Computer Science and Application 计算机科学与应用, 2023, 13(12), 2575-2586
Published Online December 2023 in Hans. https://www.hanspub.org/journal/csa
https://doi.org/10.12677/csa.2023.1312256

Hans汉斯

优化LangChain框架中的文档分割方法：方法与应用

蔡运生，穆欣宇，董　浩，陈国铨，孙　达

北京信息科技大学计算机学院，北京

收稿日期：2023年11月26日；录用日期：2023年12月22日；发布日期：2023年12月30日

摘　要

本研究旨在改进LangChain框架中的文档分割方法，以提高大型语言模型处理长文本的效率和准确性。通过分析现有的文档分割工具，发现其可能导致语义断裂和处理效率低下的问题。针对这些问题，提出了一种基于KMeans聚类算法的优化策略，以保持文本的语义连贯性和句子的原始顺序。构建了名为TextSplitter的类和名为chunk_file的函数，实现了新的文档分割和聚类方法。通过PK值评估法对优化策略的效果进行了验证，并通过实验展示了新方法相较于现有方法的优势。本研究不仅为LangChain框架的文档分割提供了有效的优化方案，也为处理大规模文本数据提供了有益的参考。

关键词

LangChain框架，文档分割方法，大型语言模型，KMeans聚类算法

Optimization of Document Segmentation Method in LangChain Framework: Methods and Applications

Yunsheng Cai, Xinyu Mu, Hao Dong, Guoquan Chen, Da Sun

School of Computer Science, Beijing Information Science and Technology University, Beijing

Received: Nov. 26th, 2023; accepted: Dec. 22nd, 2023; published: Dec. 30th, 2023

Abstract

This study aims to improve the document segmentation method in the LangChain framework to enhance the efficiency and accuracy of large language models in processing long texts. By analyz-

文章引用: 蔡运生, 穆欣宇, 董浩, 陈国铨, 孙达. 优化 LangChain 框架中的文档分割方法: 方法与应用[J]. 计算机科学与应用, 2023, 13(12): 2575-2586. DOI: 10.12677/csa.2023.1312256

ing existing document segmentation tools, issues related to semantic discontinuity and inefficiency were identified. To address these issues, an optimization strategy based on the KMeans clustering algorithm was proposed to maintain the semantic coherence and original order of the sentences. A class named TextSplitter and a function named chunk_file were constructed to implement the new document segmentation and clustering methods. The effectiveness of the optimization strategy was verified through the PK value evaluation method and the advantages of the new method over existing methods were demonstrated through experiments. This study provides not only an effective optimization solution for document segmentation in the LangChain framework but also serves as a valuable reference for processing large-scale text data.

Keywords

LangChain Framework, Document Segmentation Method, Large Language Models, KMeans Clustering Algorithm

1. 引言

1.1. 人工智能与大型语言模型的发展背景

在过去的几年里，人工智能(AI)取得了显著的进步，其中一个突出的例子是大型语言模型(LLMs)的发展。大型语言模型是一种经过大量文本数据训练的人工智能系统，它们能够理解自然语言并产生人类般的文本[1]。这些模型如 OpenAI 的 GPT 系列，已经在自然语言处理(NLP)领域中表现出了显著的优势，不仅改变了我们与机器的交互方式，还彻底改变了多个行业。

随着技术的快速发展，大型语言模型已经成为了众多应用的强有力工具，包括在医疗保健领域[2]。例如，GPT 和 LLaMA 等大型机器学习模型已经显示出改善病人疗效和转变医疗实践的潜力。大型语言模型的出现和发展是人工智能技术不断进步的直接结果，它们为处理和理解大规模文本数据提供了新的可能，也为自然语言处理领域带来了革命性的变化。

1.2. LangChain 框架及其在自然语言处理中的应用

LangChain 框架是一个开源解决方案，旨在简化大型语言模型(LLM)驱动应用的开发[3]。使构建人工智能解决方案更为容易。该框架的架构集成了几个复杂的模块，这些模块对于自然语言处理(NLP)应用的无缝执行至关重要[4]。通过 LangChain，开发者可以方便地创建和部署涵盖聊天机器人、问答系统、摘要生成和其他对话应用的 NLP 任务[5]。它提供了一种模块化的方法，允许通过模块组合使用 LLM，并与其他模块结合以创建应用。

LangChain 的工作流程是一个高度集成和自动化的过程(图 1)，简单来说就是从加载读取数据文件开始，然后对文本和问句进行分割和向量化，在文本向量中匹配出与问句向量最相似的几个部分，加入到提示中提交给大语言模型[6]。LangChain 提供支持高度自动化和可定制的文本处理流程，使开发人员可以更加轻松的管理与语言模型的交互[7]。特别是在处理长文本时，LangChain 通过提供文档分割工具，解决了大型语言模型一次处理的 token 限制问题，为自然语言处理应用提供了有效的解决方案。

