Beijing Forest Studio

北京理工大学信息系统及安全对抗实验中心


# 代码变更表示学习及其应用研究

张钊

2023年07月16日

# 内容提要

- 背景简介
- 基础概念
  - 代码变更
  - 代码变更表示学习
- 算法原理
  - FIRA
  - CCRep
- 应用总结
- 参考文献

# 问题回溯

- 总结反思：
  - 学术报告要讲解深入，注意整体架构、方法细节和关键技术的讲解
  - 要声音洪亮清晰，提高现场沟通互动频次

- 相关内容：
  - 李新帅《基于Transformer的时间序列分析》
  - 张凌浩《代码异味检测》
  - 孔令迪《源代码漏洞检测》
  - 谢宁《软件漏洞检测及其严重性评估》

# 背景简介

- 预期收获
  - 1. 了解代码变更、代码变更表示学习的背景和基本原理
  - 2. 理解代码变更表示学习的方法和难点
  - 3. 理解代码变更表示学习在提交信息生成任务中的应用
  - 4. 了解代码变更表示学习的前沿发展

# 背景简介

- 代码变更
  - 软件演化过程中的关键行为，对于理解代码库和分析软件演化过程具有重要意义
  - 用于功能需求实现、软件缺陷修复和软件架构改进等重要任务
  - 大量软件工程任务依赖于对代码变更的分析和理解
    - 即时缺陷预测需要分析代码变更的质量以预测出现缺陷的概率
    - 软件制品可溯性恢复需要理解代码变更的内容和意图，以构建代码与其他软件制品的关联
- 代码变更表示越准确，下游任务的学习或检索过程就越不具有挑战性

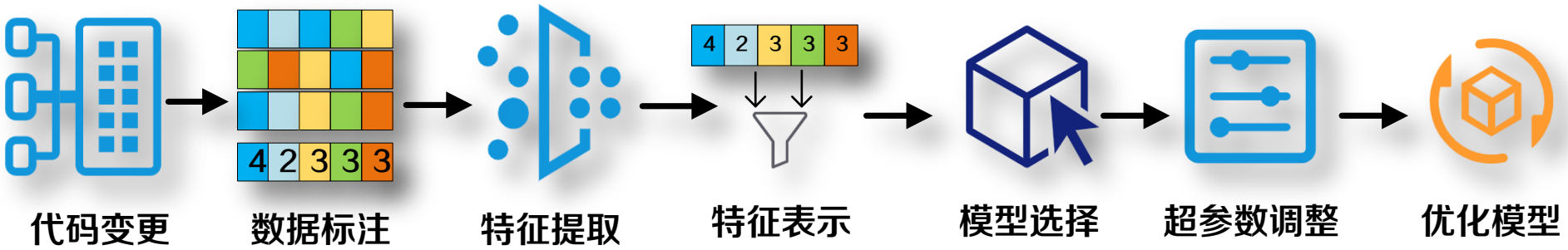




表示和建模代码变更是自动化分析和理解代码变更的基础

# 背景简介

- 早期代码变更表示技术
  - 利用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
    - 人工定义抽象语法树(AST)匹配规则提取变更前后AST的最小编辑脚本
    - 基于代码变更的扩散、规模、目标、历史和开发者经验评估缺陷引入可能性
  - 局限性：依赖手工分析，只能提取显式的、浅层的特征，难以捕捉代码变更语义隐式的、深层的信息
- 代码变更表示学习
  - 将代码变更的语义信息表示为低维稠密的实值向量，即学习代码变更的分布式表示
  - 被应用到众多软件工程任务中，包括代码提交日志生成、即时缺陷预测、即时注释更新、安全漏洞补丁识别、和冲突合并等



# 背景简介　下游任务

- 代码变更表示学习技术
  - 自动学习：利用表示学习模型从代码变更历史数据中自动学习代码变更的表示方法，无需人工定义特征
  - 端到端训练：代码变更表示学习模型可与下游任务模型一起进行端到端的训练，简化代码变更表示的训练和应用过程
  - 表示准确：基于特别设计的表示学习模型、大规模代码变更数据以及下游任务提供的反馈信息，此类技术能够更好地捕捉到更细粒度的、包含语义信息且与下游任务密切相关的高层次特征


代码变更表示学习已应用于大量软件工程下游任务

# 基础概念

- 代码变更(code change)
  - 定义：指对软件源代码的增加、删除和修改。
  - 含两部分信息，即变更前代码和变更后代码
  - 变更前代码/变更后代码：发生变更前代码库中的所有代码和发生变更后代码库中的所有代码

```
@@ -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";
```

删除

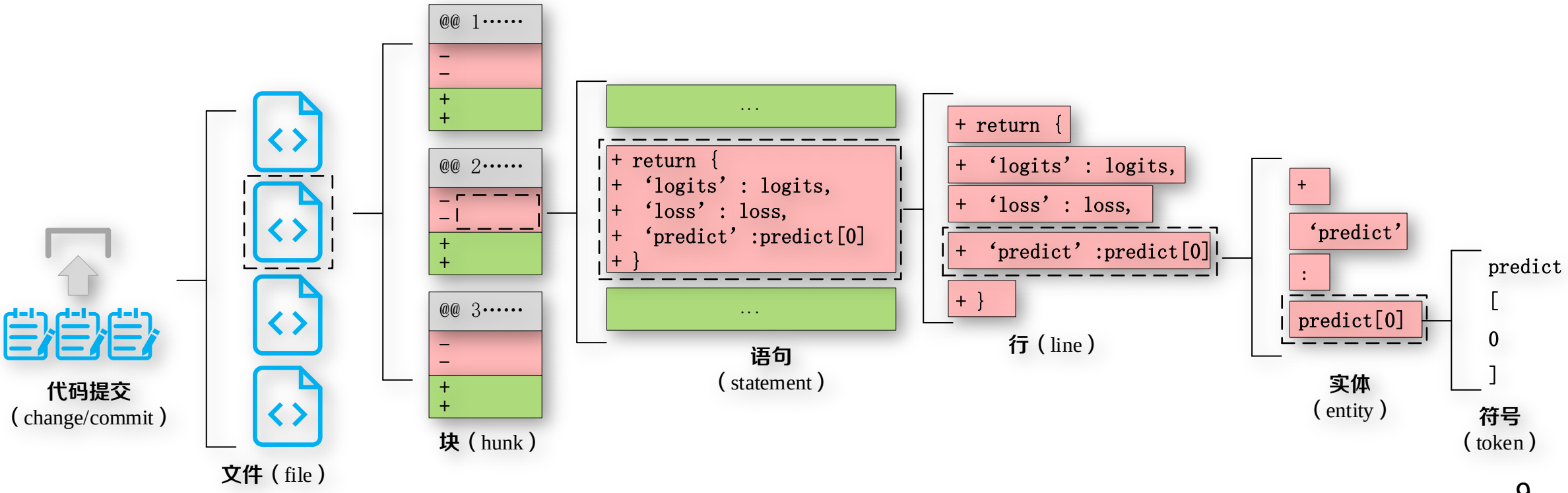
增加

提交信息

```
Commit Message:
    Make base class abstract
```

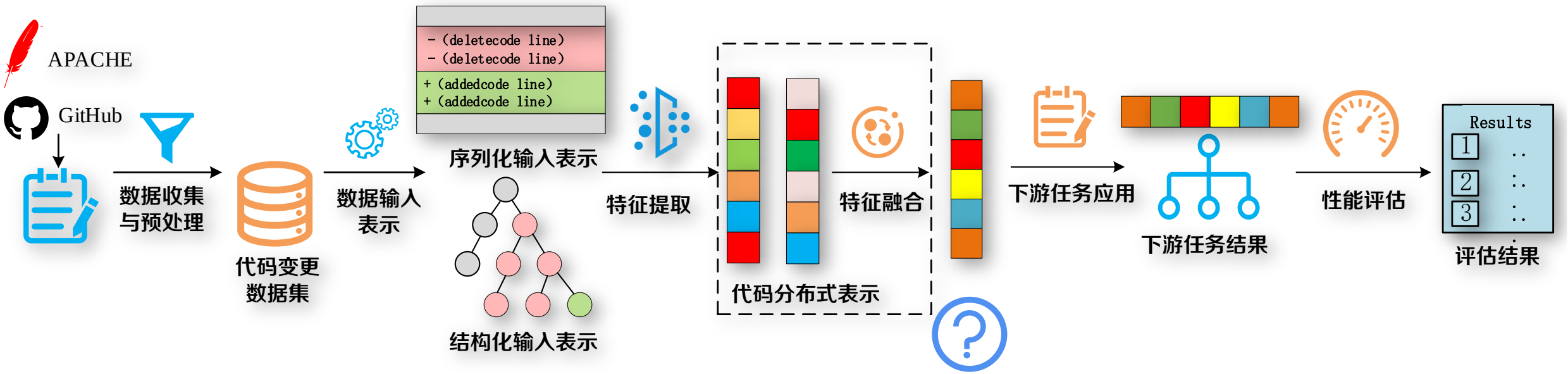
# 基础概念

- 代码变更(code change)
  - 代码变更粒度：代码的结构性使代码变更也具有层次性, 代码变更粒度按照从大到小包括: 代码提交(commit/change)、文件 (file)、块 (hunk)、语句 (statement)、行 (line)、实体 (entity) 和符号 (token)



