对人工智能领域学术论文摘要的 AIGC 自动检测研究

摘要

近年来，以 ChatGPT 为代表的 AI 大模型在多个领域具有出色表现与巨大潜力，因此迅速成为研究热点。然而，使用 AIGC 代写学术论文等滥用行为也层出不穷。为实现对 AIGC 的检测，本文基于特定领域收集到的数据，设计了 **BEM-RCNN 算法**，并与多种常用 AIGC 检测算法进行比较，再使用**高频词统计法**与 **N-gram 法**进行文本分析，最终给出 AIGC 检测建议。

针对问题一：首先，综合考虑数据易得、领域熟悉与研究意义，选定**“人工智能主题学术论文摘要”**这一特定领域，并按照引用量降序从知网上获取五个核心期刊中共 100 篇高引论文作为学者撰写的数据，同时选用多种 AI 大模型按照给定题目与提示词生成的相应内容作为 AI 生成的数据。随后，设计了 **BEM-RCNN 算法**，其分为两个模块，BEM 模块使用预训练的 **BERT 模型**并结合收集到的数据进行微调，对输入数据多层处理生成包含位置和语义信息的上下文丰富的词嵌入；RCNN 模块结合 **RNN 与 CNN** 优点，通过端到端的连接对 BEM 模块的词嵌入进行多层处理，提取深层特征完成最终的分类任务。最后，将该算法应用到数据集上，评估得到 **F1-score=0.96，Accuracy=0.96** 的高性能。

针对问题二：首先，考虑到常用的 AIGC 检测算法不能直接处理文本数据，因此进行数据预处理：在完成类别标记后，构建停用词表并使用 **Jieba 分词库**进行分词操作，随后使用 **TF-IDF** 进行文本向量化，最后使用**卡方检验**进行特征筛选并使用 **Truncated SVD** 进行降维操作。接着，将已预处理的数据应用到 8 种如朴素贝叶斯、逻辑回归、随机森林等 AIGC 检测算法上，求得对应的 F1-score、Accuracy 等指标数值。最后综合分析，发现**自设计的 BEM-RCNN 检测算法性能更佳**。

针对问题三：为了寻找肉眼识别 AIGC 的方法，深入分析两类文本内容，选定**摘要长度与句子数量、词汇特征与常用搭配**两个指标。首先利用 Python 对两类文本进行数据分析与可视化，得出**论文摘要长度过长或过短，句子数量过多或过少大概率为学者撰写**的结论。然后使用高频词统计法对高频词与独有词进行统计排序，并使用 N-gram 法对常用搭配进行排序，得出**学者撰写的文本用词常见且易于理解，其使用的常用搭配比 AI 生成的更具有实际意义**的结论。

针对问题四：结合以上三个问题的结论，从**算法检测**和**人工审核**两个角度给出建议。其中算法检测要注意数据获取、模型训练与算法优化三个环节；人工审核要注意标准设定、专业培训与质量控制三个部分。

关键字： AIGC　BEM-RCNN　TF-IDF 向量化　高频词统计法　N-gram

一、 问题重述

2022 年，随着 ChatGPT 等人工智能大模型的问世，人工智能生成内容（AIGC）迅速成为学术界和工业界的研究热点。这些大模型在语言处理、文本生成、知识问答等多个领域具备广泛的应用潜力。然而，随之而来的是一系列潜在的伦理与法律挑战，尤其是 AIGC 技术被不当利用于代写论文等学术不端行为，直接触及学术诚信的底线。《中华人民共和国学位法（草案）》明确指出，一旦发现此类行为，学位授予单位有权撤销学位。因此，如何有效识别并检测特定领域的 AIGC 内容，以保证学术内容的真实度极其重要。

在此社会需求与政策导向下，试建立数学模型分析研究下面的问题：

（1）选择一个特定的领域，收集该领域内的人工智能生成文本与人类原创文本作为研究样本，并基于所收集的数据，设计并实现一个 AIGC 检测算法，用来捕捉并区分 AIGC 与人类创作之间的差异。

（2）将设计的检测算法与现有的常用 AIGC 检测算法进行比较，分析评估各自的优势和不足。

（3）探讨是否可以通过肉眼识别等传统方法有效辨认 AIGC，分析人类直观判断的局限性与可能性。

（4）基于研究结果，提出有效的 AIGC 检测策略和建议。

二、 问题分析

2.1 问题一的分析

人工智能生成内容（AIGC）涉及到的领域众多且复杂，为了更好地开展并实现对 AIGC 检测的研究，必须选择一个熟悉、易于研究且具有价值的领域进行，因此选择人工智能主题论文的摘要部分作为研究对象。

研究数据直接影响到模型的性能和算法训练的效果，获取丰富且高质量的数据非常重要，因此选择从中国知网获取多个核心期刊上引用量高的论文作为学者撰写的数据，选择多个具有代表性的 AI 大模型并给定生成内容指令作为 AI 生成的数据。

设计的算法应实现低成本和高性能，并契合文本分类的任务，因此将 BERT 与 RCNN 等深度学习算法整合并进行微调，以完成 AIGC 检测任务。

2.2 问题二的分析

目前常用的 AIGC 检测算法无法直接处理文本数据，故必须先进行数据预处理，通过文本分词与去除停用词，文本向量化将文本语言转化为数据，再进行特征筛选与数据降维以降低复杂度与避免过拟合。使用预处理好的数据应用到所选定的 8 种流行的 AIGC 检测算法，求得相应的 F1-score（F1-值）和 Accuracy （准确度）等评估指标，完成对算法的比较。

2.3 问题三的分析

研究用传统方法肉眼识别 AIGC 需要进行深入的文本分析，挖掘学者撰写与 AI 生成文本的不同点。摘要长度与句子数量指标可通过编写 Python 代码进行统计；词汇特征与常用搭配指标可使用高频词统计法排序两类文本的高频词与独有词，再使用 N-gram 方法比较两类文本固定搭配的差异，最后总结出能用肉眼识别的方法。

2.4 问题四的分析

综合上述三个问题的结论，有效进行 AIGC 检测可从算法和人工两方面入手。算法方面要注意数据获取、模型训练与算法优化三个环节，人工方面要注重标准设定、专业培训与质量控制三个部分。

本文的研究过程如下图所示：

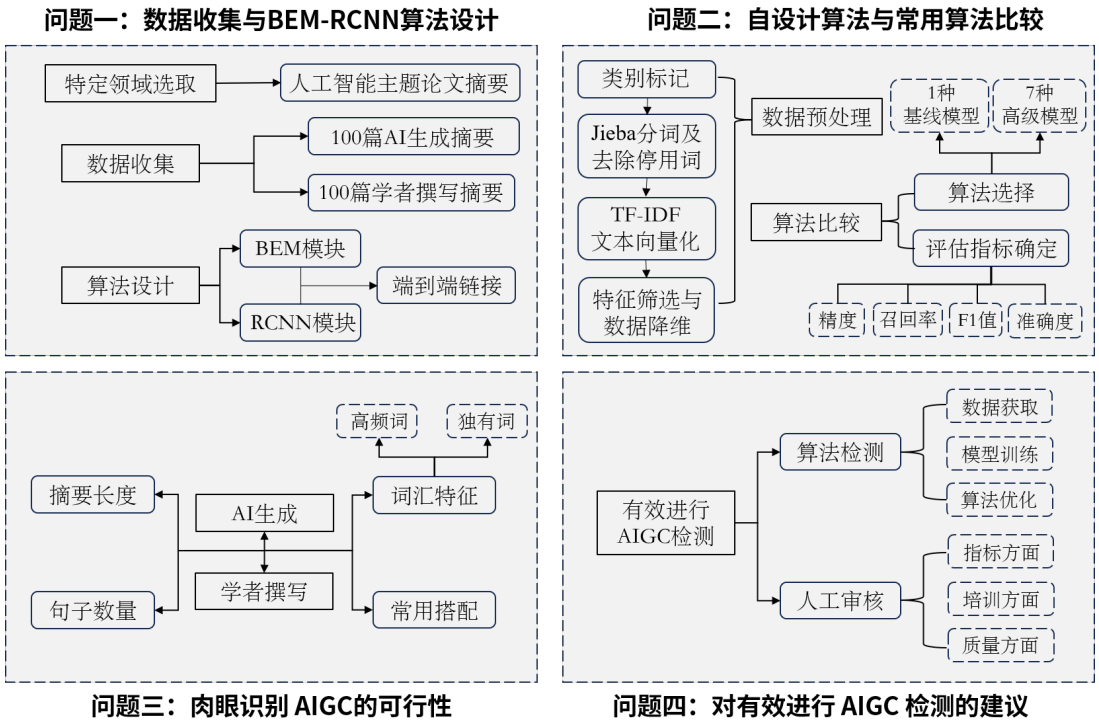

图 **1**　本文的研究过程

三、 模型假设

本文提出以下合理假设：

- **假设 1:** 收集到的高引论文可以作为目前在人工智能领域最专业水平的代表；
- **假设 2:** 生成数据所使用的 AI 可以作为目前 AI 大模型最先进水平的代表；
- **假设 3:** AI 大模型的提示词合理，可以获得理想的人工智能生成的文本数据;
- **假设 4:** 摘要长度与句子数量、词汇特征与常用搭配可以作为通过传统方法肉眼识别 AIGC 的具有代表性的维度。

