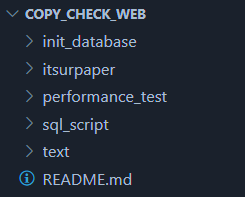
海量文本抄袭检测系统项目报告

# 系统架构与功能设计

本项目经过一次迭代，共两个版本，此报告仅针对于新版本的介绍。

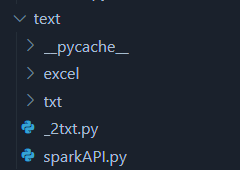
## 1.1系统架构

本项目文件层次结构如下图所示：



其中各部分分别作用于数据生成以及数据入库、Django整体框架、性能测试、数据库操作相关脚本、数据提取相关脚本。

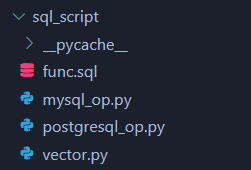
### 1.1.1数据提取



本部分中的\_2txt.py脚本负责从xlsx文件中提取内容，同时对其文本进行合理划分及处理；sparkAPI.py脚本则调用了科大讯飞的spark星火大模型用于去除废弃字符、格式标签等无关字符，然后对其处理得到文本的标题、来源、作者信息、正文等。

### 1.1.2数据库操作

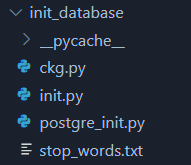
本项目经过一次迭代，共两个版本，其中旧版本使用的数据库为MySQL，新版本使用的为PostgreSQL。



本部分中的各脚本皆为对应数据库的相关操作脚本，相关操作主要包括插入、更新、查询以及执行SQL。

### 1.1.3数据生成以及数据入库

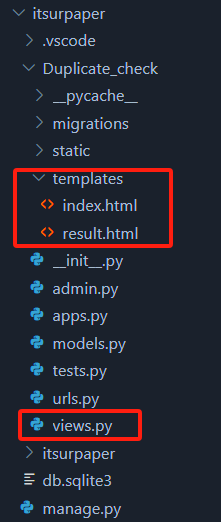
本部分为数据处理的核心部分，主要涉及将预处理完成的文本数据利用多线程等方式批量化地进行指纹生成以及入库。



其中，init.py与postgre\_init.py两个脚本分别对应两个版本的数据库需要的不同操作，ckg.py脚本文件中则包含了本项目最为核心的算法部分的各类实现。

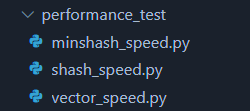
### 1.1.4 Django整体框架

本部分Django框架的整体实现，主要涉及的重点部分分别为后端实际用于处理和查询的views.py视图函数、前端分别用于输入数据的初始页面的index.html以及用于展示检测结果的result.html。