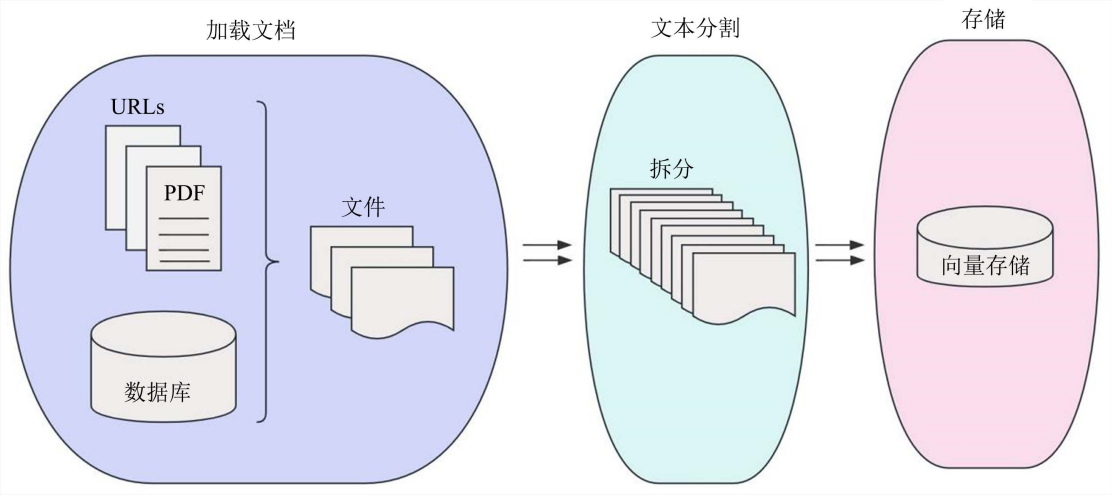

Figure 1. Working principle of the Langchain framework
图 1. Langchain 框架工作原理

1.3. 文档分割的挑战与重要性

文档分割是自然语言处理(NLP)中的一项基本任务，其目标是将较大的文本内容划分为更易于处理的小块[8]。这一过程对于保证分析工作的准确性和效率至关重要。然而，随着大型语言模型(LLM)如 GPT-3 和 GPT-4 的兴起，文档分割面临了新的挑战。这些模型虽然在理解和生成文本方面表现出色，但它们在处理大规模数据时遇到了 token 限制的问题。例如，GPT-3 的 token 限制在 4 k 到 8 k 之间，而 GPT-4 则在 8 k 到 32 k 之间，这意味着处理能力受到限制，尤其是在处理中文等字符占用更多 token 的语言时。

为了最大限度地发挥 LLM 的潜力，开发出有效的文档分割方法显得尤为重要。这不仅涉及到保证每个文本块都在模型处理能力的范围内，还要确保在分割过程中能够保留足够的上下文信息，以便获得有意义的分析结果。此外，文档分割方法的选择和优化对于提升模型的整体表现和应用效果有着直接影响。因此，为特定的 LLM 和应用场景设计合适的文档分割策略，不仅是一项技术挑战，也是推动自然语言处理技术发展的重要环节。

2. LangChain 中现有的文档分割方法

2.1. text_splitter 的基本功能与控制选项

LangChain 框架通过 text_splitter 模块实现了对长文本的高效分割，以适应大型语言模型(LLM)处理大量文本数据的需求[9]。该模块提供基础的文档分割功能，并配有多样化的控制参数，使用户能够根据特定需求定制文档分割细节，确保文档处理在 LLM 窗口限制内最优化。

text_splitter 的主要控制参数包括：

- length_function：计算每个文本块长度的方法，使开发者能够精确控制文本块大小，满足不同模型和算法的需求。
- chunk_size：设定分割文本时每块的最大长度，确保文本块适应处理系统或模型的输入限制。
- chunk_overlap：定义连续文本块之间的字符重叠量，保持文本连续性和上下文信息，特别适用于需考虑上下文的 NLP 任务[3]。

这些控制参数的灵活性和多样性使 text_splitter 成为一种简洁且高效的文档分割解决方案，允许开发者针对不同应用场景进行精细调整。

2.2. 现有文档分割工具的概述

在 LangChain 框架的 text_splitter 模块中，提供了三种主要的文档分割工具来满足不同的分割需求：RecursiveCharacterTextSplitter，CharacterTextSplitter 和 TokenTextSplitter [10]。它们为大型语言模型处理过长的文本提供了基础设施，尤其是在文档长度超过模型的最大输入长度时。