四、 符号说明

表 **1** 符号说明

符号	含义
$\boldsymbol{x}_i$	输入词
$\boldsymbol{X}_i$	$\boldsymbol{x}_i$ 独热编码表示
$\mathbf{s}_i$	句子 ID
$\mathbf{p}_i$	i 位置的编码
$\mathbf{W}_{\text{token}}$	词嵌入矩阵
$\mathbf{W}_{\text{segment}}$	段落嵌入矩阵
$\mathbf{W}_{\text{pos}}$	位置嵌入矩阵
$\mathbf{H}_i^{(l-1)}$	第 l 层 Transformer 的输入
$\mathbf{q}_i^{(l)}$	第 l 层 Transformer 中词 x_i 的查询（Query）向量
$\mathbf{k}_i^{(l)}$	第 l 层 Transformer 中词 x_i 的键（Key）向量
$\mathbf{v}_i^{(l)}$	第 l 层 Transformer 中词 x_i 的值（Value）向量
T_f	某个关键词在整篇文章中出现的频率
I_{df}	逆文档频率

五、 问题一：数据收集与算法设计

5.1 特定领域的选取

选取“以人工智能为主题的学术论文摘要”作为本次研究的特定领域。该领域具有以下几点优势：

1. 该领域具有大量的公开发表的学术论文，数据获取相对容易，可以作为训练数据集；

2. 人工智能是当前的热点话题，以该主题为研究对象具有更高价值；

3. 将研究对象细化为“摘要部分的 AIGC 自动检测”，简化了研究问题，使模型训练更可行。

5.2 数据的收集

为完成在人工智能领域学术论文摘要的 AIGC 检测，本次研究的数据来源包括以下两部分：

1. 学者撰写的论文摘要

2. 人工智能生成的论文摘要

5.2.1 学者撰写的论文摘要的收集

在中国知网中检索主题为“人工智能”、时间为“2019-2024 年”且同时被北大核心与 CSSCI 收录的专业核心期刊，从中选取《计算机工程与应用》《计算机科学》《智能系统学报》《计算机工程》和《计算机应用研究》等五种期刊作为数据来源。该五种期刊均为人工智能领域影响因子较高的期刊，其收录的论文均可在一定程度上代表该领域的高质量学术论文。将论文按知网被引频次降序排列，筛选出每种期刊 20 篇、共 100 篇论文作为研究样本（数据集 A），研究样本的具体情况如下表所示：

5.2.2 人工智能生成的论文摘要的收集

经过反复测试与对比试验，本研究使用以下提示词作为 AI 模型的输入：

“假设您是一名人工智能领域的专家，请您撰写一篇中文学术论文的摘要部分。我将提供给您所需的论文题目与摘要关键词，您需要根据指定的题目与对应的关键词为我撰写相应的论文摘要。该论文题目为：*XXX*；摘要关键词为：*XXX*。”

基于给定的论文标题、摘要关键词及生成提示词，输入 GPT-4o、Claude、文心一言、通义千问与星火认知等五种截止目前国内外存在的最先进、最强大的 AI 大模型，获取论文摘要的人工智能生成内容（数据集 B），以增强人工智能生成内容的数据来源的广泛性，增加模型的稳定性与结果的可信度。

表 **2**　收集的五种核心期刊对应论文数量及平均被引频次

期刊名称	论文数量	平均被引频次
《计算机工程与应用》	20	173.60
《计算机科学》	20	110.00
《智能系统学报》	20	103.55
《计算机工程》	20	43.75
《计算机应用》	20	39.45

5.3 算法的设计

5.3.1 整体架构

基于 BEM-RCNN 的 AIGC 检测算法的整体架构如下图所示：

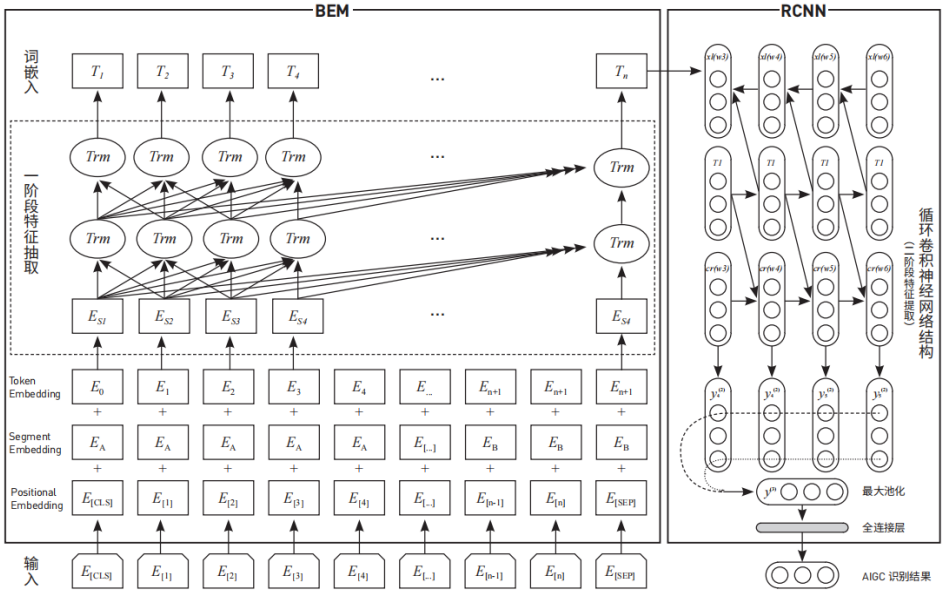

图 **2**　基于 **BEM-RCNN** 的 **AIGC** 检测算法的整体架构

由图可知，首先通过 BEM（BERT-Generated Word Embedding Model）模块提取已收集的数据作为输入并生成词嵌入，然后使用 RCNN（Recurrent Convolutional Neural Network）模块进一步提取词向量中的上下文深层特征，最后通过最大池化层与全连接层进行 AIGC 的检测。其中，为了更好地完成检测任务，本研究采用端到端（End-to-end）结构将 BEM 模块与 RCNN 模块进行连接。

5.3.2 BEM 模块

BERT 生成的词嵌入模型（BEM）是 BEM-RCNN 算法的关键组成部分，负责将收集到的数据作为输入并进行初始特征提取和词嵌入生成，可利用预训练的 BERT 模型并根据收集到的特定数据进行微调，以达到更好的效果。

BEM 模块由包括词嵌入（Token Embedding）、段落嵌入（Segment Embedding）、位置嵌入（Positional Embedding）和 Transformer 层等若干层组成，共同生成包含位置和语义信息的上下文丰富的词嵌入。

1. 词嵌入（Token Embedding）

每个输入词 $\boldsymbol{x}_i$ 首先转化为稠密向量表示的 $\mathrm{E}_{\text{token}}\left(x_i\right)$，并使用预训练的嵌入矩阵：

$$\mathbf{E}_{\text{token}}\left(x_i\right)=\mathbf{W}_{\text{token}}\cdot\mathbf{x}_i \tag{1}$$

其中，$\mathbf{W}_{\text{token}}$ 是词嵌入矩阵，$\boldsymbol{X}_i$ 是 $\boldsymbol{x}_i$ 的独热编码表示。

