开发日志

2025-12-20：技术选型与环境搭建
1. 阶段目标
2. 分析与决策
2025-12-21：数据读取和分词处理功能实现
1. 阶段目标
2. 分析决策
3. 测试方案
4. 结果验证
2025-12-22：核心数据结构设计：滑动窗口与热词统计
1. 阶段目标
2. 分析决策
3. 问题与解决
2025-12-23：顶层类的组装、可视化处理和处理数据方式的选择
1. 阶段目标
2. 分析决策
3. 功能实现
2025-12-24：性能测试及报告书写和架构图、类图的绘制
1. 阶段目标
2. 分析决策
3. 问题及功能实现
2025-12-25：打包跨平台测试和系统设计文档的编写
1. 阶段目标
2. 分析决策
2025-12-26：README说明的编写及作业提交
1. 阶段目标
2. 分析决策

2025–12–20：技术选型与环境搭建

1. 阶段目标

- 确定开发语言与基础架构，搭建 WSL 开发环境。

2. 分析与决策

- **语言选择**：题目对性能有要求， C++ 更有优势，而且有较为熟悉的STL可供使用。但考虑到涉及外部库的导入（分词等）和数据可视化的要求，python方便导入库以及用外部库来实现可视化效果，而且若后期要将其升级为 web 应用，本人掌握的flask框架也是基于python的，于是选择了python作为本次作业的语言。
- **环境选择：**本人习惯WSL下的开发，上手速度快，只需要保证后期在windows环境下测试通过即可，同时要注意中文编码在两种环境下的差异，要保证显示地使用UTF-8格式读取和保存中文结果
- **可视化方案**：由于时间不充足，难以同时完成前后端的工作，故笔者借助AI找到了streamlit这个简单构建web可视化的外部库并使用其来进行可视化
- **决策**：最终选择 **WSL + uv + Python+streamlit**。
- **补救措施**：为了弥补 Python 解释器的性能劣势，决定在数据结构设计上严格控制复杂度

使用uv 进行依赖管理，但在提交时导出为标准 `requirements.txt` 以兼容验收环境。

2025-12-21：数据读取和分词处理功能实现

1. 阶段目标

- 实现数据读取和分词处理功能，并对其进行测试

2. 分析决策

- **字符串处理** ： 由于 python 中字符串无法像 C++ 一样当作字符数组处理，故采用正则表达式来提取输入信息中的时间戳、文本、topK 的值等信息；而且由于正则表达式采用匹配的方式提取信息，遇到数据格式不正确的内容会直接跳过
- **内存优化**：同时考虑到 python 作为解释型语言，运行时内存相较 C++ 会更大，在读取外部文件的时候，我选择了写一个生成器，每次读一行并将读取内容yield到外部，以此保证每次处理只有一条文本的内存占用

3. 测试方案

- 实现读取文件和分词处理的两个类后，在 `main.py` 中实例化对象，并在 `data/input.txt` 中写入测试用例

```
[0:04:26] 诸葛家当时也是三家下注，诸葛家龙虎狗
[0:04:27] 你们三兄弟长的一点都不一样
[ACTION] QUERY K=6
[0:04:27] 诸葛瑾确实是神级管家！其他能力就一般了
[0:04:27] 那三弟呢，在哪儿
[0:04:28] 刘备这都不敢问出口了，声音听着那么打颤
[0:04:28] 孔明乃。。。中少爷。。
```

4. 结果验证

```
Prefix dict has been built successfully.
诸葛
家
当时
三家
下注
诸葛
家
龙虎
狗
三
兄弟
长
一点
诸葛瑾
确实
神级
管家
能力
那三弟
刘备
不敢
问
出口
声音
听
打颤
孔明乃
中
少爷
```