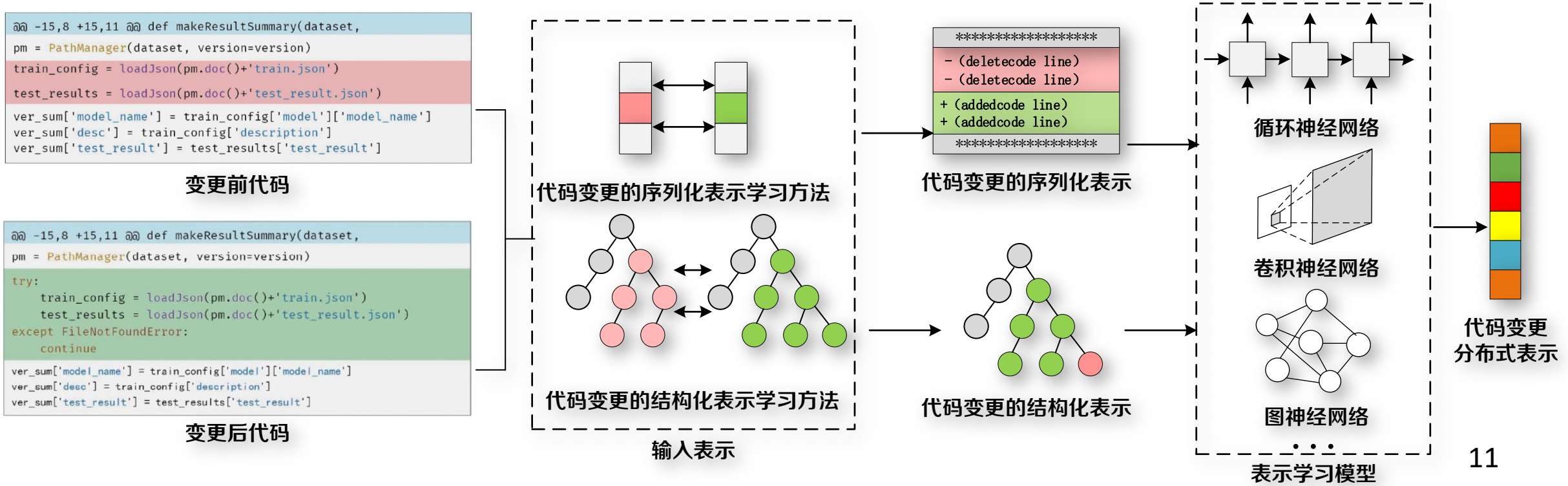
# 基础概念

- 代码变更表示学习
  - 指利用机器学习模型将代码变更的语义信息表示为低维稠密实值向量
  - 代码变更表示学习及其应用的一般流程框架
  - 数据收集和预处理、数据输入表示、特征提取、特征融合、下游任务应用、性能评估



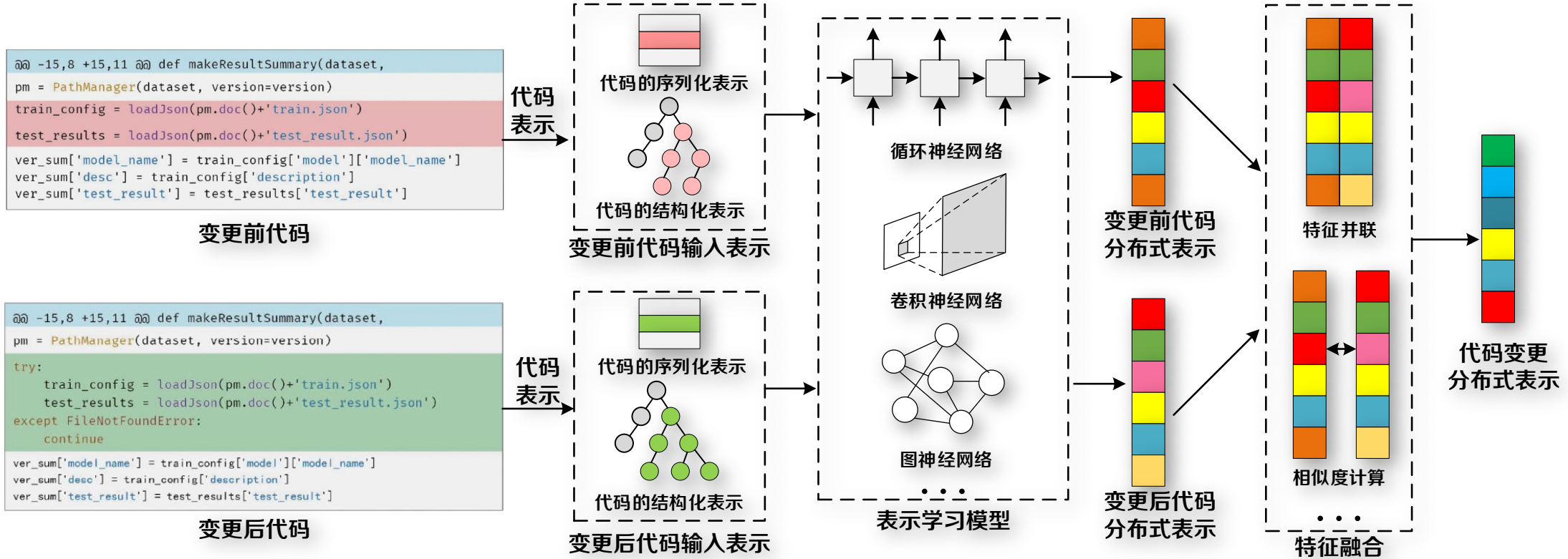
# 基础概念

- 基于显式信息交互的代码变更表示学习
  - 特点：在数据输入表示阶段进行显式地比对与融合
  - 先通过 git diff 和 AST 差分等方式显式地比对和提取代码变更的内容和结构信息，并将其转换为特殊的数据形式（例如 diff），然后利用表示学习模型从此输入中提取特征向量作为代码变更的分布式表示。即，“先交互，再提取”



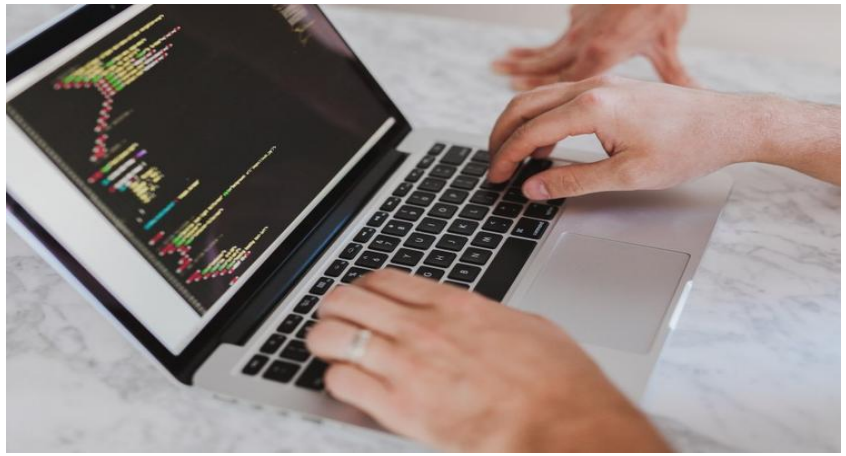
# 基础概念

- 基于隐式信息交互的代码变更表示学习
  - 特点：在特征融合阶段进行隐式地比对和融合
  - 先利用表示学习模型分别提取变更前后代码的特征向量，利用不同特征融合方法（例如特征并联和相似度计算等）将变更前和变更后代码的分布式表示对比融合为单一的特征向量作为代码变更的分布式表示。即，“先提取，再交互”。



# 基础概念

- 提交信息生成
  - commit message generation
  - 自动为版本管理系统中的代码提交生成描述其内容和意图的自然语言语句，减轻开发者手动编写代码提交日志的负担
  - 输入：代码提交中的代码变更
  - 输出：相应的代码提交日志



```
@@ -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**

