基于大语言模型的智能司法判决书生成与优化系统

成鑫 刘思齐 丁佳悦 麦洧煽 周宸宇

（上海大学 理学院 计算机工程与科学学院）

摘要 本项目旨在构建一个智能司法判决书生成与优化系统，利用大语言模型和多源数据整合技术，实现从案件信息提取到判决书生成与润色的全流程自动化。系统通过整合案件关键信息、相似案例和高频法律条文，结合精心设计的 Prompt，生成符合中国法院规范的判决书初稿，并对其进行润色和格式优化。模块设计严格遵循事实约束，杜绝虚构或推断未提供的内容，确保生成文书的法律合规性和专业性。系统支持多案件类型的批量处理，能够高效生成大规模司法文书，为智能司法文书生产提供了高效、规范的解决方案。

关键词 司法文书生成 相似案例分析 高频法律条文 大语言模型 自动化司法系统

目录

1. 研究背景和意义

随着生成式人工智能技术的快速发展，人工智能在各行各业的应用不断深化。司法作为国家治理体系的重要组成部分，也正面临前所未有的智能化转型机遇。近年来，我国法院系统面临案件数量持续增长与法官编制有限之间的矛盾，尤其是在基层法院，审判压力尤为突出。例如，北京朝阳区法院在 2023 年受理案件超过 13 万件，平均每位法官每日需处理近两个案件，工作强度极大。据预测，2024 年全国法院新收案件总量有望突破 5000 万件，司法系统运行已趋近饱和状态[1]。

为缓解这一压力，人工智能在司法辅助领域的应用被广泛关注。通过引入 AI 进行类案推送、法律文书生成与判决辅助，不仅可以提高审判效率，还能够在一定程度上实现“同案同判”，降低人为因素带来的不确定性，提升司法公正性。已有数

据显示，在部分地区引入 AI 系统后，案件审理周期平均缩短 30%，纠纷调解成功率提升至 75%，有效缓解了法院的审判压力，并从源头减少了诉讼增量。

1.1 AI 司法的现状和挑战

尽管人工智能在司法领域的探索不断深入，其实际应用仍面临诸多挑战和限制。技术层面，当前 AI 系统普遍存在“黑箱”特性，即其决策过程缺乏透明度与可解释性。这使得在判决文书生成和裁判建议中，AI 可能在不被察觉的情况下引入训练数据中的偏见，影响判决公正性。2024 年多起因 AI 生成内容导致的版权与隐私纠纷案件，也暴露了生成式模型在语义准确性与合规性方面的不足[2]。

在法律层面，现有司法体系尚未对 AI 辅助裁判或 AI 生成证据的合法性与效力作出明确规定，相关法律制度仍处于探索阶段。例如，AI 辅助生成的判决内容是否可以直接用于裁判文书、是否具备法律约束力，其法律地位尚不明晰，亟需制度设计与立法规范。在伦理层面，AI 的介入可能弱化司法人员的独立判断力，导致过度依赖技术系统，进而削弱司法的人文关怀。此外，由于技术发展水平在地区之间存在差异，发达地区法院更容易获取先进 AI 资源，可能进一步加剧司法服务在城乡、区域间的不平衡，加重数字鸿沟问题[3]。

尽管如此，国际上对 AI 司法应用的探索仍在不断推进。2024 年，美国亚利桑那州高级法院首次在量刑听证中采纳了利用生成式 AI 重建的遇害人“数字陈述”作为法庭证据，引发全球法律界广泛关注。这一案例不仅展示了 AI 在法庭程序中的实际应用潜力，也凸显了其背后技术可信性与伦理规范的重要性。

总体来看，AI 赋能司法具有广阔前景，尤其是在法律文书生成、类案推送、量刑辅助等具体应用方面，已逐步展现出可行性和实用价值。然而，要实现 AI 与司法的深度融合，仍需在技术标准、法律制度、伦理框架等方面持续完善与突破。本项目正是在此背景下提出，致力于探索生成式人工智能在判决书自动生成中的应用路径，力求在提升审判效率的同时确保内容的准确性、合法性与可解释性，为推进我国司法智能化建设提供有益示范。

1.2 项目目标和研究任务定义

本项目旨在研究并实现一个基于生成式人工智能的判决书自动生成系统，探索 AI 技术在司法文书智能化生成中的可行性与实际应用效果。通过引入自然语言处理、文书结构建模与类案数据分析等技术手段，系统可根据案件事实自动生成符合司法规范的裁判文书草稿，辅助法官高效、规范地完成判决书撰写任务。

项目的核心目标包括：第一，构建一个适用于司法场景的判决书生成框架，涵盖案情分析、法条适用、裁判逻辑梳理与文书结构组织等关键模块；第二，提升生成内容的法律准确性与逻辑一致性，避免 AI 生成常见的幻觉与语义错误；第三，通过类案分析与法律知识增强，提高生成内容的可解释性和法律可信度；第四，探索 AI 辅助文书撰写对审判效率、文书质量及司法资源配置的影响，验证系统在实际应用场景下的实用价值。

通过上述研究，本项目希望为人工智能在司法文书自动化生成领域的实践落地提供理论支撑与技术方案，为缓解当前司法系统人力资源紧张、文书处理效率低等问题提供创新路径，并推动我国司法系统在智能化、数字化方向上的持续发展。

2. 系统原理与关键技术分析

2.1 法律语义向量构建方法（基于 BERT）

BERT（Bidirectional Encoder Representations from Transformers）是一种革命性的预训练语言模型，凭借深度双向 Transformer 架构生成蕴含丰富上下文语义信息的文本向量表示。其输入层借助 WordPiece 分词为每个子词生成初始嵌入向量，同时加入位置编码和段落类型向量，三者相加形成输入 Token 的初始表示，为模型提供词义、词序及句子关系的基础信息。BERT 的核心在于堆叠的多层 Transformer 编码器，每层由多头自注意力机制和前馈神经网络构成，并配备残差连接与层归一化。多头自注意力机制使模型在处理序列中每个词时，能同时关注其前后文信息，充分融合左右语境，捕捉词语间的复杂依赖关系。经过逐层抽象和融合，最终输出层的每个 Token 位置对应一个高维稠密向量，即其深度语义表示。

