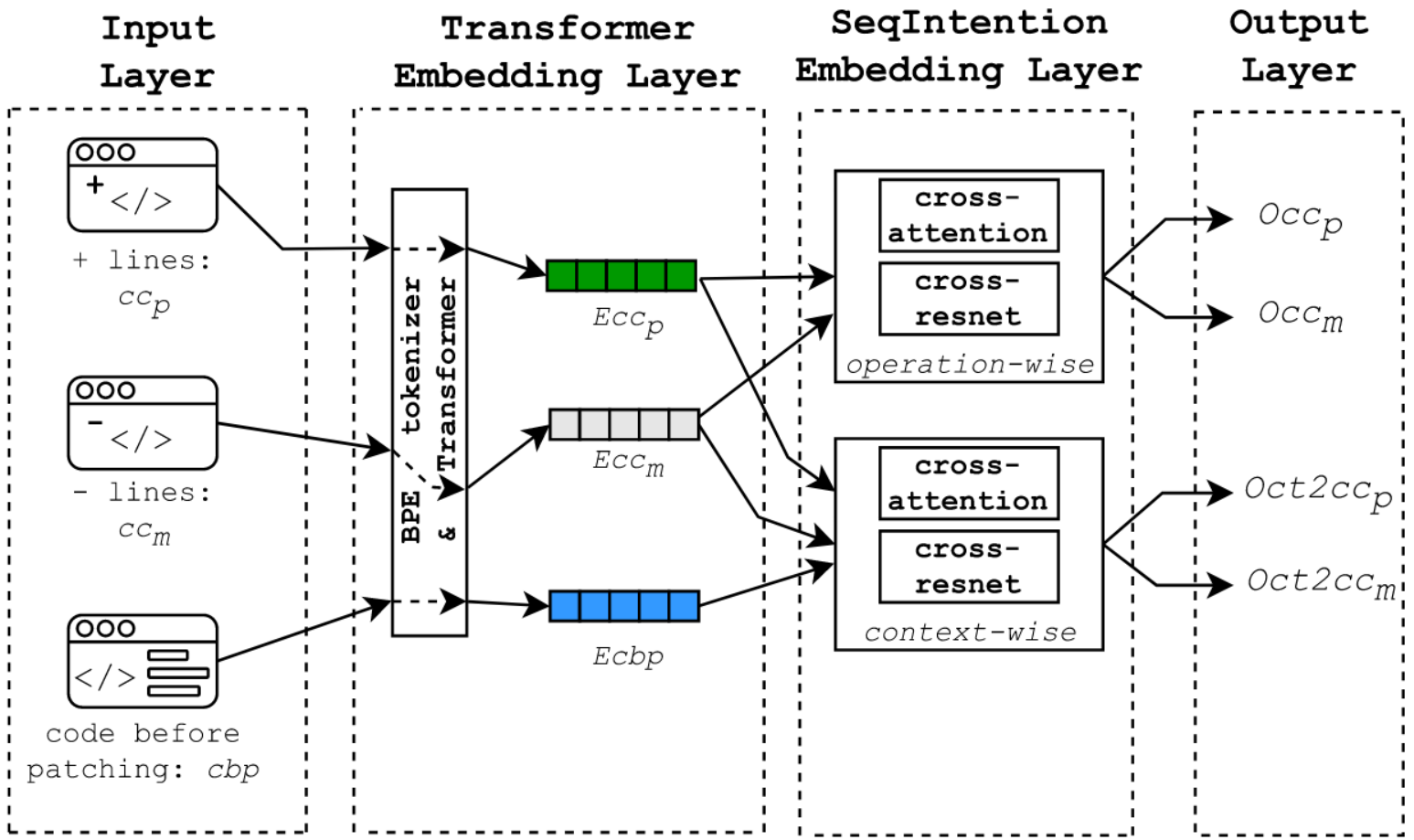
算法原理　预处理

- 拆分变更
  - 拆出新增（+）和删除（-）行
  - 含行内容及行号
- 构造上下文
  - 提取变更前后未变更的代码块
  - 缺点：结构元素缺失，AST无法生成
  - 解决：同一文件中查找周围行进行补充，仍不完整，则不使用图意图编码
- 生成变更前后AST
- 构建词汇表
  - 根据训练数据中的代码，使用字节对编码（BPE）算法构建词汇表