2. 段落嵌入（Segment Embedding）

为了更好地区分输入中的不同段落，即本研究中每篇摘要中不同的句子，再进行段落嵌入 $\mathbf{E}_{\text{segment}}\left(x_i\right)$：

$$\mathbf{E}_{\text{segment}}\left(x_i\right)=\mathbf{W}_{\text{segment}}\cdot\mathbf{s}_i \tag{2}$$

其中，$\mathbf{W}_{\text{segment}}$ 是段落嵌入矩阵，$\mathbf{s}_i$ 表示句子 ID。

3. 位置嵌入（Positional Embedding）

为了捕获序列中词的位置信息，进行位置嵌入：

$$\mathbf{E}_{\text{pos}}(i)=\mathbf{W}_{\text{pos}}\cdot\mathbf{p}_i \tag{3}$$

其中，$\mathbf{W}_{\text{pos}}$ 是位置嵌入矩阵，$\mathbf{p}_i$ 是 i 位置的编码。

4. 综合嵌入

每个词 $\boldsymbol{x}_i$ 的最终嵌入表示为词嵌入、段落嵌入与位置嵌入的总和，即：

$$\mathbf{E}\left(x_i\right)=\mathbf{E}_{\text{token}}\left(x_i\right)+\mathbf{E}_{\text{segment}}\left(x_i\right)+\mathbf{E}_{\text{pos}}\left(i\right) \tag{4}$$

5. Transformer 层

完成综合嵌入后，通过多层 Transformer（表示为 Trm）进行处理。每个 Transformer 层包含多头自注意力机制和前馈神经网络，共同完成一阶段特征抽取。

单个词 $\boldsymbol{x}_i$ 的单层 Transformer 输出可表示为：

$$\mathbf{H}_i^{(l)}=\mathrm{Trm}^{(l)}\left(\mathbf{H}_i^{(l-1)}\right) \tag{5}$$

其中，$\mathbf{H}_i^{(l-1)}$ 是第 l 层 Transformer 的输入，$\mathbf{H}_i^{(0)}=\mathbf{E}\left(x_i\right)$。

自注意力机制计算每对词的注意力分数 $\alpha_{ij}^{(l)}$，用于加权其他词对词 $\boldsymbol{x}_i$ 表示的贡献:

$$\alpha_{ij}^{(l)} = \frac{\exp\left(\mathbf{q}_i^{(l)} \cdot \mathbf{k}_j^{(l)}\right)}{\sum_{k=1}^{n} \exp\left(\mathbf{q}_i^{(l)} \cdot \mathbf{k}^{(l)}\right)} \tag{6}$$

$$\mathbf{z}_i^{(l)} = \sum_{j=1}^{n} \alpha_{ij}^{(l)} \mathbf{v}_j^{(l)} \tag{7}$$

其中，$\mathbf{q}_i^{(l)}$、$\mathbf{k}_i^{(l)}$ 和 $\mathbf{v}_i^{(l)}$ 分别是第 l 层 Transformer 中词 x_i 的查询（Query）、键（Key）和值（Value）向量。

最终 Transformer 层的输出提供了输入序列中每个词的上下文化嵌入，有效地捕捉了局部和全局上下文信息。这些嵌入随后传递到后续的 RCNN 模块，以进行进一步处理和特征提取。

5.3.3 RCNN 模块

RCNN（Recurrent Convolutional Neural Network）模块是 BEM-RCNN 算法的另一个关键组成部分，可进一步处理来自 BEM 模块的词嵌入，以提取深层上下文特征。RCNN 模块结合了循环神经网络 (RNN) 和卷积神经网络 (CNN) 的优点，不仅能够有效地捕捉长文本的上下文信息，同时解决了 RNN 训练速度慢和 CNN 固定窗口大小的问题。

RCNN 模块由循环神经网络层、卷积层、池化层和全连接层等部分组成，各层协同工作，从 BEM 模块生成的词嵌入中提取深层特征，并进行最终的分类任务。

1. 循环神经网络层（完成二阶段特征提取）

RNN 层用于捕捉输入序列的上下文信息。给定输入序列的词嵌入 H_i，RNN 生成每个时间步的隐藏状态 h_i。公式如下:

$$\mathbf{h}_i = \sigma\left(\mathbf{W}_h \mathbf{h}_{i-1} + \mathbf{W}_x \mathbf{H}_i + \mathbf{b}\right) \tag{8}$$

其中，W_h 和 W_x 是权重矩阵，b 是偏置，σ 是激活函数。

2. 卷积层

卷积层用于从 RNN 生成的隐藏状态中提取局部特征。给定一个窗口大小 k，卷积操作可以表示为:

$$\mathbf{c}_i = f\left(\mathbf{W}_c \cdot \mathbf{h}_{i:i+k-1} + \mathbf{b}_c\right) \tag{9}$$

其中，W_c 是卷积核，b_c 是偏置，f 是非线性激活函数，$\mathbf{h}_{i:i+k-1}$ 表示从 i 到 $i+k-1$ 的隐藏状态序列，

3. 池化层

池化层用于对卷积层输出的特征进行降维。常用的池化方法是最大池化，公式如下:

$$\mathbf{p}_i = \max\left(\mathbf{c}_{i:i+k-1}\right) \tag{10}$$

其中，$\mathbf{C}_{i:i+k-1}$ 表示卷积层输出的特征序列。

4. 全连接层

池化层的输出传递到全连接层，用于最终的特征分类。给定池化层输出 P，全连接层的计算如下:

$$\mathbf{y} = \mathbf{W}_f \cdot \mathbf{p} + \mathbf{b}_f \tag{11}$$

其中，W_f 是全连接层的权重矩阵，b_f 是偏置。

RCNN 模块将上述各层集成在一起，从 BEM 模块生成的词嵌入中提取深层特征，并完成最终的 AIGC 检测任务。

5.3.4 端到端的连接

BEM-RCNN 算法采用了端到端的连接结构，使模型的各个部分可以共同优化，从而提升该算法的整体性能。端到端连接的设计允许 BEM 模块生成的词嵌入直接输入到 RCNN 模块，进行进一步的特征提取和分类，确保了在训练过程中 BEM 和 RCNN 模块的参数能够协同调整，以最佳方式完成我们的 AIGC 检测任务。

其突出优势在于：

1. BEM 模块和 RCNN 模块的参数在训练过程中可以同时更新，允许在大规模语料库和收集到的人工智能领域特有数据集上同时学习，能在获得一般语言特征的同时针对具体研究问题进行微调，极大地避免了过拟合与欠拟合风险。

2. 减少了独立模块之间的数据传输和处理误差，从而提高了特征提取和分类的精度。

3. 端到端的连接不需要中间步骤的独立优化和处理，简化了训练和测试流程，使算法的整体架构更加清晰高效。

5.4 评估指标的确定

本研究选择了 Precision（精度）、Recall（召回率）、F1-score（F1-值）和 Accuracy（准确度）等四个指标来评估各种算法模型的性能并加以比较。这些指标被广泛用于评估机器学习和深度学习模型的性能，公式如下所示：

$$\text{Precision} = \frac{TP}{TP + FP} \tag{12}$$

$$\text{Recall} = \frac{TP}{P} \tag{13}$$

$$\text{F1-score} = \frac{2\ \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \tag{14}$$

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{15}$$

其中，TP 表示将正样本预测正确的数值，TN 表示将负样本预测正确的数值，FP 表示将负样本错误的预测成了正样本的数量，FN 表示将正样本错误的预测为负样本的数值。

计算 BEM-RCNN 算法在该四个指标上的数值，得出 Precision = 0.97，Recall = 0.96，F1-score = 0.96，Accuracy = 0.96。

六、 问题二：自设计算法与常用算法比较

本文选择了当前常用的几个 AIGC 检测算法，使用经过预处理的数据进行测试，通过与自设计的基于 BEM-RCNN 的 AIGC 检测算法进行比较，以得出结论。

6.1 数据预处理

6.1.1 类别标记

本文研究的“在人工智能领域学术论文摘要的 AIGC 检测”可转化为二分类问题。对收集到的数据分为两类，并做如下处理：合并数据集 A 与数据集 B 为数据集 X，并分别用类标号“1”标记学者撰写内容，类标号“0”标记人工智能生成内容。生成的数据集 X 共 200 行 2 列，其中行用于独立存储每个摘要，列包括摘要列和对应的标签列。

