An Empirical Study of Language features usage in C/C++

An Empirical Study of Language features usage in C/C++

- Topic introduction
  - Abstract
  - Target
- Design
  - Tools for generating trees
    - Tools we have explored
    - Why Tree-sitter?
  - C/C++ Features
  - Data set
  - CCScaner
    - Expected functionality
    - Project structure
      - Preliminary preparation
      - Mid-term data processing
      - Post analysis
    - Analysis methods
      - Query
      - Tree cursor
  - Technical route
- Division of labor & Contribution ratio
- Implementation
  - Inner structure of CCScaner
  - How to analyze a repository?
- Data & Analysis
  - Raw data
    - Repositories selected
    - Raw data
  - Analysis
- Conclusion
  - A summary of what we have done
  - Deficiencies and prospects
- Reference

Topic introduction

Abstract

本项目旨在分析目标语言`C/C++`的各项语言特性和机制在开源项目中的使用情况。在实现中将目标特性划分为了 Template & Modular Programming 、 Concurrency & Multithreading 、 Memory Management、Exception Handling、Polymorphism & Overloading、Reference Control、Function、Type System 等八个类别，并在每一类别中选取了相应的若干特征项；继而利用我们完成的工具 CCScaner 对一批开源项目仓库进行了统计分析，得到统计结果。本项目将基于这一统计结果进行进一步的分析与讨论，以加深对这一经典语言的诸多特性的理解。

Target

1. 整理汇总我们所关注的 C/C++ 语言特征，划分成为不同类别，并在每一类别中选取若干具有代表性的特征项
2. 基于我们完成的工具 CCScaner 分析 C/C++ 的各项语言特征在开源项目中的使用情况，获得基础数据
3. 对得到的基础数据进行分析与讨论，关注：
    1. C/C++ 语言中的哪些特性具有比较显著的影响，并得到大量应用
    2. 不同领域语言使用者的特性使用倾向以及其内在原因

    等等。

Design

Tools for generating trees

Tools we have explored

我们所考虑的工具均是`python`库的形式，即以`python`语言为基础，其原因是`python`具有脚本语言特有的便捷性，此外对于数据的分析与处理也具有很好的支持。

- `pycparser`

  `pycparser`可将`C`语言代码解析为`AST`，可以作为`C`编译器或者其他分析工具的前端。

  - 只支持`C`语言

- `pygccxml`

  基于`GCC-XML`，后者后来被`Castxml`所替代。其目的是从`GCC`的内部表示生成 C++ 程序的 XML 描述。并且由于 XML 易于解析，因此其他开发工具将能够与 C++ 程序一起使用，而无需复杂的 C++ 解析器的负担。

  - 支持的版本落后，`c++98`能够全面支持，但`c++11`与`c++14`只能做到运行通过测试，更新的版本甚至未经过测试

- `lib-clang`

  基于`C`语言的`libclang`，后者又基于`clang`。`libclang`是`clang`的稳定的高级 C 接口。

然而以上工具普遍存在以下缺点：

1. 支持的语言或者版本有较大限制
2. 容错性不好，对于稍不符合要求的语法（比如新版本加入的特性）就完全无法运行
3. 头文件展开会导致代码中加入大量库文件内容，影响分析
4. 运行速度慢，难以在合理的时间内对大规模的仓库展开统计
5. 如欲对仓库展开分析，需要引入预编译过程，可能需要自己调配编译参数

所以，我们希望能够找到一个容错性好、不引入大量库文件内容、运行速度快、不涉及直接操作仓库过程的语言分析工具。

Why Tree-sitter?

> Tree-sitter 是一个解析器生成器工具和增量解析库。 它可以为源文件构建具体的语法树，并在源文件编辑时有效地更新语法树。

基于探索部分寻找到的工具的种种缺陷，以及前述的诉求，我们最终找到了一个新的替代工具，即 Tree-sitter 。此工具具有如下优点：

- 普适性好：虽然在此次实验中我们所完成的是CCScaner，但只要在相同的框架下简单更改项目核心文件ccscaner.py中的树访问逻辑与cctable.py中的特征表条目，就可以轻松移植成为RustScaner、GoScaner、HaskellScaner、MyLanScaner等等。

  > 这里选用CCScaner的原因是在探索过程中我们还不确定方案的可行性，所以选择了一个大家相对较熟悉的语言作为分析对象

- 分析快捷：结合query查询与cursor的方式可以实现目标信息的快速提取，因而有能力对较大型的仓库进行分析
- 鲁棒性好：容错性极佳，对于任何文件均可以流畅运行，对于语法错误或者不支持的语法只会造成ERROR结点并在分析时被忽略，不会因为少数的语法问题造成整个仓库的分析无法运行。
- 依赖项少：只需要编译生成一个针对目标语言的动态链接库，并在python中安装对应库即可运行。
- 不引入预编译过程：不会由于头文件展开导致代码中加入大量库文件内容；不需要针对仓库单独配置编译参数。

然而上述优点部分是建立在如下的牺牲上的：

- Tree-sitter是一个词法分析器，生成的结果是CST而非AST，因而造成：
  - 缺乏一些重要的语义信息，比如类型标注，继而造成识别能力有限
  - 由于缺乏预编译过程，在不引入库文件内容的同时也造成存在于代码中的宏无法展开
  - 由于C/C++的宏展开替换过于灵活，对于某些宏在应用中的特殊情形容易误识别造成统计的偏差

因而需要在Tree-sitter与pygccxml/lib-clang两类工具中进行权衡。最后综合以下考虑，我们选择使用Tree-sitter作为主要的分析工具：

- 目标语言C/C++相对来说很多特性可以通过关键字或者局部词法信息判断
- 为了在给定时间内分析达到满足统计要求的数据量，需要选择性能相对较优秀的分析工具
- 尽量减少人为对目标仓库的操作，比如配置编译参数，因为不同的编译参数经过预编译后对应不同的代码，其特性分布也有所不同
- 目标仓库列表中包含跨越多个C/C++版本的语法，而工具即使是最新版本，其覆盖范围仍是有限的，因而要求必须具有一定的容错能力

既然已经选用Tree-sitter，针对前述的种种缺陷，需要对选择的特征项进行一定约束，规则如下：

- 目标特征项一定是在词法上可以区分的
- 并且这种区分是局部性的，即不可跨越多个文件，这基于两个方面的考虑：

    - 目标仓库内容繁多，`ccscaner`的处理是面向单个文件的，过程中会记录大量信息并存储，可能产生大量文件，如果对这些文件进行综合分析，时间开销比较大
    - 跨越多个文件的处理有时可能会在信息上相互依赖并利用到一些语义信息进行区分，这是工具所不能够支持的
- 特征尽量不易受宏展开以及`typedef`等操作对词法分析带来的影响；比如一些仓库中常常将一些基本数据类型定义为其他符合使用情境的名称，宏定义函数也易于与真正的函数调用混淆等等。

