清華大學深圳国际研究生院

Tsinghua Shenzhen International Graduate School

《土木与建筑工程 CAE》大作业总结报告

研究课题：基于 BGE 嵌入与 Dynamo 集成的建筑规范智能审查方法研究

学生姓名：古镇彰（2024214897）

课程名称：土木与建筑工程 CAE（70030082-200）

教师姓名：胡振中

完成时间：二〇二五年六月

基于 BGE 嵌入与 Dynamo 集成的建筑规范智能审查方法研究

1 引言

1.1 研究背景

在政策驱动与行业转型的双重背景下，建设工程审查领域迎来变革关键期。据住建部《“十四五”住房和城乡建设科技发展规划》[1]部署，“建设项目智能化审查和审批关键技术”被明确列为十四五重点任务，凸显了国家对建筑业数字化转型的战略导向。然而，当前审查制度面临着严峻的现实挑战，当前的审查方式仍以人工核验二维图纸和计算书为主，手工逐条核对的传统审查流程往往耗时费力，审查周期长且规范条文的主观解读易引发审查结果差异[2]。

面对行业痛点，技术突破带来智能化转型机遇。近年来，建筑信息模型（Building Information Modeling, BIM）技术在行业内获得广泛关注与应用[3]，作为建筑信息载体的 BIM 技术通过构建机器可读的三维数据模型，为审查流程革新奠定基础[4]。不少研究[4][5][6][7]指出，基于 BIM 的自动合规审查能够帮助突破人工效率瓶颈，实现数据驱动的标准化校验。其核心价值在于将二维图纸与三维模型的信息割裂转化为全生命周期数据贯通，避免了信息的低效利用，提升审查效率的同时保障标准执行的一致性。建筑规范合规性审查的智能化转型，逐渐成为建筑业从传统模式向数字化、工业化升级的重要路径[8]。

1.2 研究工作与创新性

本研究提出 BGE（BAAI General Embedding）嵌入+Dynamo 集成的智能审查框架，通过三阶段技术融合实现语义-构件自动化关联。创新性构建了“语义嵌入-参数匹配-实时反馈”的闭环机制，突破传统审查依赖人工经验、难以复用的技术瓶颈。主要研究工作如下：

1）基于 OCR（Optical Character Recognition）和自然语言处理技术，将原始建筑规范文件转化为结构化层级文本；

2）基于 BGE 模型生成中文语义向量，通过计算规范关键词与构件参数间的余弦相似度，实现二者智能语义关联；

3）基于 Dynamo 可视化编程，依托参数化规则驱动构件属性识别，实现三维模型构件智能动态隔离，为智能审查提供精准靶向。

2 研究方法

2.1 文本预处理

建筑规范文档多以扫描件或图片型 PDF 形式存储，文字无法直接编辑或分析，需通过文本预处理实现从图像到结构化文本的转化。本研究针对规范文档的典型特征（中英文混排、多层级标题、表格干扰），设计了“OCR 识别+结构化清洗”的两阶段预处理流程，为后续语义解析与智能审查提供高质量输入。

2.1.1 OCR 图像文本转换

考虑到规范文档常以高分辨率 PDF 形式存档（含扫描件或矢量图），直接提取文本难度大，本研究采用 OCR（光学字符识别）技术实现图像到文本的转换。具体流程可见附录 A。

首先，通过 pdf2image 库将 PDF 页面转换为高分辨率图像（DPI 设为 400），确保文字边缘清晰，避免因分辨率不足导致的识别误差；其次，对图像进行预处理：转换为灰度图以降低计算复杂度，采用 5×5 高斯模糊核降噪，减少扫描噪声对文字识别的干扰；最后，基于 Tesseract 开源 OCR 引擎（配置中文简体+英文混合语言包 chi_sim+eng），使用页面分段模式 6（--psm 6，假设文本为

统一块分布）进行识别，兼顾中英文混合场景的识别精度。该步骤通过 ocr_pdf 函数实现，支持批量 PDF 处理并输出结构化 TXT 文本，解决了传统人工转录效率低、易出错的问题。