6.1.2 分词及去除停用词

1. 为减少模型的噪音并提高计算效率，将标点符号与无实际意义的词汇纳入到停用词表。

2. 由于中文文本中没有显式的词边界，因此需要将连续的文本切分成独立的词语。使用 Python 中的 Jieba 中文分词库，利用基于前缀词典的最大概率路径，进行文本分词处理。

6.1.3 文本向量化

选择 TF-IDF（Term Frequency-Invers Document Frequency，词频-逆文档频率）作为文本向量化方法，将具有高度模糊性的文本语言转化为计算机可以理解的数据。TF-IDF 是一种用于信息处理和数据挖掘的加权技术，可根据词组在文本中出现的次数和在整个语料中出现的文档频率来计算一个词组在整个语料中的重要程度。其中，TF（Term Frequency）表示某个关键词在整篇文章中出现的频率，IDF（InversDocument Frequency）表示逆文档频率，计算公式如下所示：

$$T_f(\text{ 词频}) = \frac{n_{i,j}(\text{ 某个词组}i\text{ 在文章}j\text{ 中出现的次数})}{\sum_{z=1}^{z=k} n_{z,j}\left(\text{第篇文章的所有词组}k\text{ 出现的次数}\right)} \tag{16}$$

$$I_{df}(\text{ 逆文档频率}) = \log \frac{|D|\text{ (语料库中文章总数)}}{1+|D_{t_t}|\left(1+\text{ 文章中包含特征词}t_i\text{ 的数量}\right)} \tag{17}$$

$$T_{f-idf} = T_f \times I_{df} \tag{18}$$

6.1.4 特征筛选与数据降维

在文本向量化后，获得了 200 行 2191 列数据。为降低模型的复杂度、避免引入过多的噪声数据、减小发生过拟合的风险，以达到更佳的分类算法效果，因此对数据进行进一步特征筛选与数据降维处理。

1. 特征筛选

特征筛选是选择原始特征集中的一个子集，保留最重要的特征，以去除冗余或无关的特征，提高模型的性能和计算效率。

在本研究中，使用卡方检验评分函数，来评估每个特征的重要性，并选择得分最高的前 1000 个特征。

卡方检验用于检测两个分类变量之间的关联程度，其基本原理如下：

（1）计算观测频率和期望频率

对于每个特征 i 和类别 j，计算观测频率 θ 和期望频率 $E_(ij)$。

（2）计算卡方统计量

$$\chi^2 = \sum \frac{(O_{ij} - E_{ij})^2}{E_{ij}} \tag{19}$$

其中，$O_(ij)$ 是观测频率，$E_(ij)$ 是期望频率。

（3）评估特征重要性

根据卡方统计量的大小来评估每个特征的重要性，卡方统计量越大，说明特征与类别之间的关联性越强。

2. 数据降维

数据降维是将原始特征集映射到一个低维空间，创建新的特征。该方法不直接删除特征，而是通过数学变换降低数据的维度。

在本研究中，使用 Truncated SVD（又称为潜在语义分析，LSA）指定 $n_{components} = 100$ 进行数据降维操作，降维后得到 200 行 100 列数据。

Truncated SVD 的基本原理如下：

（1）奇异值分解（SVD）

对于一个矩阵 A(m × n)，可以分解为三个矩阵的乘积:

$$\mathbf{A} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T \tag{20}$$

其中，U 是 m × m 的正交矩阵；Σ 是 m × n 的对角矩阵，包含 A 的奇异值；V 是 n × n 的正交矩阵。

（2）截断奇异值分解（Truncated SVD）

通过只保留前 k 个奇异值和对应的奇异向量，对 A 进行近似:

$$\mathbf{A_k} \approx \mathbf{U_k}\boldsymbol{\Sigma_k}\mathbf{V_k}^T \tag{21}$$

其中，U_k、Σ_k、V_k 分别是 U、Σ、V 的前 k 列。

（3）潜在语义分析（LSA）

在文本数据处理中，使用 Truncated SVD 对词-文档矩阵进行降维，捕捉潜在的语义结构。通过将高维空间中的文档和词语映射到低维潜在语义空间中，可以更好地发现词语和文档之间的语义关系。

6.2 算法比较

目前，许多流行算法在文本分类任务上表现出色。针对人工智能领域学术论文摘要的 AIGC 自动检测任务，本研究选择 8 种当前常用的 AIGC 检测算法与自设计的基于 BEM-RCNN 的 AIGC 检测算法进行比较。

6.2.1 算法选择

在所选的八种 AIGC 检测算法中，1 种为基线模型：朴素贝叶斯（Naive Bayes）；7 种为高级模型：逻辑回归（Logistic Regression）、随机森林（Random Forest）、支持向量机（SVM），K 近邻分类法（KNN），轻量梯度提升机（LightGBM），极端梯度提升（XGBoost）与自适应增强（AdaBoost）。

其中，特别考虑了作为基线模型的朴素贝叶斯算法。朴素贝叶斯分类器是一种基于概率论的监督学习方法，使用贝叶斯定理并假设特征之间相互独立。其基本原理如下：

（1）贝叶斯定理

贝叶斯定理用于计算条件概率，即在已知某些条件下事件发生的概率。贝叶斯定理的一般形式为：

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)} \tag{22}$$

其中，$P(A \mid B)$ 是在事件 B 发生的条件下 A 发生的概率（后验概率），$P(B \mid A)$ 是在事件 A 发生的条件下 B 发生的概率（似然度），$P(A)$ 是事件 A 发生的概率（先验概率），$P(B)$ 是事件 B 发生的概率（证据）。

（2）朴素贝叶斯分类器

对于分类问题，根据特征向量 $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ 来预测类别 C_k，其中 kk 表示类别标签。朴素贝叶斯分类器通过最大化以下条件概率来做出决策：

$$P(C_k \mid \mathbf{x}) = \frac{P(\mathbf{x} \mid C_k)\, P(C_k)}{P(\mathbf{x})} \tag{23}$$

由于分母 $P(\mathbf{x})$ 对于所有类别都是相同的，并且只起到规范化的作用，因此可以忽略它来进行最大化的比较：

$$\arg\max_k P(C_k \mid \mathbf{x}) = \arg\max_k P(\mathbf{x} \mid C_k)\, P(C_k) \tag{24}$$

由于朴素贝叶斯假设特征之间相互独立，可以将联合概率分解为各个特征的独立概率：

$$P(\mathbf{x} \mid C_k) = \prod_{i=1}^{n} P(x_i \mid C_k) \tag{25}$$

因此，最终的分类决策可以表示为：

$$\hat{C} = \arg\max_k \left(\prod_{i=1}^{n} P(x_i \mid C_k) \right) P(C_k) \tag{26}$$

6.2.2 评估指标的计算

对自设计的 BEM-RCNN 模型和选定的八个模型分别计算问题一中选择的 Precision（精度）、Recall（召回率）、F1-score（F1-值）和 Accuracy（准确度）等四个指标的具体数值，来评估各种算法模型的性能并加以比较，具体结果如下表所示：

由表可以看出，自设计的 BEM-RCNN 算法在所有指标上均表现出色，Precision、Recall、F1-score 和 Accuracy 分别达到了 0.97、0.96、0.96 和 0.96，显示了该算法在处理该数据集时的卓越性能。

相比之下，作为基线算法的朴素贝叶斯（Naive Bayes）在所有指标上的得分均最低，Precision、Recall、F1-score 和 Accuracy 分别为 0.68、0.68、0.67 和 0.68，表明其在该数据集上的表现最差。

对于其他算法，逻辑回归（Logistic Regression）和随机森林（Random Forest）在所有指标上都表现出色，Precision、Recall、F1-score 和 Accuracy 均为 0.95，接近 BEM-RCNN 算法的性能。支持向量机（SVM）和极端梯度提升（XGBoost）的表现次之，各项指标均在 0.92 以上。K 近邻分类法（KNN）的表现略显逊色，Precision、Recall、F1-score 和 Accuracy 分别为 0.88、0.85、0.85 和 0.85。轻量梯度提升机（LightGBM）和自适应增强（AdaBoost）同样也表现良好，各项指标均在 0.90 以上。