> 这一点不能够完全保证，因为从词法和语法上来说很多特征的识别都能够受到宏展开的影响。但要注意，分析过程中目标仓库往往代码比较规范，一些极端特殊、人们一般不会使用的情形并不会涉及。所以，只要对测试仓库在分析结束后将分析过程记录的具体信息条目索引到源代码进行比对并确认是否识别错误，如果识别不准确在根据错误原因对特征的选取与识别逻辑进行调整，直到在抽取的数据中鲜少出现识别错误的情况，就可以认为特性的识别是相对准确的。

C/C++ Features

特征表格如下：

Category	Language Features	iIllustration/Utility	Reference denominator可参考分母
Template & Modular Programming	Standard Template Library (STL)	容器（Containers）:stack、vector、set、map、priority_queue	文件数
	template 范式	泛型编程：创建适用不同数据类型的泛型函数和类	函数定义数
	lambda表达式	构造闭包：能够捕获作用域中的变量的无名函数对象	代码行数取对数
	using namespace	命名空间控制	文件数
	Macro宏中的##拼接	避免命名冲突，更灵活地生成代码，但容易出bug	宏定义数
Concurrency &Multithreading	thread_local	变量声明为线程存储期	变量定义数取对数
	votatile	多任务共享变量、多线程并发访问变量修饰	变量定义数取对数
Memory Management	析构函数~function	当一个对象被删除或离开其作用域时，会自动调用其析构函数进行资源回收	类定义数
	Smart Pointers	shared_ptr/unique_ptr/weak_ptr (通过指针释放对象可能造成内存泄漏)	变量定义数取对数
	直接初始化:T 对象（实参）	从明确的构造函数实参的集合初始化对象，无需右值复制	变量定义数取对数
Exception Handling	Try-Catch Blocks	catch 语句捕捉 try 块中抛出的特性异常 (std::exception)	代码行数取对数

Category	Language Features	iIllustration/Utility	Reference denominator可参考分母
Polymorphism & Overloading	noexcept 说明符	noexcept 是一个函数后缀，指示该函数是否可能抛出异常，减少不必要的异常处理开销	函数定义数
	Nested Class	支持嵌套类、不支持嵌套函数	类定义数
	Operator Overloading	运算符重载	函数定义数
	virtual&override&final	override 指定一个虚函数覆盖另一个虚函数 final指定某个虚函数不能在派生类中被覆盖，或者某个类不能被派生	函数定义数
	Polymorphic Type Conversion	dynamic_cast/static_cast/const_cast	变量定义数取对数
	function overloading	同名不同参数个数或类型的函数声明	函数定义数
Reference Control	using	跨类或跨空间引用成员对象	变量定义数取对数
	this	this指针指向隐式对象形参（在其上调用非静态成员函数的对象）的地址	类定义数
	friend友元声明	允许类的非公有成员被一个类或者函数访问	函数定义数
Function	可变参数	int main (int argc, char *argv[ ]) { 函数体 } 变长实参（...）	函数定义数
Type System	typedef	允许用户自定义类型别名，增强代码可读性	文件数
	union	共用体类型提高内存利用率	文件数
	decltype	泛型编程：创建模板时易于获得表达式类型	函数定义数
	using	创建类型别名，增加代码可读性	变量定义数取对数
	votatile	提醒编译器该变量易变，无需优化，系统总是重新从它所在的内存读取数据	变量定义数取对数
	constexpr	指定变量或函数的值可以在常量表达式中出现，减少重复计算	变量定义数取对数

Data set

我们希望能够综合考虑更多的应用场景，因而计划将收集到的仓库列表分成了 14 个大类，如下：

1. 操作系统
2. 软件工程
3. AI与机器学习
4. 加密系统
5. 游戏开发
6. 金融领域
7. 嵌入式系统
8. 网络通信

9. 量子计算
10. 编译系统
11. 自动化
12. 编码技术
13. 图像处理与计算机视觉
14. 开发者工具与插件

具体的项目条目将在实现环节逐步加入。

CCScaner

在分析目标、分析方式、分析工具悉数确定之后，就可以着手构建实现目标的代码框架。

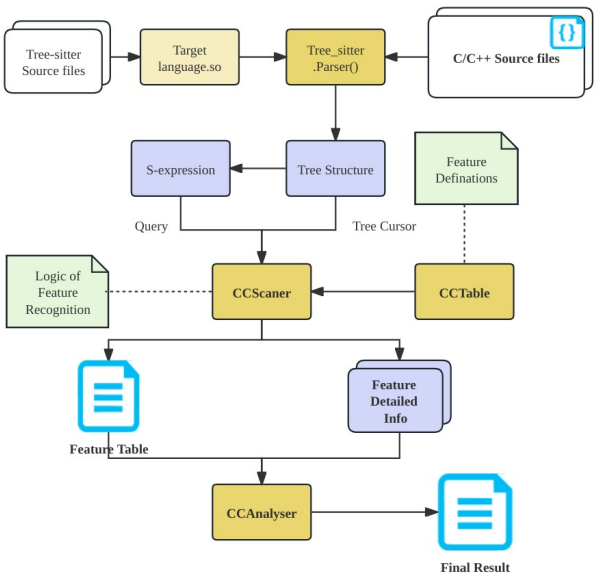
Expected functionality

期望达成的主要功能：

1. 树分析：对目标文件生成分析树并针对目标特征表项进行树访问提取出目标特征。
2. 数据获取与存储：记录各个文件中各个目标特征出现的次数并累加，并记录其所在仓库名、文件路径、行列位置与局部字节串信息以便于索引。具体存储格式在`doc/storage-format.md`中有具体说明。

Project structure

设计思路如下：



Preliminary preparation

1. 前期选定目标语言，利用`Tree-sitter`中对应语言部分的源码编译出动态链接库，作为`Parser`的输入。
2. 利用动态链接库为`Tree-sitter`库中的`Parser`设定语言类型之后，利用`Parser`分析仓库中的 C/C++ 源代码文件得到树形结构，获得根节点。
3. 对树中的每一个结点，可以获取到其代表的词法单元对应的行列坐标范围的字节串，同时还可以这一部分字串的词法结构，其存储结构是S-表达式，这是一种形式表达半结构化数据的约定，在 Lisp 语言中广为应用。其结构大致如下：