### 1.1.5性能测试

本部分主要用于指纹生成算法的性能测试，分别利用脚本计算各指纹生成脚本处理同样的1000条文本数据的平均每条耗时。



## 1.2功能设计

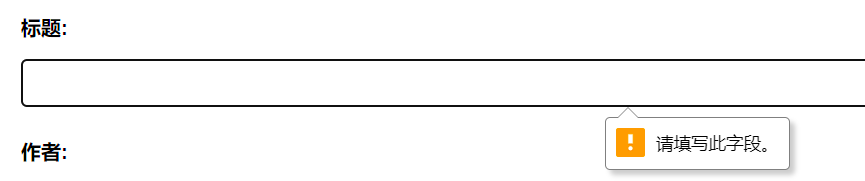
### 1.2.1开始界面

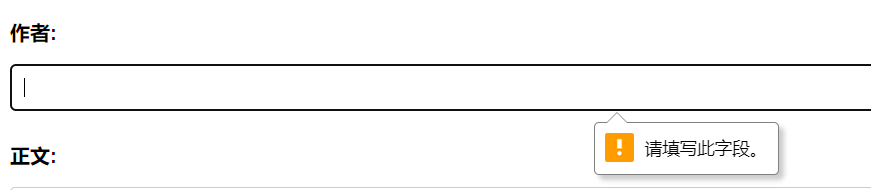


该页面为项目的首页，其项目启动后即可进入的页面。

#### a)必填限制

其中各“标题”和“作者”文本框供用户输入所传文本的标题和作者相关信息，并且带有必填性质，于前端部分设置了对应的限制功能，实际效果如下图：





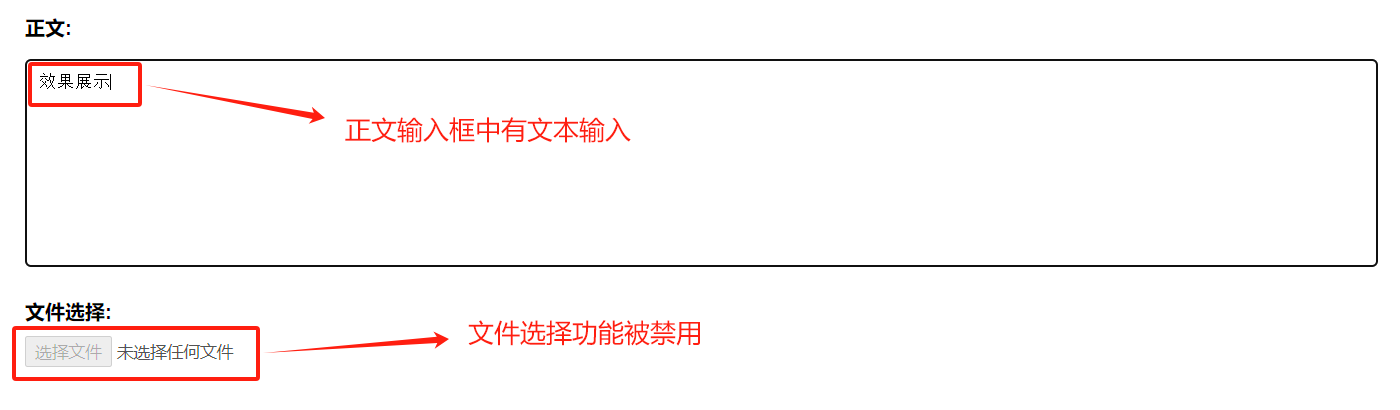
并且在已输入标题和作者但未上传文件并未于正文输入框中输入的情况下点击提交按钮时，弹出同样的“请填写此字段。”提示：



#### b)输入框、文件上传二选一

正文输入框和文件选择功能为二选一的功能，当且仅当有正文文本输入至正文输入框或传入了txt文件后才可以点击提交按钮进行提交以及检测。

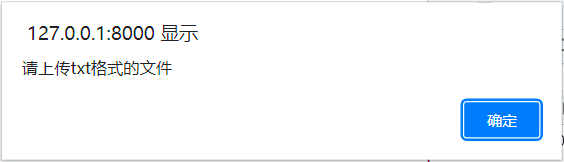
同时，为了防止两个部分传入了不同的文本内容，设置了使用了其中一个输入功能，则自动禁用另一输入功能的限制，具体效果如下图所示：





#### c)文件类型限制

为防止用户传入其他类型的文件带来的错误，以及目前暂时不支持处理txt以外的文本文件，故为文件上传功能设置了文件类型限制，当选择的文件非txt类型时，弹出警告弹窗，实际效果如下图所示：