表 **3**　算法各指标比较结果

算法	Precision	Recall	F1-score	Accuracy
BEM-RCNN	**0.97**	**0.96**	**0.96**	**0.96**
朴素贝叶斯（Naive Bayes）	0.68	0.68	0.67	0.68
逻辑回归（Logistic Regression）	0.95	0.95	0.95	0.95
支持向量机（SVM）	0.93	0.93	0.92	0.93
随机森林（Random Forest）	0.95	0.95	0.95	0.95
K 近邻分类法（KNN）	0.88	0.85	0.85	0.85
轻量梯度提升机（LightGBM）	0.92	0.89	0.90	0.90
极端梯度提升（XGBoost）	0.93	0.93	0.92	0.93
自适应增强（AdaBoost）	0.93	0.93	0.92	0.93

七、 问题三：肉眼识别 AIGC 的可行性

7.1 摘要长度与句子数量

摘要长度指一篇完整摘要的中文字符个数。通过编写 Python 代码分别对人工智能生成的摘要和学者撰写的摘要长度进行统计，并绘制了正态分布拟合直方图如下所示:

统计结果显示，人工智能生成的摘要长度均值与标准差分别为：217.00 个中文字符与 29.05 个中文字符；而学者撰写的摘要长度均值与标准差分别为：279.93 个中文字符与 95.60 个中文字符。

进一步，绘制 AI 生成与学者撰写摘要长度的箱线图如下所示：

由图可知，AI 生成的摘要长度最大为 381 个中文字符，最小为 135 个中文字符，极差为 246 个中文字符；而学者撰写的摘要长度最大为 659 个中文字符，最小为 21 个中文字符，极差为 638 个中文字符。

通过对比分析可知，学者撰写的摘要普遍比 AI 生成的摘要更长，且长度的分布相对分散。这说明学者撰写摘要的异质性程度更高，可能是由于不同作者的学术背景和写作风格有差异所致，而 AI 大模型的输出会出现高度的同质化，故其写作风格更加相近。

分析结果表明，从肉眼上看，长度相对较长、特别是极长或极短的摘要大概率由学

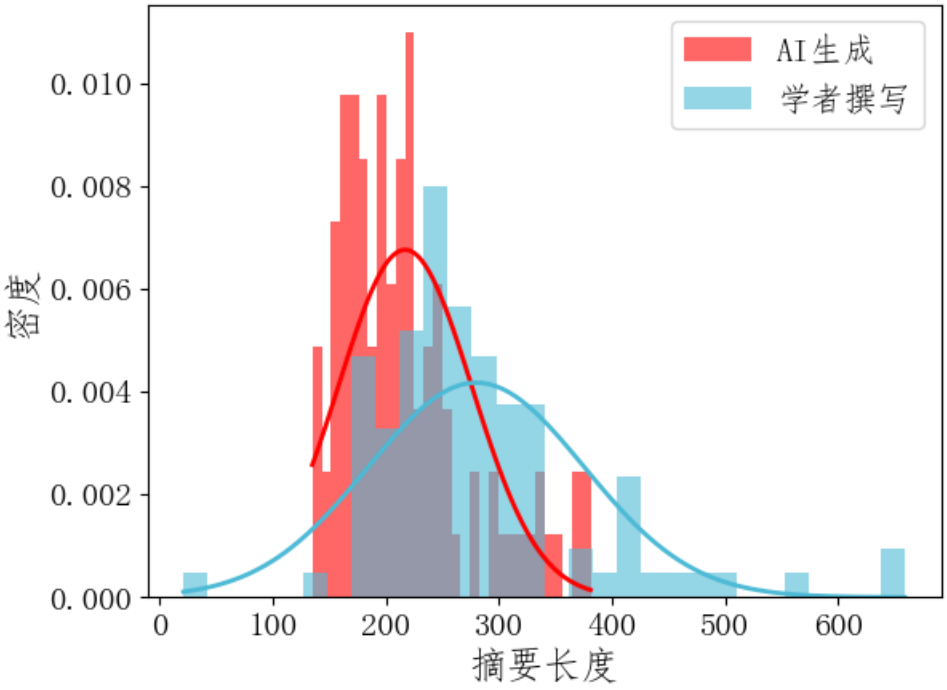

图 **3**　摘要长度的正态分布拟合直方图

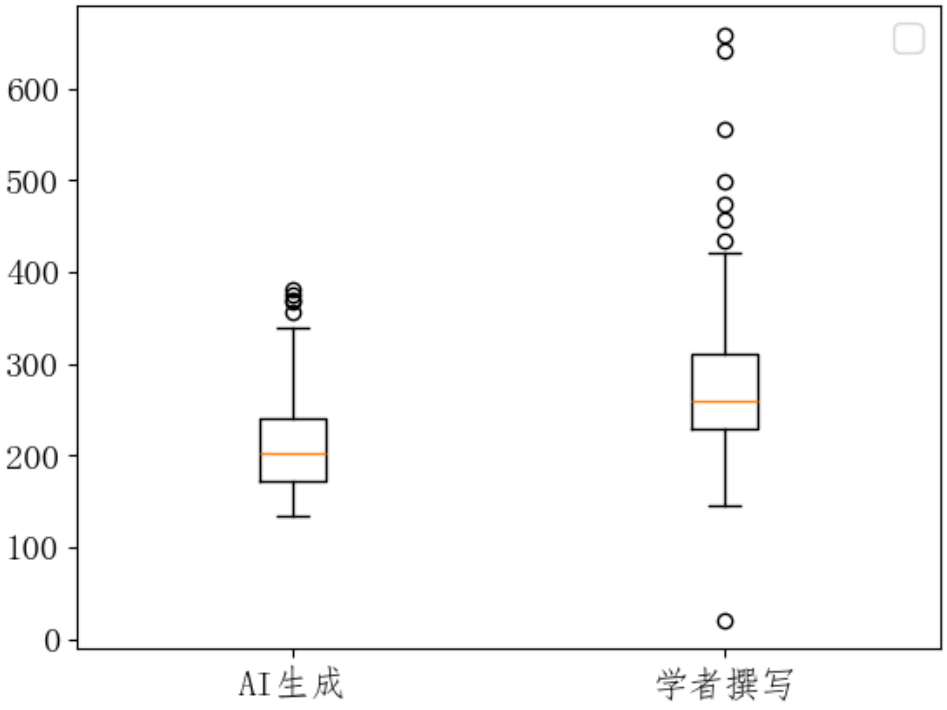

图 **4**　摘要长度的箱线图

者撰写。

接下来，对摘要中的句子（以句号为分隔单位）的数量进行统计分析，结果如下图所示：

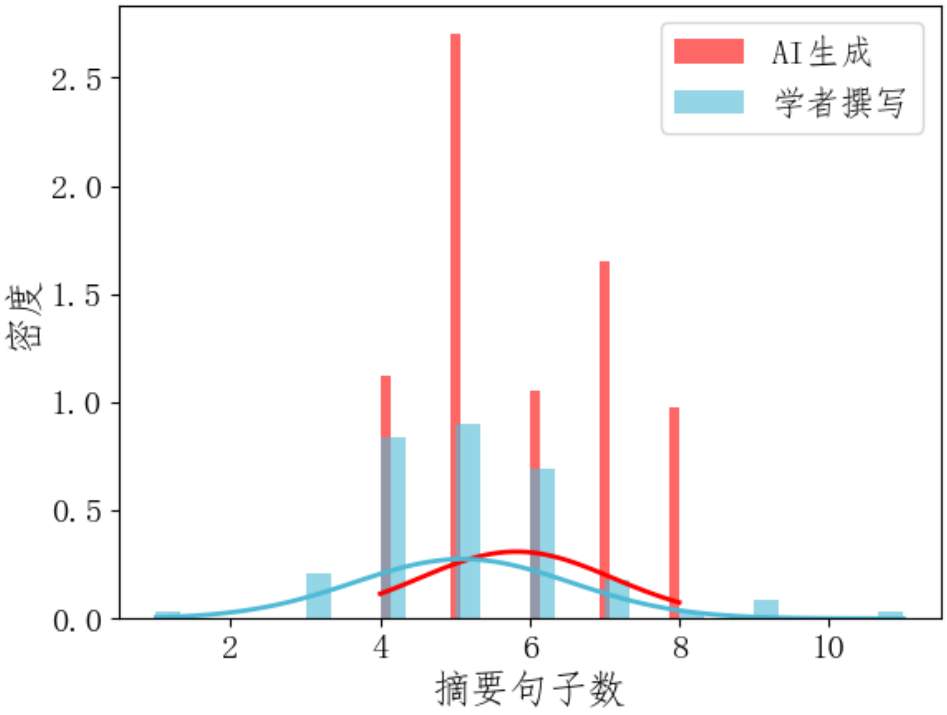

图 **5**　摘要句子数量的正态分布拟合直方图

结果显示，AI 生成的摘要句子数量通常为 4-8 个，且以 5 和 7 个居多，其偏态为 0.28，峰度为-1.14。相比之下，学者撰写的摘要句子数量通常为 3-9 个，且以 4-6 个居多，其偏态为 0.96，峰度为 2.68。由此可知，学者撰写的摘要相较于 AI 生成的摘要，句子数量的分布更加不对称，且更容易出现句子数量极多或极少的情况，分布较分散。