```
@@ -219,7 +220,7 @@ public class FeatureToggles extends
SherlockActivity{
                        newTab.setText("Text!");
                    }
                }
+   newTab.setTabListener(FeatureToggles.this);
    getSupportActionBar().addTab(newTab);
```

**Commit Message:**
Add **tab** listener for feature **toggles**

# FIRA

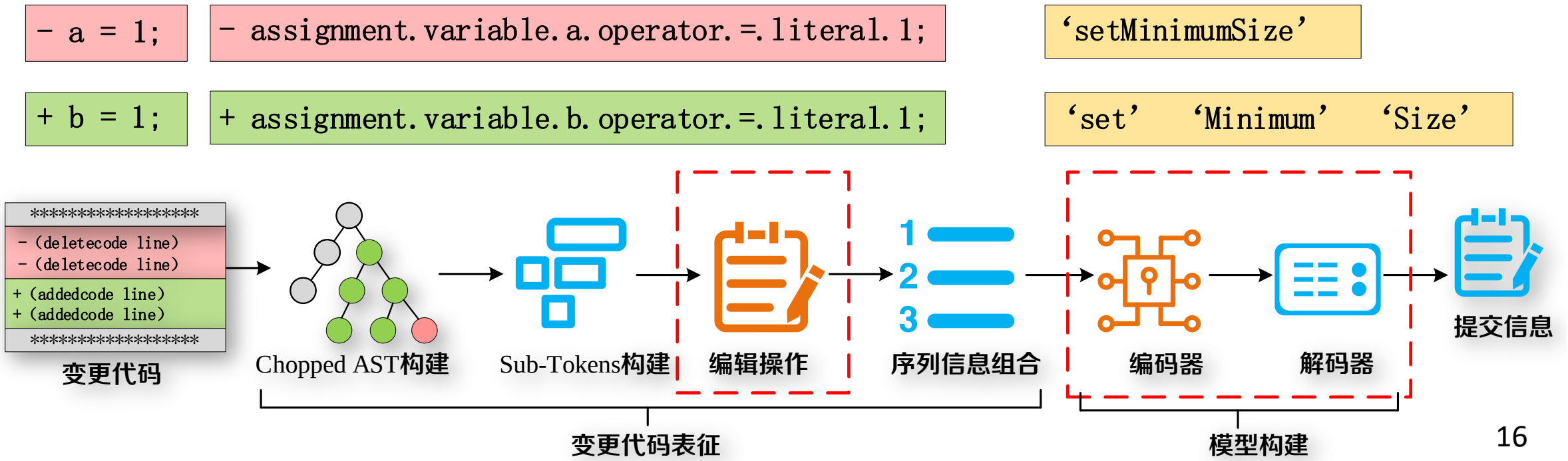
# FIRA TIPO

| | | |
|---|---|---|
| T | 目标 | 准确生成代码变更的提交信息 |
| I | 输入 | 源代码的变更文件 |
| P | 处理 | 1. 根据变更类型划分hunk，提取chopped AST，建立匹配图<br>2. 将Token拆分获取Sub-Token，利用编辑节点表示代码变更操作，构建变更匹配序列<br>3. 构建图神经网络编码器和Transformer解码器，结合双复制机制生成提交信息 |
| O | 输出 | 代码提交信息 |

| | | |
|---|---|---|
| P | 问题 | 现有粗粒度代码变更表示，导致提交信息生成性能准确性弱 |
| C | 条件 | 变更代码可解析为有效的AST |
| D | 难点 | 如何更加准确的描述不同粒度的代码编辑操作和代码Token |
| L | 水平 | ICSE2022（CCF-A） |

# FIRA 算法原理

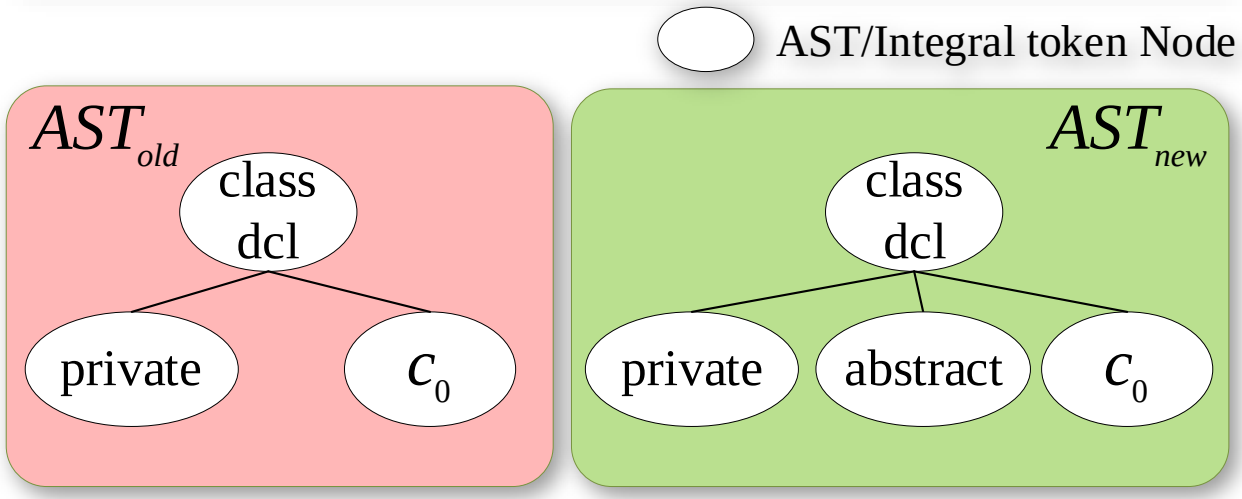
- FIRA (用于自动提交消息生成的基于细粒度图的代码变更表示)
  - 现有方法仅合并变更前后代码，未提取细粒度的代码编辑操作，导致提交信息生成准确性弱
  - 提交信息包含代码子Token情况下，仅提取粗粒度的完整Token，导致提交信息生成的可用成分缺失



- 1. 变更代码表征：chopped AST
  - 按照变更类型划分hunk
  - 构建hunk的chopped AST
    - $AST_{new}$：添加hunk的chopped AST
    - $AST_{old}$：删除hunk的chopped AST
    - $c_0$：指类名FormAuthClientBase的占位符

```
@@ -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";
```

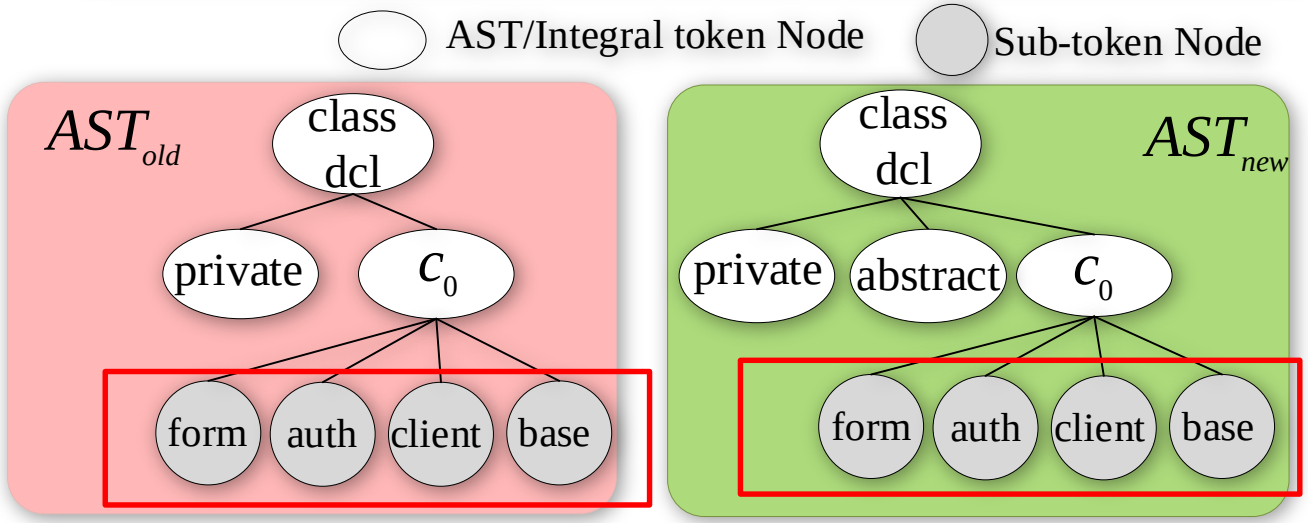
**Commit Message:**
Make base class **abstract**



- 2. 变更代码表征：Sub-Token
  - 构建细粒度的令牌表征
  - 完整令牌：
    - FormAuthClientBase
  - 子令牌
    - from auth client base