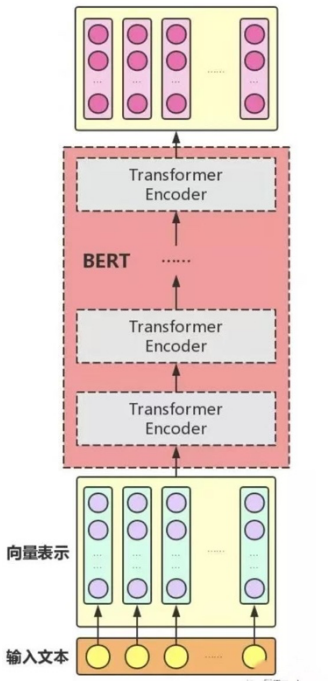

图 2-1 Bert 原理图

BERT 强大的表示能力源于其创新的预训练任务，主要在掩码语言模型（MLM）和下一句预测（NSP）两大无监督任务上训练。MLM 任务随机遮蔽输入序列中约 15%的 Token，迫使模型依据双向上下文预测原始 Token，以学习强大的上下文表示能力；NSP 任务则让模型判断两句子的先后关系，助力学习句子间关系和篇章结构信息。通过在海量无标注文本数据上完成预训练，BERT 的模型参数得以优化，能够生成高质量、上下文感知的文本向量。

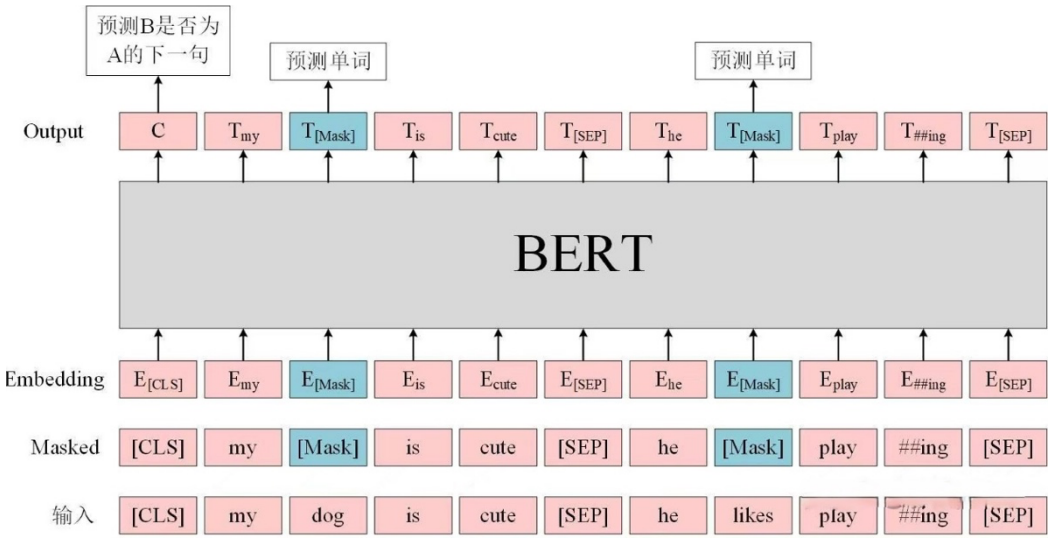

图 2-2 Bert 原理图

在生成文本向量表示时,BERT 有多种策略。常用的是提取特定 Token 的输出向量，如用特殊标记[CLS]对应的向量表示整个序列，或直接使用各 Token 位置的最终层输出向量。也可采用平均池化，对序列中所有 Token 的输出向量取平均，得到序列的单一向量表示。

BERT 文本向量生成理论的核心在于：利用深度双向 Transformer 的自注意力机制,借助 MLM 和 NSP 预训练任务,学习大规模语料中词语的深层语义关联和句间关系，生成高度语境化、富含语义信息的稠密向量。这些向量作为特征表示，可迁移应用于多种下游 NLP 任务，提升性能。其生成的向量是动态且上下文相关的，同一词在不同句子中会有不同向量，这是相较于传统静态词嵌入模型的关键优势。

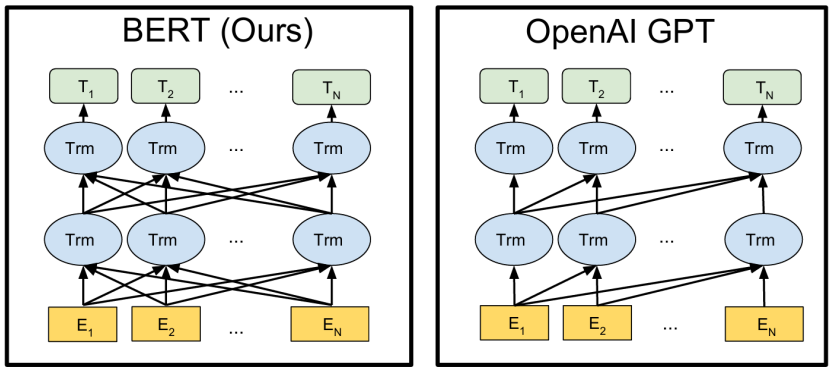

图 2-3 Bert 原理图

在法律人工智能领域，案例检索与相似度匹配环节对文本的高效、精准向量化表示要求极高。本系统采用的基于 BERT 模型的文向量生成方法，已在多项自然语言处理任务中取得突破性性能。相较于传统的静态词嵌入模型（如 Word2Vec、GloVe），其生成的词元向量是上下文感知且动态的，能精准捕捉法律文本中复杂多变的概念和表述。例如，“银行”一词在涉及金融交易与描述方位的句子中，其向量蕴含的语义信息显著不同。

具体操作流程是将经过分词处理的文本输入 BERT 模型，模型输出与输入分词序列等长的隐藏状态序列。以本系统采用的 BERT-base-chinese 模型为例，每个隐藏状态都是蕴含丰富上下文信息的 768 维高维向量，这些向量可直接作为下游任务的特征输入（如文本相似度计算、聚类等）或文本表示向量。