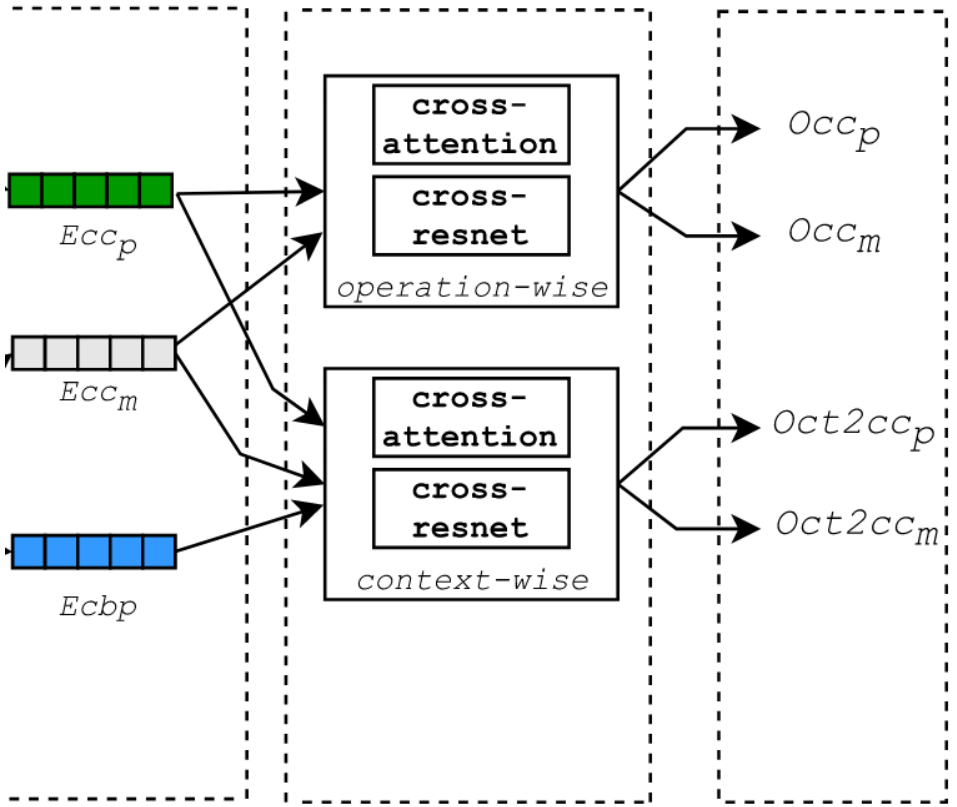
算法原理　序列意图编码

- 双层学习
  - 第一层：把三类输入 ccp / ccm / cbp 转变为具备上下文语义的token向量
  - 第二层：显式建模“新增”和“删除”之间、“变更内容”和“上下文”之间的关系



