# 基于IDE开发动作序列的意图预测

## 课题概述

在现代软件开发中，集成开发环境（Integrated Development Environment, IDE）已经成为程序员不可或缺的工具。它不仅提供代码编辑功能，还集成了编译、调试、版本控制等一整套开发工具，大大提高了开发效率。随着人工智能和机器学习技术的发展，将各类智能模型融入IDE，以辅助开发者进行更高效和智能的编码工作，成为了当前研究的一个热点。本课题旨在探索通过分析开发者在IDE中的交互动作序列来预测其编程意图的可能性。

具体来说，我们将研究如何表示、捕捉和存储开发者在IDE中的各种操作行为，如代码编写、文件切换、命令执行等，形成一系列的动作序列数据。这部分我们称为“前端部分”。

然后，对于收集到的开发者行为数据，需要进行一系列预处理和整理工作，比如数据清洗、数据转换、特征工程。之后需要选择和应用适当的模型算法来对整理后的序列数据进行建模，从中挖掘出能够反映开发者意图的模式和特征。这部分我们称为“后端部分”。

最后，在成功实现数据处理与建模两大部分后，将这两部分进行整合，构建出一个能够实时监测并解析开发者行为的系统原型。该系统基于开发者的历史行为模式和当前的动作序列，预测其下一步可能的操作或需求的功能，并据此提前准备或推荐相关资源，为用户提供个性化的开发辅助。通过这种预判性支持，系统不仅可以减少开发者的重复劳动，还提供了更加流畅的工作流程，增强了用户体验。后续，还可以期待此系统与IDE内置智能助手的集成可能性。通过向智能助手提供实时的用户意图数据，可以使助手生成的内容和服务更贴合用户的实际需要，进一步提升开发效率和代码质量。

## 前端技术路线

### 2.1 总体功能

前端以vscode插件的形式进行开发，通过IPC与后端进行通信。主要功能是收集开发者的原始开发动作序列，并以json格式发给后端。

### 2.2 插件数据收集

为了系统地采集来自不同教育背景和技能水平的开发者的编程行为数据，我们开发了一个基于vscode插件的开发者行为自动化采集工具。该工具通过调用多种IDE的数据收集API，捕获开发者在编程过程中的各种行为，包括代码编写、文件操作、调试过程等。

在数据收集中，除了直接调用VS Code提供的API，还将引入代码静态分析和代码仓搜索等技术。代码静态分析可以帮助工具获取代码的结构和质量信息，如代码复杂度、依赖关系、注释情况等。

插件将捕获的行为和相关的数据保存为json文件。其中，需要记录的行为根据性质的不同被分为了几类事件。

#### 文件级事件

|  |  |  |
| --- | --- | --- |
| 编号 | 名称 | 符号 |
| 1-1 | 打开文本文件 | OpenTextDocument |
| 1-2 | 关闭文本文件 | CloseTextDocument |
| 1-3 | 切换文本编辑器 | ChangeTextDocument |
| 1-4 | 新建文件 | CreateFile |
| 1-5 | 删除文件 | DeleteFile |
| 1-6 | 保存文件 | SaveFile |
| 1-7 | 重命名文件 | RenameFile |
| 1-8 | 移动文件 | MoveFile |

#### 文本内容相关事件

|  |  |  |
| --- | --- | --- |
| 编号 | 名称 | 符号 |
| 2-1 | 添加文件内容 | AddTextDocument |
| 2-2 | 删除文件内容 | DeleteTextDocument |
| 2-3 | 修改文件内容 | EditTextDocument |
| 2-4 | 重做文件内容 | RedoTextDocument |
| 2-5 | 撤销文件内容 | UndoTextDocument |
| 2-6 | 选中文本 | SelectText |
| 2-7 | 鼠标悬停 | MouseHover |

#### 终端事件

|  |  |  |
| --- | --- | --- |
| 编号 | 名称 | 符号 |
| 3-1 | 打开终端 | OpenTerminal |
| 3-2 | 关闭终端 | CloseTerminal |
| 3-3 | 切换终端 | ChangeActiveTerminal |
| 3-4 | 执行终端命令 | ExecuteTerminalCommand |
| 3-5 | 获取调试控制台输出 | DebugConsoleOutput |

#### 执行菜单项事件

|  |  |  |
| --- | --- | --- |
| 编号 | 名称 | 符号 |
| 4-1 | 执行菜单项 | ExecuteMenuItem |

### 2.3 插件数据保存

对于不同的事件，需要记录的信息也不同，根据这些事件进行归纳整理，我们设计了通用数据结构来记录所有事件的信息。其中所有事件都需要包含信息被称为通用属性。除此之外某些事件具有复杂的属性（比如文本编辑事件），对此我们用额外的属性来记录这些信息。