对于篇幅较长的法律文书（如判决书、起诉状），因其通常超出 BERT 模型的最大输入长度限制（如 512 tokens），系统采用分块处理与向量合成策略。先利用 jieba 进行中文分词，再依据 BERT Tokenizer 计算的 token 数量，结合预设块大小（默认 400 tokens，预留空间给特殊 token），将长文本分割成多个较小且语义完整的文本块。然后对每个独立文本分块输入 BERT 模型，获取所有 token 的隐藏状态向量。为将序列融合为单个固定维度向量，采用平均池化操作，计算分块内所有 token 向量的平均值，得到分块向量。最后，按分块长度加权平均所有分块向量，每个分块向量的权重为其实际包含的 token 数量，以生成代表整个长文本的单一向量。

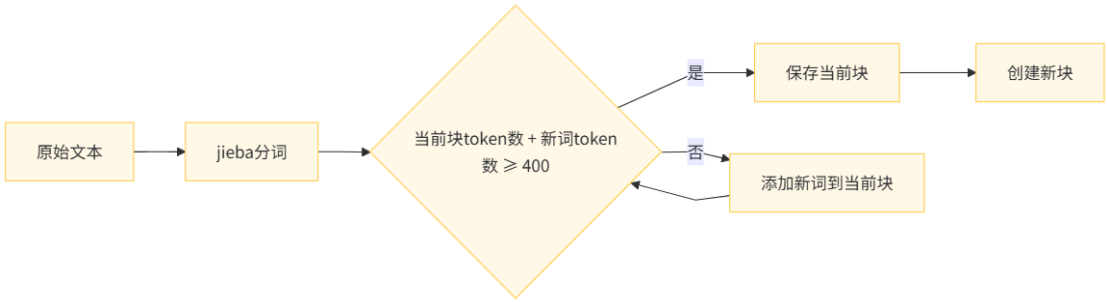

图 2-4 处理流程图

这种基于 BERT 的动态上下文感知向量生成与高效的长文本处理机制，为后续的案例检索、相似度分析及法律要素提取提供了坚实的技术支撑，是实现司法 AI 系统智能化处理海量法律文本的关键环节。

图 2-5 BERT 文本向量匹配效果展示（以附件三为例）

2.2 关键信息提取与向量化索引机制

2.2.1 Faiss 原理简介

在大模型处理高维数据任务中，相似性搜索（Similarity Search）始终是一项核心挑战。特别是在司法场景中进行案例比对、文本向量相似度计算时，面对成千上万篇长文本的向量表示，如何在保证语义匹配准确性的同时，实现高效、可扩展的搜索能力，成为系统性能优化的关键瓶颈。为应对这一挑战，本项目引入了由 Meta AI 团队研发的开源高效相似度搜索库 Faiss（Facebook AI Similarity Search）。该工具专为高维稠密向量的高效近似最近邻搜索设计，结合多种量化技术与倒排机制，可在精度略微损失可接受的前提下，实现数量级的加速效果，极适合本项目中对历史案例库和法条知识库的向量匹配任务。

暴力搜索方法虽然能获得完全精确的结果，但其 O(m*n)的时间复杂度在实际应用中往往由于速度、内存上的限制无法接受。可见，允许一定程度精度妥协时，基于相似性搜索速度可获得数量级的提升。在 Faiss 实现中，这种加速极大地依赖于数据集的预处理过程——索引构建，以及数据的查找过程——向量检索。

索引构建的核心算法原理称为乘积量化(PQ) 。乘积量化操作中，数据流中的每个向量被拆为特定个数的子向量，每个子向量依次使用 k-means 进行聚类。易得聚类得到的子向量映射并未减少存储大小，因此 Faiss 使用将子向量映射成类目 ID，即每

组子向量中的每个向量均用类目中心表示，继而得到原始数据数位上压缩。拆分子向量的个数与聚类个数均由经验值得出，通常取 8 和 256。

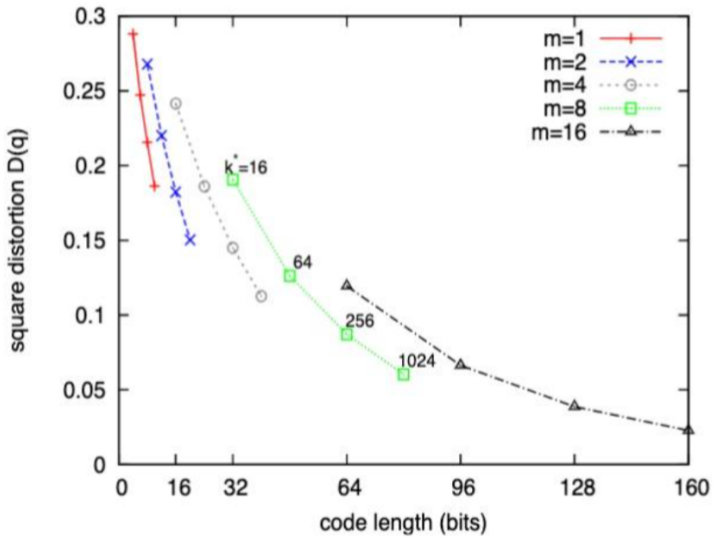

图 2-6 与子向量拆分个数 k 及聚类个数 m 相关的量化误差

向量距离计算的原理与乘积量化有所相似，同样对拆分的子向量进行聚类，然后计算每段子向量与索引中预训练中心的距离。因此在查询时，仅需查表得到待查向量的每段子向量与其中心距离之和即可，大大减少了向量个数较大时的距离计算与查表次数。

2.2.2 向量检索与高效相似度计算

PQ 优化了向量距离计算的过程，但如若库中的向量特别多，依然难以避免遍历整个库的过程，效率依旧较低高，因此引入 Faiss 用到的另外一个关键技术——倒排乘积量化(Inverted File System)。其相对于 PQ 的优化在于加速原理：暴力搜索是在全空间进行搜索，而 PQ 通过对全空间分割，即聚类，在搜索时快速确定子空间并加以遍历来提速。由上一段落可以看出，虽然距离已预先计算，每个样本到查询样本的距离之和仍需一一相加。但查询过程中，机器真正感兴趣的查询对象是那些与查询样本相近的样本，依次相加因而做了很多无用功。若能通过某种手段快速将全局遍历锁定为感兴趣区域，则可以舍去不必要的全局计算以及排序。倒排 PQ 乘积