   ```
   (translation_unit (preproc_include path: (system_lib_string)) (function_definition type:
   (primitive_type) declarator: (function_declarator declarator: (identifier) parameters:
   (parameter_list)) body: (compound_statement (declaration type: (primitive_type) declarator:
   (init_declarator declarator: (identifier) value: (number_literal))) (return_statement
   (number_literal)))))
   ```

   利用这一表示形式，可以快速通过`query`对目标结构进行查找，这在下面会有介绍。

Mid-term data processing

1. 基于我们已经划分出来的特征表条目确定项目核心类`CCTable`的内容，其中包括：
   1. 各个表项的定义
   2. 统计基数的定义与处理
   3. 存储文件的维护与更新

   同时注意要限制每一个特征对应的局部`content`即字节序列的长度，对较大的字节序列进行裁剪，以防止空间效能过差。
2. 得到树结构与结点对应的 `S-expression` 之后，可以使用核心分析脚本`CCScaner`进行树结构分析，寻找目标特性，可以利用的方式有`query`查询与树访问，分别基于`S-expression`与树结构。这一过程受到我们对每个特征项定义的特征识别逻辑的指导。

   分析结果为`Feature Table`与`Feature Detailed Info`，前者记录特征的使用次数，以供统计分析；后者是特征每一次使用的相关信息，以供深入分析。

Post analysis

1. 基于 CCScaner 对仓库中大量文件的分析结果进行分析，构想是采用一个 CCAnalyser 完成：
   1. 汇总分析
   2. 对可能跨文件的特征进行统计

   并预留了接口。

   但实际上对于第二点，由于目标特征中并不存在这一情形，加上时间比较紧迫，所以并没有加入仓库。对于第一点，由于仓库数量并不多，使用表格统计的方式更加灵活高效，所以也没有进行代码实现。

   因而尽管设计时这一文件预留了接口，但实际没有启用，项目仍然以 CCScaner 为核心。

2. 统计出 Final Result ，以供最终的汇总分析。

Analysis methods

这里介绍特征识别的两种主要方式。

Query

使用`query`对结点的`S-expression`进行查询。

基本模式如下：

```
    def find_MEMORY_directinit(self):
        query = CPP_LANGUAGE.query(
            """(declaration
                (type_qualifier)*
                type: (_)
                declarator: [
                    (init_declarator
                    declarator: (identifier) @identifier_type1
                    value: [(argument_list (_)+)
                            (initializer_list (_)+)])
                    (function_declarator
                    declarator: (identifier) @identifier_type2
                    parameters: (parameter_list (_)+))])
                (new_expression
                    type: (_)
                    arguments: (argument_list (_)+)) @new_expression
                (lambda_expression
                    captures: (lambda_capture_specifier (_)+)) @lambda_expression
                ......(其它的query项省略)""")
        captures = query.captures(self.root)
        for capture in captures:
            # '=' can't show up between identifier and initializer_list
            if capture[1] == "identifier_type1":
                # ......(省略)
            # 记录所需要的具体信息:
            location = str(tuple(x + 1 for x in capture[0].start_point)) + \
                       '-' + str(tuple(x + 1 for x in capture[0].end_point))
            content = capture[0].parent.text
            # ......(省略)
            self.cctable.table["MEMORY"]["directinit"].append((location, content)) # 加入特征表条目
```

匹配过程中允许一些类似于正则式的表达方式。

上述的代码可以匹配类似于`T object ( arg1, arg2, ... );`，`T object { arg1, arg2, ... };`，`new T(args, ...)`，`[arg](){...}`的形式的子 S-表达式（其它一些这里省略了），对应于 C++ 手册中直接初始化的`(1)`，`(2)`，`(5)`，`(7)`情形。

由此可以得知 `query`的表达力。

Tree cursor

使用 Tree cursor 对树结构进行快速遍历。

```
    def find_POLYMORPHISM_nestedclass(self):
        # [DFS] traverse the tree to find class_specifier (outmost layer)
        # when find a class_specifier, use query to find the inner class_specifiers
        # and don't visit its subnodes
        self.cursor.reset(self.root)
        # .......(省略)
        while True:
            if flag_subtree_visited:
                flag_subtree_visited = False
                if not self.cursor.goto_next_sibling():
                    # .......(省略)
                continue
            elif self.cursor.node.type == "class_specifier":
                # 遇到一个类结点后用 query 访问其文本内容，不再继续向下遍历
                query = CPP_LANGUAGE.query("""(class_specifier) @class_specifier""")
                for child in self.cursor.node.children:
                    captures = query.captures(child)
                    for capture in captures:
                        content = capture[0].text
                        # .......(添加特征表项，过程省略)
                self.cursor.goto_parent()
                flag_subtree_visited = True
            elif self.cursor.goto_first_child():
                continue
            else:
                if not self.cursor.goto_next_sibling():
                    if not self.cursor.goto_parent():
                    # .......(省略)
                continue
        return
```

这里是一个深度优先遍历的情形，遇到类结点之后对其标识的 S-表达式范围进行 query 查询，统计嵌套类个数。

上述两个例子能够展现使用 Tree-sitter 所能够进行的基本操作。

Technical route

1. 明确目标语言：C/C++
2. 讨论语言特性，选择关注点并进行汇总：

   目标特征类： Template & Modular Programming 、Concurrency & Multithreading、Memory Management 、 Exception Handling 、 Polymorphism & Overloading 、 Reference Control 、 Function、Type System 。
3. 建立`CCScaner`的项目框架，选择树分析工具，完成外层数据处理脚本

   树分析工具确定为 Tree-sitter 。

项目结构的介绍前后均有提及。

4. 确定数据处理与存储流程，完成对特性表类`cctable`的搭建

   其中存储的格式见 `doc/storage-format.md`。

5. 基于特性汇总表完成对`demo/`目录的填充，将特征的定义细化到语言实例

   其目的是便于确定目标特性的出现模式，从而确定代码逻辑；同时也用于代码测试。

6. 完成`CCScaner`项目核心类`ccscaner`，确定对每一个特性项的处理逻辑

7. 分别对单文件和单目录进行调试测试，据此完善项目的每一个组件

8. 对测试仓库进行处理，得到相应数据，从而对特性表、特性处理方法进行反复的再调整

   对于一些明显不符合实际认知的数值通过查看过程中记录的具体信息找到误识别的位置，修改代码逻辑以规避这种情形。

9. 利用已完善的工具`CCScaner`对大量仓库进行分析汇总

10. 对数据进行处理与人工分析，得到初步结论

Division of labor & Contribution ratio

Student ID	Name	Github id	Work content	Ratio
PB21061327	王志成	Starrybook	项目框架搭建与部分识别函数	33.3%
PB21111656	余淼	Ymm-cll	部分识别函数与可视化表格	33.3%
PB21111682	龚劲铭	Gjmustc	C++特性及仓库分类统计分析，工具使用分析C++代码举例demo编写	33.3%