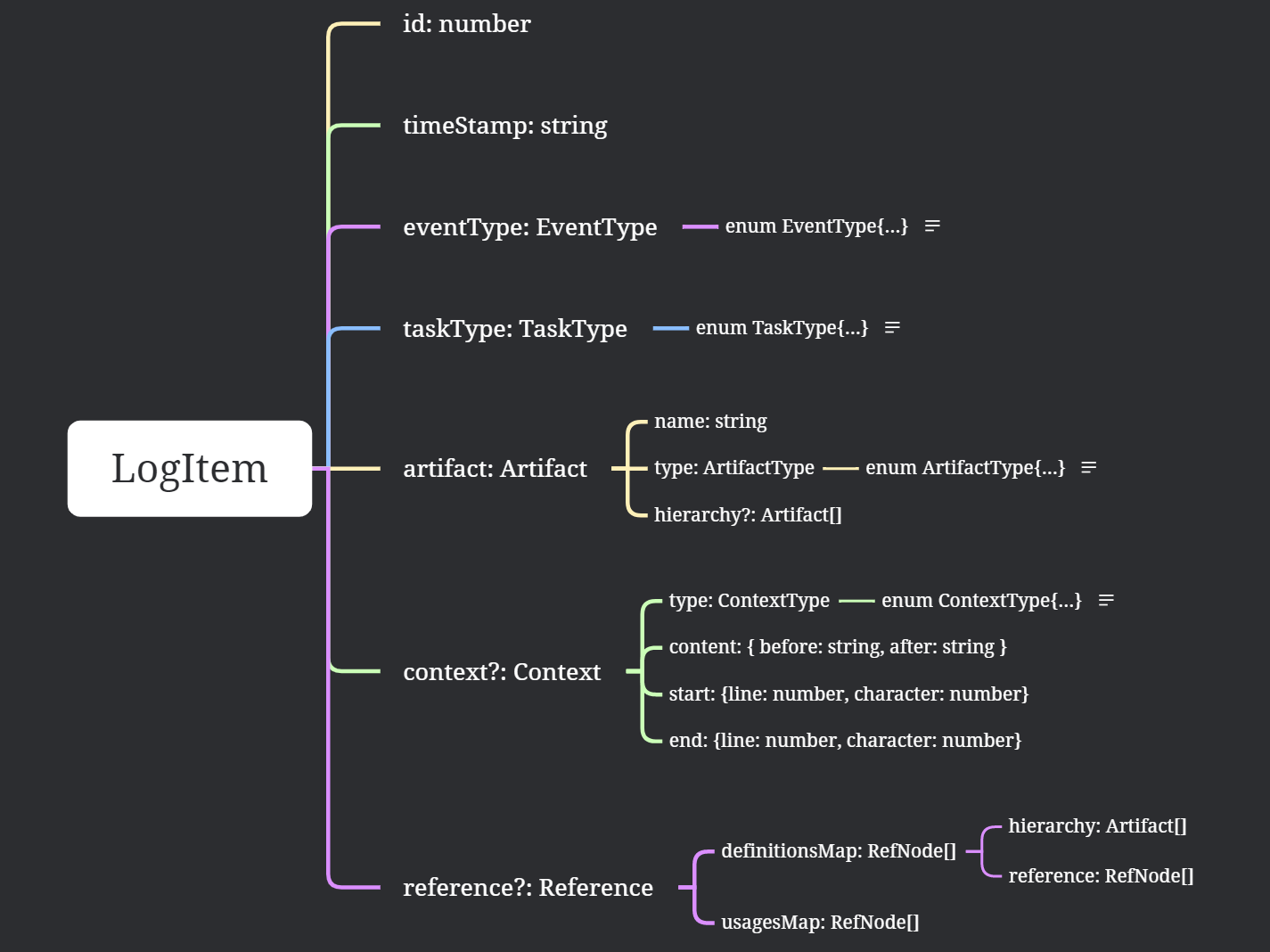
#### 通用属性

以下属性为通用属性，每个事件类型都会包含

* id: number 在本次记录中的序号
* timeStamp: string 记录本事件的时间戳
* eventType: EventType 事件类型
* taskType: TaskType 所处任务类型
* artifact: Artifact 操作工件
  + name: string 工件名称
  + type: ArtifactType 工件类型
  + hierarchy?: Artifact[] 该工件的层级（可能没有）