量化的“倒排”，正是这样一种思想的体现。在具体实施手段上，采用的是通过聚类的方式实现感兴趣区域的快速定位。在所有的中心点距离中选取距离最近的 N 个点，再使用 PQ 计算查询向量与它们的距离。使用了 IVF 过后，需要计算距离的向量个数就少了几个数量级，继而使向量检索就变成一个很快的操作。

2.3 法律助手 AI 模型

我们使用基于 RAG 框架，结合 Langchain + Faiss + DeepSeek 搭建的本地知识库提取法律案例关键信息。RAG 技术是一种结合了检索和生成能力的新型语言模型应用方式。其核心在于，首先使用一个检索器从知识库中获取与查询相关的文档片段，然后基于这些检索到的上下文，利用语言模型（LLM）生成回答。这种方式显著提高了回答的准确性，因为它能够实时地、基于事实地、动态地生成响应。

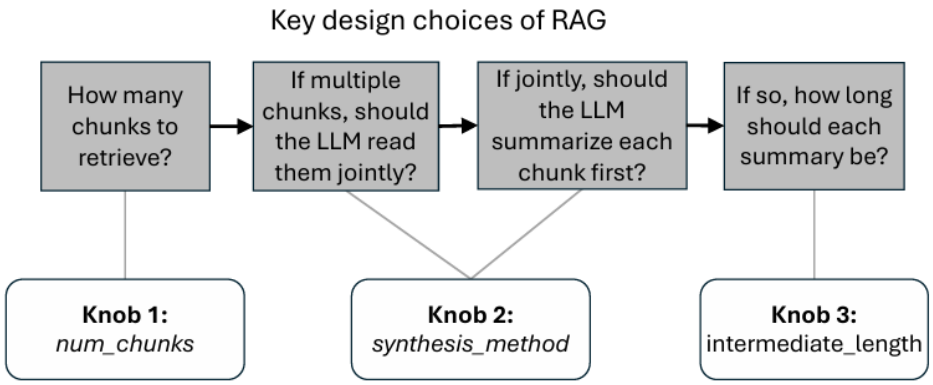

图 2-7 源自 RAG 系统关键设计选择的配置选项

在构建 RAG 系统时，选择合适的技术工具至关重要，我们的系统使用了以下几种关键技术：Langchain 是基于大语言模型开发的框架，提供了大量的开发工具，可以将语言模型与其他的数据源连接、也运行语言模型与其他环境进行交互，提供本地向量存储数据和文档检索操作；Faiss 提供向量相似性检索工具，其原理上一小节已有说明；DeepSeek 提供预训练的自然语言处理模型，具有强大的语言理解和生成能力，使得系统能够更准确地理解用户的查询需求，并生成上下文相关的，自

然语言的答复。此外，还有提供文本转化向量服务的 Sentence Transformers 和深度学习框架 PyTorch，在此不加赘述。

2.4 系统整体流程图

为更清晰地展示本系统从原始案件材料到判决书生成的整体流程，我们绘制了系统流程图（如图 2-7 所示）。该流程图全面描述了各模块之间的逻辑关系与数据流动路径，包括基础信息提取、关键信息抽取、相似案例检索、法条分析、判决书生成与润色等关键环节。通过这一端到端的自动化处理流程，系统实现了司法文书生产的高效化与规范化，确保了内容生成的准确性、合规性与可读性。

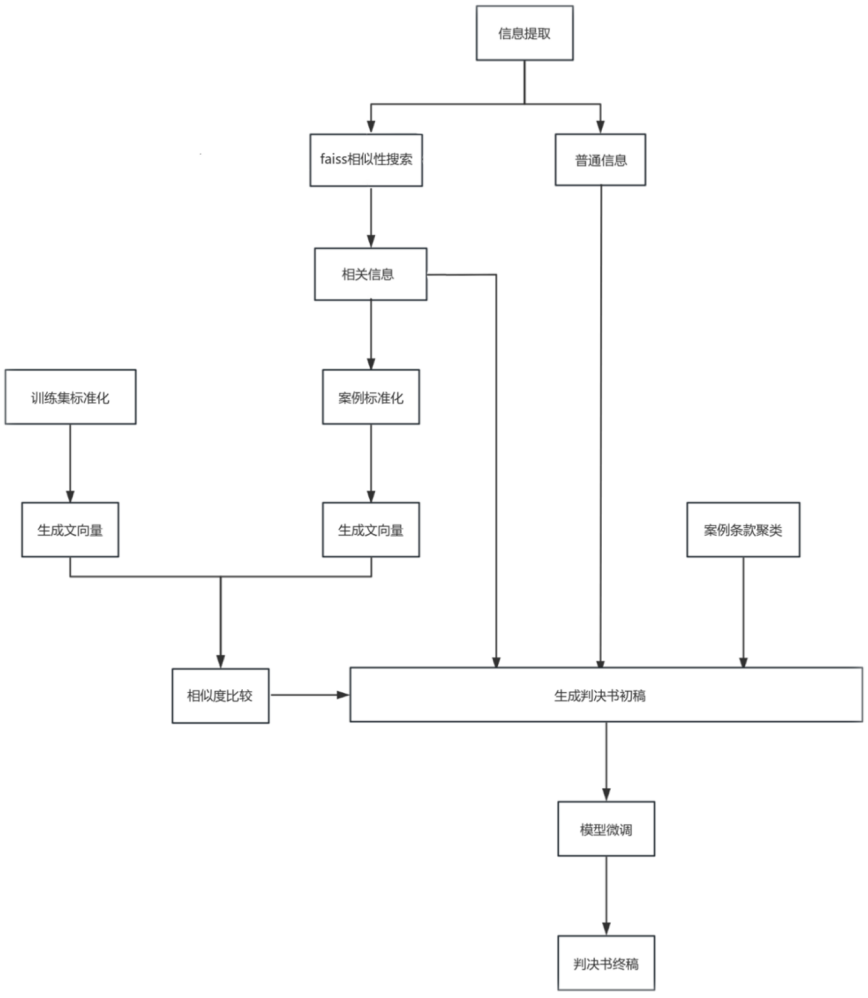

图 2-8 项目系统流程图

3. 系统实现与实验流程

3.1 基础与关键信息提取模块

信息的检索是信息提取环节的核心组成部分，直接影响后续判决书生成的准确性、完整性与法律合规性。在司法场景中，案件材料通常结构复杂、信息密集，不仅包含大量与审理相关的核心内容，还包含部分格式性、结构性信息。因此，为提高系统在不同类型案件下的泛化能力与实用效果，本项目将信息检索模块划分为两个子模块，分别针对不同类型的信息进行定向提取。