进一步，通过绘制 AI 生成与学者撰写摘要句子数量的箱线图如下所示：

由图可知，AI 生成的摘要句子数量最多为 8 个，最少为 4 个，极差为 4；而学者撰写的摘要句子数量最多为 11 个，最少为 1 个句子，极差为 10。这同样说明，学者写作异质性程度高，且当论文摘要中句子数量极多（大于 8 句）或者极少（小于 4 句），大概率是学者撰写而非 AI 生成的。

7.2 词汇特征与常用搭配

文本中出现次数较多的词汇往往能够反映出该文本主题的核心内容或观点。利用预处理后的数据，使用高频词统计法，对 AI 生成和学者撰写摘要的高频词与独有词进行统计，并按照出现频率进行排序。结果如下表所示：

由表可知，在排名前 15 的高频词中，尽管相对排名略有差异，但是有 12 个词（占比 80%）是 AI 生成和学者撰写论文中共同出现的高频词，包括“学习”“方法”“深度”

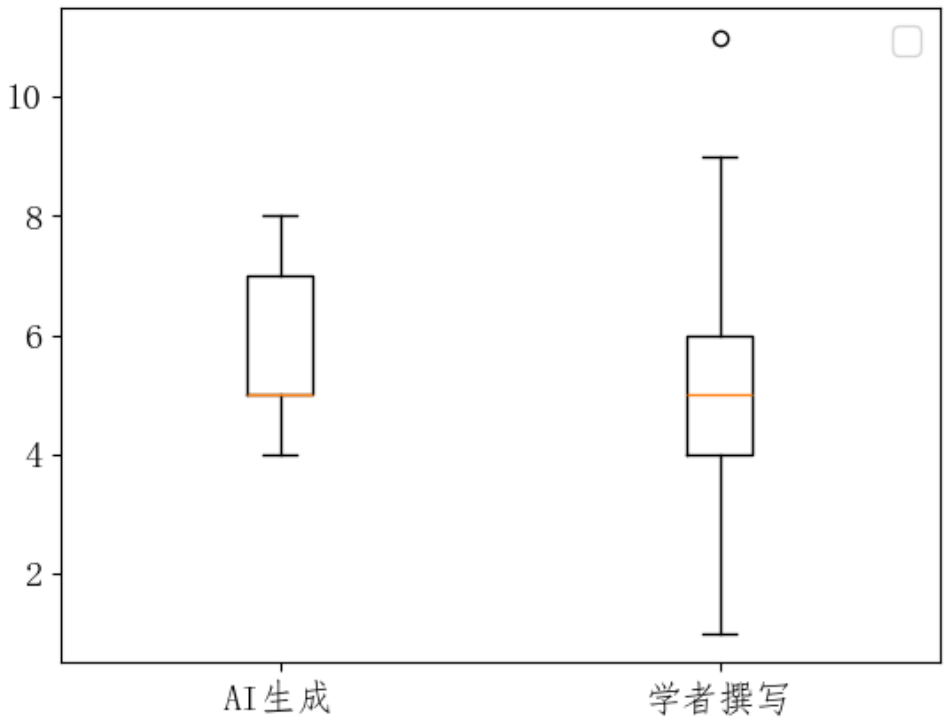

图 **6**　摘要句子数量的箱线图

等。此外，剩下的 3 个词（占比 20%，表中粗体显示）存在差异，其中 AI 生成的高频词为“本文”“方向”“未来”等在论文中常见的泛化表达，而学者撰写的高频词为“数据”“算子”等人工智能领域经常研究讨论的主题。

对独有词进行分析，发现 AI 生成的独有词如“系统阐释”“层次化”“技术手段”等用法更正式，含义较空洞；而学者撰写的独有词如“代表性”“分为”“角度”等更加日常化与口语化。

综上，在肉眼识别 AIGC 时，由于 AI 生成与学者撰写摘要的高频词高度重合，因此从该角度出发不易辨别，但是仍然可能从文本用词的风格上得以分辨。当摘要用词生硬晦涩时，大概率是由 AI 生成；反之，若用词大都常见且易于理解，则大概率为学者撰写。

接下来，使用 N-gram 比较 AI 生成和学者撰写的摘要中固定搭配的差异。N-gram 指的是语料库或文本中连续出现的由 n 个词组成的的序列。N-gram 模型可以产生自然语言序列，即可通过给定序列的出现概率预测下一个序列的出现概率。

使用 N-gram（$2 \leq N \leq 10$）的比较结果如下表所示：

由表可知，从统计结果来看，排名前 15 的搭配中，有 9 个（占比 60%）是相同的搭配，如“深度学习”“目标检测”“计算机视觉”“卷积神经网络”等，均为人工智能领域的专业术语。而剩下 6 个（占比 40%）是不同的搭配，其中 AI 生成的常用搭配更倾向于使用归纳总结型的学术话语体系，更加格式化，如“本文综述”“本文研究”“本文提出”“最后本文”等；而学者撰写的常用搭配往往着重强调实际含义，如“检测算

表 **4**　**AI** 生成与学者撰写摘要的高频词和独有词

序号	AI 生成			学者撰写		
	高频词	词频	独有词	高频词	词频	独有词
1	**本文**	**244**	揭示	学习	218	代表性
2	方法	168	涵盖	方法	170	归纳
3	应用	156	系列	深度	159	文中
4	学习	152	最新进展	**进行**	**149**	目前
5	深度	115	交通系统	算法	139	分为
6	研究	107	表现出色	目标	137	角度
7	领域	88	系统阐述	研究	137	问题
8	算法	84	现代	检测	134	梳理
9	模型	83	局限	模型	102	节点
10	检测	82	层次化	领域	95	相机
11	技术	78	高精度	**数据**	**89**	科学
12	**方向**	**73**	优异	技术	86	准则
13	目标	72	新兴	**算子**	**83**	途径
14	**未来**	**71**	技术手段	应用	81	统一
15	分析	69	联邦	分析	75	人工

法”“智能体”“路径规划”等。

因此，在肉眼识别两类文本时，由于使用专业术语的频率均较高且规范，因此从该角度出发不好区分，但显然学者撰写所使用的常用搭配比 AI 生成内容中的搭配更加具有实际意义。

表 **5**　**AI** 生成与学者撰写摘要内容中的常用搭配

序号	AI 生成		学者撰写	
	常用搭配	频次	常用搭配	频次
1	深度学习	91	深度学习	106
2	目标检测	55	目标检测	92
3	机器学习	42	强化学习	50
4	**本文综述**	**41**	计算机视觉	38
5	实际应用	35	**检测算法**	**28**
6	强化学习	33	**目标检测算法**	**27**
7	**未来研究**	**30**	卷积神经网络	26
8	**最后本文**	**30**	智能体	26
9	研究方向	29	机器学习	24
10	**实际应用中**	**29**	提出一种	20
11	卷积神经网络	28	**路径规划**	**19**
12	**本文提出**	**28**	**检测方法**	**19**
13	计算机视觉	26	研究方向	18
14	发展方向	25	**进行分析**	**18**
15	提出一种	24	发展方向	17

八、 问题四：对有效进行 AIGC 检测的建议

8.1 算法检测

1. 数据获取方面：用于训练模型的数据的数量与质量对算法检测的影响极大。因此，要广泛地收集大量高质量数据，同时确保后续预处理过程的规范性，以保证数据的高质量。

2. 模型训练方面：要选择合适的模型，并结合多种不同的模型和算法来增加准确性与覆盖面。训练过程必须规范合理，既要避免过拟合而导致误将人工撰写识别为 AI 生

成，也要避免欠拟合而导致无法准确识别 AI 生成的内容。

3. 算法优化方面：使用如准确率、召回率、F1 分数等多个指标交叉检验模型的性能，根据评估结果对模型进行调优，并通过特征工程、调整参数、集成设计等方面来进行算法的优化。