2.1.2 文本结构化清洗与解析

OCR 输出文本虽已转化为可编辑格式，但仍存在冗余空格、层级混乱、非目标章节干扰等问题，需进一步结构化处理。本研究针对建筑规范核心章节（通常为条款起始部分），设计了多级清洗策略，具体实现过程可见附录 B。

（1）内容定位：通过正则表达式匹配条款起始章节标题（如“3 基本规定”）及其变体，精准截取目标章节内容，排除规范文件中的非核心文本（如“本标准用词说明”）；

（2）空格优化：针对中文字符间冗余空格，采用正则替换([\u4e00-\u9fa5])\s+([\u4e00-\u9fa5])，保留中英文间的合理空格；

（3）层级解析：通过正则匹配数字编号（如“3.1”“3.1.1”），识别二级、三级标题层级，构建标题-内容的树状结构；同时过滤表格干扰（以“表”开头的行），避免非文本信息混入；

（4）格式统一：清理编号冗余空格（如“3 . 1”→“3.1”）、保留“注”开头的技术说明行，并通过合并连续文本行，最终输出层次分明、格式规范的结构化文本。该过程通过 process_regulations 函数实现，确保后续语义分析的准确性与效率。

文本预处理流程通过 OCR 解决图像转文本的核心问题，结合结构化清洗适配建筑规范的格式特征，为后续规范条款与 BIM 构件的语义关联奠定了高质量数据基础。

2.2 文本向量化

建筑规范条款与 BIM 构件的语义关联需通过数值化表征实现，文本向量化是连接自然语言与机器理解的核心桥梁。本研究针对规范文档的中文特性与专业术语密集特点，设计了“分词-嵌入-相似度计算”的三级向量化流程，其中 BGE 模型作为核心嵌入工具，有效解决了传统词袋模型语义表征不足的问题。以下从分词工具选择、嵌入技术演进及 BGE 核心技术展开详述。

2.2.1 中文分词与关键词提取：Jieba 工具的适配性

中文文本的向量化需以精准分词为前提。规范文档包含大量专业术语（如“抗震等级”“防火分区”）及长句结构，传统基于空格的分词方式无法满足需求。本研究选用 Jieba 分词工具，其采用基于前缀词典的最大概率分词算法，支持自定义词典扩展，在专业术语识别上表现优异。

具体的代码实现中（可见附录 C），通过“jieba.lcut”对 OCR 清洗后的规范内容进行精确模式分词，结合正则表达式过滤非中文字符（如数字、符号），并从分词结果中提取关键词（如“结构柱”“配筋率”）。代码中“process_regulation_content”函数通过正则匹配“### 条款 ID”后的核心内容，结合“re.sub”去除编号干扰，最终得到去重后的关键词列表。该步骤通过 Jieba 的高精度分词能力，为后续嵌入提供了语义粒度适宜的输入。

2.2.2 文本嵌入技术：从通用模型到 BGE 的专业适配

文本向量化本质是将自然语言映射到低维连续向量空间，使语义相似性可通过向量距离量化。早期常用方法（如 TF-IDF、Word2Vec）虽能实现文本数值化，但存在显著局限。TF-IDF 着重于词频统计衡量重要性，无法捕捉句中词语间的语义关联；Word2Vec 虽通过上下文学习词向量，但其“一词一义”的静态表征难以适应建筑规范中同一术语的多义性。因此，需引入更贴合自然语言语义特性的嵌入（Embedding）技术，通过动态学习文本的上下文语义，将词语或短语映射为富含语义信息的向量。