Implementation

Inner structure of CCScaner

这里给出了一个比设计部分更为具体的描述：

```
~/Documents/ccscaner$ tree
.
├── README.md                          : 仓库介绍
├── 05-language-features-usage-in-Cpp.md    : 小组工作介绍
├── build                              : 从 tree-sitter 源码编译出针对目标语言的动态链接库
│   └── my-languages.so                : 可为任意语言，本项目中为 c/c++
├── doc                                : 文档
│   ├── binding.pyi                    : tree-sitter 的 python 绑定
│   ├── Cppfeatures.md                 : c/c++ 目标特性表，即本项目分析目标
│   ├── grammar.js                     :
│   ├── Report.md                      : 报告【当前文件】
│   ├── scripts-examples.md            : 执行脚本的使用示例与相关说明
│   ├── scripts.md                     : 脚本的接口说明
│   ├── sources.md                     : tree-sitter 官方文档等参考信息
│   └── storage-format.md              : 数据存储格式的统一定义
├── demo                               : 测试文件，实质上是对抽象特征表条目的具象化描述
│   ├── FEATURE-CLASS                  : 特征大类
│   │   ├── ......                     : 大类下的具体条目
│   │   └── feature-item.cpp
│   └── ......                         : 其他特征大类
├── data                               : 最终分析结果原始数据的存放位置
│   ├── feature-tables                 : 目标仓库的特征统计表
│   │   ├── ......                     : 以仓库名标识的特征统计表
│   │   └── repo-name.csv
│   ├── ......                         : 每个仓库的具体信息条目做成压缩包的形式存放于此
│   └── repo-name-res.zip
├── res                                : 程序运行时的分析结果存放处，是运行时维护的共享区域
│   ├── feature-tables                 : 目标目录/文件的特征表存放处
│   │   ├── ......                     : 以目录/文件名称命名的特征统计表文件
│   │   └── repo-name.csv
│   └── repos-info                     : 目录/文件具体信息条目存放处
│       ├── ......                     : 按照 特征大类-特征具体条目 标识的 csv 文件
│       └── FEATURECLASS_featureItem.csv
├── ccanalyser.py                      : 基于 ccscaner 的结果进行综合分析，设计时预留了接口，但最后并未启用
├── ccscaner.py                        : 项目核心，包含对每一个特征条目的处理逻辑，依赖于 cc
├── cctable.py                         : 项目核心，定义了特征表的数据结构，并承担存储部分的工作
├── ccprinter.py                       : 实现分析树可视化，辅助调试
├── test-file.sh                       : 对单个指定的文件进行分析，多用于调试，也作为 test-dir 的子过程
├── test-dir.sh                        : 对目录下的所有符合后缀要求的文件进行分析，依赖于 test-file
└── clean-all.sh                       : 清除 ./res/ 下的分析产出
```

How to analyze a repository?

仓库核心脚本文件`ccscaner.py`、`test-file.sh`、`test-dir.sh`在同目录下的[doc/scripts.md](doc/scripts.md)文件中有接口说明，并在同目录下的[doc/scripts-examples.md](doc/scripts-examples.md)文件中有具体的使用说明与使用示例。处理过程则在前面有所介绍。

两个文件的描述都相对详尽，故而这里就不再赘述。

Data & Analysis

Raw data

Repositories selected

在设计部分提到过目标仓库大类的划分，这里是具体条目的填充：

领域	项目名称	项目简介
操作系统	torvalds/linux	Linux 内核的源代码
	reactos/reactos	专注于兼容自Windows NT到XP的开源操作系统
软件工程	facebook/react-native	Facebook 的开源框架，用以构建移动应用
	cinder/Cinder	一套开源的、专为创意编程而设计的C++库
AI与机器学习	opencv/opencv	开源计算机视觉库，其中包含了多种机器视觉算法
	tensorflow/tensorflow	Google 的深度学习框架
	apache/mxnet	深度学习框架，它让数据科学家可以构建深度学习模型
加密系统	openssl/openssl	包含了加密算法和用于创建SSL和TLS协议的工具库
数据库	rocksdb	C++构建的，针对快速存储场景的嵌入式数据库
	sqlitebrowser	浏览和操作SQLite数据库的工具
	typesense	快速、打字错误兼容的、开源搜索引擎
	arrow	提供了一套用于处理大规模数据的开源工具
	duckdb	是一个为分析查询优化的内存原生数据库
	mysql	关系型数据库管理系统
游戏开发	cocos2d-x	一个用于构建2D游戏的开源框架
	CnC_Remastered_Collection	《命令与征服》的开源游戏代码
	Hazel	易用的游戏制作引擎
	godotengine/godot	全功能的开源游戏开发引擎，支持2D和3D游戏的开发
金融领域	lballabio/QuantLib	用于金融工程的自由/开源库
	bitcoin/bitcoin	Bitcoin核心维护团队官方维护的比特币项目源码
嵌入式系统	rogerclarkmelbourne/Arduino_STM32	用于在STM32基板上运行的Arduino适配
	ETLCPP/etl	一种嵌入式模板库
网络通信	chromium/chromium	Chrome浏览器的开源项目
	mozilla/gecko-dev	Firefox的开源项目

领域	项目名称	项目简介
	webkit/webkit	开源的浏览器引擎，支持HTML和JavaScript，广泛应用于Apple的Safari浏览器以及很多其他的开源浏览器
量子计算	eclipse/xacc	开源的，为量子加速器设计的编程框架
编译系统	llvm/llvm-project	LLVM项目，包含了一套编译工具链，用于从多种语言生成机器代码
	gcc-mirror/gcc	免费的编程语言编译器，支持如C、C++、Java、Fortran等多种编程语言，并可跨平台使用。
	Kitware/CMake	跨平台的自动化建构系统
自动化	FreeCAD/FreeCAD	开源的3D 建模软件
	ros-planning/moveit	用于机器人移动规划的工具集
编码技术	PointCloudLibrary/pcl	用于2D/3D 图像和点云处理的开源库
	Kitware/VTK	用于处理和显示科学数据的开源软件系统
图像处理与计算机视觉	opencv	跨平台的计算机视觉库
开发者工具与插件	swift	Apple 的开源编程语言
	json	轻量级的数据格式
	grpc	Google 的开源远程过程调用系统
	flatbuffers	Google 的开源内存高效的序列化库
	spdlog	极其快速、灵活且方便的C++日志库
	foundationdb	分布式数据库，设计用于大规模的数据存储
	incubator-weex	提供一个框架，使开发人员能够使用现代的Web开发经验来构建安卓、iOS以及web的高性能应用程序