2.2.1. RecursiveCharacterTextSplitter

在 LangChain 框架中，RecursiveCharacterTextSplitter 作为一种专门的文本分割工具，它的主要特点是递归地基于特定字符来进行文本切割[3]。首先，它会按照预先设定或用户自定义的字符，例如换行符和空格，对文本进行初步分割。若分割后的文本块长度超过设定的 chunk_size 值，则继续使用下一个字符进行更细致的分割，直至文本块长度接近于 chunk_size 值。此方法通过递归切割和合并，尽可能保证文本块的长度接近设定值，以适应后续处理的需求。

可以用如下代码来导入 RecursiveCharacterTextSplitter (图 2)：

```
Python
from langchain.text_splitter import RecursiveCharacterTextSplitter

text_splitter = RecursiveCharacterTextSplitter(
    chunk_size = 200,
    chunk_overlap  = 20,
)
```

Figure 2. Code for importing RecursiveCharacterTextSplitter
图 2. RecursiveCharacterTextSplitter 导入代码

2.2.2. CharacterTextSplitter

CharacterTextSplitter 是基于特定字符进行文本切割的工具。它主要利用 Python 内置的 split 方法，以预定义或用户指定的字符(如换行符)为基础对文本进行分割。不同于 RecursiveCharacterTextSplitter，它仅执行一次分割，简化了处理流程，但可能会导致文本块的长度不一致。该工具提供了一种快速而简单的方式来处理文本分割的需求，尤其是在字符分割符明确的场景中。

2.2.3. TokenTextSplitter

TokenTextSplitter 是 LangChain 中基于语义的文本切割工具。它首先将文本内容转换为 Token，再基于这些 Token 进行文本切割。每个 Token 通常对应于自然语言处理中的一个最小语义单位。TokenTextSplitter 的设计考虑了不同语言模型可能采用不同的 Token 编码方法(表 1)，提供了一种从语义层面进行文本切割的策略。

Table 1. Common Token encoding in OpenAI models
表 1. OpenAI 模型常用 Token 编码

编码名称	OpenAI 模型
Cl100k_base	gpt-4, gpt-3.5-turbo, text-embedding-ada-002
p50k_base	Codex models, text-davinci-002, text-davinci-003
r50k_base (or gpt2)	GPT-3 models like davinci

与基于字符的分割方法相比，TokenTextSplitter 能够提供更为准确和高效的处理效果，有助于保持文本的语义完整性。

3. 文档分割方法的局限性与挑战

3.1. 各工具的主要缺陷

3.1.1. RecursiveCharacterTextSplitter

LangChain 框架中的 RecursiveCharacterTextSplitter 工具以字符为单位对长文本进行分割[11]。此工具内置一个字符数组，用于存储可能的分割字符，通过递归方式尽可能将长文本切割成等长的文本块。然而，这种递归的分割方法时间复杂度较高，具体为 O(m*n)，其中 m 是 splits 数组的长度，n 是字符数组的长度。此外，该方法可能会导致文本的语义断裂，因为它完全依赖于字符级别的分割，忽略了文本的语义结构。

3.1.2. CharacterTextSplitter

CharacterTextSplitter 与 RecursiveCharacterTextSplitter 类似，也是基于字符进行文本分割。区别在于，CharacterTextSplitter 要求用户指定分割字符。此方法简单快速，但分割结果可能会超过指定的 chunk_size，且容易导致语义的撕裂，特别是在无法找到合适的分割字符时。

3.1.3. TokenTextSplitter

TokenTextSplitter 则是基于 token 进行文本分割，每个 token 通常对应英文单词的一部分或一个中文字符。相比字符分割，token 分割更能保留文本的语义完整性。然而，由于 token 的不确定性，可能会在单词或句子的边界处产生分割，导致性能下降。

3.2. 实际应用中遇到的问题与挑战

在实际应用中，文档分割方法的选择对大型语言模型处理长文本的效果有显著影响。由于大型语言模型的 token 数限制，需要将长文本分割为多个较小的文本块以适应模型的处理能力。不同的分割方法会影响分割结果的质量和处理效率。例如，递归的分割方法虽然能够得到较为均匀的文本块，但时间复杂度高，效率较低。同时，文档分割可能会导致原始语义的丢失或歧义，影响模型的输出质量。目前，除了 LangChain 框架中的分割工具外，还有诸如 NLTK 和 Spacy 的句子分割器等，但它们也存在类似的问题和挑战。为了改善文档分割的质量和效率，本文基于 KMeans 聚类算法，提出了一种新的基于语义相似度的文档分割方法。