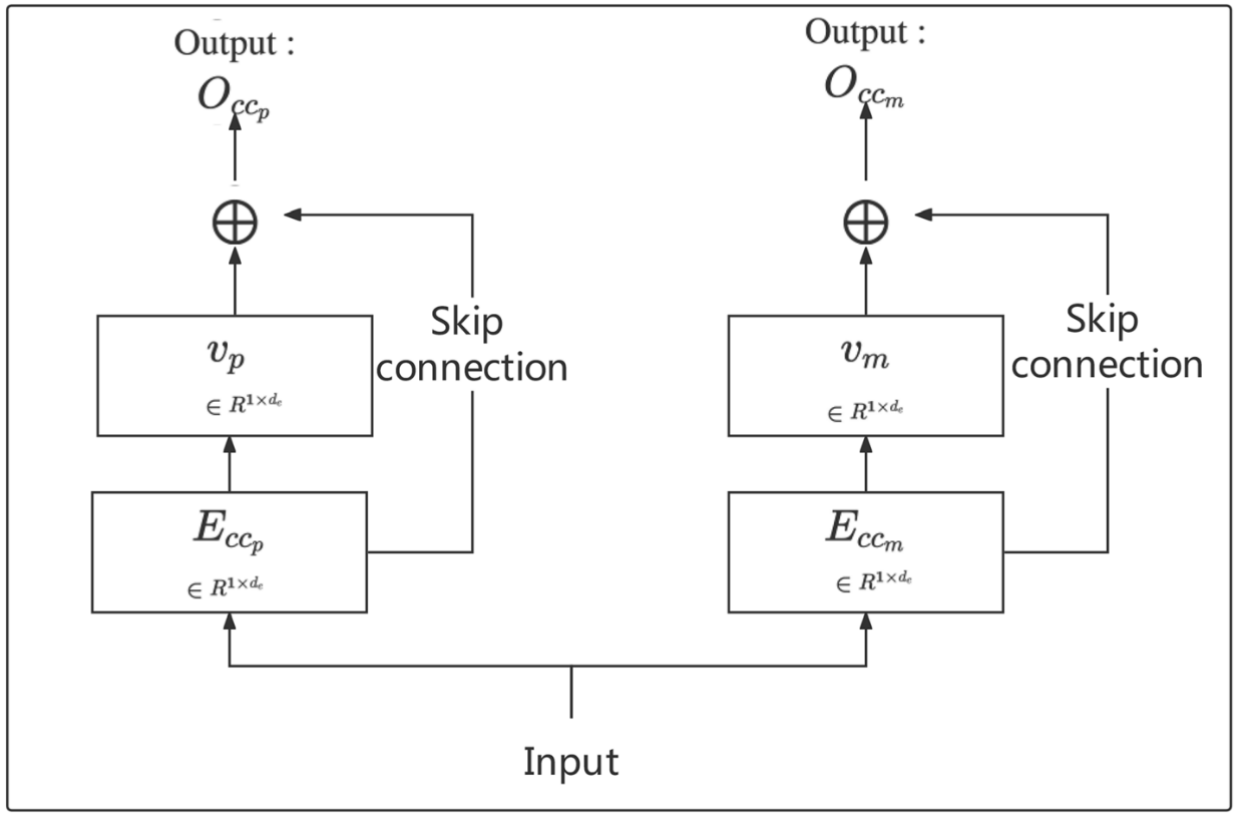
算法原理　序列意图编码

- 序列意图编码
  - Operation-wise：建模 ccp 和 ccm 的关系
    - Query来自删除行，Key/Value来自新增行
      - 注意力权重矩阵：每个删除token对每个新增token的关注强度
      - 更新后的V矩阵：被新增语义增强后的删除表示
    - 反之得到：被删除语义增强后的新增表示
  - Context-wise：建模 cbp 与变更内容的关系
    - 动机：不能只看到变更之间的关系，还要判断变更的意图
  - 输出：变更交互后的+表示、变更交互后的-表示、上下文增强后的+表示、上下文增强后的-表示