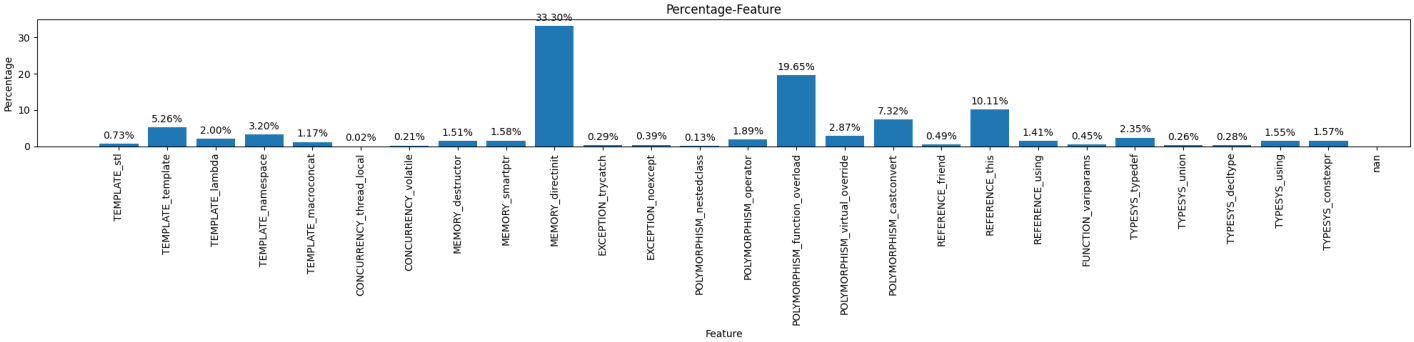
Raw data

每个数据表最终的特征统计表的存储位置位于 `data/feature-tables/`目录下。

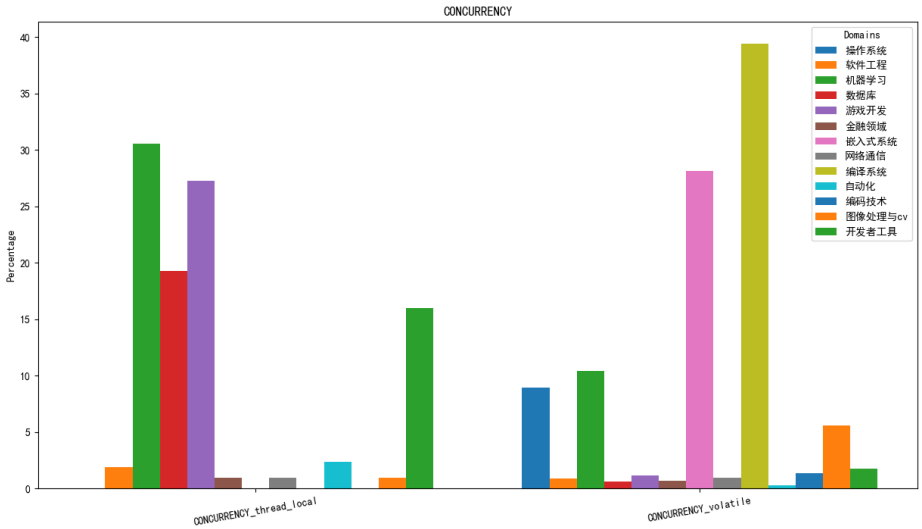
最终的汇总统计表为data/statistic.xlsx。

由于内容过于庞杂，这里难以一一列举。请查看对应链接。

Analysis

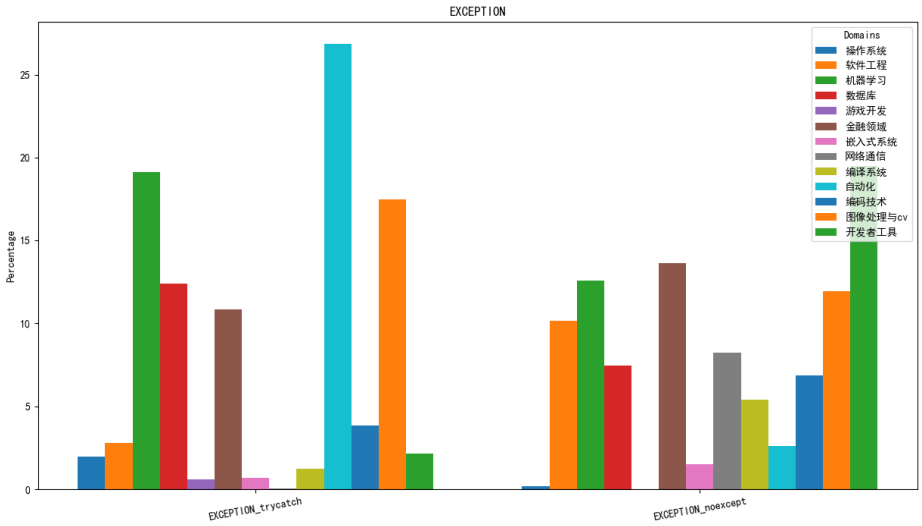


- 内存优化中的直接初始化和多态中的函数重载是最普遍被使用的C++高级特性，说明在实际编程中语言使用者更喜欢语法简单、内存开销小、且在运行时可复用性高的编程特性。
- 除直接初始化外，TEMPLATE模版编程类和POLYMORPHISM多态类占据了特征计数的主要比例，说明大多数C++ developers能抓住并使用C++最突出的核心特性。
- COCURRENCY并发类和EXCEPTION异常处理类占比较小，说明在考察的仓库语料中，对C++并行加速、存储生存期等这种与底层硬件控制相关的特性使用较少，体现了高级语言的封装性；同时大部分的单文件中对异常情况的处理较少较简单，程序的健壮性有待提高。
- POLYMORPHISM_nestedclass 、 REFERENCE_friend 、 REFERENCE_using 、 FUNCTION_variparams这几种C++支持的特殊用法在实际编程中使用率较小，第一个增加了类的复杂性，可能会造成作用域混淆，第二和第三个则破坏了类或函数的封装性，最后一个增加了函数参数的不确定性，这些都容易在实际运行中提高程序出错风险，语言设计者可以根据特性使用情况适当修改完善相关特性。