#### d)提交按钮

当用户填入的数据符合上述限制并按下提交按钮后，前端将所输入数据以表单形式，通过向后端的视图函数以POST的方式提交至后端进行处理以及后续的抄袭检测。

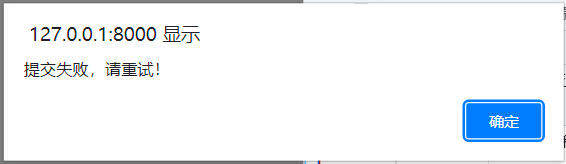
#### e)加载中蒙版

当用户填入的数据符合上述限制并按下提交按钮后，展示加载中蒙版，以此起到限制用户于加载过程中使用初始页面各功能而引起错误，实际效果如下图所示：



#### f)提交失败警告

当用户填入的数据符合上述限制并按下提交按钮后，如遇到了网络异常或是文件读取错误等情况时，弹出提交失败的警告弹窗，实际效果如下图所示：

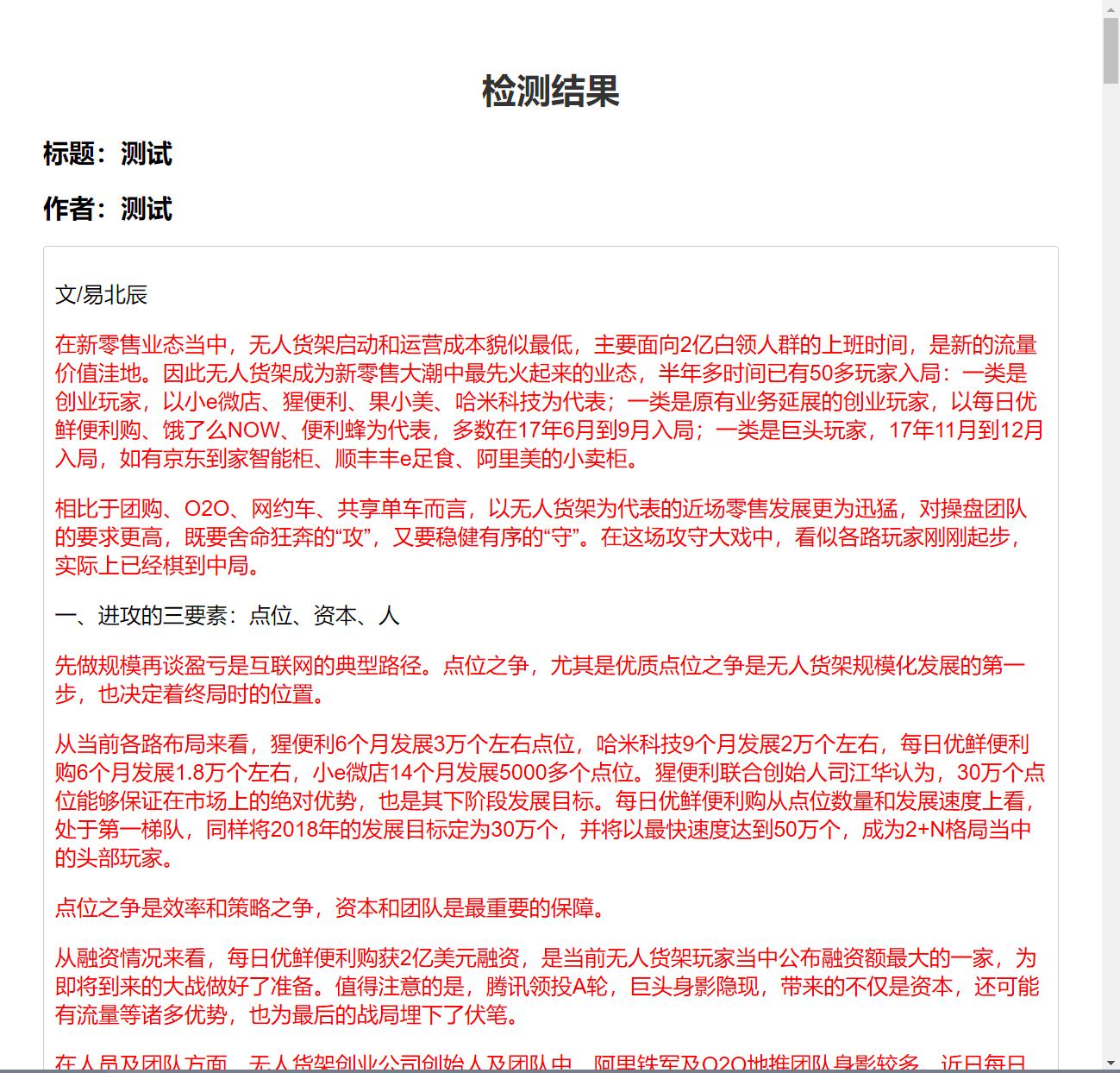


### 1.2.2检测结果页面

当用户成功提交后，后端在完成了相应的处理以及查询后，将返回的数据以参数的形式于result.html中返回，前端再将收到的响应用于替换当前网页html内容，实现页面切换。

#### a)原文抄袭部分标红

在此页面中，包含了传入的标题与作者，同时完整的展示了传入的原文，并以红色字体颜色标出检测为疑似抄袭的部分。实际效果如下：



#### b)具体抄袭来源展示

于正文下方的表格中展示原文中检测为抄袭的句子，以及该句子于数据库中查询道德对应的疑似抄袭来源。实际效果如下：



#### c)返回按钮

在表格下方有“返回”按钮，点击即可跳转回初始页面。



#### d)未检测到抄袭效果页面

当传入的文本经检测后未找到涉嫌抄袭的部分，则通过前端修改页面，隐藏该表格，并显示“未检测到可能抄袭的文段”的提示语。



### 1.2.3多端适配

本项目前端设置了自动页面尺寸，可以根据用户所使用设备自动调整页面尺寸，以达到多端适配的效果，手机端展示如下：



# 二、语料库构建与预处理方法

本次实验从github选取了内含一万篇文本的《SmoothNLP专栏资讯数据集样本10k》。文件为.xlsx文件，为对其文本进行合理划分及处理，我们使用了科大讯飞的spark星火大模型去除废弃字符、格式标签等无关字符，然后对其处理得到文本的标题、来源、作者信息、正文。该版本将文本内容划分到句级。