首先是基本信息提取模块，其目标是从案件材料中提取如当事人姓名、性别、民族、出生日期、联系方式、住址等标准化结构化信息。这些信息虽不直接影响判决结果，但在正式司法文书中是不可或缺的构成要素，关系到文书的完整性与规范性。其次是关键信息提取模块，重点聚焦于案件事实、争议焦点、诉讼请求、证据清单、裁判要点等对判决逻辑具有决定性影响的内容。该模块通过结合向量检索与大模型语义理解能力，自动提炼案件中的核心法律要素，为后续的相似案例匹配与法律条文引用提供坚实的数据支持。

通过“基本信息 + 关键信息”的双模块提取体系，系统不仅实现了案件信息的全面捕捉，也为下游任务提供了结构清晰、语义明确的输入基础，确保最终生成的判决书具备较高的专业水准与实用价值。

3.1.1 对于基本信息的提取

由于“姓名”，“住址”，“生日”，“联系电话”，“民族”等信息都是一些与判案无关但是在判决书生成时很必要的信息，所以单独把它提取出来。

具体通过 *extract_baseinfo.py* 实现：

```python
# 结果保存到 src/result/baseinformation.json
    result_dir = os.path.abspath(os.path.join(os.path.dirname(__file__), '../result'))
    if not os.path.exists(result_dir):
        os.makedirs(result_dir)
    output_path = os.path.join(result_dir, "baseinformation.json")
    with open(output_path, "w", encoding="utf-8") as f:
        json.dump(result, f, ensure_ascii=False, indent=2)
    print(f"已用大模型结构化抽取并保存到 {output_path}")
```

图 3-1 extract_baseinfo.py 代码细节

这个程序首先通过 *PyPDFLoader* 批量读取指定目录下各个案例文件夹中的"起诉状.pdf"文件（前将附件一中 DOCX 文件转换为 PDF 文件），将 PDF 内容转换为文本格式，然后调用 DeepSeek 大语言模型的 API 接口，使用提示词模板指导模型从起诉状文本中准确提取原告和被告的详细信息（包括姓名、性别、民族、生日、住址、身份证号、联系电话等七个关键字段），模型返回的结构化 JSON 数据经过格式清理和解析后，程序将所有案例的提取结果汇总保存到 *baseinformation.json* 文件中，整个过程也采用异常处理机制确保程序稳定运行，即使某个 PDF 文件不存在或 API 调用失败也会记录相应的错误信息，这确保了提取信息的准确性。这为后续的判决书生成等功能提供了标准的基础支撑。结果如右图所示（以离婚案为例）：


```json
"divorce": {
  "原告": [
    {
      "姓名": "肖某某",
      "性别": "男",
      "民族": "汉族",
      "生日": "1955年11月25日",
      "住址": "重庆市江津区油溪镇",
      "身份证号": "5102251955xxx",
      "联系电话": "191xxx5537"
    }
  ],
  "被告": [
    {
      "姓名": "黄某某",
      "性别": "女",
      "民族": "汉族",
      "生日": "1952年9月14日",
      "住址": "重庆江津区德感镇长冲街道",
      "身份证号": "5102251952xxx",
      "联系电话": ""
    }
  ]
}
```


图 3-2 baseinformation.json

3.1.2 对于关键信息的提取

关键信息的提取经历了两个阶段，第一个阶段是直接通过 Prompt 用 DeepSeek 提取这三个待判案例的关键信息，但效果不尽如人意，因此采用了 Faiss 向量检索方法，在增强其检索速度的基础上还大幅度提高了其在语义理解上的强度。

具体通过 FaissExtractor.py 实现：

```python
# 构建FAISS索引
    vector_store = FAISS.from_documents(texts, embeddings)
    vector_store.save_local("faiss_index")
    print(f"FAISS索引构建完成，包含 {vector_store.index.ntotal} 个向量")
    return vector_store, embeddings
# 通过faiss索引检索案件信息
def retrieve_case_context(vector_store, case_name, top_k=10):
    # 使用案件名称作为查询
    docs = vector_store.similarity_search(case_name, k=top_k)
    # 获取文件基本名（不含路径和扩展名）
    base_name = os.path.splitext(os.path.basename(case_name))[0]
    # 过滤出属于该案件的文本块
    case_context = []
    for doc in docs:
        # 从元数据获取源文件名
        source_path = doc.metadata.get('source', '')
        source_name = os.path.splitext(os.path.basename(source_path))[0]
        # 使用文件名基本部分匹配
        if source_name == base_name:
            case_context.append(doc.page_content)
    return "\n\n".join(case_context)
```

图 3-3 FasissExtractor.py 代码细节

这个程序集成了先进的向量检索技术和大语言模型分析能力。程序首先使用 *LangChain* 框架批量加载指定目录下的所有 PDF 法律文档，通过 *RecursiveCharacterTextSplitter* 按照中文标点符号进行智能分块处理，然后利用中文文本向量化模型 “shibing624/text2vec-base-chinese”将文档转换为高维向量表示，构建高效的 FAISS 向量索引库以支持快速相似性检索；在信息提取阶段，程序通过向量检索定位到特定案件的相关文档片段，再调用 DeepSeek 大语言模型 API，使用多层次提示词模板，从法律文书中深度提取案件事实、争议焦点、案由、案件类型、当事人详细信息、诉讼请求、裁判要点、证据清单、庭审过程摘要等十大类关键法律要素，特别强调对金额、时间、数量等具体数字的准确提取，最终输出结构化的 JSON 格式数据，保存到 *result.json* 文件中，作为待判案件的关键信息，这解决了问题一。结果如下所示（以离婚案的案情摘要为例）：

```
"争议焦点": [
        "夫妻感情是否确已破裂",
        "被告提及的借款事实是否属实",
        "原告的收入情况"
    ],
    "诉讼请求": [
        "判决离婚"
    ],
    "裁判要点": [
        "原告未能提供充分证据证明夫妻感情确已破裂，故依法驳回原告的诉讼请求"
    ],
    "其他关键信息": [
        "结婚证字号：J500116-2016-xxx",
        "婚后无子女",
        "被告提及向邱某某借款共计17000元，向黄某某借款20000元，但未提供借条或转账记录",
        "原告月收入为1842元"
    ]
```