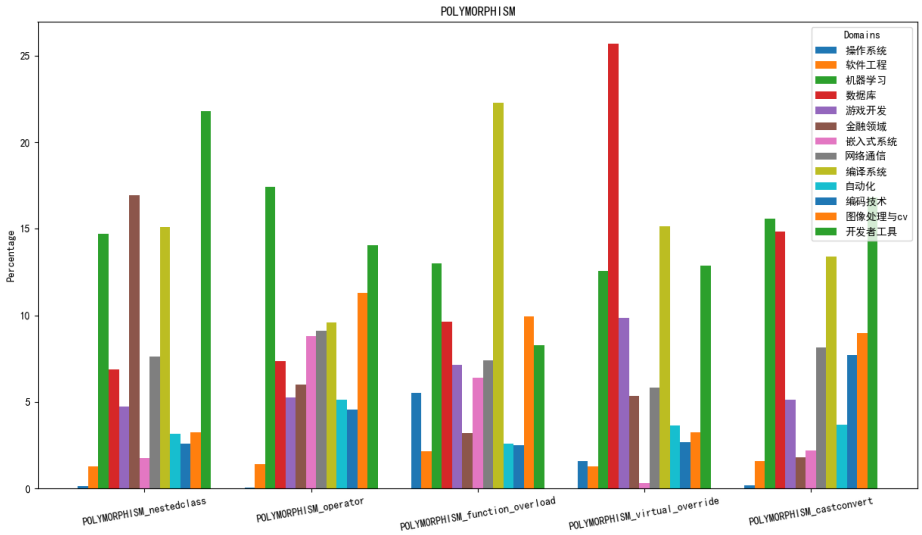


- thread_local:与线程加速有关的特性在机器学习、数据库、游戏开发这些对运行性能或实时性要求极高的领域。
- volatile：与数据存储生存期控制有关的特性在编译系统和嵌入式系统这些与底层硬件设计息息相关的系统软件领域。

- 在主体仓库多基于 C++ 而非 C 的领域中，软件工程、机器学习、游戏开发、数据库等比较注重运行时性能的领域对线程控制相关的 thread_local 关键字的使用显著多于其它领域，而在主体仓库多基于 C 而非 C++ 的领域中，操作系统、嵌入式系统、编译系统等比较注重运行时性能的领域对并发访问相关的 volatile 关键字的使用显著多于其它领域。综合来看，这说明这些领域相对于金融、网络通信、自动化、图像处理等领域更注重利用并行的方式提高效能。



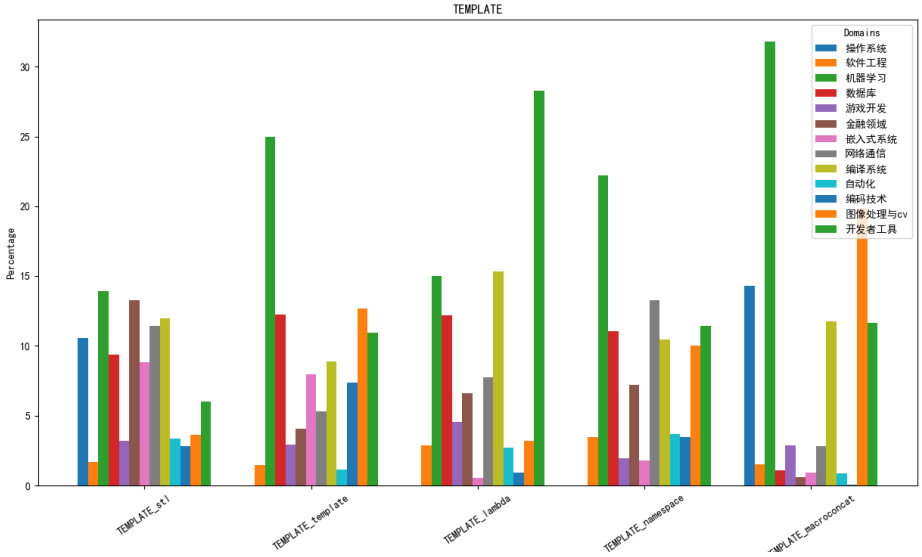
- try-catch:在机器学习、自动化和计算机视觉领域使用try-catch异常控制较多，这可能是由于在这些领域的出错代价较高，设计者一般会力求降低产品在实际运营中的出错率。
- noexcept：显式说明noexcept在开发者工具项目中使用明显高于其他领域，表明该领域设计者一般能巧妙利用该特性让编译器避免某些优化，这些优化包括省略异常处理代码等，从而减少栈展开操作，提高这些辅助开发工具的运行效率。



- operator 运算符重载在机器学习和开发中工具中使用较多，通过重载运算符，开发人员可以编写更具表达力和简洁性的代码，从而更容易处理复杂的数据结构和算法。这可以在机器学习项目中产生更具可读性和可维护性的代码。在开发工具中，运算符重载可以用来定义运算符在特定库或框架的上下文中的行为。这允许在处理某些类型的数据或对象时更直观、更自然地使用运算符。