2025-12-22：核心数据结构设计：滑动窗口与热词统计

1. 阶段目标

- 实现滑动窗口与热词统计，并对其进行测试

2. 分析决策

- **滑动窗口设计**：要在源源不断地数据流到来时只保留规定窗口大小的信息，很容易想到使用双端队列来实现， python 中可以用 `collections` 库中的 `deque` 模块，每次添加词汇前将过期词汇剔除即可
- **热词统计设计**：热词本质是个字符串，统计其次数即为统计不同字符串的出现次数，同样能够想到使用哈希表来计数，在 python 中可以使用 `collections` 库的 `defaultdict` 模块，其功能与普通的字典类似，不过访问不存在的键时不会报错
- **topK查找**：需要求出一堆数据中前K个数据，且排序的依据有多个时，在 C++ 中容易想到用快速排序或者优先队列实现；而在数据不断更新时，使用堆变得更加合理。在python中可以使用 `heapq` 模拟小根堆，将统计次数取负值，当次数相同时按照热词的字典序升序排序，具体实现采用了 `heapq` 的 `nsmallest` 方法
- **单元测试**：用 pytest 对**删除过期词汇、热词计数、topK排序**等功能进行了单元测试，通过测试发现了一些问题并解决

3. 问题与解决

- 问题：排序的结果与实际结果相反
- 解决方案：发现使用了 `heapq` 的 `nlargest` 方法，以大根堆的方式来排序导致的结果相反问题，改用 `nsmallest` 方法即解决

> 问题如下：

```
tests/test_timewindow.py:36: KeyError
_____________________________________________ test_top_k_sorting _____________________________________________

window = <src.timewindow.TimeWindow object at 0x76e775923350>

    def test_top_k_sorting(window):
        """测试 Top-K 排序是否正确"""

        window.add_word(1, "A")
        window.add_word(1, "B")
        window.add_word(2, "A")
        window.add_word(2, "B")
        window.add_word(2, "C")
        window.add_word(3, "C")

        # A=2, B=3, C=2
        # 按照出现次数降序排序，若出现次数相同则按字典序升序排序
        top_3 = window.get_top_k(3)

>       assert top_3[0][0] == "B"
E       AssertionError: assert 'C' == 'B'
E
E         - B
E         + C

tests/test_timewindow.py:61: AssertionError
========================================== short test summary info ==========================================
FAILED tests/test_processor.py::test_process - AssertionError: assert '结果' in ['测试', '一行', '文本', '分词']
FAILED tests/test_timewindow.py::test_expired_words - KeyError: 'banana'
FAILED tests/test_timewindow.py::test_top_k_sorting - AssertionError: assert 'C' == 'B'
```

> 解决后的运行成果：

```
(hotwordsanalysissystem) lyclyc@DESKTOP-EFS6UF4:~/workspace/HotWordsAnalysisSystem$ pytest
========================================== test session starts ==========================================
platform linux -- Python 3.11.13, pytest-9.0.2, pluggy-1.6.0
rootdir: /home/lyclyc/workspace/HotWordsAnalysisSystem
configfile: pyproject.toml
collected 3 items

tests/test_processor.py .                                                                          [ 33%]
tests/test_timewindow.py ..                                                                        [100%]

=========================================== 3 passed in 0.47s ===========================================
```

2025-12-23：顶层类的组装、可视化处理和处理数据方式的选择

1. 阶段目标

- 完成顶层类的组装，将结果保存下来并进行可视化处理，以及考虑输入输出形式

2. 分析决策

- 首先是可视化，由于时间和技术能力不足，我没有选择使用web前端的方式。通过询问AI找到了streamlit这个可以将数据用web方式显示而无需用到HTML、CSS和JS的python库；我设想将每次查询的结果保存下来，每次查询用一张柱形图表显示，借助AI完成了这项任务

- 然后是输入输出以及交互的方式，我计划了两种模式：auto自动模式和interactive交互模式。auto模式下将弹幕流数据和查询一同放在txt里，程序自动完成查询；interactive允许用户在导入弹幕流后，在终端窗口中手动查询，每次会在终端显示结果