8.2 人工审核

1. 要制定清晰具体的审核指标与标准，保证人工审核时的一致性与公平性。

2. 要对审核人员提前进行专业的培训，如熟悉审核标准、广泛阅读人工撰写和 AI 生成内容的差异点，以便更好地完成检测任务。

3. 由于人眼识别准确率不高且具备一定主观性，因此要实施严格的审核质量控制措施，及时对检测结果进行检查、反馈与改进，以避免误伤与漏放。

九、 优缺点分析

9.1 优点

1. 收集到的数据质量高，包括专门筛选的核心期刊的高引论文作为学者撰写摘要的数据，以及专门设置提示词的多个最先进大模型的生成结果作为 AI 生成摘要的数据。

2. 数据预处理工作全面且规范，包括数据标注、分词与去除停用词、TF-IDF 向量化以及特征筛选与降维四个步骤，每一步均使用合适且最优的处理方法。

3. 自设计的基于 BEM-RCNN 的 AIGC 检测算法集合了多种当下最先进的深度学习算法的优势，同时采用端到端连接的设计思路，提高了算法的性能，降低了模型训练的成本。

4. 摘要长度、句子数量、词汇特征以及常用搭配等指标覆盖了肉眼识别文本可用的的绝大多数特征，且对数据文本的分析深入全面。

9.2 缺点

1. 本研究仅以人工智能领域的高引论文的摘要部分作为研究样本，对于其它学科领域以及论文中的其它部分暂未涉及，因此结论的普适性仍需进一步研究验证，后续还应对不同学科领域以及论文中的其它部分进行进一步研究。

2. 收集数据时仅获取高引论文 100 篇，数据量较少，因此自设计算法的普适性与稳健性较低。后续应更广泛地收集高质量数据，在更大的样本集上训练与检测模型，以提高模型的泛化能力。

参考文献

[1] 王一博, 郭鑫, 刘智锋, 等.AI 生成与学者撰写中文论文摘要的检测与差异性比较研究 [J]. 情报杂志,2023,42(09):127-134.

[2] 邓胜利, 汪璠, 王浩伟. 在线社区中人工智能生成内容的识别方法研究 [J]. 图书情报知识,2024,41(02):28-38+149.DOI:10.13366/j.dik.2024.02.028.

[3] 潘雪峰, 王超. 学者撰写与 AI 生成内容的差异性与识别研究——以图书馆健康服务研究领域为例 [J]. 图书情报导刊,2024,9(03):54-60.

[4] https://github.com/goto456/stopwords

附录

表 6　支撑材料文件列表

文件列表名	
问题一	学者撰写的论文摘要.xlsx
	AI 生成的论文摘要.xlsx
	带标签的论文摘要.xlsx
	stopwords.txt
	BEM-RCNN 算法.ipynb
	特征储存.npy
	标签储存.npy
问题二	数据预处理.ipynb
	常用的 AIGC 检测算法.ipynb
问题三	摘要长度与句子数量分析.ipynb
	词汇特征与常用搭配分析.ipynb

附录 A BEM-RCNN 算法

```
import pandas as pd
import jieba
from collections import Counter

scholar_data = pd.read_excel('学者撰写的论文摘要.xlsx')
ai_data = pd.read_excel('AI生成的论文摘要.xlsx')

scholar_abstracts = scholar_data['论文摘要']
ai_abstracts = ai_data['论文摘要']

# 加载停用词表
def load_stopwords(filepath):
with open(filepath, 'r', encoding='utf-8') as file:
stopwords = set(file.read().strip().split('\n'))
return stopwords
```

```
stopwords = load_stopwords('stopwords.txt')

# 分词并去除停用词
def jieba_tokenizer(text):
words = jieba.cut(text)
return ' '.join([word for word in words if word not in stopwords])

scholar_abstracts = scholar_abstracts.apply(jieba_tokenizer)
ai_abstracts = ai_abstracts.apply(jieba_tokenizer)

import torch
import torch.nn as nn
from transformers import BertModel, BertTokenizer

class BEM(nn.Module):
def __init__(self, pretrained_model_name='bert-base-uncased'):
super(BEM, self).__init__()
self.bert = BertModel.from_pretrained(pretrained_model_name)

def forward(self, input_ids, attention_mask, token_type_ids):
outputs = self.bert(input_ids=input_ids,
attention_mask=attention_mask,
token_type_ids=token_type_ids)
last_hidden_state = outputs.last_hidden_state
return last_hidden_state

class RCNN(nn.Module):
def __init__(self, embed_size, hidden_size, num_layers, num_classes, dropout=0.5):
super(RCNN, self).__init__()
self.rnn = nn.LSTM(embed_size, hidden_size, num_layers, batch_first=True,
    bidirectional=True, dropout=dropout)
self.conv = nn.Conv1d(in_channels=hidden_size*2, out_channels=hidden_size*2, kernel_size=3,
    padding=1)
self.max_pool = nn.MaxPool1d(kernel_size=2)
self.fc = nn.Linear(hidden_size*2, num_classes)

def forward(self, x):
# x: [batch_size, seq_length, embed_size]
h_rnn, _ = self.rnn(x)
# h_rnn: [batch_size, seq_length, hidden_size*2]
h_rnn = h_rnn.permute(0, 2, 1)
# h_rnn: [batch_size, hidden_size*2, seq_length]
h_conv = torch.relu(self.conv(h_rnn))
# h_conv: [batch_size, hidden_size*2, seq_length]
h_pool = self.max_pool(h_conv)
# h_pool: [batch_size, hidden_size*2, seq_length//2]
```

```
h_pool = h_pool.permute(0, 2, 1)
# h_pool: [batch_size, seq_length//2, hidden_size*2]
h_pool = torch.mean(h_pool, dim=1)
# h_pool: [batch_size, hidden_size*2]
output = self.fc(h_pool)
# output: [batch_size, num_classes]
return output

embed_size = 768
hidden_size = 256
num_layers = 2
num_classes = 10
rcnn_model = RCNN(embed_size, hidden_size, num_layers, num_classes)

batch_size = 2
seq_length = 20
dummy_embeddings = torch.randn(batch_size, seq_length, embed_size)

output = rcnn_model(dummy_embeddings)

class BEM_RCNN(nn.Module):
def __init__(self, pretrained_model_name='bert-base-uncased', hidden_size=256, num_layers=2,
    num_classes=10, dropout=0.5):
super(BEM_RCNN, self).__init__()
self.bem = BEM(pretrained_model_name)
self.rcnn = RCNN(embed_size=768, hidden_size=hidden_size, num_layers=num_layers,
    num_classes=num_classes, dropout=dropout)

def forward(self, input_ids, attention_mask, token_type_ids):
embeddings = self.bem(input_ids, attention_mask, token_type_ids)
output = self.rcnn(embeddings)
return output

bwem_rcnn_model = BEM_RCNN()

input_ids = torch.randint(0, 1000, (batch_size, seq_length))
attention_mask = torch.ones(batch_size, seq_length)
token_type_ids = torch.zeros(batch_size, seq_length)

import torch.optim as optim

num_epochs = 10
learning_rate = 1e-4

model = BEM_RCNN()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
```

```
for epoch in range(num_epochs):
model.train()
optimizer.zero_grad()

outputs = model(input_ids, attention_mask, token_type_ids)
labels = torch.tensor([0, 1])
loss = criterion(outputs, labels)

loss.backward()
optimizer.step()

print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}")

torch.save(model.state_dict(), 'bem_rcnn_model.pth')
```