#### 完整属性

除了上述属性，对于涉及到具体文本操作的事件，需要记录文本操作内容和操作内容涉及到的工件关系，需要额外的属性来进行记录。最终我们得到的通用数据结构如下所示。



## 后端技术路线

### 3.1 总体设计

后端采用python环境，拟通过IPC与插件前端进行通信。同时方便离线和在线AI模型的训练和部署。主要分为五个模块：数据预处理、行为模式识别、行为总结、操作预测、能力分析。拟向上提供三种能力，即上层@VirtualMe时能够选择输入的三个问题：过去一段时间内开发者的行为总结、开发者的操作预测、开发者能力分析。

### 3.2 各模块方案

#### 数据预处理

收到插件前端发来的json格式的原始数据后，在预处理模块进一步处理：

1. 格式转换：前端收集的事件类型基于API，json文本能进一步总结和转换成python对象方便后续处理。

2. 剔除冗余数据：看数据集情况

3. 形成历史操作件artifact\_history：用于后续的预测检索

4. 形成历史命令cmd\_history：用于后续的预测检索

处理后，形成了python对象的序列化数据。

拟向上提供三种能力，即上层@VirtualMe时能够选择输入的三个问题：过去一段时间内开发者的行为总结、开发者的操作预测、开发者能力分析。

#### 行为模式分析

根据程序员开发过程中普遍的几种行为类型，初版原型工具设定如下几种固定的模式/状态，状态内的操作都有明显的类型倾向。方案拟采用预训练的液态神经网络CFC（Closed-form continuous-time neural networks）对序列数据进行实时分类。

**模式分类**

主要讲行为模式分为四类，第五类为其他或中性操作：

1. **配置环境**  
   执行的命令类型多为install、可能出现静态检查问题、操作的文件多为配置文件。  
   安装依赖包：pip, mvn, conda, npm等  
   编辑配置文件：packge.json, requirement.txt, cmakelist.txt, build.gn等  
   编辑源代码的依赖相关部件：import, include, 增删部分用到依赖包的地方等
2. **调查阅读**  
   鼠标滚轮滑动、选中文本但不编辑、文件跳转、代码符号跳转、搜索事件密集，编辑操作少。  
   单文件阅读代码：鼠标停留，查看悬停文档，滚轮滑动  
   跨文件阅读代码：前面的行为+文件跳转  
   查找关联部件：代码符号跳转、文件搜索、全局搜索、文件跳转、查找引用
3. **编写内容**  
   一段时间内主要事件都是文本变化。  
   增加：总体上新增文本内容/新增文件，体现为artifact的增加  
   修改：改变代码编排结构/移动重命名文件，比如修改某个func的出入参时，要修改func内逻辑、调用func的artifacts的逻辑，包含了重构行为  
   减少：总体上删除文本内容/删除文件，体现为artifact的减少  
   注释：增改注释
4. **执行验证**  
   频繁执行命令和调试，可能配合少量编辑操作，原因复杂，这种情况可能是在调整运行效果、验证某些功能，也可能是在修复某些问题，直接原因可能来自调试输出或终端也可能不是，因此统称为“验证”。  
   修改代码：不同于前面的编写，这里来源于执行的结果  
   阅读代码：不同于前面的阅读，这里在思考问题原因等  
   执行命令：g++, gcc, python, 启动debug调试等
5. **其他操作**  
   不明确属于任何一类模式，有可能什么都不做，有可能是以下没有明显意义的操作，可能穿插在以上所有模式中：  
   文件开关切换、终端开关切换：这些切换操作在每个模式里都可能出现，可能代表注意力的转移但是注意什么我们不知道  
   常见终端命令、未知终端命令：cd/pwd/su等过于常见的命令、规则库之外的终端cmd

**模式识别**

拟采用CFC模型进行操作序列的分类即模式识别。具体的特征编码策略为：将每个操作提取出三个特征：

1. 事件类型特征：使用one-hot编码
2. 操作件特征：针对每个事件里具体操作的对象（文件、代码语义符号等），提炼操作件类型、操作件功能、与前面操作间是否存在引用关系及深度、与前面操作件的操作耦合度（参考论文《Recognizing Developer Activity Based on Joint Modeling of Code and Command Interactions》和《Recording and Interpreting Developer Behaviour in Programming Tasks》中的指标）。
3. 上下文特征：某些事件存在对应的上下文，例如终端执行的上下文是执行输出是否成功；文本编辑操作的上下文是编辑内容量和范围