# 三、特征提取与指纹生成算法原理

我们在构建系统时依据以分段为颗粒度和分句为颗粒度分析指纹都做了尝试。

我们首先以段级为文本划分生成指纹的颗粒度，考虑段级文本长度够长，可以得到充分的特征来对其生成文本指纹。

## 3.1预处理：

在进行指纹提取与指纹生成算法之前，首先对文本进行预处理操作。预处理操作分为下面数据清洗和分词两部分：

a)数据清洗，去除停用词。

b)由于文本为中文文本，采用jieba、THULAC等工具进行中文分词、关键词提取。

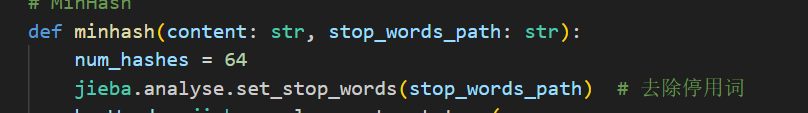
### 3.1.1数据清洗

由于我们的文本为中文文本，因此在github中找到中文常用停词表，下载并放在本地init\_database文件夹下，后续在特征提取部分调用进行停词去除，废弃字符格式标签等无关字符在前一步中已经使用大模型处理。

在simhash和minhash算法函数中调用：

descript

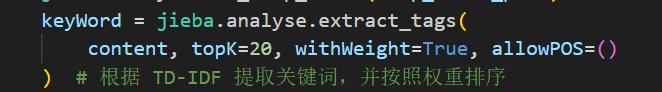
在simhash实现函数中调用去除



在minhash实现函数中调用去除

## 3.2特征提取：

根据TD-IDF提取进行分词并特征提取按照权重排序



## 3.3指纹生成算法：

使用simhash和minhash算法，算法的实现位于init\_database\ckg.py中。

在使用向量数据库时,使用bge embedding技术生成文档指纹，该算法位于sql\_script\vector.py，可以正常生成文档的指纹，格式为1024位向量。但由于需调用大模型计算时间过长，不适合我们快速查询的要求。且其维度过多，基于段级和句级颗粒度查询不适应，在性能评估后部将其作为其余两种算法性能的对比，不于后续的实际系统中使用。

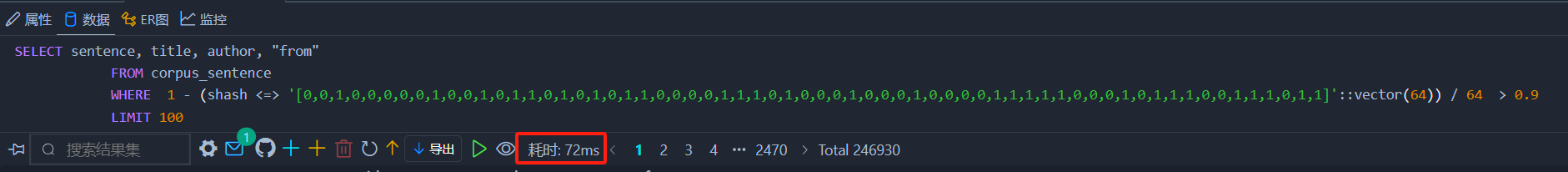
# 四、检索优化与相似度计算方法

为了提高检索速度，我们通过调查研究以及参考老师给出的建议，进行了相关的改进。

## 4.1向量数据库postgresql和pgvector插件

使用向量数据库postgresql和pgvector插件，将数据库中simhash和minhash存储为向量。同时基于欧几里得距离和余弦相似度构建索引，并在基于相似度查询处使用了其封装好的余弦相似度操作符计算相似度，提高检索速度。