附录 B 数据预处理

```
import pandas as pd
import jieba

data = pd.read_excel('带标签的论文摘要.xlsx')

# 加载停用词表
def load_stopwords(filepath):
with open(filepath, 'r', encoding='utf-8') as file:
stopwords = set(file.read().strip().split('\n'))
return stopwords

stopwords = load_stopwords('stopwords.txt')

# 分词并去除停用词
def jieba_tokenizer(text):
words = jieba.cut(text)
return ' '.join([word for word in words if word not in stopwords])

data['tokenized_abstract'] = data['论文摘要'].apply(jieba_tokenizer)

print(data['tokenized_abstract'].head())

# 使用TF-IDF向量化
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(data['tokenized_abstract'])
```

```
y = data['标签']

print(X.shape, y.shape)

from sklearn.feature_selection import SelectKBest, chi2

k = 1000
selector = SelectKBest(chi2, k=k)
X_kbest = selector.fit_transform(X, y)

print(X_kbest.shape)

from sklearn.decomposition import TruncatedSVD

n_components = 100
svd = TruncatedSVD(n_components=n_components)
X_reduced = svd.fit_transform(X_kbest)

print(X_reduced.shape)

# 保存处理后的数据
import numpy as np

np.save('X_reduced.npy', X_reduced)
np.save('y.npy', y.values)
```

附录 C 常用的 AIGC 检测算法

```
import numpy as np

X_reduced = np.load('X_reduced.npy')
y = np.load('y.npy')

print(X_reduced.shape)
print(y.shape)

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_reduced, y, test_size=0.2,
    random_state=42)

from sklearn.metrics import classification_report, accuracy_score

# Logistic Regression
from sklearn.linear_model import LogisticRegression
```

```
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)
y_pred = log_reg.predict(X_test)

print('Logistic Regression')
print('Accuracy:', accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

# SVM
from sklearn.svm import SVC

svm = SVC()
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)

print('SVM')
print('Accuracy:', accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

# Random Forest
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier()
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)

print('Random Forest')
print('Accuracy:', accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

# Naive Bayes
from sklearn.naive_bayes import GaussianNB

nb = GaussianNB()
nb.fit(X_train, y_train)
y_pred = nb.predict(X_test)

print('Naive Bayes')
print('Accuracy:', accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

# KNN
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
```

```
y_pred = knn.predict(X_test)

print('KNN')
print('Accuracy:', accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

# LightGBM
import lightgbm as lgb

lgb_train = lgb.Dataset(X_train, y_train)
lgb_test = lgb.Dataset(X_test, y_test)

params = {
   'objective': 'binary',
   'metric': 'binary_error',
   'verbosity': -1
}

lgb_model = lgb.train(params, lgb_train, valid_sets=lgb_test, num_boost_round=1000)

y_pred = lgb_model.predict(X_test, num_iteration=lgb_model.best_iteration)
y_pred = np.round(y_pred)

print('LightGBM')
print('Accuracy:', accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

# XGBoost
import xgboost as xgb

xgb_train = xgb.DMatrix(X_train, label=y_train)
xgb_test = xgb.DMatrix(X_test, label=y_test)

params = {
   'objective': 'binary:logistic',
   'eval_metric': 'error'
}

xgb_model = xgb.train(params, xgb_train, num_boost_round=1000, evals=[(xgb_test, 'test')])
y_pred = xgb_model.predict(xgb_test)
y_pred = np.round(y_pred)

print('XGBoost')
print('Accuracy:', accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

# AdaBoost
```

```
from sklearn.ensemble import AdaBoostClassifier

ada = AdaBoostClassifier()
ada.fit(X_train, y_train)
y_pred = ada.predict(X_test)

print('AdaBoost')
print('Accuracy:', accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

附录 D 摘要长度与句子数量分析

```
import pandas as pd

scholar_data = pd.read_excel('学者撰写的论文摘要.xlsx')
ai_data = pd.read_excel('AI生成的论文摘要.xlsx')

scholar_abstracts = scholar_data['论文摘要']
ai_abstracts = ai_data['论文摘要']

def calculate_abstract_lengths(abstracts):
lengths = [len(str(abstract)) for abstract in abstracts]
return lengths

scholar_lengths = calculate_abstract_lengths(scholar_abstracts)
ai_lengths = calculate_abstract_lengths(ai_abstracts)

import matplotlib.pyplot as plt
import numpy as np
import scipy.stats as norm
import matplotlib

matplotlib.rc("font",family='FangSong')

# 绘制直方图
plt.hist(ai_lengths, bins=30, density=True, alpha=0.6, color='r', label='AI生成')
plt.hist(scholar_lengths, bins=30, density=True, alpha=0.6, color='#4DBAD6',
    label='学者撰写')

# 拟合正态分布曲线并绘制
mu, sigma = norm.norm.fit(ai_lengths)
x = np.linspace(min(ai_lengths), max(ai_lengths), 100)
p = norm.norm.pdf(x, mu, sigma)
plt.plot(x, p, 'r', linewidth=2)
```

```
mu, sigma = norm.norm.fit(scholar_lengths)
x = np.linspace(min(scholar_lengths), max(scholar_lengths), 100)
p = norm.norm.pdf(x, mu, sigma)
plt.plot(x, p, '#4DBAD6', linewidth=2)


# 添加标签和图例
plt.xlabel('摘要长度', fontsize = 17)
plt.ylabel('密度', fontsize = 17)
plt.legend(fontsize = 15)
plt.xticks(fontsize = 15)
plt.yticks(fontsize = 15)

# 显示图形
plt.show()
```

附录 E 词汇特征与常用搭配分析

```
import pandas as pd
import jieba
from collections import Counter

scholar_data = pd.read_excel('学者撰写的论文摘要.xlsx')
ai_data = pd.read_excel('AI生成的论文摘要.xlsx')

scholar_abstracts = scholar_data['论文摘要']
ai_abstracts = ai_data['论文摘要']

# 加载停用词表
def load_stopwords(filepath):
with open(filepath, 'r', encoding='utf-8') as file:
stopwords = set(file.read().strip().split('\n'))
return stopwords

stopwords = load_stopwords('stopwords.txt')

# 分词并去除停用词
def jieba_tokenizer(text):
words = jieba.cut(text)
return ' '.join([word for word in words if word not in stopwords])

scholar_abstracts = scholar_abstracts.apply(jieba_tokenizer)
ai_abstracts = ai_abstracts.apply(jieba_tokenizer)

# 统计词频函数
```

```
def count_words(texts):
all_words = []
for text in texts:
all_words.extend(text.split())
return Counter(all_words)

scholar_words = count_words(scholar_abstracts)
ai_words = count_words(ai_abstracts)

# 输出词频最高的前15个词
print('学者撰写的论文摘要:')
print(scholar_words.most_common(15))
print('AI生成的论文摘要:')
print(ai_words.most_common(15))

# 统计独有词
scholar_unique_words = set(scholar_words.keys()) - set(ai_words.keys())
ai_unique_words = set(ai_words.keys()) - set(scholar_words.keys())

# 分别输出独有词频数最高的前15个词
print('学者撰写的论文摘要独有词:')
print(Counter({word: scholar_words[word] for word in scholar_unique_words}).most_common(15))
print('AI生成的论文摘要独有词:')
print(Counter({word: ai_words[word] for word in ai_unique_words}).most_common(15))

import nltk
from nltk import ngrams

# 定义函数来生成 N-gram，并计算频率
def generate_ngrams(tokens, n):
n_grams = ngrams(tokens, n)
return [' '.join(gram) for gram in n_grams]

# 定义函数来统计 N-gram 的频率并返回前 top_n 个
def top_ngrams_by_frequency(abstracts, min_n, max_n, top_n):
all_ngrams = Counter()
for abstract in abstracts:
words = abstract.split()
for n in range(min_n, max_n + 1):
ngrams_list = generate_ngrams(words, n) # 生成 N-gram
all_ngrams.update(ngrams_list)
top_ngrams = all_ngrams.most_common(top_n)
return top_ngrams

# 定义输出 top_n 个高频搭配的函数
def print_top_ngrams(top_ngrams, category):
print(f"Top {len(top_ngrams)} {category} 搭配: ")
```

```
for ngram, freq in top_ngrams:
print(f"{ngram}: {freq}")

# 统计学者撰写的摘要中的 N-gram 频率并输出前15个
scholar_top_ngrams = top_ngrams_by_frequency(scholar_abstracts, 2, 10, 30)
print_top_ngrams(scholar_top_ngrams, "学者撰写")

# 统计AI生成的摘要中的 N-gram 频率并输出前15个
ai_top_ngrams = top_ngrams_by_frequency(ai_abstracts, 2, 10, 30)
print_top_ngrams(ai_top_ngrams, "AI生成")
```