3. 功能实现

- 由于streamlit需要单独一个命令运行，于是在main.py中使用了subprocess库来实现内部调用保证可以在终端一条命令完成热词统计以及可视化服务启动
- 对于命令行参数的读取使用了argparse库，可以自定义输入的模式以及窗口大小

2025-12-24：性能测试及报告书写和架构图、类图的绘制

1. 阶段目标

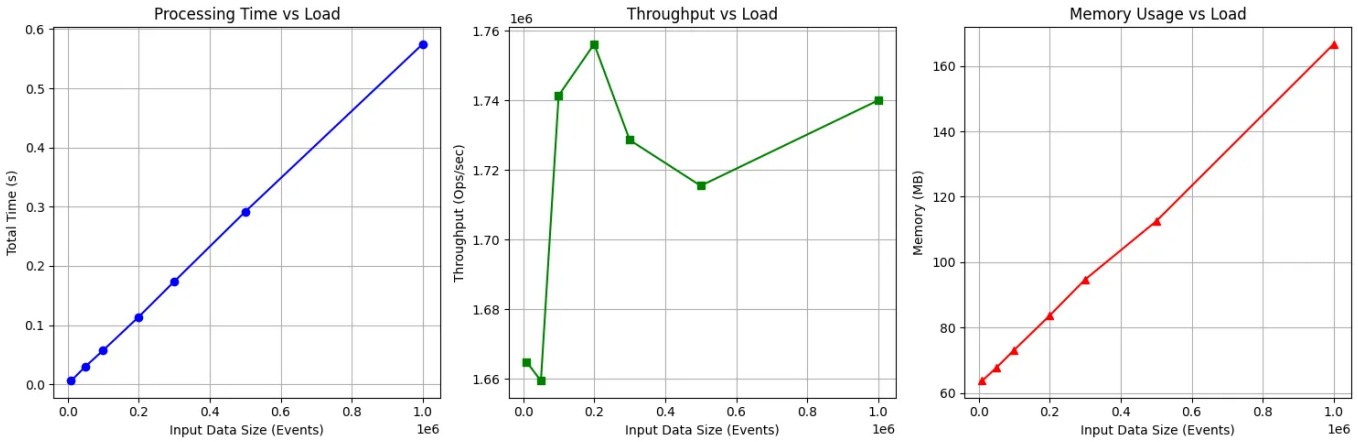
- 进行性能测试，写性能测试报告
- 绘制架构图和类图

2. 分析决策

- 计划使用脚本模拟生成从10,000到100,0000 行多个规模的文本数据，包含时间戳、随机组合的中文/英文词汇，以及周期性的 `[ACTION] QUERY` 指令。
- 考虑到交互模式的打印柱形图以及分词处理的时间占用较长，我分成了两种情况进行测试
  - **纯算法吞吐**：屏蔽 IO 和 Jieba 分词，直接向 `TimeWindow` 灌入数据。
  - **全链路吞吐**：包含文件读取、分词处理、窗口计算和结果输出。
- 对于架构图和类图的绘制，选择用mermaid语言，可以通过在线编辑器直接生成图表，很清晰

3. 问题及功能实现

- 在性能测试完后只有几条延迟、吞吐量的数据，并不直观，于是我使用了python中的matplotlib库来绘制出多个规模文本数据的图表
- 显示效果如下：



2025-12-25：打包跨平台测试和系统设计文档的编写

1. 阶段目标

- 将项目打包上传至云端仓库，并在windows环境下测试效果
- 编写系统设计文档

2. 分析决策

- 考虑到项目的持久化和公开性，我将项目上传至github
- 考虑在windows环境下，没有命令行基础的使用人员能够快速上手，这里我选择写了两个bat脚本，分别对应auto模式和interactive模式，用户只需要修改txt文件内容后执行bat文件就可以了

2025-12-26：README说明的编写及作业提交

1. 阶段目标

- 为代码添加部分注释
- 编写README.md说明文档

2. 分析决策

- 最后一个步骤，编写项目说明书，阐明项目内容结构和使用方法