```
@@ -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";
```

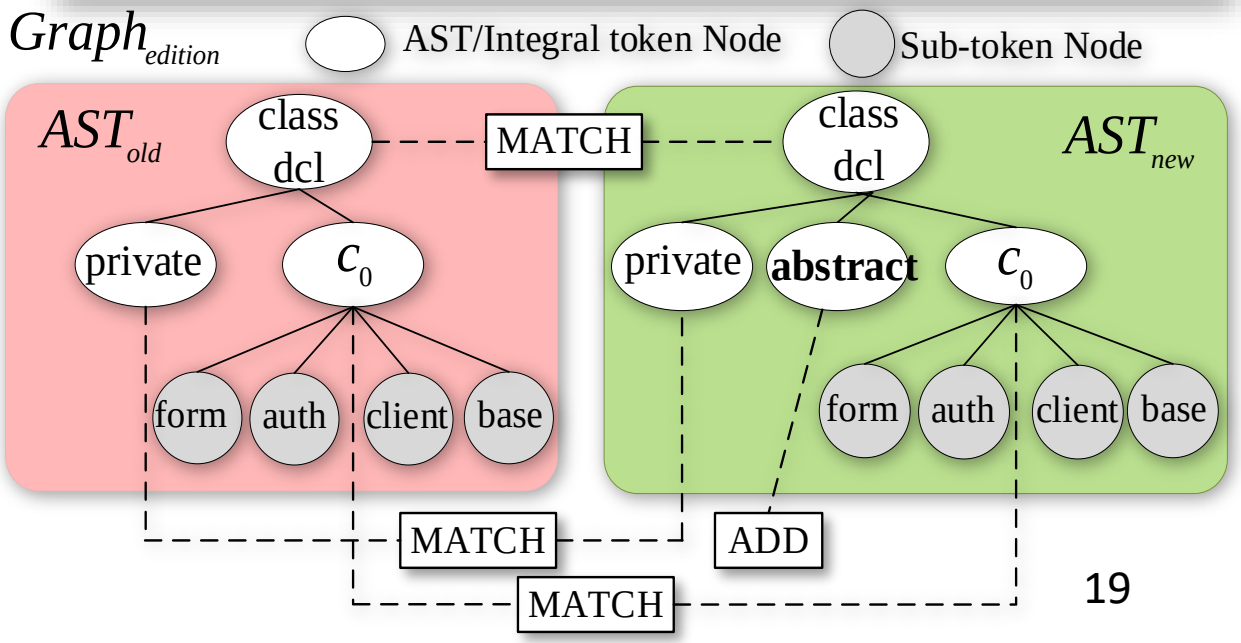
**Commit Message:**
Make base class **abstract**



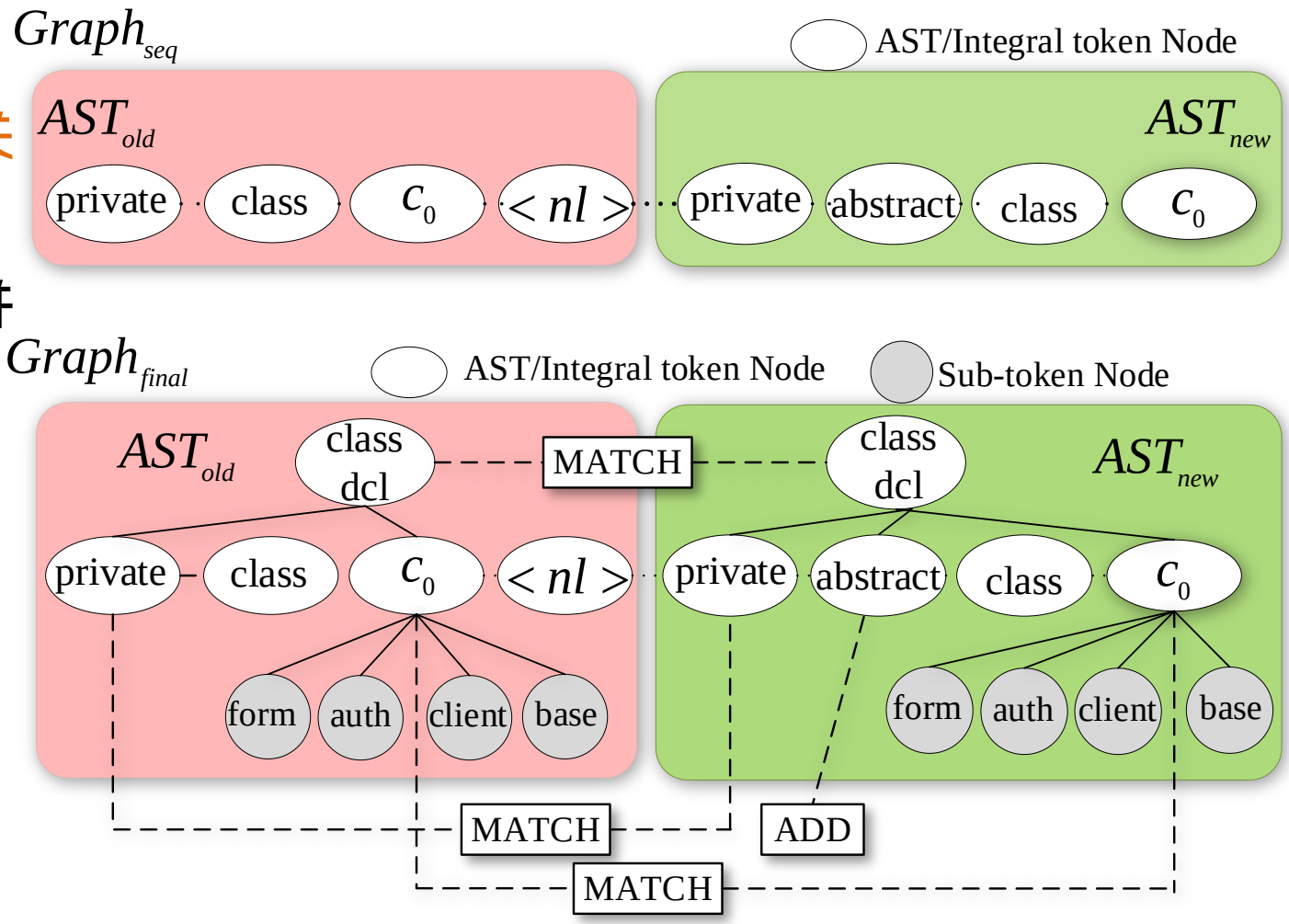
- 3. 变更代码表征：编辑操作
  - $v_{ADD}$：节点$v$在$AST_{new}$，但不在$AST_{old}$，与$v_{ADD}$连接
  - $v_{DEL}$：节点$v$在$AST_{old}$，但不在$AST_{new}$，连接$v_{DEL}$
  - $v_{MOVE}$：节点$v$在$AST_{old}$和$AST_{new}$中，其子树被移动，连接$v_{MOVE}$
  - $v_{UPDATE}$：节点$v$在$AST_{old}$和$AST_{new}$中，其值被更新，连接$v_{UPDATE}$
  - $v_{MATCH}$：节点$v$在$AST_{old}$和$AST_{new}$中，值和位置不变，连接$v_{UPDATE}$
  - 构建$Graph_{edition}$

```
@@ -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**

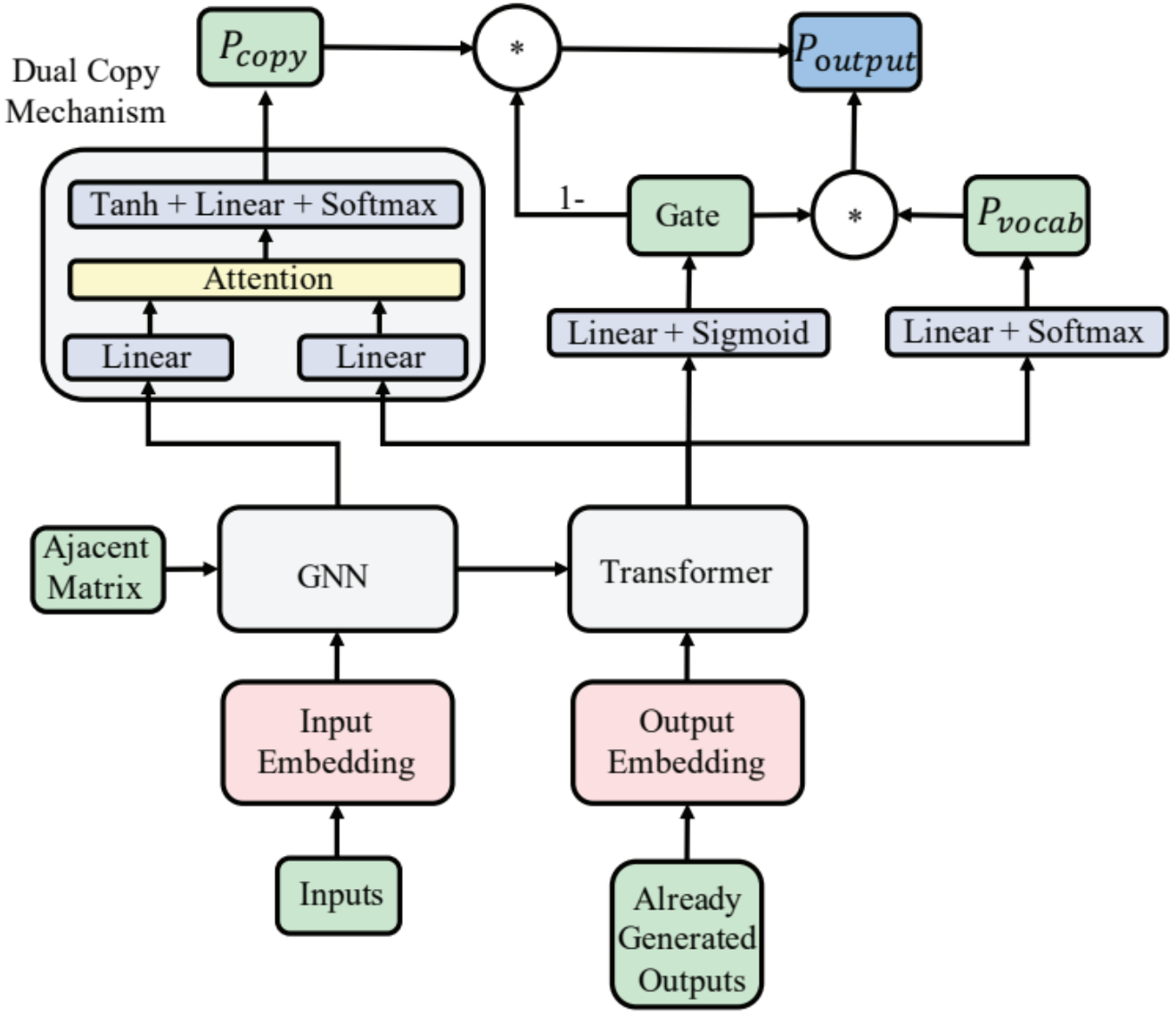


- 4. 变更代码表征：合并额外的序列信息
  - 序列信息可以保留Token的相邻关系和顺序，有助于提交信息生成
  - 通过将每个Token视为一个节点并连接两个相邻节点，构建单行图 $Graph_{seq}$
  - 构建$Graph_{final}$：利用锚节点合并$Graph_{edition}$和$Graph_{seq}$