4. 优化 LangChain 的文档分割方法

4.1. 优化策略与方法的提出

在处理大规模文本时，传统的文档分割方法面临诸如破坏语义完整性和效率低下的问题。为此，本文提出了一种基于 KMeans 聚类算法的优化策略，旨在通过维持文本的语义连贯性和句子的原始顺序来改善文档分割方法。

KMeans 是一种广泛应用的聚类算法，其核心思想是通过迭代优化的方式，将数据点分配到最近的聚类中心，从而将数据分为 K 个聚类[12]。具体来说，KMeans 聚类算法的目标是最小化每个点到其聚类中心的距离之和，数学上可以表示为：

$$J = \sum_{i=1}^{n} \sum_{j=1}^{k} \omega_{ij} \left\| x_i - \mu_j \right\|^2$$

其中：n 是数据点的数量，

k 是聚类的数量，

ω_{ij} 是指示变量，如果数据点 i 在聚类 j 中，则 ω_{ij} = 1，否则 ω_{ij} = 0，

x_i 是数据点，

μ_j 是聚类 j 的中心。

然而，直接将 KMeans 应用于句子分割可能会打破句子的原始顺序，因为语义相似但在文本中位置较远的句子可能会被划分到同一聚类。

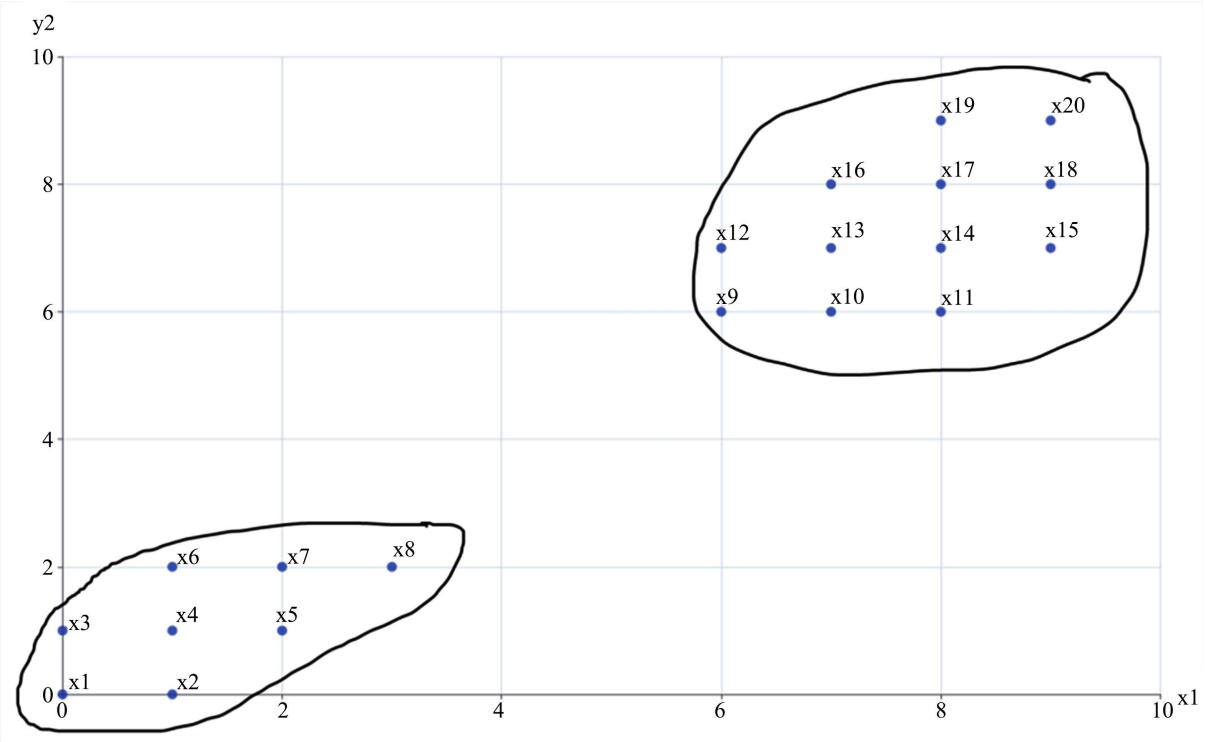

Figure 3. Principle of the KMeans algorithm
图 3. KMeans 算法原理

为了维护句子的原始顺序和提高文档分块的语义连贯性，我们采用了一种基于 KMeans 聚类算法的优化策略(图 3)。该策略以相邻句子的语义相似度作为聚类依据。在实施过程中，我们遍历分割后的句子集合，计算每对相邻句子之间的语义相似度。当这些相似度超过预设阈值时，这些句子被归入同一文本块；否则，它们被划分到不同的文本块中。在每次迭代中，聚类中心通过以下公式进行更新：

$$C = \frac{1}{n}\sum X_i$$

其中：C 表示聚类中心，n 是聚类中的数据点数量，$\sum X_i$ 表示聚类中所有数据点的总和。

完成一轮聚类之后，我们会检查每个生成的文本块的长度。如果发现文本块长度不符合预期(过短或过长)，我们会调整语义相似度的阈值，并重新进行聚类。这个过程重复进行，直至所有文本块的长度均接近于预设的目标长度。这样，我们既保留了文本的语义连贯性和句子的原始顺序，又确保了文本块长度的均匀性，有效解决了传统文档分割方法可能遇到的问题。方法流程图(图 4)如下：