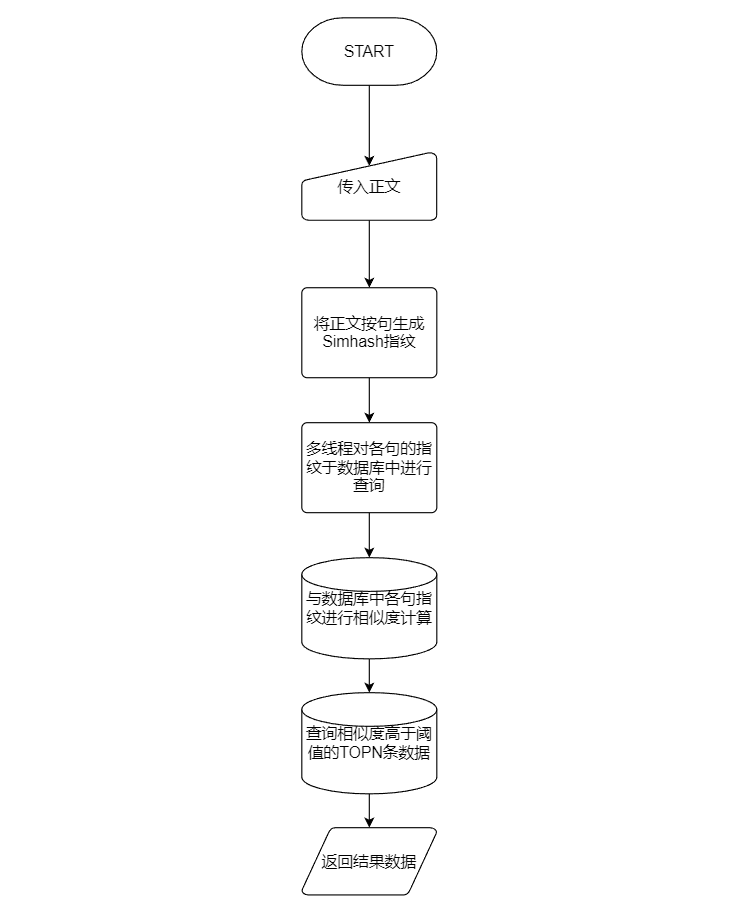
## 4.2使用多线程并行检索



通过使用多线程的并行方式，数据库中数据库中检索相似度超过阈值的查询操作可以到达秒级。并且，相似度计算在使用mysql数据库中欧几里得距离操作符构成海明距离相似度，余弦相似度操作符实现余弦相似度。

# 抄袭检测流程

本项目抄袭检测流程如下流程图所示：



# 六、系统性能分析

系统性能分析我们分为了理论性能分析以及系统实际整体性能分析两部分：

## 6.1理论性能分析

在此理论性能分析部分，我们对比评估了SimHash与MinHash的计算效率、碰撞概率、空间复杂度等文本指纹和特征提取性能指标。

计算效率脚本位于performance\_test文件夹下，碰撞概率通过数据库查询重复的文档指纹测试，空间复杂度由算法原理分析。其结果如下：

### 6.1.1计算效率

我们随机选取1000条文本去做文本划分，预处理，特征提取及指纹生成，多次实验并求出平均每条处理速度，即计算效率结果，得到如下表格：

|  |  |  |
| --- | --- | --- |
|  | SimHash | MinHash |
| 平均每条处理速度  （第一次实验） | 0.002556s | 0.004420s |
| 平均每条处理速度  （第二次实验） | 0.002562s | 0.004608s |
| 平均每条处理速度  （第三次实验） | 0.002539s | 0.004429s |
| 平均值 | 0.002552s | 0.004486s |