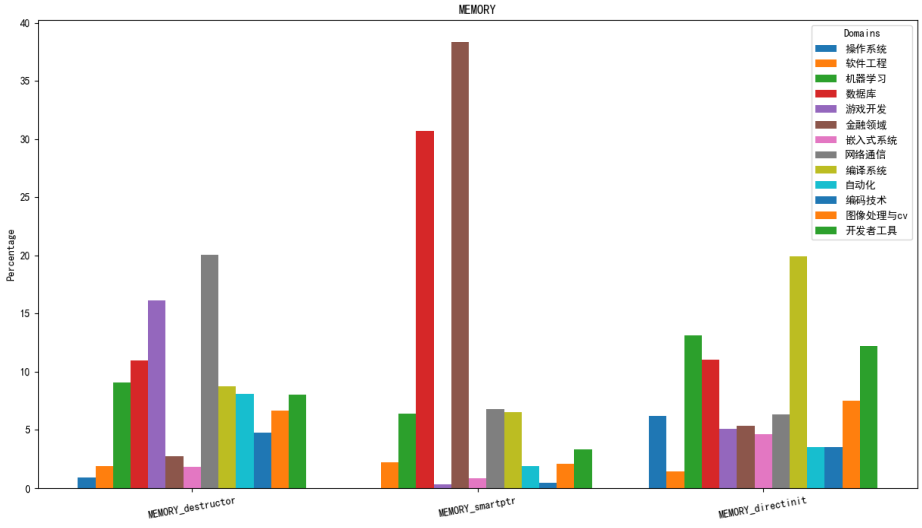
（chatgpt）

- 函数重载在编译系统中使用较多，
- 虚函数重写在数据库和编译系统中使用较多

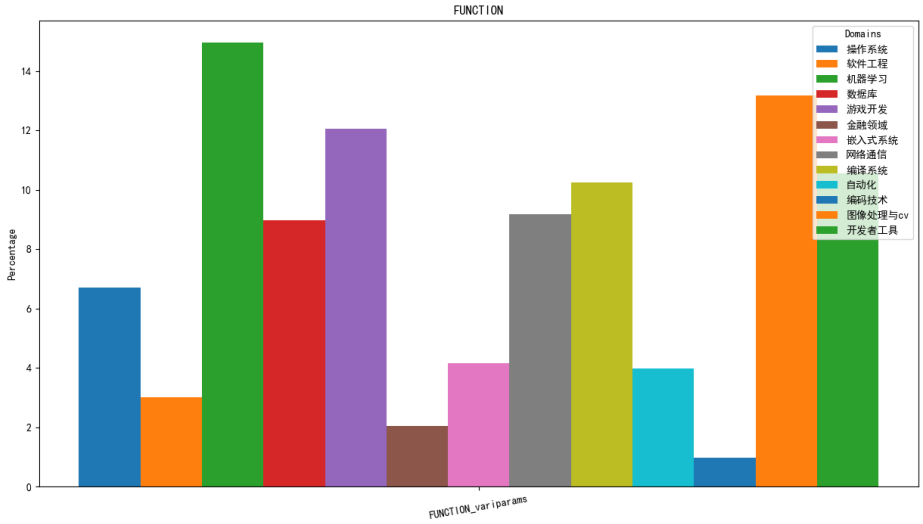


- 模板化编程特性在机器学习领域出现频率明显高于其他，符合我们的常规印象，机器学习和深度学习的模型编程都是具有高度模版性和封装性的。

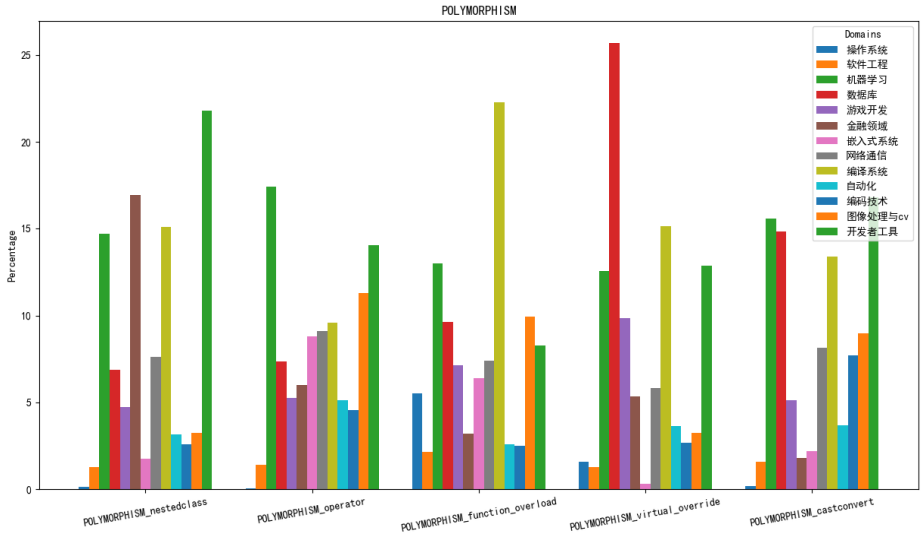
一些使用习惯（惯例）：

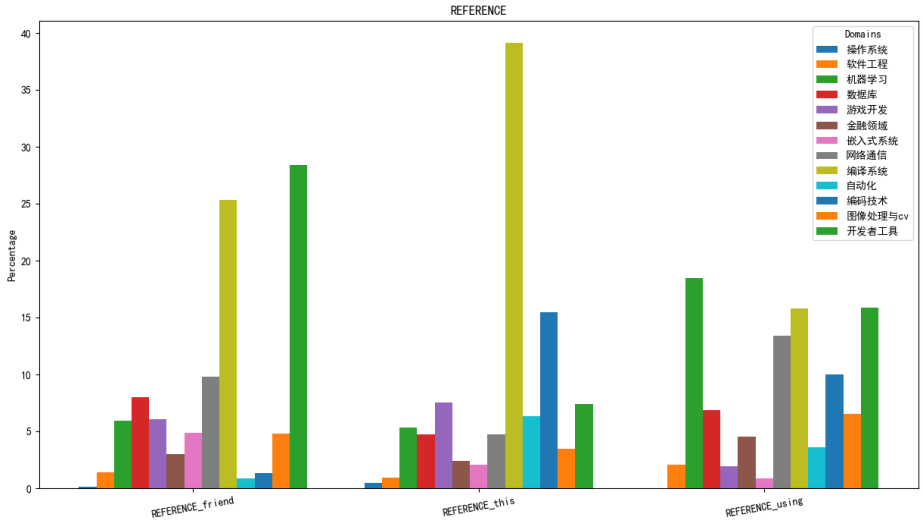


- smartptr：智能指针在数据库和金融领域使用较多。

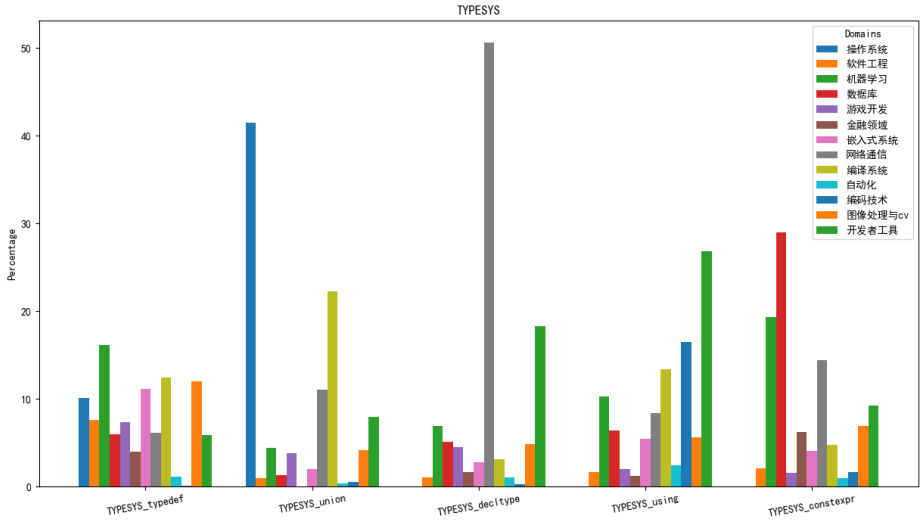


- variparams: 可变长省略参数在机器学习和软件工程领域使用较多。

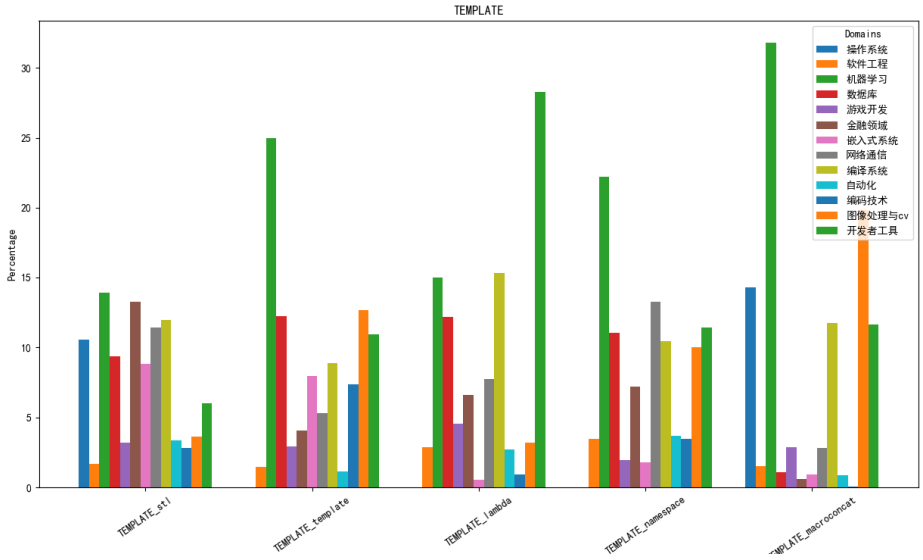




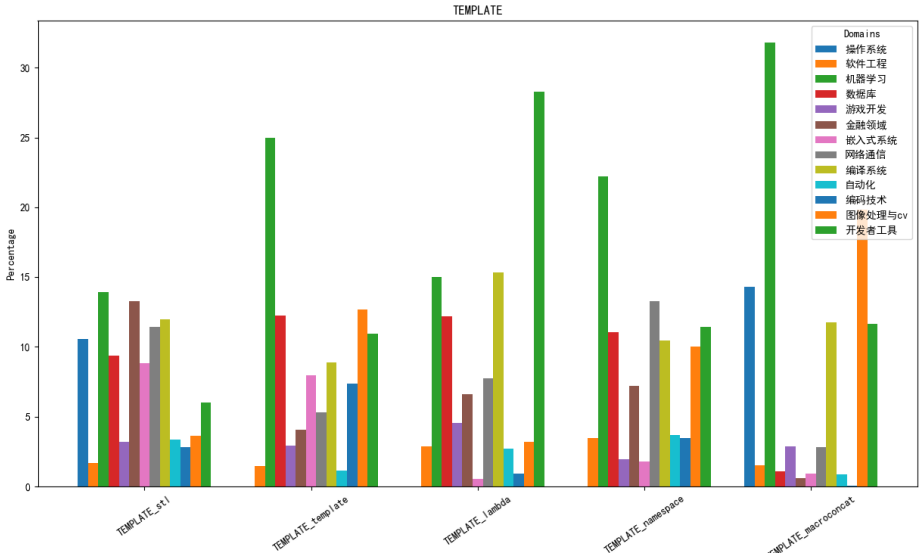
- this: 在编译系统中this指针的使用率远高于其他领域，这也非常符合我们本学期的编程体验。



- decltype:在网络通信工程中使用较多。
- union：在操作系统中使用较多（主要来源于C语言特性）。



- lambda表达式在开发者工具中使用较多。


TEMPLATE
Percentage
0
5
10
15
20
25
30
Domains
操作系统
软件工程
机器学习
数据库
游戏开发
金融领域
嵌入式系统
网络通信
编译系统
自动化
编码技术
图像处理与cv
开发者工具
TEMPLATE_stl
TEMPLATE_template
TEMPLATE_lambda
TEMPLATE_namespace

Conclusion

A summary of what we have done

本项目中我们实现了如下目标:

1. 成功构建了一个对 C/C++ 语言具有大规模分析能力的语言特性分析工具：CCScaner