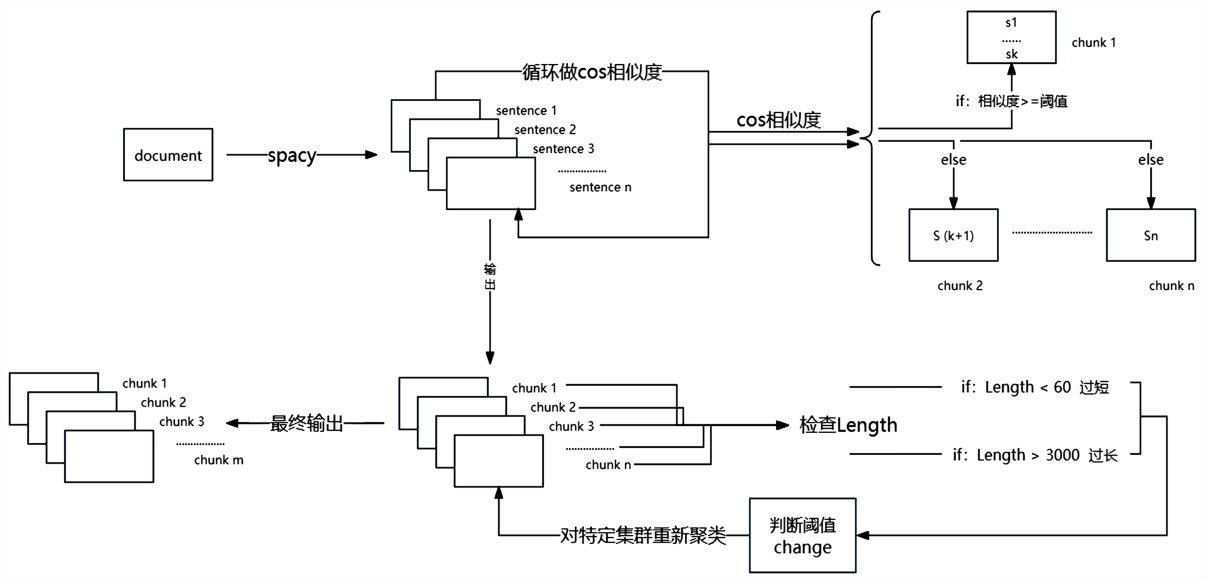

Figure 4. Adjacent sentences KMeans algorithm
图 4. 相邻句子 KMeans 算法

4.2. 优化后的工具与方法的实现

为了实施我们的优化策略，我们开发了一个名为 TextSplitter 的类，它包含两个主要方法：process 和 cluster_text。此外，我们设计了 chunk_file 函数，用于在 File 实例上执行文档的分块和聚类。这个函数接收多个参数，如块大小、重叠度、分割方法和相似度阈值，允许对文档的处理进行精细调整。

通过 chunk_file 函数，TextSplitter 类的方法可以应用于 File 实例中的每个文档，从而执行内容分割和聚类，并生成包含处理后文档的新 File 实例。这种设计不仅提高了文本分割和聚类过程的可配置性和灵活性，而且便于进行后续的文本处理和分析。

4.2.1. 文本分割与向量计算

Process 方法(图 5)首先依据传入的 method 参数(可为按句子、段落或特定关键字分割)调用 split_text 方法来分割文本。接着，该方法为每个分割出的文本段计算向量表示，这一步通过 Spacy 库的 NLP 对象完成，生成固定维度的向量。为了后续相似度计算的准确性，这些向量被单位化，即除以它们的模长。最终，process 方法输出分割的文本段及其相应的单位化向量，为后续步骤做好准备。

```
Python
def process(self, text, method):
    segments = self.split_text(text)  # 根据所选方法分割文本
    # 计算每个段落/句子的向量，并进行单位化
    vecs = np.stack([nlp(segment).vector / nlp(segment).vector_norm for segment in segments if nlp(segment).vector_norm > 0])
    return segments, vecs
```