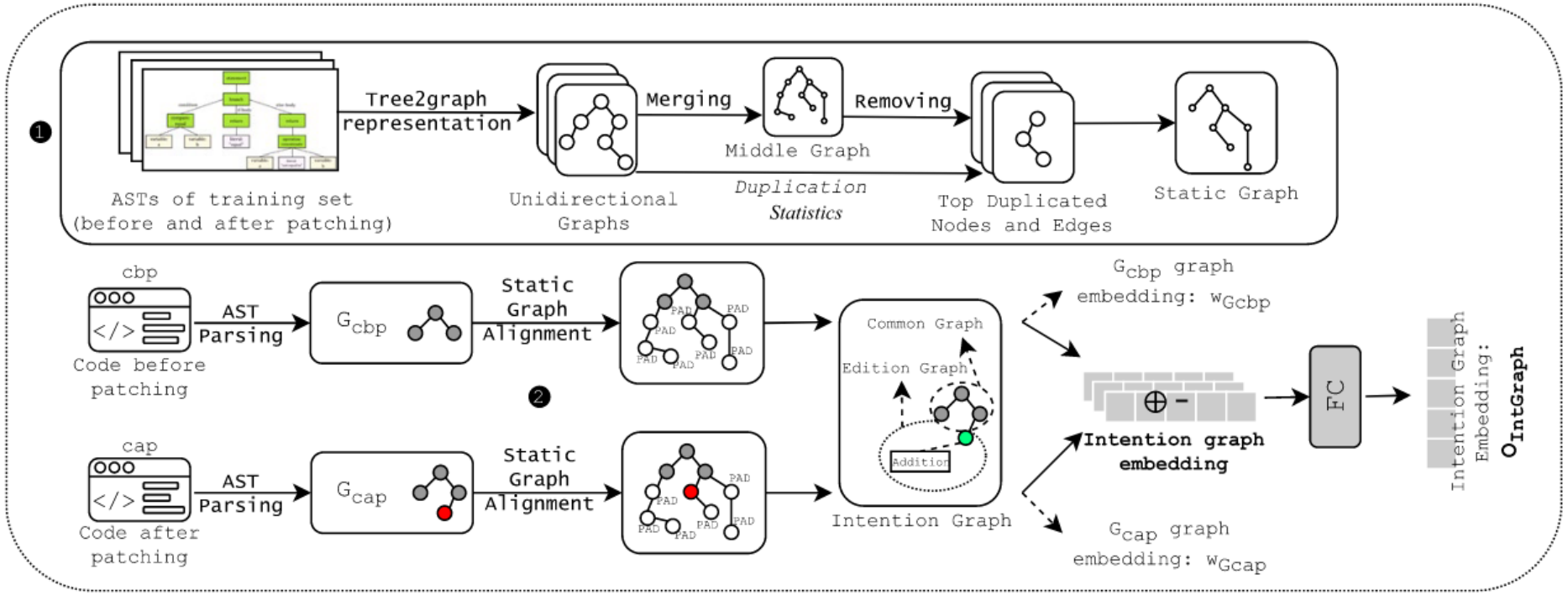
算法原理　序列意图编码

- 残差模块
  - 动机：
    - 交叉注意力得到的是“另一侧的信息”
      - 防止语义淡化
    - 残差学习更适合学“变化量”而不是重写全部表示
      - 本身与建模变更契合，代码变更的本质是在原始代码语义上发生局部变化
    - 训练稳定