# FIRA 算法原理

- 模型构建
  - 编码器
    - 嵌入层
    - 图神经网络：GCN
  - 解码器
    - Transformer层
    - 双复制机制

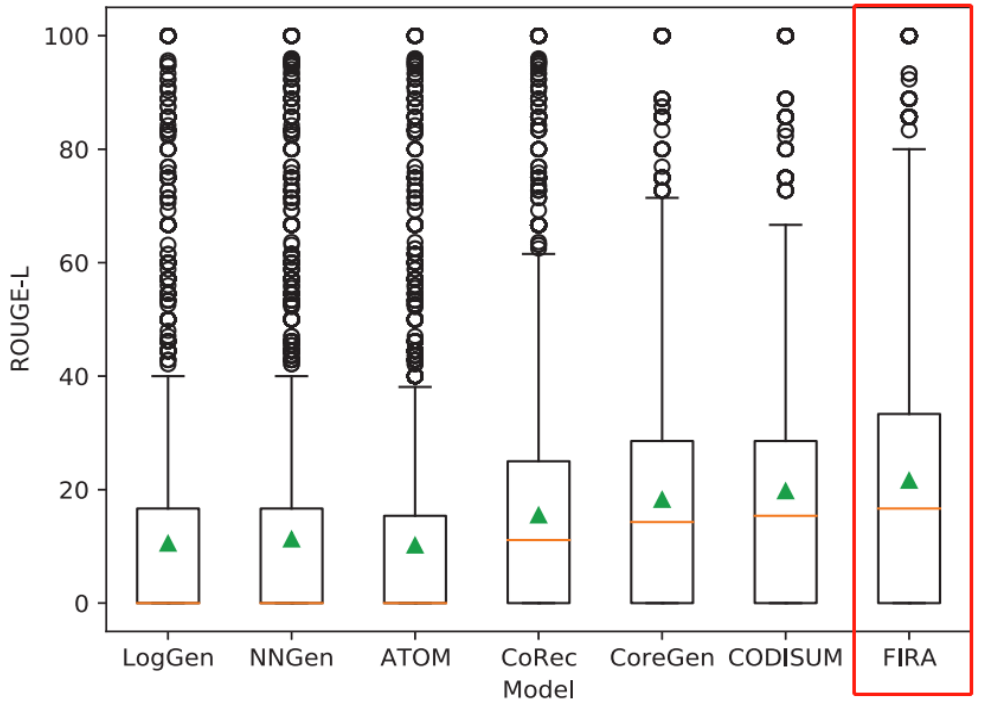


- 实验设计
  - 数据集
    - 1个基准数据集（1000个项目，含90661个提交）
    - 训练集（随机选择75000个提交）、验证集（8000个提交）、测试集（ 7661 个提交）
  - 对比方法
    - 基于信息检索的方法*2：NNGen、LogGen
    - 基于学习的方法*4：CODISUM、ATOM、CoreGen、CoRec
  - 评价指标
    - BLEU：比较生成的提交信息和参考信息的n-gram重合程度
    - ROUGE-L：比较生成的提交信息和参考信息的最长公共子序列重合程度。最长公共子序列是指生成的摘要和参考摘要之间以相同顺序出现的最长连续单词序列
    - METEOR： METEOR综合考虑了多个因素，包括精确度、召回率和单词级别的对齐。它使用了单词级别的对齐信息，将生成的文本与参考文本之间的匹配进行比较

# FIRA 实验结果

- 对比实验
  - FIRA在所有指标上，性能均优于其他六种技术
    - 基于检索的LogGen和NNGen方法仅能检索现有信息，不能生成新的提交信息
  - FIRA在生成不同长度的提交信息方面始终有效

| Model | BLEU | ROUGE-L | METEOR |
|---|---|---|---|
| LogGen [16] | 8.95 | 10.50 | 8.34 |
| NNGen [29] | 9.16 | 11.24 | 9.53 |
| CoreGen [32] | 14.15 | 18.22 | 12.90 |
| CODISUM [44] | 16.55 | 19.73 | 12.83 |
| ATOM [28] | 8.35 | 10.17 | 8.73 |
| CoRec [41] | 13.03 | 15.47 | 12.04 |
| FIRA | **17.67** | **21.58** | **14.93** |



# FIRA 实验结果

- 消融实验1-编辑操作的性能
- 实验设置
  - $FIRA_{edit-}$：在代码变更图中删除编辑操作
  - $FIRA_{sub-}$：在双复制机制中不再复制sub-token和integral token
  - $FIRA_{both-}$：移除以上两个组件
- FIRA可准确生成提交信息，删除编辑操作导致无法生成准确信息
- 明确表示编辑操作可以帮助模型捕获细粒度的代码变更，实现精确的提交信息生成

```
@@ -290,7 +290,7 @@ public abstract class PlaybackControlGlue {
        throw new IllegalStateException("Fragment
        OnItemViewClickedListener already present");
    }
    mFragment.setOnItemViewClickedListener(mOnItemViewClickedListener);
-   if (mFragment.getInputEventListener() != null) {
+   if (mFragment.getInputEventHandler() != null) {
        throw new IllegalStateException("Fragment InputEventListener
        already present");
    }
    mFragment.setInputEventHandler(mInputEventHandler);
@@ -264,7 +264,7 @@ public class PlaybackOverlayFragment extends
DetailsFragment {
-   public final InputEventHandler getInputEventListener() {
+   public final InputEventHandler getInputEventHandler() {
        return mInputEventHandler;
    }
```

**Commit message:**

| | |
|---|---|
| Ground Truth: | Rename **getInputEventListener** to **getInputEventHandler** |
| FIRA: | Rename **getInputEventListener** to **getInputEventHandler** |
| $FIRA_{edit-}$: | Fix a potential npe |
| NNGen: | Allow fadeout when fadeenabled is false |
| CODISUM: | Fix a typo in PlaybackControlGlue |

| Model | BLEU | ROUGE-L | METEOR |
|---|---|---|---|
| $FIRA_{edit-}$ | 17.39 | 21.15 | 14.54 |
| $FIRA_{sub-}$ | 17.36 | 20.97 | 14.09 |
| $FIRA_{both-}$ | 16.82 | 20.15 | 13.42 |
| FIRA | **17.67** | **21.58** | **14.93** |

# FIRA 实验结果

- 消融实验2-复制sub-token的性能
  - FIRA可以有效利用输入代码中的sub-token
  - $FIRA_{sub-}$在没有双重复机制下，无法将低频的sub-token复制到提交信息
  - NNGen：生成的提交信息相关性弱
  - CODISUM：无法生成含低频sub-token的提交信息
- FIRA可以有效利用sub-token，生成准确提交信息

```
@@ -6,6 +6,7 @@
 package net.java.sip.communicator.impl.gui.main.call;
+import java.awt.*;
 import java.awt.event.*;
 import java.util.*;
@@ -39,6 +40,8 @@ public class TransferCallDialog{
     this.setOkButtonText(GuiActivator.getResources()
         .getI18NString("service.gui.TRANSFER"));
+    this.setMinimumSize(new Dimension(300, 300));
     addOkButtonListener(new ActionListener()
     {
       public void actionPerformed(ActionEvent e)
```

**Commit message:**

| | |
|---|---|
| Ground Truth: | **Set minimum size** for **transfer call dialog** |
| FIRA: | **Set minimum size** for **TransferCallDialog** |
| $FIRA_{sub-}$: | Set dialog size to the dialog |
| NNGen: | Remove border from dialpad button to hide the button style on ubuntu |
| CODISUM: | Set the size of the size |

| Model | Copy Ratio (%) | #Different Sub-tokens | Occurrence Frequency |
|---|---|---|---|
| NNGen | 10.53 | 436 | 689 |
| CODISUM | 3.77 | 115 | 1097 |
| $FIRA_{sub-}$ | 5.40 | 159 | 1118 |
| FIRA | **11.95** | **454** | **643** |

# FIRA 实验结果

- 开发人员评估
  - 随机选择100个提交，6位具备3-5年行业经验的java编程语言开发人员
  - 利用Wilcoxon 符号秩检验，证明95%置信度下，评分具有统计学意义

```
@@ -88,6 +88,7 @@ public class DeepLearningAutoEncoderTest extends
TestUtil {
      // cleanup
      mymodel.delete();
+     frame.add("dummy", resp);
      frame.delete();
      p.delete();
      l2_frame.delete();
@@ -95,7 +96,6 @@ public class DeepLearningAutoEncoderTest extends
TestUtil {
      reconstructed.delete();
      ((Frame)DKV.get(Key.make("Difference")).get()).delete();
      diff.delete();
-     resp.remove(null);
    }
  }
```

**Commit message:**

Ground Truth: Fix **memory leak**
FIRA: Fix **memory leak** in DeepLearningAutoEncoderTest
NNGen: Add missing npe checks
CODISUM: Fix test case

| 分数 | 定义 |
|---|---|
| 0 | 在语义上不相关，也没有共享的标记 |
| 1 | 语义上无关的，但有一些共同的标记 |
| 2 | 语义部分相似，但每一个都包含排他性信息 |
| 3 | 语义高度相似但不完全相同 |
| 4 | 语义相同 |

| Model | Low (%) | Medium (%) | High (%) | Average Score |
|---|---|---|---|---|
| NNGen | 71.3 | 13.2 | 15.5 | 0.98 |
| CODISUM | 38.0 | 19.8 | 42.2 | 2.06 |
| FIRA | **35.5** | **20.3** | **44.2** | **2.15** |