Figure 5. Code for text segmentation and vector calculation
图 5. 文本分割与向量计算代码

4.2.2. 文本聚类

Cluster_text 方法(图 6)启动时，将第一个文本段设为一个独立的类别。随后，该方法遍历所有文本段，通过计算相邻文本段向量的点积来评估它们之间的相似度。若相似度低于预设的阈值，便开始一个新的聚类；相反，如果相似度高，当前文本段则加入到现有的聚类中。这样，相似的文本段聚集在一起形成一个聚类，而不相似的则被分到不同的聚类里。

```
Python
def cluster_text(self, segments, vecs, threshold):
clusters = [[0]]  #初始化聚类列表，开始于第一个段落/句子
for i in range(1, len(segments)):
            # 通过比较相邻段落/句子的向量点积，决定是否开始新的聚类
if np.dot(vecs[i], vecs[i-1]) < threshold:
        clusters.append([])
clusters[-1].append(i)  #否则继续向当前聚类添加段落/句子
 return clusters
```

Figure 6. Text clustering code
图 6. 文本聚类代码

4.2.3. 聚类和长度检查

对每个生成的聚类，我们将聚类中的文本段合并成一个单独的文本，并检查其长度。根据文本的长度，我们决定是否保留该聚类、继续细分或跳过。如果聚类文本过长，我们通过降低相似度阈值，重新进行聚类，以期得到更细粒度的文本分割(图 7)。

```
Python
for cluster in clusters:
   cluster_txt = ' '.join([segments[i] for i in cluster])  # 生成聚类文本
   cluster_len = len(cluster_txt)
   # 检查聚类文本长度，以决定是否继续分割或直接保存
   if cluster_len < chunk_size / 4:
      continue  # 跳过过短的聚类
   elif cluster_len > chunk_size:
      new_threshold = 0.4  # 降低相似度阈值
      # ... (省略重新聚类和保存文档的代码)
```

Figure 7. Clustering and length check code
图 7. 聚类和长度检查代码

通过这种方式，chunk_file 函数为我们提供了一种灵活的文档处理框架，我们可以通过调整参数来实现不同程度的文本分割和聚类，以满足特定的应用需求。

4.3. 优化效果的验证与评估

为了评估优化策略的有效性，我们采用了 PK 值评估法。这个过程首先涉及对选择的长文本进行人工分块，以创建基准块，并使用自动分块方法生成系统块。在评估阶段，我们对比了原始文本中相邻段落在基准块和系统块中的分布情况。在基准块中，使用 is_new_block 函数标记了每个新文本块的开始，其中新块的开头标记为 true，非开头标记为 false。系统块也进行了相同的标记。通过这些标记的比较，我们能够观察自动分块和人工分块之间的差异。通过计算 PK 值，我们可以量化分块性能，其中较小的 PK 值表示更好的分块性能。我们提供了一个计算 PK 值的代码实现，通过 calculate_pk 函数计算得出，以量化分块性能(图 8)。

```
Python
def is_new_block(sentence):
# 检查句子是否以特定字符开始，以此来判断是否为一个新的文本块的开始
    return sentence.startswith("1.") or sentence.startswith("●") or
sentence.startswith("*^&")
# ...省略部分代码...
def calculate_pk(auto_segmentation, true_segmentation, window_size):
    num_mismatches = 0 # 不匹配的文本对数量
    num_pairs = 0  # 检查的文本对总数
# 遍历自动分块结果，以窗口大小为范围
    for i in range(len(auto_segmentation) - window_size + 1):
        for j in range(i, i + window_size - 1):
            for k in range(j + 1, i + window_size):
                num_pairs += 1 # 增加文本对计数
# 比较自动分块和真实分块中的相应文本对，记录不匹配的情况
                if (auto_segmentation[j] == auto_segmentation[k]) !=
(true_segmentation[j] == true_segmentation[k]):
                    num_mismatches += 1
# 计算并返回 PK 值，即不匹配对占总对数的比例
    return num_mismatches / num_pairs if num_pairs > 0 else 0
# ...省略部分代码...
```

Figure 8. Code for calculating PK values
图 8. PK 值计算代码

为了实施此评估，我们选择了不同类型和来源的文档(如新闻、医学、法律、科技等)作为原始数据集，并分别采用了三种不同的分块方法进行实验：RecursiveCharacterTextSplitter、TokenTextSplitter 和基于 KMeans 聚类算法的优化方法。通过比较这些方法的实验结果，我们能够展现出基于 KMeans 聚类算法优化方法相较于其他方法的优势。以下是我们挑选后具有代表性的几组数据(表 2)：