于是对于每个原子操作可以编码为[事件类型, 操作件, 上下文]向量。标签为收集数据时用户的打标tag，即模式的类别。

#### 行为总结

对于每一种行为模式，执行不同方案的行为总结：

**环境配置**

内容：

1. 添加的依赖包和版本+是否成功

2. 配置的项目参数和指标

3. 使用的工具链和框架

方案：

1. 规则匹配历史命令：pip/conda/npm/cnpm./yarn install xxx

2. 读取packages.json：检测修改过的包+版本号

3. 读取BUILD.gn/CMakeLists.txt等编译脚本：检测引用的包

4. 读取其他项目配置文件或特殊命令，根据支持的项目类型进行规则适配

**调查阅读**

内容：

1. 重点关注的文档+行号范围

2. 重点关注的内容的摘要

方案：

1. 频度统计最近最常浏览的范围

2. 从前端获取这些文档的内容，调用LLM总结这些范围的摘要

**编写代码**

内容：

1. 重点编辑的源代码+行号范围

2. 重点编辑过的代码变更摘要

方案：

1. 频度统计最近最常编辑代码文件

2. 如果是git仓库：先git log 重点源代码获得修改记录过滤出最近时间的commit内容，然后git diff HEAD获得最近未commit的更改记录，综合得到重点源代码文件的变更记录

3. 如果不是git仓库：回溯操作，从某个时间点阈值开始回溯式合并文件变更操作，形成文本变更记录

4. 前端获得变更记录后，调用LLM总结这些变更的摘要

**执行验证**

内容：

1. 经常执行过的命令+是否成功

2. 伴随执行操作而修改过的代码变更摘要

方案：

1. 频度统计最近最常执行的命令

2. 读取对应的命令行输出，判断是否成功，如果不成功，摘出traceback

3. 代码变更摘要参考前面方案

**其他操作**

内容：

1. 重点编辑的源代码+行号范围

2. 经常执行过的命令+是否成功

方案同上。

#### 操作预测

预测开发者的意图，具体包括：

1. 工作模式：调用模式识别模块
2. 操作：操作对象、命令。

其中，因为要考虑预测的操作件是不是前面操作过的，所以设计两种预测方式。首先判断下一步操作件是已知的还是未知的，如果是已知的，则执行检索方式，提供artifact\_history、cmd\_history中的检索结果；如果不是，则指导前端进一步根据repo内容和行为模式，进一步获取信息并提供结果。最终的输出结果是否同时包含新的和旧的预测结果，取决于新旧预测的置信程度，具体阈值拟根据数据集验证情况设定。

**新旧操作件预测**

拟采用简单的二分类神经网络，进行增量式学习，实时更新模型参数。对于序列中的每个操作件，他相对于前缀序列都有一个唯一的新旧标签可以自动标出，实现无监督学习。

模型的输入为前述的对于操作件的特征编码向量。输出则为one-hot编码的新旧二分类向量。

**旧操作件检索**

拟采用规则+检索的方式。对于操作件的预测：使用简单的逻辑回归模型对操作件历史列表的项进行打分，模型输入与前述的操作件特征向量相同，输出为分数，其中对于每一种行为模式都设置一个单独的模型，为了加强区分不同模式下对操作件语义的不同理解。对于终端命令的预测：直接使用频度统计+模式相关度匹配的方式。

**新操作件获取**

拟采用规则匹配的方式指导前端获取不存在于历史列表的里的操作件和命令。配置环境时，提供仓库内于环境配置有关的文件和参数值；调查阅读时，提供旧操作件涉及的依赖项；执行验证时，提供traceback中抛出异常的有关操作件等。

#### 能力分析

该模块拟通过前述的预处理后的数据进行分析和特征提取，匹配到PMMI等开发者能力评估模型中，具体的方案有待完善。

## 当前成果

对于前端部分，经过了多轮迭代，我们已经构建了开发者在IDE内行为的表示方法，并在基于VS Code开发了数据收集插件。该插件经过了多轮迭代，正在不断地进行完善。在完成插件构建后，需要使用该插件收集数据并构建数据集，当前数据收集正在进行中。



对于后端部分，我们搭建了项目的整体框架，并完成了数据预处理和行为模式分析部分，但目前还缺少高质量数据，其他部分具体代码还没有编写完成，模型训练还没有开始。

## 后续规划

前端部分：继续完善VS Code插件，确保其稳定性和数据收集的准确性。针对开发者行为的不同场景进行测试，以保证插件能够在各种开发环境下可靠运行。继续数据收集，期望建立一个丰富且多样化的数据集。

后端部分：一旦获得了足够的高质量数据后，将开始尝试模型训练，寻找最适合开发者意图预测任务的模型架构并优化模型性能。完成模型训练后，将前端的数据收集模块与后端的预测分析模块集成，形成完整的预测系统。