# FIRA 优劣分析

- 优势
  - 细粒度的代码变更表征，有利于提交信息的准确生成

- 劣势
  - 要求代码规范性，需要满足特定的函数名、类等名称设置
  - 代码变更无法解析为AST时，FIRA有效性弱
  - 训练集包含与给定的代码变更高度相似的低频代码变更（例如，只有1次），FIRA性能弱于基于检索的方法
  - 提交信息包含词汇表和输入代码中都没有的Token时，FIRA无法生成这类提交信息

# CCRep

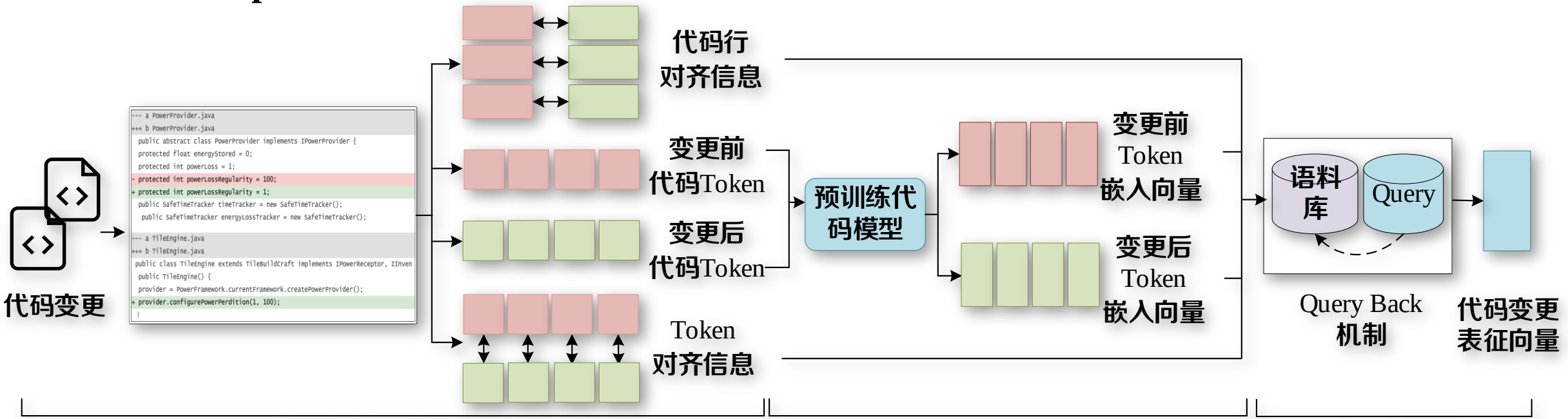
# CCRep TIPO

| | |
|---|---|
| T | 提出基于学习的通用代码变更表示方法 |
| I | 代码变更数据集 |
| P | 1. 代码变更预处理：利用代码行对齐、分词、扁平化、Token对齐技术，获取代码变更前和变更后的行对齐信息<br>2. 上下文代码嵌入：利用预训练模型获取代码变更Token序列的上下文嵌入向量<br>3. QueryBack机制：输入变更代码Token序列嵌入向量和变更代码片段对齐信息，生成代码变更向量表示 |
| O | 代码变更向量 |

| | |
|---|---|
| P | 现有方法忽略代码上下文信息、变更代码表示不明确，缺乏变更代码与整个代码变更文件之间的明确交互，难以捕获代码变更有效信息 |
| C | 利用预训练模型获取代码变更Token的嵌入向量 |
| D | 如何在代码变更中自适应选择重要信息，建立代码变更的准确向量表征 |
| L | ICSE2023（CCF-A） |

# CCRep 算法原理

- CCRep：一种代码变更表示学习方法
  - 现有方法忽略代码上下文信息、变更代码表示不明确，缺乏变更代码与整个代码变更文件之间的明确交互，难以捕获代码变更有效信息
  - 提出一种编码代码变更的QueryBack机制，自适应选择代码变更的重要信息
  - CCRep由预训练模型和查询返回机制组成，即插即用

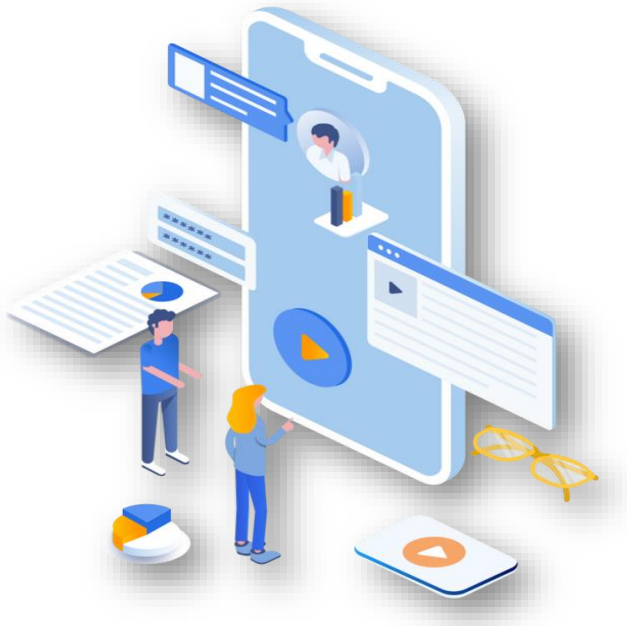


- 代码变更预处理
  - 代码行对齐(Line Aligning)

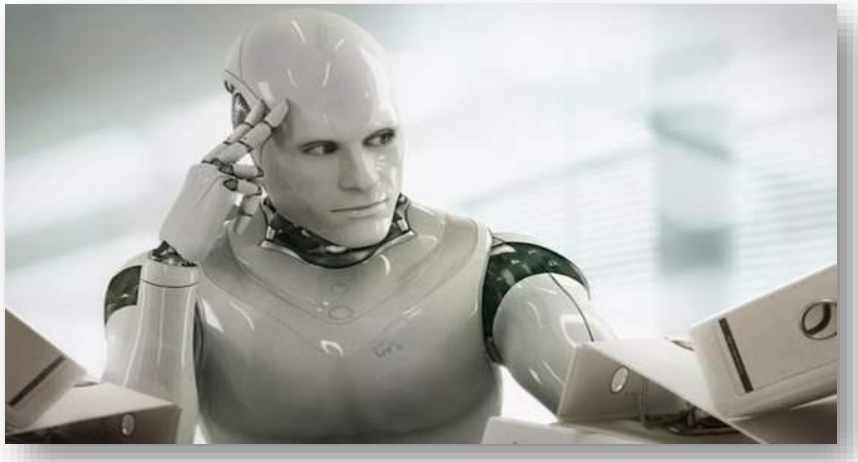
| Line Pair Index | Before-Change Code Line | After-Change Code Line | Line Change Type | $l_i^b$ | $l_i^a$ |
|---|---|---|---|---|---|
| 1 | If(cursor != null) { | If (cursor != null && cursor.moveToFirst()) { | Replace | 1 | 1 |
| 2 | cursor.moveToFirst(); | - | Delete | 2 | N/A |
| 3 | Int idx = cursor.getColumnIndex(); | Int idx = cursor.getColumnIndex(); | Keep | 0 | 0 |
| 4 | - | If (idx != -1) | Add | N/A | 4 |
| 5 | result = cursor.getString(idx);} | result = cursor.getString(idx);} | Keep | 0 | 0 |

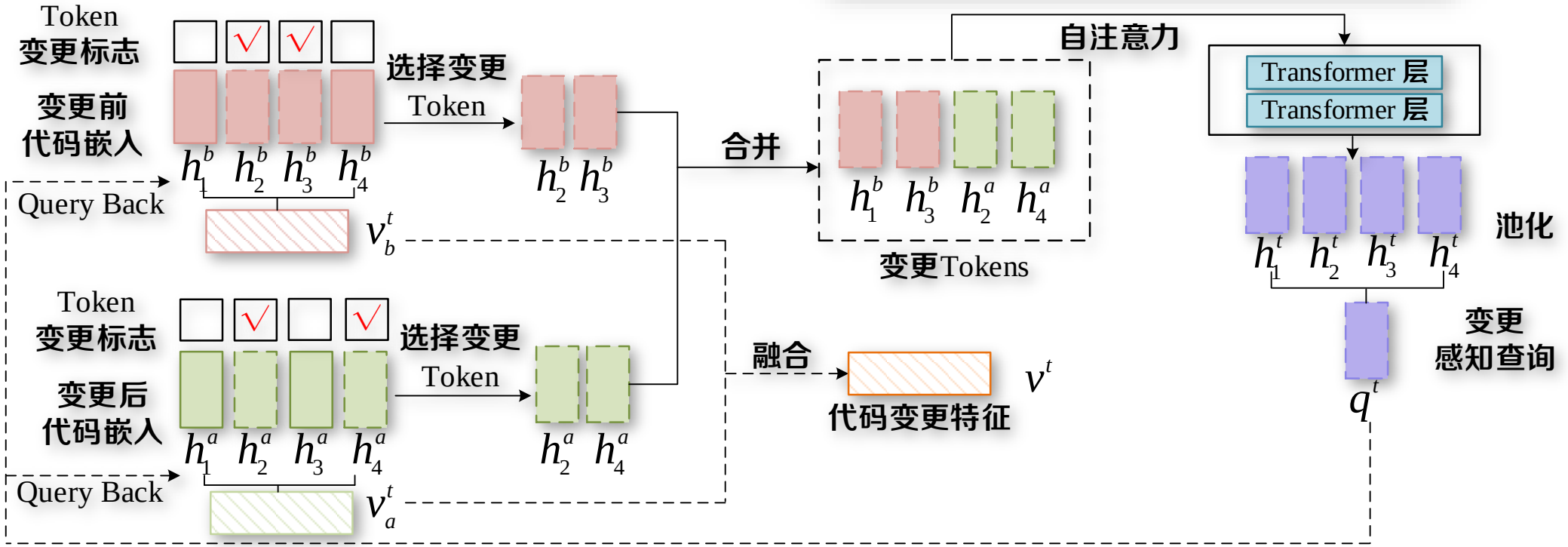
  - 分词(Tokenizing)：利用预训练模型分词器将每个代码行划分为Token序列
  - 扁平化(Flattening)：构建变更前后Token序列
    - 变更前token序列：$T^b = \left[t_1^b, t_2^b, \cdots, t_{|T^b|}^b\right]$
    - 变更后token序列：$T^a = \left[t_1^a, t_2^a, \cdots, t_{|T^a|}^a\right]$