Table 2. Evaluation of PK values of different algorithms
表 2. 不同算法的 PK 值评估

分割方法 / PK 值 / 序号	RecursiveCharacterTextSplitter	TokenTextSplitter	基于 KMeans 聚类算法的优化方法
1	0.54	0.49	0.44
2	0.44	0.49	0.36
3	0.33	0.46	0.21
4	0.43	0.43	0.36
5	0.22	0.59	0.33
6	0.67	0.57	0.25

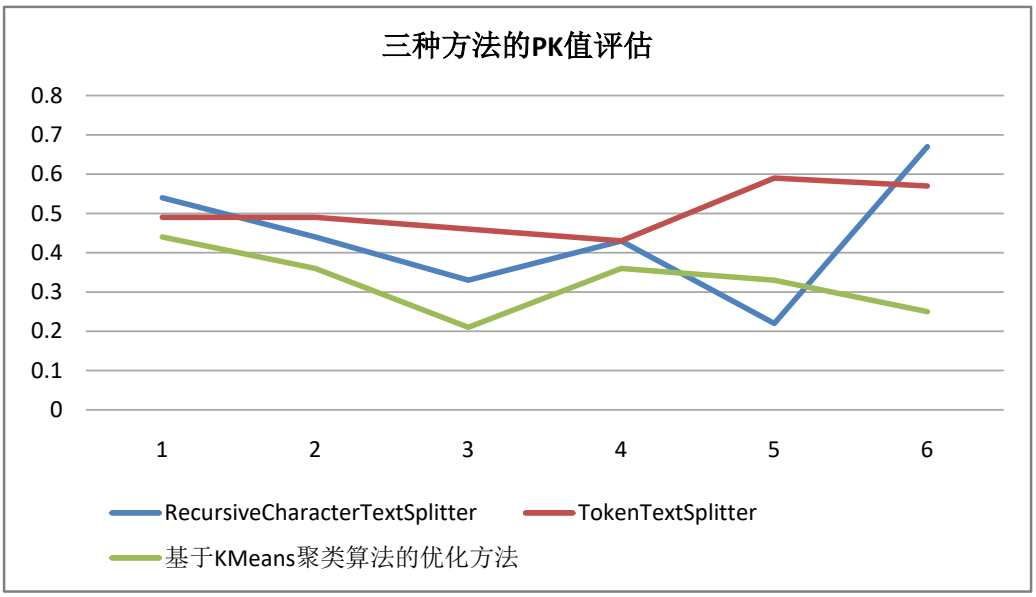

Figure 9. Evaluation of PK values of three methods
图 9. 三种方法的 PK 值评估

通过上述评估方法和实验结果，见表 2 及图 9，我们可以清晰地看到基于 KMeans 聚类算法的优化策略在实际应用中的效果和优势，为未来的研究和优化提供了有益的参考。

5. 实际应用与案例分析

5.1. 优化后的文档分割方法在实际项目中的应用

本研究提出了一种基于 KMeans 聚类算法的文档分割优化方法，并将其成功应用于一个基于 Langchain 框架的实际知识问答系统[13]。在数据预处理阶段，我们引入了预爬取的长文本数据，并使用 spacy 库将其细化为代表基本语义单位的句子数组。随后，应用优化的文档分割方法，对句子进行聚类分析，根据语义关联性划分为不同的文本块，以提高每个块的内聚性，并为后续处理提供结构化的文本数据。这些文本块通过 Embedding 技术转化为向量并存储在数据库中。用户提问时，系统将问题转化为向量，并与数据库中的文本块向量进行相似度比对，选择最匹配的向量作为语境，结合问题向量进行处理，并以自然语言形式输出精确、语境相关的答案。

5.2. 与现有方法的对比与分析

在探索文档分割技术时，我们通过相邻语句的聚类分析，将语义相似的句子聚集成文本块，不仅提升了分割准确性，也成功融合到基于 Langchain 框架的知识问答系统中。本节将此优化方法与 Langchain 原有的文档分割方法对比。

首先，Langchain 原有方法如 RecursiveCharacterTextSplitter 和 CharacterTextSplitter，主要依赖字符或词汇分割，忽略了文档的语义结构，导致文本块内句子语义不连贯。相比之下，本优化算法通过相邻句子的语义相似度进行聚类，保留了文本的语义结构，确保了文本块内句子的语义连贯性。

与 Langchain 的 TokenTextSplitter 方法相比，本算法在初步聚类后引入了长度检查机制，对生成的文本块进行长度检查与调整，保证文本块长度相似，提升了文档分割的均匀性。同时，这种基于语义的聚类分析也弥补了在保证文本块内语义连贯性方面的不足。

通过在知识问答系统中的应用实践，我们发现该优化方法不仅提高了文档分割的准确性，而且将文本块转化为向量形式并存储于向量数据库，极大地提升了问答系统在处理长文本数据时的效率和准确性。