2. 利用 CCScaner 对分属 14 个大类的超过三十个中大型规模的仓库、接近六万个文件、两千五百万行代码进行了统计分析并得到原始数据
3. 基于原始数据进行了大量的分析，并得到了一系列结论

整理来看，整个过程符合我们的预期，得到了很多有意义的结果，也使得我们对于语言特性的设计、影响、使用模式有了更加直接深入的了解。

Deficiencies and prospects

当然，本项目中存在着许多值得深入考量的地方，这里给出我们的一些想法:

1. 在提取特性时引入`pygccxml`/`libclang`进行辅助性的语法分析得到 AST，从而能够对特性实现更加准确的捕捉
2. 借助预编译过程解决宏展开对特性识别的干扰问题
3. CCScaner 可以通过并行化的方式加速以解决引入语法分析与预编译过程后分析速度慢的问题
4. 对项目进行迭代，根据数据分析的结果对原有的特性项与识别逻辑进行反思与改进（探索时拟定的特征项可能不够精炼），从而得到更具代表性的特性与更有效的识别逻辑
5. 可以对 C 与 C++ 进行更细致的区分
6. 基于项目分析得到的数据的讨论可以更加深入，并与一些其它语言的相关研究进行对照
7. 本项目也可以横向迁移，对于其他语言进行探索性的特征统计

上述的想法或由于时间的限制、工具本身的限制，或由于最初探索时对处理过程的不了解而没能够达成，是这个项目的一些遗憾，不过同时也是值得展望的地方，希望我们一个月来的探索与尝试能够为相关的研究提供一些有意义的经验。

Reference

An Empirical Study for Common Language Features Used in Python Projects

An Empirical Study of Type-Related Defects in Python Projects

An Empirical Study of C++ Programs

An Empirical Study of Function Overloading in C++

Github repository: PyScan

Tree-sitter official documentation