- 代码变更预处理
  - 标记对齐(Token Aligning)：利用python difflib 工具逐个识别变更Token，建立变更标志序列，变更Token标志为1，未变更为0
- 上下文代码嵌入
  - 输入变更前和变更后的token序列，将 $T^b$和$T^a$独立编码为上下文嵌入序列：
    - 变更前代码嵌入序列： $H^b = \left[h_1^b, h_2^b, \cdots, h_{|T^b|}^b\right]$
    - 变更后代码嵌入序列： $H^a = \left[h_1^a, h_2^a, \cdots, h_{|T^a|}^a\right]$
- Query Back机制
  - Token级Query Back机制
  - Line级Query Back机制
  - Hybrid的Query Back机制



- Token级Query Back机制
  - 变更Token选择
  - Query构造
  - Query Back

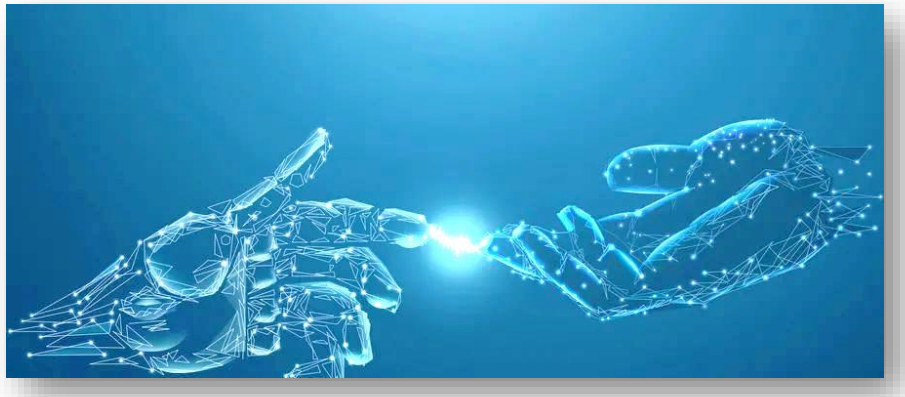


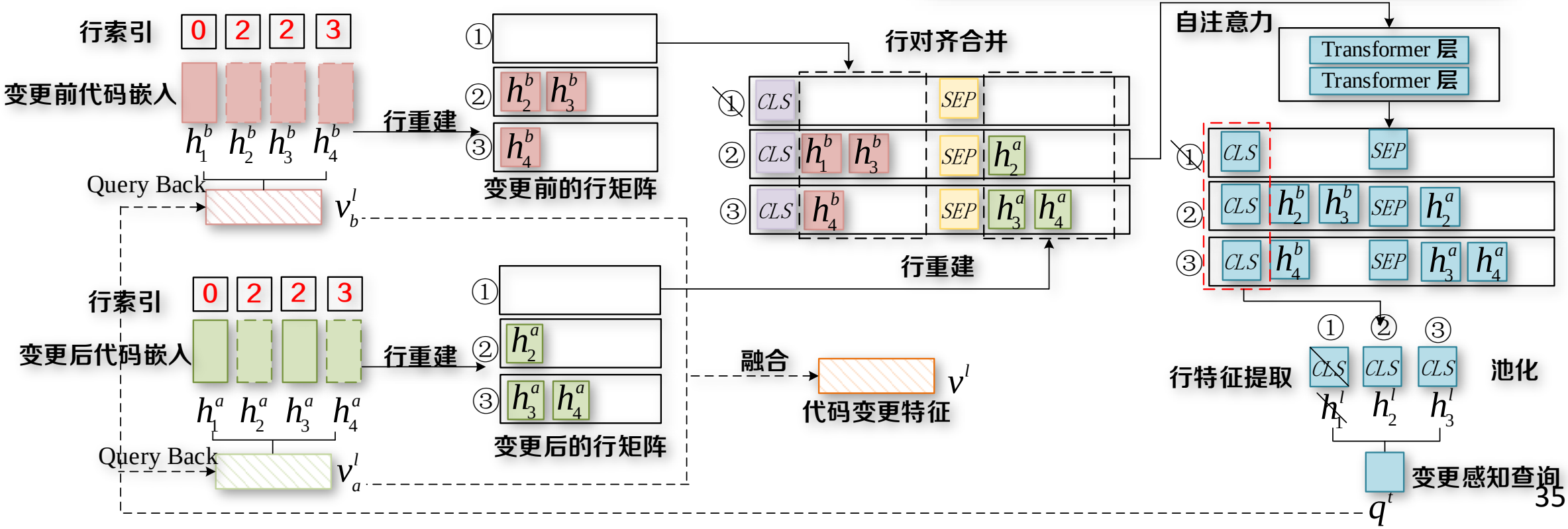


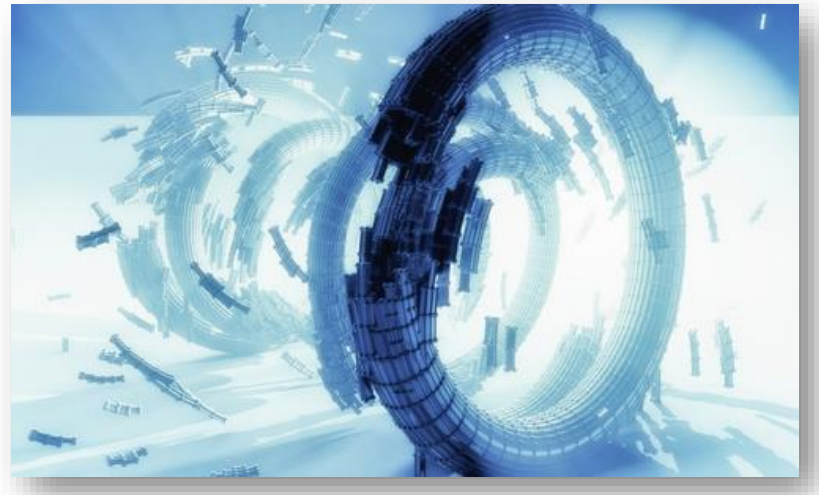
- **Token级Query Back机制**
  - **变更Token选择：构建变更Token的flag序列，选择与合并变更Token的上下文嵌入**
    - **变更前上下文嵌入序列：** $[h_1^{b'}, h_2^{b'}, \cdots, h_{n_b}^{b'}]=[h_1^b, h_2^b, \cdots, h_{|T_b|}^b] \otimes [m_1^b, m_2^b, \cdots, m_{|T_b|}^b]$
    - **变更后上下文嵌入序列：** $[h_1^{a'}, h_2^{a'}, \cdots, h_{n_a}^{a'}]=[h_1^a, h_2^a, \cdots, h_{|T_a|}^a] \otimes [m_1^a, m_2^a, \cdots, m_{|T_a|}^a]$
    - **合并后新的嵌入序列：** $H' = [h_1^{b'}, h_2^{b'}, \cdots, h_{n_b}^{b'}, h_1^{a'}, h_2^{a'}, \cdots, h_{n_a}^{a'}]$
  - **构造Query：利用Transformer提取$H'$特征$H^t = [h_1^t, h_2^t, \cdots, h_{n_b+n_a}^t]$，平均池化得$q^t$**
    - **变更感知特征向量：** $H^t = Transformer(H')$
    - **Query向量：** $q^t = Pooling(h_1^t, h_2^t, \cdots, h_{n_b+n_a}^t)$
  - **Query Back：通过注意力检索变更前后代码（语料库）中的相关信息**
    - **变更前代码中检索到的特征向量：** $v_b^t = MultiHead(q^t, H^b)$
    - **变更后代码中检索到的特征向量：** $v_a^t = MultiHead(q^t, H^a)$
  - **特征融合：获取最终的代码表征向量：** $v^t = v_b^t + v_a^t$