综上所述，本算法相较于 Langchain 框架原有的文档分割方法，具有更强的语义保留和更高的分割准确性。同时，也为进一步优化知识问答系统中的文档处理流程提供了有力的实证支持。

6. 结论与未来工作

6.1. 本研究的主要贡献与意义

本研究专注于文档分割技术的优化及其在知识问答系统中的应用。研究首先深入分析 Langchain 框架及其文档分割技术，对比不同方法后，提出一种基于无监督学习的新算法——相邻句子聚类算法。该算法通过分析相邻语句的语义相似度，将相似度高的语句聚集到同一文本块中，显著提升文档分割的准确性和效率。实验评估表明，这一算法相比 Langchain 框架原有的基于字符或词汇的分割方法，更能有效保留文本的语义结构，确保文本块内句子的语义连贯性。算法还引入长度检查机制，确保文本块长度的一致性，提高了文档分割的均匀性。

本研究最终成功将此文档分割方法整合进基于 Langchain 框架的知识问答系统，丰富了文档分割技术领域的研究，为基于大语言模型的知识问答系统提供了有价值的参考。这不仅推动了相关技术领域的进步，还为后续研究和实际应用提供了重要的参考价值。

6.2. 对未来研究与应用的建议与展望

本研究成功地将聚类算法应用于文档分割，并在知识问答系统中取得了良好的效果，但仍存在进一步优化和拓展的空间。首先，未来可以探索更多的无监督学习算法以进一步提升文档分割的准确性和效率。同时，对于聚类算法的优化也是未来研究的重要方向，以期降低算法的计算复杂度和提升实时性。

此外，该聚类算法的应用不应局限于文本分割和自然语言处理领域，它在图像、音频等多媒体数据处理中也具有广泛的应用前景。未来，可以探索将本研究中的聚类算法应用到更多领域，以实现多媒体数据的高效处理和分析。

总的来说，本研究为未来相关领域的研究和应用提供了有益的思路和基础，期待通过持续的研究和探索，推动相关技术向前发展，实现更多实际应用的价值。

基金项目

本文由北京信息科技大学促进高校分类发展－大学生创新创业训练计划项目——计算机学院(5112310855)支持。

参考文献

[1] Tardif, A. (2023) Unveiling the Power of Large Language Models (LLMs). https://www.unite.ai/large-language-models/#:~:text=Updated%20on%20April%2022%2C%202023,machines%20and%20revolutionizing%20various%20industries

[2] Briganti, G. (2023) A Clinician's Guide to Large Language Models. https://www.futuremedicine.com/doi/full/10.2217/fmai-2023-0003#:~:text=The%20rapid%20advancement%20of%20artificial,without%20a%20background%20in

[3] LangChain. https://python.langchain.com/docs/modules/data_connection/

[4] Sharma, R. (2023) Leveraging LangChain for Next-Gen Language Models. https://markovate.com/blog/langchain-for-language-models/#:~:text=The%20LangChain%20framework%20is%20an,NLP%29%20applications

[5] Lancaster, A. (2023) Beyond Chatbots: The Rise of Large Language Models. https://www.forbes.com/sites/forbestechcouncil/2023/03/20/beyond-chatbots-the-rise-of-large-language-models/?sh=20cdb9d92319

[6] Ali, M. (2023) How to Build LLM Applications with LangChain. https://www.datacamp.com/tutorial/how-to-build-llm-applications-with-langchain

[7] Enterprise DNA Experts (2023) What Is LangChain? A Beginners Guide with Examples. https://blog.enterprisedna.co/what-is-langchain-a-beginners-guide-with-examples/

[8] Todeschini, S. (2023) How to Chunk Text Data—A Comparative Analysis. https://towardsdatascience.com/how-to-chunk-text-data-a-comparative-analysis-3858c4a0997a

[9] Hashemi-Pour, C. (2023) What Is Generative AI? Everything You Need to Know. https://www.techtarget.com/searchEnterpriseAI/definition/LangChain#:~:text=LangChain%20is%20an%20open%20source,powered%20applications

[10] AI 让世界更懂你. 文本分割(话题分割)的 6 种评估性能的方法[EB/OL]. https://blog.csdn.net/qq_35082030/article/details/105410478, 2020-04-09.

[11] Hearst, M.A. (1997) TextTiling: Segmenting Text into Multi-Paragraph Subtopic Passages. *Computational Linguistics*, **23**, 33-64.

[12] Beeferman, D. (1999) Statistical Models for Text Segmentation. *Machine Learning*, **34**, 177-210.

[13] Manathunga, S. (2023) Knowledge GPT. https://github.com/mmz-001/knowledge_gpt