图 3-4 提取出来的 Json 格式关键信息

3.2 语义匹配与相似度计算模块

针对问题 2，根据提取案例的关键信息，从附件 2 中查找相关案例。我们采用的是量化的方法，根据前面提到的生成文向量的方法，对附件二中所有案例分别生成了对应的 768 维文向量。为了让附件一和附件二之间的对比有意义，先将附件一中的各个待判案件中的案情摘要，起诉状和庭审笔录总结为一段有逻辑的话，然后用同样的方法生成三个 768 维向量。用附件二生成的文向量为 *document_vectors.json*，附件一生成的文向量为*附件 1 vec.json*。接下来采用余弦相似度算法逐一计算目标案例与历史案例库中每个案例的语义相似程度，通过向量空间中的角度距离来衡量案例内容的相似性，程序还会为每个目标案例找出相似度最高的前 20 个历史案例并按相似度降序排列，最终将检索结果以结构化 JSON 格式保存到 *similar_cases.json* 文件中，每个结果包含案例编号和精确的相似度数值，也就解决了问题二。在这也存在优化，原来采取暴力搜索的方法直接使用余弦相似度计算（如图 3-3 所示），现在可以通过 Faiss 方法技巧性实现余弦相似度计算，关键技巧是当两个向量都被 L2 归一化以后，他们的内积就等于余弦相似度，大大提高了计算效率，具体见 2.2.2 向量检索与高效相似度计算。

采用余弦相似度的原因是其专门设计用于处理高维空间中的相似性计算，能够有效避免"维度诅咒"问题，而且法律文本中，真正重要的是语义内容的相似性，而不是字面文字的匹配。余弦相似度更能捕捉到这种深层语义关联。

具体代码实现是在 *search_similar_cases.py* 中实现的。


```
"劳务合同.docx": [
    {
      "编号": "1720.docx",
      "相似度": 0.9903035759925842
    },
    {
      "编号": "0890.docx",
      "相似度": 0.9900975823402405
    },
    {
      "编号": "0889.docx",
      "相似度": 0.9900875687599182
    },
```


图 3-5 相似度提取结果


```
"劳务合同.docx": [
  {
    "编号": "1720.docx",
    "相似度": 0.9903036726004268
    "相似度": 0.9903035759925842
  },
  {
    "编号": "0890.docx",
    "相似度": 0.9900976890400193
    "相似度": 0.9900975823402405
  },
  {
    "编号": "0889.docx",
    "相似度": 0.9900877399916211
    "相似度": 0.9900875687599182
  },
```


图 3-6 传统方法和采用 Faiss 后结果对比图

```python
# 反向：对每个附件1 vec.json中的文档，找出与document_vectors.json中最相似的前20个文档
    results = {}
    for idx, (doc2_name, doc2_vec) in enumerate(zip(doc2_names, doc2_vectors)):
        sims = [cosine_sim(np.array(doc2_vec), np.array(doc1_vec)) for doc1_vec in doc1_vectors]
        topN = 20
        top_idx = np.argsort(sims)[-topN:][::-1] if sims else []
        similar = []
        for i in top_idx:
            similar.append({"编号": doc1_names[i], "相似度": float(sims[i])})
        results[doc2_name] = similar
        if idx < 3:
            print(f"[{doc2_name}] sims前5: {sims[:5]}")
```

图 3-6 search_similar_cases.py 代码细节

通过 Faiss 建立索引之后，相似度搜索在结果准确性不受到明显影响的情况下，效率得到大幅度的提高，程序运行时间加快了 7.1 倍，恰当地说明了 Faiss 在此类任务中的高效性和实用性

```
=== 3. 结果对比验证 ===
查询: 劳务合同.docx
传统方法前5个: ['0.990304', '0.990098', '0.990088', '0.990049', '0.989807']
Faiss方法前5个:  ['0.990304', '0.990098', '0.990088', '0.990048', '0.989807']
✅ 两种方法结果一致!

修复后的Faiss结果已保存到: D:\XZB\xinzhibeigit\src\result\similar_cases_faiss_fixed.json
性能对比: 传统方法 0.343秒 vs Faiss方法 0.048秒
性能提升: 7.1倍
传统方法结果已保存到: D:\XZB\xinzhibeigit\src\result\similar_cases_traditional_verified.json

=== 检索完成 ===
```

图 3-7 增加 Faiss 之后的运行结果对比

为了做好第二题与第三题的衔接。编写了 *law_article_analysis.py* 程序。

这个程序可以说是一个法条关联分析模块，专门用于从附件二中大量判决书文档中智能提取法律条文引用信息并进行统计分析。程序首先采用双重策略提取法条信息：优先从判决书结尾的"依照……判决如下"段落中精确提取法院最终适用的法条及其具体条号（判决的核心依据），通过复杂的正则表达式模式匹配识别《法条名称》和条号组合，并支持同一法条多个条号的智能归并处理；若判决书格式不标准无"依照"段落，则回退到全文扫描模式，使用更宽泛的正则表达式在整个文档中搜索法条引用；接着程序遍历指定目录下的所有 Word 格式判决书文档，构建一个案例-法条二值矩阵（行代表案例文件，列代表不同法条，值为 0 或 1 表示该案例是否引用了该法条），然后基于这个矩阵计算每个法条在所有案例中的出现概率（反映法条使用频率）和法条间的皮尔逊相关系数矩阵（揭示哪些法条经常被同时引用），最终将案例-法条矩阵、法条概率统计和法条相关性分析结果分别保存为 CSV 文件，分别为 *case_law_matrix.csv*，*law_article_prob.csv*，*law_article_corr.csv*。