# CCRep 算法原理

- Line级Query Back机制
  - 变更行选择
  - 构建Query
  - Query Back融合



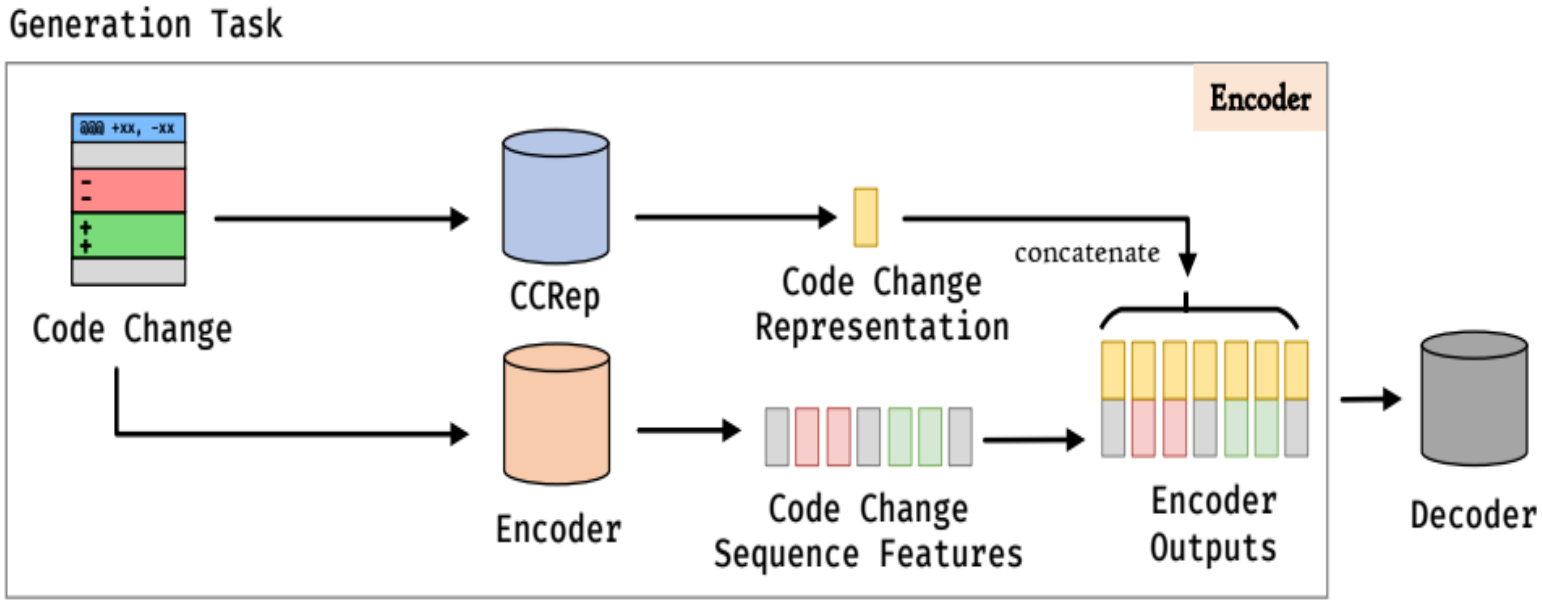




- **Line级Query Back机制**
  - **变更行选择**
    - **代码行矩阵：** $Ls^b = \left[h_1^b, h_2^b, \cdots, h_{|T^b|}^b\right] \odot \left[l_1^b, l_2^b, \cdots, l_{|T^b|}^b\right]$
    - **代码行矩阵：** $Ls^a = \left[h_1^a, h_2^a, \cdots, h_{|T^b|}^a\right] \odot \left[l_1^a, l_2^a, \cdots, l_{|T^a|}^a\right]$
    - **成对代码行：** $Ls = [CLS, Ls^b, SEP, Ls^a]$
  - **Query构造：** $q^l = MaskPooling(q^l, mask^l)$
  - **Query Back & Merging：同Token级Query Back机制**
- **Hybrid Query Back机制**
  - **结合Token级和Line级的代码变更表示，构建混合代码变更表征向量**
  - $v^h = LayerNorm(W_t^T v^t) + LayerNorm\left(W_l^T v^l\right)$

# CLARE 实验设计

- 实验设置
  - 结合CCRep和预训练模型CodeBERT输出的特征向量，输入解码器以生成提交信息
  - 两个数据集：CoReC数据集（20.5K个commit）和FIRA数据集（90.6K个commit）
  - 三个评估指标：BLEU、METEOR和ROUGE-L
- 基线方法
  - FIRA
  - NNGen
  - CODISUM
  - CoReC
  - LogGen



# CLARE 实验结果

- 对比实验
  - 在CoReC数据集上，Token级CCRep在三项指标中，比性能最优的CoReC分别高出11.8%、10.8%和15.1%
  - 在FIRA数据集中，Token级CCRep在三项指标中，比性能最优的FIRA分别高出12.8%、9.0%和10.3%
  - 利用自举重采样随机抽取1000个样本进行统计显著性检验，所有p值均小于0.001，与基线方法性能差异显著

CMG: EVALUATION RESULTS ON THE COREC DATASET

| Model | BLEU | METEOR | ROUGE-L |
|---|---|---|---|
| LogGen [8] | 9.41 | 5.31 | 12.13 |
| NNGen [20] | 23.29 | 14.26 | 28.68 |
| CODISUM [37] | 13.15 | 7.35 | 16.49 |
| CoReC [13] | 25.27 | 15.34 | 29.73 |
| $CCRep^{token}$ | **28.24** | **16.99** | **34.23** |
| $CCRep^{line}$ | 26.65 | 15.74 | 32.01 |
| $CCRep^{hybrid}$ | 27.25 | 16.58 | 33.30 |

CMG: EVALUATION RESULTS ON THE FIRA DATASET

| Model | BLEU | METEOR | ROUGE-L |
|---|---|---|---|
| LogGen [8] | 8.95 | 8.34 | 10.50 |
| NNGen [20] | 9.16 | 9.53 | 11.24 |
| CODISUM [37] | 16.55 | 12.83 | 19.73 |
| CoReC [13] | 13.03 | 12.04 | 15.47 |
| FIRA [35] | 17.67 | 14.93 | 21.58 |
| $CCRep^{token}$ | **19.93** | **16.27** | **23.81** |
| $CCRep^{line}$ | 19.79 | 16.06 | 23.60 |
| $CCRep^{hybrid}$ | 19.70 | 15.84 | 23.41 |

# CLARE 实验结果

- 消融实验
  - CCRep采用Token级的CCRep进行实验
  - CCRep-CodeBERT：利用RoBERTa基础模型替代CodeBERT
  - CCRep-QueryBack：删除CCRep中的QueryBack机制，将CodeBERT编码器输出的代码变更表征向量，输入解码器以生成commit信息
  - 相比CCRep-QueryBack， CCRep的p值均小于0.001。与CCRep-CodeBERT相比，CCRep的p值均小于0.05，表明CodeBERT在多数情况下是有益的

| Model | BLEU | METEOR | ROUGE-L |
|---|---|---|---|
| CCRep – CodeBERT | 27.90 | 16.56 | 33.36 |
| CCRep – QueryBack | 26.23 | 15.72 | 31.78 |
| CCRep | **28.24** | **16.99** | **34.23** |

| Model | BLEU | METEOR | ROUGE-L |
|---|---|---|---|
| CCRep – CodeBERT | 18.77 | 15.54 | 22.37 |
| CCRep – QueryBack | 17.88 | 14.78 | 21.33 |
| CCRep | **19.93** | **16.27** | **23.81** |

# 应用总结

- 优势
  - 提出一种多粒度变更代码表征方法，通用性强，适用软件工程领域多种下游任务
  - 可根据任务需求，嵌入多种预训练模型，包括CodeBERT等
- 劣势
  - 如果代码变更中上下文代码较少，Query Back机制的有效性会降低
  - CCRep可处理代码变更长度受预训练模型限制，即变更前后代码长度不超过预训练模型的长度限制，通常为512

# 前沿发展

- 未来工作
  - 代码变更的预训练
    - 大多数代码变更相关的软件工程任务难以收集到大量有标签的代码变更 (例如安全漏洞补丁)，而开源项目的代码仓库中包含丰富的无标签代码变更数据。
    - 意味着预训练技术有望提升代码变更表示学习方法的性能
  - 多任务代码变更表示学习
    - 自然语言处理等领域，多任务学习的有效性已经反复得以验证
    - 多任务学习能帮助代码变更表示学习模型分辨不同任务相似却不同的信息需求，帮助模型捕捉到代码变更中多种维度的特征
  - 代码变更层次信息的利用
    - 代码变更中的信息是层次化的，例如一次代码提交可能改动多个文件，一个被改动的文件内可能包含多个变更的代码块等。层次化信息有望提升代码变更表示学习的性能并帮助设计出适用于多种代码变更粒度的通用技术

# 参考文献

[1] Dong J, Lou Y, Zhu Q, et al. FIRA: fine-grained graph-based code change representation for automated commit message generation[C]//Proceedings of the 44th International Conference on Software Engineering. 2022: 970-981.

[2] Liu Z, Tang Z, Xia X, et al. FIRA: Fine-Grained Graph-Based Code Change Representation for Automated Commit Message Generation[C]//Proceedings of the 44th International Conference on Software Engineering. 2023.

# 谢谢！

大成若缺，其用不弊。大盈若冲，其用不穷。大直若屈。大巧若拙。大辩若讷。静胜躁，寒胜热。清静为天下正。