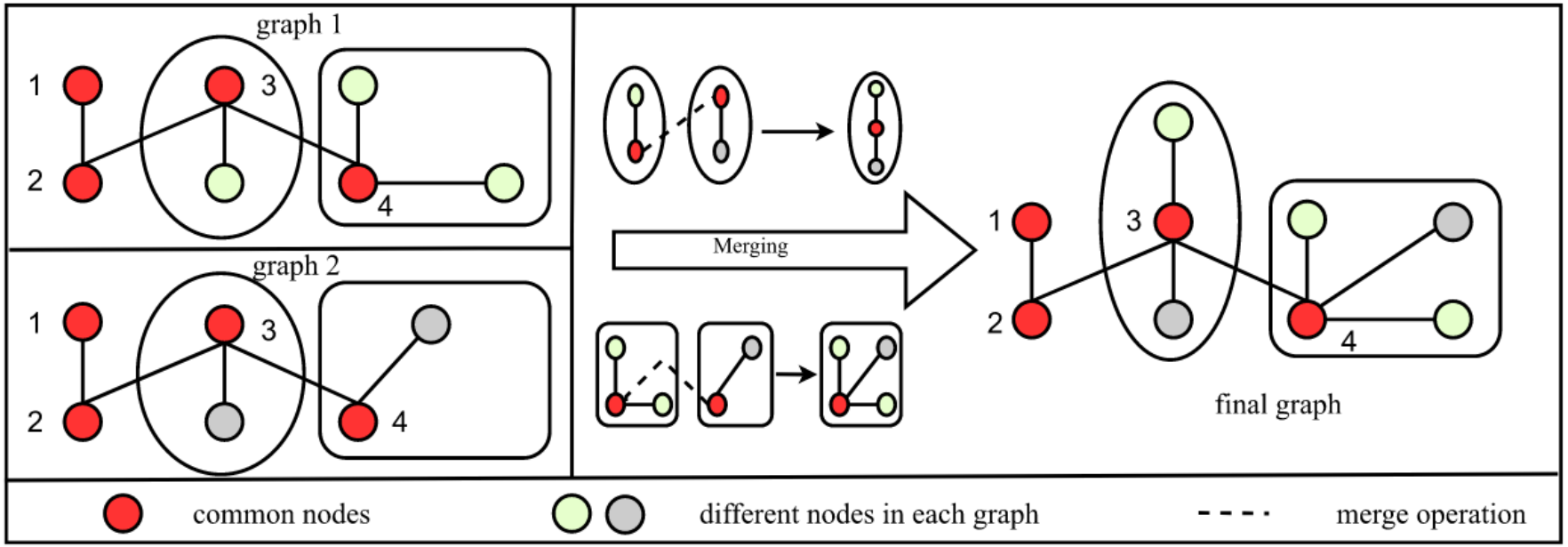
算法原理　图意图编码

- 动机
  - 解决变更在结构上发生了什么语义变化
  - 增加结构维度的表示
- 图构建
  - 目的：解决变更对应的AST大小不一问题，方便GCN训练



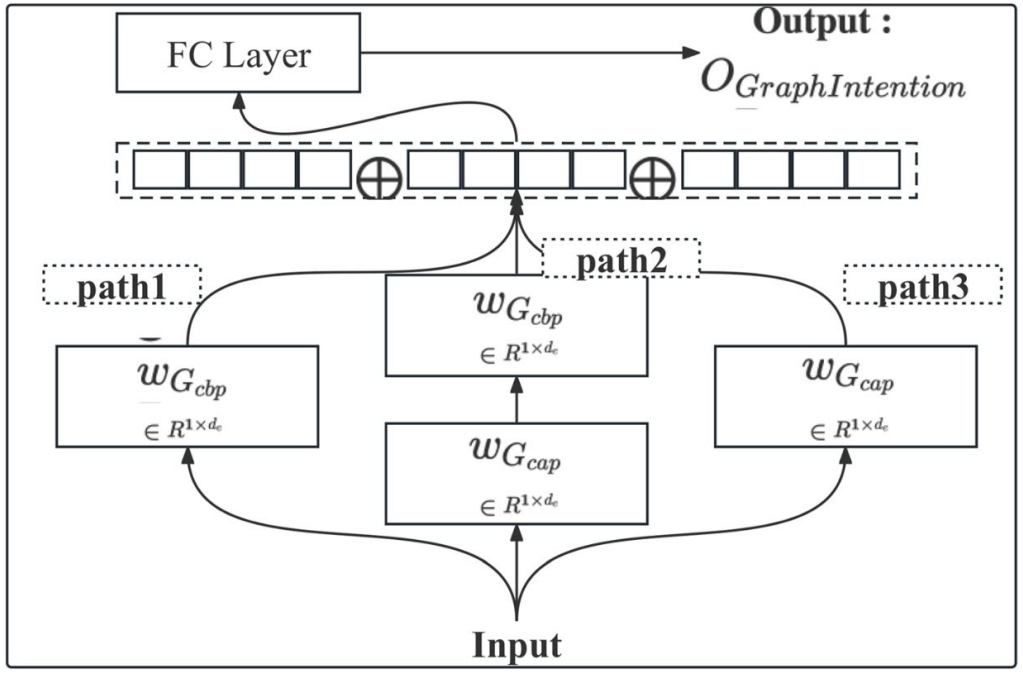
算法原理　图意图编码

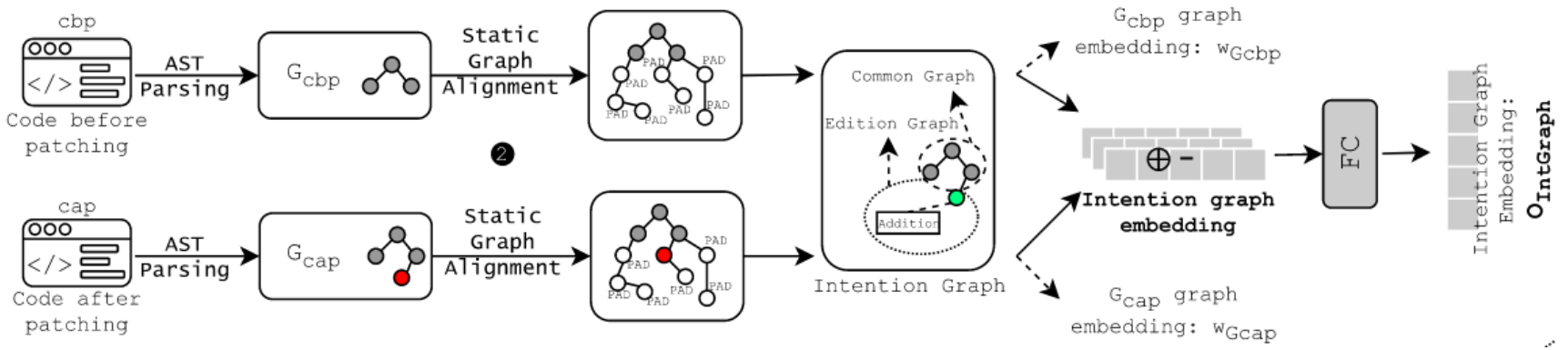
- 图构建
  - 第一步：合并变更前后的2张局部AST
  - 第二步：把训练集所有局部AST合并为1张全局静态图
  - 第三步：删除噪声节点
  - 操作：保留公共节点、合并共同节点、连接所有邻居
  - 效果：构造了一个全局结构模板，每个变更并非各自学习，而是先映射到统一模板