如下图 3-6 显示了附件二中引用最多的五条法文：

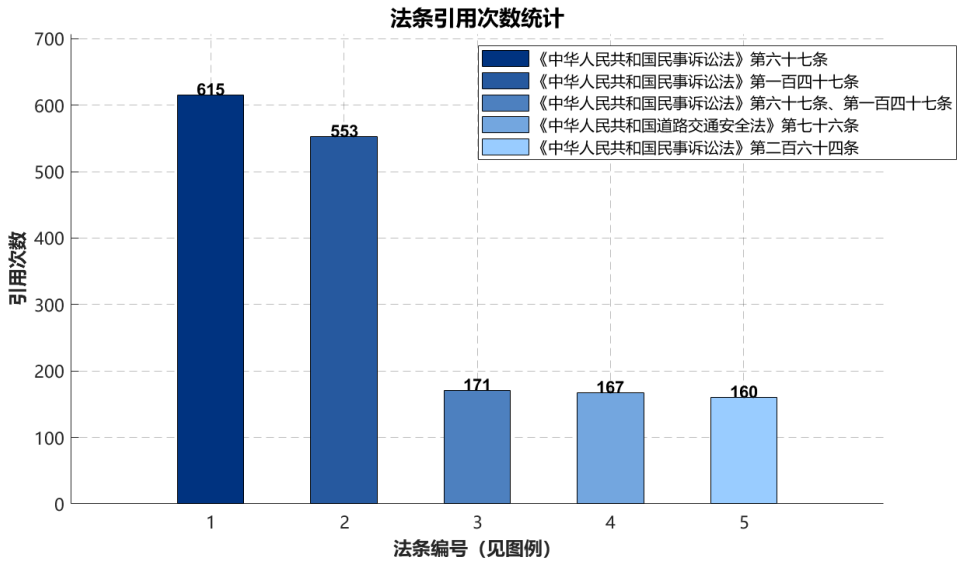

图 3-8 法律条文引用次数统计

3.3 判决书生成与优化模块

判决书生成与润色模块是本项目的重要组成部分，旨在通过自动化手段生成符合中国法院规范的判决书初稿，并对其进行润色和规范化处理。模块的设计目标是提升司法文书生产的效率与质量，同时确保内容的专业性和法律合规性。整个模块由“判决书生成”和“判决书润色”两个子模块构成，分别负责初稿生成和后续优化。

判决书生成模块的核心功能是根据案件关键信息、相似案例和法律条文，自动生成结构化的判决书初稿。判决书生成的基础是多种数据来源的整合与处理。模块从以下几个主要数据文件中提取信息：首先是案件关键信息：从*result.json*中提取案件的核心信息，包括案情摘要、起诉状内容和庭审笔录。这些信息涵盖了案件类型、案由、当事人信息、争议焦点、诉讼请求、裁判要点及其他关键信息。例如，在“离婚”案件中，案情摘要中明确了夫妻感情破裂、借款事实争议等焦点问题，并提供了相关证据和背景信息；其次是相似案例：从*similar_cases.json*中读取与当前案件相关的历史案例，提取其裁判要点和案情摘要。这些相似案例为判决书的法律适用部分提供了参考依据，增强了说理的深度和权威性；另外还参考了当事人信息：从*baseinformation.json*中获取详细的当事人信息，包括原告和被告的身份信息。模块将这些信息与案件关键信息进行整合，确保判决书中当事人信息的完整性和准确性；随后参考了高频法律条文：通过分析*case_law_matrix.csv*中相似案例的引用情况，统

计出高频出现的法律条文。这些条文被用于判决书的法律适用部分，确保引用的法律依据具有代表性和相关性。同时最终要的判决依据主要由法律助手模型生成，我们将分类过的附件二进行遍历，检索出附件二中与待判案件同类型的案例涉及到附件三的法律条文共有八个，故在魔改法律助手模型后，结合这八个法文生成了*最终版case_law_result.json*文件作为最终的判案指导。

```
case_law_path = os.path.join(os.path.dirname(__file__), "..", "background", "最终版case_law_results.json")
    with open(case_law_path, "r", encoding="utf-8") as f:
        case_law_results = json.load(f)
    # 读取result.json、similar_cases.json、baseinformation.json、case_law_matrix.csv都从src/result目录
    result_dir = os.path.join(os.path.dirname(__file__), "..", "result")
    with open(os.path.join(result_dir, "result.json"), "r", encoding="utf-8") as f:
        case_infos = json.load(f)
    with open(os.path.join(result_dir, "similar_cases.json"), "r", encoding="utf-8") as f:
        similar_cases = json.load(f)
    with open(os.path.join(result_dir, "baseinformation.json"), "r", encoding="utf-8") as f:
        baseinfo_all = json.load(f)
```

图 3-9 生成过程中参考的信息

Prompt明确要求模型严格按照中国法院判决书的标准结构输出内容，包括案由、当事人信息、诉讼请求、争议焦点、查明事实、举证责任分析、法律适用与裁判理由以及判决结果等部分。生成过程严格禁止模型虚构、补充或推断未在输入信息中出现的内容，确保生成内容的客观性和法律合规性。最终，将每个案件类型的判决书内容保存为Markdown文件，便于阅读，同时，将所有案件的判决书内容汇总保存为*judgement_draft.json*文件，便于后续处理和润色。该模块整合案件要点、相似案例和高频法律条文，确保生成内容全面、准确。判决书生成严格依赖输入数据，杜绝虚构或推断，确保法律合规。支持多类型案件的批量处理，高效生成大规模司法文书。引用相似案例与高频条文，有力提升说理部分的深度与权威性。

在生成初稿的基础上，判决书润色模块进一步对文书进行表达和格式的优化，使其更加正式、规范。润色模块首先从指定目录的docx文件中提取案情摘要，为润色过程提供上下文支持。随后，通过调用DeepSeek的API，模块对判决书初稿进行润色，优化其语言表达和格式。润色过程严格遵循“只优化表达和格式，绝对禁止虚构、

