# VirtualMe 后端模型设计

目录

[VirtualMe 后端模型设计 1](#_Toc185782652)

[一、 总体设计 1](#_Toc185782653)

[二、 各模块方案 2](#_Toc185782654)

[2.1 数据预处理 2](#_Toc185782655)

[2.2 行为模式识别 2](#_Toc185782656)

[2.2.1 模式分类 2](#_Toc185782657)

[2.2.2 模式识别 4](#_Toc185782658)

[2.3 行为总结 4](#_Toc185782659)

[2.4 操作预测 6](#_Toc185782660)

[2.4.1 新旧操作件预测 6](#_Toc185782661)

[2.4.2 旧操作件检索 7](#_Toc185782662)

[2.4.3 新操作件获取 7](#_Toc185782663)

[2.5 能力分析 7](#_Toc185782664)

## 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。

## 各模块方案

### 2.1 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。

### 2.2 行为模式识别

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。

#### 2.2.1 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

#### 2.2.2 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。

### 2.3 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。

### 2.4 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。

#### 2.4.1 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。

#### 2.4.2 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。

#### 2.4.3 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。

### 2.5 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。