算法原理　图意图编码

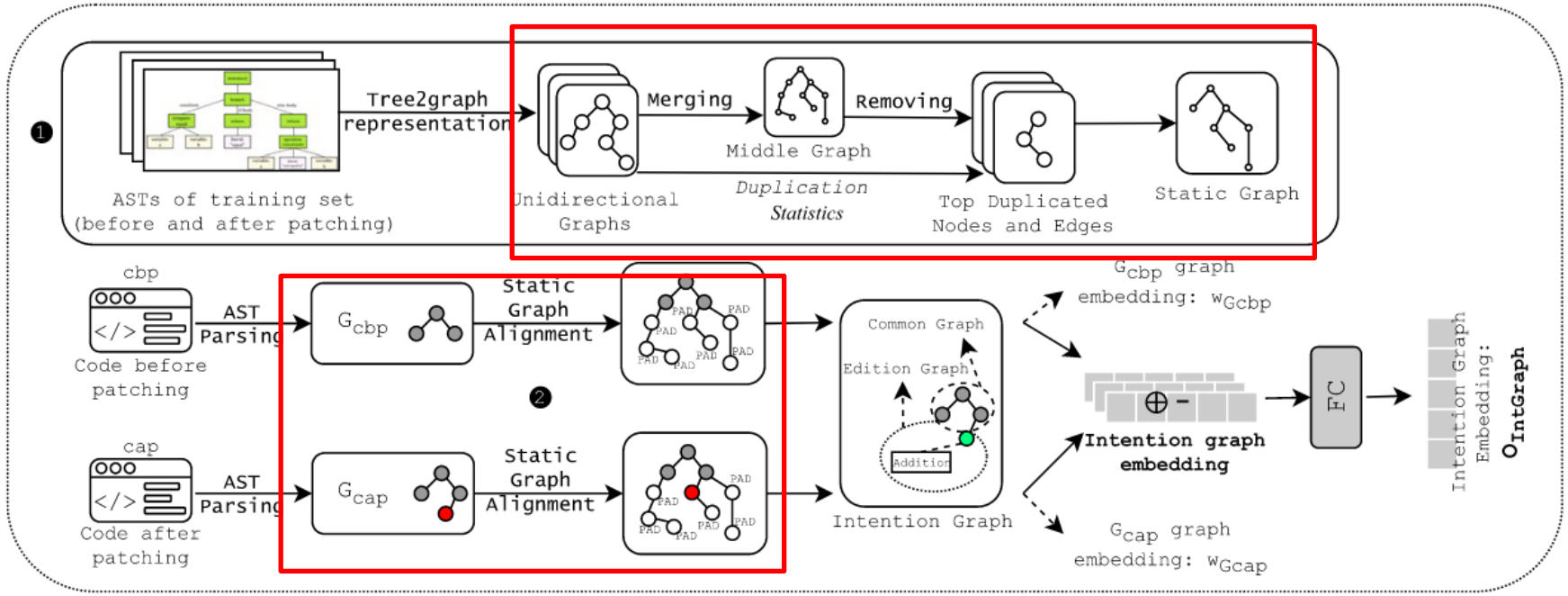
- 图学习
  - 构造输入
    - 使用图匹配算法，在全局图中找到最相似的子图
    - 填充其他节点至全局图大小
  - GCN学习：得到子图（AST）表示
  - 意图编码
    - graph-cross-resnet：分别对应变更前的结构、变更后的结构、中间发生了什么结构变化





算法原理　图意图编码

- 思考
  - 不同样本的AST子图合并成一个大图，有何意义？
    - 本质在于希望构造一个统一的结构坐标系或结构词典，并非还原真实项目的完整程序图
    - 类比：词表，并非来自同一句话，但可作为所有句子的公共离散空间
    - 缺陷：牺牲了程序级真实上下文，不是程序语义图，而作为结构模式表征