补充或推断未在原文中出现的内容”的原则，确保润色后的文书内容真实可靠。润色完成后，结果以新的Markdown文件保存，形成最终版本。

判决书生成与润色模块通过大语言模型驱动的自动化技术，为司法文书的生产提供了高效、规范的解决方案。其技术路线兼顾了法律合规性、内容专业性与自动化能力。

4. 系统创新点介绍

4.1 针对司法文本的定制化向量

本系统创新性地采用了基于 BERT 模型的动态上下文感知文向量生成方法，专门针对司法文本的复杂性和专业性进行了优化。通过结合中文分词工具（如 jieba）和 BERT-base-chinese 模型，系统能够精准捕捉法律术语的语义和上下文关联。例如，同一词语（如“银行”）在不同法律语境下的含义差异可通过动态向量准确区分。此外，针对长文本（如判决书）超出 BERT 最大输入长度限制的问题，系统设计了分块处理与加权平均池化策略，将长文本分割为语义完整的片段并生成全局向量表示。这种方法显著提升了法律文本的语义表示质量，为后续的案例检索和相似度分析奠定了坚实基础。

4.2 高效的案例相似度检索和匹配

系统通过集成 Faiss（Facebook AI Similarity Search）向量检索技术，实现了海量法律案例的高效相似度匹配。Faiss 的倒排乘积量化（IVFPQ）算法将高维向量空间划分为多个子空间，通过聚类快速定位相似案例，避免了传统暴力搜索的高计算成本。实验表明，该系统能够在毫秒级时间内完成百万级案例库的检索，且准确率显著优于基于关键词匹配的传统方法。此外，系统结合余弦相似度算法，从语义层面衡量案例相似性，而非依赖表面文字匹配，从而更贴合司法实践中“类案同判”的需求。

4.3 结合 AI 生成的法律文书自动化流程

本系统首创了“检索-生成-润色”一体化的司法文书自动化流程。基于 RAG（Retrieval-Augmented Generation）框架，系统首先从本地法律知识库中检索相关案例和法条，再通过 DeepSeek 大语言模型生成符合中国法院规范的判决书初稿。生成过程中，系统严格遵循事实约束，禁止虚构未提供的内容，确保法律合规性。最后，润色模块对初稿进行语言表达和格式优化，提升文书的专业性和可读性。这一流程不仅实现了从案件信息到判决书的端到端自动化处理，还通过多源数据整合（如当事人信息、相似案例、高频法条）增强了生成内容的权威性和说服力。

5. 项目总结

5.1 主要工作回顾

在本题的解决过程中，我们的工作经历了由“初步实现”到“高效优化”的持续迭代。最初阶段，我们采用了一种较为机械的方式，即逐一遍历全部文档，提取所需信息。该方法虽然能够保证结果的完整性，但处理效率低，且在面对大规模数据时计算成本较高，难以适应竞赛中对响应速度和准确率的双重要求。

为提升整体性能，我们对流程进行了多轮调整与测试，逐步实现了从“全量检索”向“目标化筛选”的转变。通过分析文档特征，我们构建了针对性更强的信息提取机制，有效压缩了处理范围，大幅提高了运行效率与判定准确率。

在处理附件三的过程中，我们也尝试引入创新策略以进一步优化算法。例如，考虑利用附件二中法条的出现频率构建权重模型，通过计算法条之间的相关性系数，筛选出可能性更高的法条，从而减少检索范围。该思路在理论上具备较好的时间复杂度表现，但实际效果不佳，主要问题在于相关性模型缺乏逻辑支撑，导致准确率下降，因此未作为最终方案采用。我们也对于附件二的文向量进行了 PCA 数据降维操作，即将其降到 50 维，但发现若把附件一对应文向量也降到 50 维，最后生成时准确度大大降低，故也未作为最终方案采用。

基于上述探索，我们最终选择以关键词分类为核心手段，对附件二中出现的法条内容进行语义聚类，将其划分为若干法律主题类别，并将对应类别的法律条文作为输入交由法律助手模型进行判定生成。该策略在信息组织和法律推理方面均表现良好，为模型提供了更具针对性的上下文背景。

需要说明的是，由于《合同法》《婚姻法》《劳动合同法》等已被统一编入《中华人民共和国民法典》，我们在生成判决结果时统一以《民法典》条文进行引用，从而确保输出文本的法律规范性与当前法制体系的一致性。

5.2 存在不足和未来展望

5.2.1 存在不足

首先是数据覆盖有局限性：目前系统仅支持劳动纠纷、离婚、民间借贷三大类案件，对于刑事案件、行政诉讼、知识产权纠纷等其他重要案件类型尚未涵盖，限制了系统的适用范围和普及程度。除此之外，语义理解深度也有不足：虽然采用了中文文本向量化模型，但在处理复杂法律概念、专业术语的细微差别和上下文关联方面仍存在局限，可能影响相似案例检索的精确度和法条适用分析的准确性。

5.2.2 未来展望

引入多模态处理技术：集成 OCR 技术和图像识别算法，提升对扫描文档、手写材料的处理能力；结合语音识别技术，支持庭审录音的自动转写和信息提取，实现更全面的案例信息采集；深化法律语义理解：探索融合法律知识图谱的深度学习模型，构建包含法条关系、案例先例、司法观点的知识网络，提升系统对复杂法律逻辑的理解和推理能力；拓展应用场景：将系统应用扩展至法学教育、律师培训、法律咨询等领域，为法律职业共同体提供全方位的智能化支持，推动整个法律行业的数字化转型。

参考文献

[1] 陈韬. AI 在司法中的应用[J]. 服务外包, 2025(02): 25–28.

[2] 刘霞. 生成式 AI 司法应用引争议[N]. 科技日报, 2025-06-25(004).

[3] 曲忠芳. AI 生成版权纠纷涌现著作权保护难题待解[N]. 中国经营报, 2024-12-09(C02). DOI: 10.38300/n.cnki.nzgjy.2024.002863.

[4] Douze M, Guzhva A, Deng C, Johnson J, Szilvasy G, Mazaré P E, Lomeli M, Hosseini L, Jégou H. The Faiss library[J/OL]. arXiv:2401.08281 [cs.IR], 2024. https://arxiv.org/abs/2401.08281.

[5] Ray S, Pan R, Gu Z, Du K, Ananthanarayanan G, Netravali R, Jiang J. RAGServe: Fast Quality-Aware RAG Systems with Configuration Adaptation[J/OL]. arXiv:2412.10543 [cs.CL], 2024. https://arxiv.org/abs/2412.10543.