从测试结果可以得知SimHash的处理速度较MinHash更加快速。

### 6.1.2碰撞概率

我们选取碰撞概率来评估指纹生成算法可能出现的指纹重复问题，基于其原理分析即有：

误报率：计算不同文本产生相同指纹的概率。通过统计最终生成的相同指纹的数量进行计算最终得到如下结果：

SimHash：0.27%

MinHash：0.25%

二者非常接近且都非常的小，且由于是句级，有很少部分很相似的句子会产生相同的指纹，可以忽略不记。

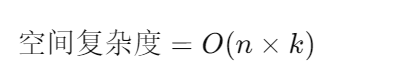
### 6.1.3空间复杂度

SimHash：

·SimHash生成一个固定长度的二进制字符串（通常为64位或128位）。

·如果我们使用k位的SimHash，那么每个文本的指纹占用的空间为k位。

对于n个文本：



这里的k是SimHash的位数。

MinHash：

·MinHash生成多个哈希值作为指纹，每个哈希值对应一个哈希函数。

·如果我们使用m个哈希函数，每个哈希值占用固定的空间，那么每个文本的指纹占用的空间为m个哈希值。

对于n个文本：

descript

这里的m是哈希函数的数量，hash\_size是每个哈希值的位数。

我们通过让哈希函数的个数与SimHash位数一致，可以得出MinHash的空间复杂度是更大的，这也符合其两种不同算法的优缺点和使用场景：

·SimHash倾向于更简洁的表示，适合快速比较和处理大规模数据。

·MinHash通过多个哈希值捕捉更多细节，相对更准确，适合需要高精度相似性检测的场景

## 6.2系统实际整体性能分析

为了对系统整体的性能进行优化，我们对系统整体的性能进行了分析。我们设计了基于阈值过滤、TopN筛选等策略的候选文本快速筛选方法，用于测试评估查全率和查准率，以及进行了线程数量与运行时间在短文本和长文本下进行了实验场景测试。

### 6.2.1查准率对比分析

通过随机从数据库中提取句子并进行同义修改，多次调整相似度阈值设置进行查全率和查准率测试。测试结果如下表所示：

通过上述测试可以分析得知，simhash与海明距离计算相似度配合在查全率与查准率的表现都更好，且在保证查全率查准率平衡时在三者中都保持了最好的效果。分析原因是由于句级生成而minhash更适合全局特征和相互特征等查询，且simhash的文本指纹和海明距离结合更加合适。综上分析，我们最终在系统中默认使用了simhash和海明距离计算相似度的配合。

### 6.2.2线程数量与系统整体运行速度

在调试过程中，我们发现利用多线程进行的处理操作的线程数量并非与最终系统的运行速度成正比。为此我们设计了一系列实验，对该现象进行深入的研究。

我们通过传入相同的文本，记录在不同的线程数量下的系统整体检测速度。其中为增加实验的可靠性，我们制作了两篇测试文本，分别为用于生成数据库中的数据的其中一篇原文构成的短文本，以及采用了数据库中原句、原句同义修改、自制非抄袭文本多种类型内容构成的长文本。

实验结果如下图所示：

由实验结果可知，传入短文本时的系统整体的检测耗时在线程数大约为12时达到最小值，传入长文本时的系统整体的检测耗时则在线程数大约为20时达到最小值。同时，在此过程中，我们观察到高线程数运行时的内存占用较高，接近溢出，甚至在某些时刻发生栈溢出的错误。对此我们推测当线程数超过某一值时，更多线程导致的额外堆栈使内存处于溢出边缘，此时可能存在的某一机制对内存进行限制，亦或是内存频繁地进行分页的换入换出以换取足够的空间，因此降低内存性能从而导致系统整体检测速度降低。

故我们得出结论，本项目系统的线程数量设置需要根据所部署的机器的内存相关性能进行对应调试，保证其能够在内存性能最佳的情况下运行。

## 6.2.3总结

综合所有性能测试，选择文本指纹和特征提取性能和查全率查准率效果最好的simhash配合海明距离作为相似度方法，在最终结果显示时使用阈值过滤、TopN筛选，用于最终系统。

在进行系统测试中发现如下问题并思考分析，考虑段级，若有人将一段中部分删除，将此时该段作为一个句子，此时原句级相似度下降，会导致在我们现有句级检索的原合适相似度阈值下查不到，但是降低相似度又会导致查准率降低。为应对上述问题所以我们保留迭代前段级查询的版本，由于段级查询的数据同时导入数据库后，会影响查询效率，所以我们选择不同时导入段级和句级的数据，而是保留了段级查询的版本。

其余算法算法在前面都已实现，可以在其余场景下使用，如对整篇搜索或者长度很长的文本，使用bge embedding技术非常适合，同时如果要长文本精确搜索，minhash和bge embedding技术结合使用会有更好的效果。

# 七、分工详情

## 赵志宸：

* 背景调研
* PPT制作
* 报告大纲拟定
* 指纹生成算法处理速度测试
* MySQL数据库相似度计算函数

## 刘函城：

* 指纹生成算法调研及实现
* 相似度算法调研及实现
* PostgreSQL数据操作相关脚本
* 算法优化对比分析
* 检索优化方法调研及实现
* 指纹生成算法性能测试
* 系统整体性能测试
* 报告编写

## 倪必凯：

* 数据库批量灌库脚本
* 数据预处理与处理脚本
* MySQL数据操作相关脚本
* Django整体开发与实现
* 后端视图函数实现
* 前端实现
* 检索优化方法实现
* 指纹生成算法性能测试
* 系统整体性能测试
* 报告编写