算法原理　预训练

- 目的
  - 设计一个任务无关的代码变更表征模型
- 预训练任务
  - 训练方式：掩码遮盖
  - 训练框架：Patcherizer encoder + Transformer decoder
  - 数据来源：代码变更描述生成训练集、代码变更正确性评估训练集
- 下游任务微调
  - 代码变更描述生成：目标为最大化正确描述序列的生成概率
  - 代码变更正确性评估
    - 二分类任务，是否正确修复
    - 输入额外增加bug描述，由单独的预训练BERT编码
  - 代码变更意图检测：三类意图分类（新增、删除、更新）

实验设计

- 代码变更描述生成
  - 数据集：90,661条变更-描述对
  - 对比方法：通用+专用
  - 指标：ROUGE-L、BLEU、METEOR
- 代码变更正确性评估
  - 数据集：11,352 条代码变更，其中 9,092 条错误，2,260条正确
  - 对比方法：CC2Vec、CCRep、BERT
  - 指标：AUC、F1、+Recall、-Recall
- 代码变更意图检测
  - 数据集：前2个任务数据集抽取得到，保留新增、删除、更新三类意图，共572条
  - 对比方法： CC2Vec、CCRep
  - 指标： Precision、Recall、F1

实验设计　对比实验

- 代码变更描述生成
  - 整体效果最好
  - BLEU/METEOR提升大，说明生成的token更准确、局部表达更贴近人工描述
  - ROUGE-L高说明整体描述序列和参考消息的长子序列重合度更好

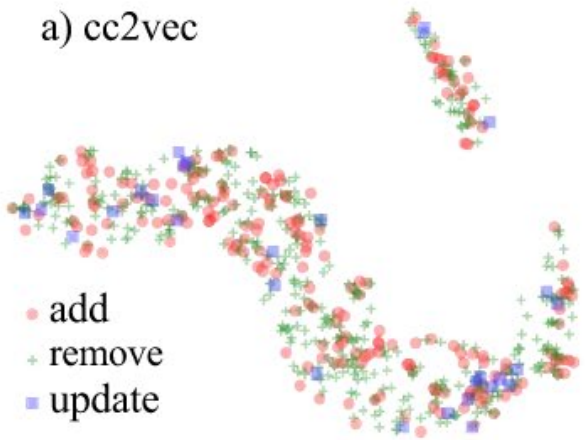
Model	Low (%)	Medium (%)	High (%)	Average Score
NNGen	70.5	15.3	14.2	0.96
CODISUM	37.6	21.4	41.0	2.03
FIRA	34.0	21.8	44.2	2.12
Patcherizer	32.8	20.5	46.7	2.19

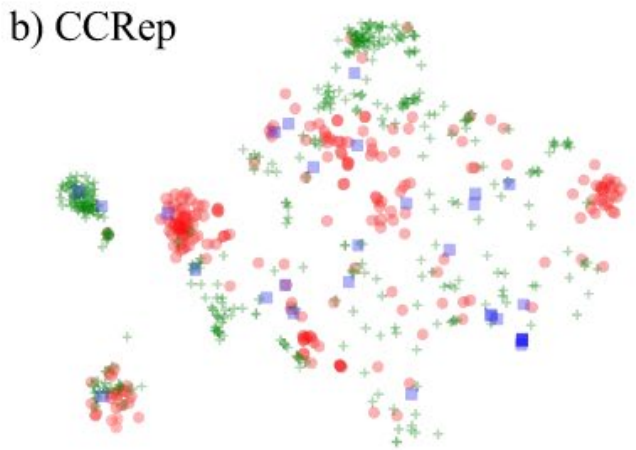
Type	Approach	Rouge-L (%)	BLEU (%)	METEOR (%)
Generation	NMT (Jiang et al. 2017)	7.35	8.01	7.93
	Codisum (Xu et al. 2019)	19.73	16.55	12.83
	ATOM (Liu et al. 2020c)	10.17	8.35	8.73
	FIRA (Dong et al. 2022)	21.58	17.67	14.93
	CoreGen (Nie et al. 2021) (Transformer)	18.22	14.15	12.90
	CCRep (Liu et al. 2023)	23.41	19.70	15.84
	CCBERT (Zhou et al. 2023)	20.74	16.98	14.25
	CCT5 (Lin et al. 2023)	21.13	17.11	14.38
	CodeT5 (Wang et al. 2021c)	21.26	17.33	14.52
	Patcherizer	**25.45**	**23.52**	**21.23**
Retrieval	CC2Vec (Hoang et al. 2020)	12.21	12.25	11.21
	NNGen (Liu et al. 2018)	9.16	9.53	16.56
	CoRec (Wang et al. 2021a)	15.47	13.03	12.04
	Patcherizer	**17.32**	**15.21**	**17.25**

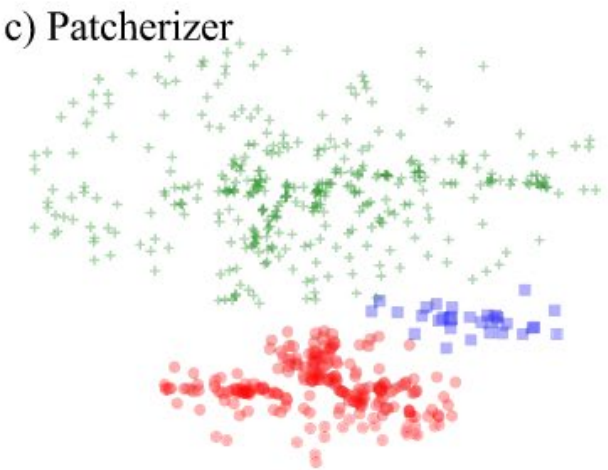
实验设计　对比实验

- 代码变更正确性评估
  - +Recall 高：更擅长识别真正正确的修复
  - -Recall 高：对错误修复的过滤能力强
- 代码变更意图检测
  - 降维可视化

Classifier	Model	AUC	F1	+Recall	-Recall
LR	CC2Vec	0.75	0.49	0.47	0.85
	BERT	0.83	0.58	0.81	0.65
	CCRep	0.86	0.67	0.74	0.83
	Patcherizer	**0.96**	**0.82**	**0.87**	**0.91**
XGB	CC2Vec	0.81	0.55	0.50	0.89
	BERT	0.84	0.61	0.64	0.85
	CCRep	0.82	0.63	0.59	0.88
	Patcherizer	**0.90**	**0.67**	**0.66**	**0.90**







实验设计　消融实验

- 代码变更描述生成

Model	ROUGE-L (%)	BLEU (%)	METEOR (%)
Patcherizer $_{GraphIntention-}$	20.10	16.50	15.40
Patcherizer $_{SeqIntention-}$	18.44	14.70	16.20
Patcherizer $_{both-}$	15.00	13.00	12.00
Patcherizer	25.45	23.52	21.23

- 代码变更正确性评估
- 代码变更意图检测

Model	Silhouette Score	Clustering Accuracy (%)
Patcherizer	**0.42**	**81.3**
Patcherizer $_{GraphIntention-}$	0.31	72.6
Patcherizer $_{SeqIntention-}$	0.25	65.4
Patcherizer $_{both-}$	0.17	58.2

Classifier	Model	AUC	F1	+Recall	-Recall
LR	CC2Vec	0.75	0.49	0.47	0.85
	BERT	0.83	0.58	0.81	0.65
	CCRep	0.86	0.67	0.74	0.83
	Patcherizer	**0.96**	**0.82**	**0.87**	**0.91**
	Patcherizer $_{GraphIntention-}$	0.88	0.70	0.80	0.78
	Patcherizer $_{SeqIntention-}$	0.84	0.62	0.75	0.72
XGB	CC2Vec	0.81	0.55	0.50	0.89
	BERT	0.84	0.61	0.64	0.85
	CCRep	0.82	0.63	0.59	0.88
	Patcherizer	**0.90**	**0.67**	**0.66**	**0.90**
	Patcherizer $_{GraphIntention-}$	0.86	0.64	0.68	0.82
	Patcherizer $_{SeqIntention-}$	0.83	0.60	0.65	0.80

特点总结与未来展望

算法总结

- 算法创新
  - 把代码变更表征从“描述改了什么”推进到“建模为什么这样改”
  - 序列表征上，建模增删交互和修改-上下文交互
  - 结构建模上，面向补丁前后的结构变化学习图意图表示
- 算法优劣
  - 优势：多任务适用，鲁棒性强
  - 劣势：不能严格证明提升来自“意图”，大图构建复杂
- 未来工作
  - 改进建图模式；真实泛化性检验；探索与预训练代码大模型结合

参考文献

[1] Xuzhu T, Haoye T, et al. Learning to represent code changes[J]. Empirical Software Engineering, 2026, 31(50).

[2] Yuze J, Beijun S, Xiaodong G. Just-in-time software defect prediction via bi-modal change representation learning[J]. Journal of Systems and Software, 2025, 219:112253.

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！