为进一步适配中文专业文本的语义复杂性，本研究选择 BGE 模型作为核心嵌入工具。BGE 是北京智源研究院针对中文场景优化的通用嵌入模型，其训练数据涵盖百科知识、专业文档、对话语料等多领域文本，通过改进的对比学习目标（Contrastive Learning）学习语义表征——模型在训练中不断拉近语义相似文本的向量距离、推远不相关文本的距离，从而在向量空间中保留更精准的语义关系。在本研究中，BGE 的核心作用是将规范关键词与 BIM 构件类别词映射为同一向量空间中的向量，通过向量间的余弦相似度量化二者的语义关联。

2.2.3 相似度计算：余弦相似度的语义关联量化

向量化的最终目标是通过数值运算衡量文本间的语义关联。本研究采用余弦相似度作为度量指标，其通过计算两个向量夹角的余弦值，反映向量在方向上的一致性（值越接近 1，语义越相似）。具体实现中（可见附录 C），首先将规范关键词与需要审查的模型构件类别词分别输入 BGE 模型编码，得到两组向量矩阵；随后通过“sklearn.metrics.pairwise.cosine_similarity”计算交叉相似度矩阵。代码中“_calculate_stats”函数进一步对相似度矩阵进行统计（最大值、平均值等），并通过 CSV 文件输出，为智能审查提供“哪些构件与选定条款强相关”的量化依据。

文本向量化流程通过 Jieba 实现精准分词与关键词提取，借助 BGE 模型将中文文本映射为富含语义的向量，最终通过余弦相似度量化规范条款与审查维度的语义关联，为后续“语义嵌入-参数匹配-实时反馈”的智能审查闭环提供了关键技术支撑。

2.3 可视化编程

2.3.1 Dynamo 工具的选用价值与核心特性

Dynamo 作为基于 Revit 的可视化编程工具，其核心优势在于“可视化编程”与“BIM 深度集成”的双重特性。其节点式编程界面通过拖拽功能节点快速构建数据处理流程，无需编写原始代码，显著降低技术门槛；同时嵌入 Revit 环境，可实时获取模型参数与视图信息，为“模型-文本-分析”跨领域数据融合提供原生支持。

本研究选用 Dynamo 的核心原因有三。其一，图形化节点替代代码编写，使建筑专业人员可直接参与数据处理流程设计；其二，可视化连接与实时错误提示，可快速定位逻辑漏洞（如路径错误、参数缺失）；其三，兼容 TXT/CSV/Excel 等文本数据及 Revit 几何属性数据，满足本研究对规范条款（CSV）、分析结果（JSON）与模型元素的多源输入需求。

2.3.2 ImportTXT.dyn：规范文本的标准化导入与外部分析联动

ImportTXT.dyn 文件是本研究实现“BIM 模型构件类别提取-规范文本读取-外部分析启动”流程的核心工具，其节点布置图（见图 2.1）完整呈现了从 Revit 视图中提取构件类别、导入规范文本到触发外部算法的全链路逻辑。

图 2.1 ImportTXT.dyn 节点布置图

（1）BIM 模型构件类别提取：流程起始于“All Elements In Active View”节点，该节点用于获取当前 Revit 激活视图中的所有构件元素；通过“Element.GetParameterValueByName”节点，从构件中提取所属类别词，输出为数组形式的数据。

（2）类别词数据分类：提取的类别词数组经“String from Array”节点转换为独立字符串，随后通过“Extraction of categories”节点进一步分类。

（3）外部分析启动：经预处理的规范文本，提取出的构件类别词列表以及结果输出路径将共同输入至“ExternalAnalysisLauncher”节点。该节点作为 Dynamo 与外部 Python 脚本的接口，将数据传递给外部分析工具，触发文本向量化、相似度计算等核心算法执行；最终分析结果（如相似性矩阵、统计摘要）回传至 Dynamo 流程。

2.3.3 IsolateElements.dyn：规范条款的筛选与目标构件隔离

IsolateElements.dyn 文件聚焦“规范条款筛选-相似性分析-目标构件隔离”的核心逻辑，其节点布置图（见图 2.2）通过模块化节点串联了从数据输入到结果输出的全流程。

图 2.2 IsolateElements.dyn 节点布置图

（1）数据输入：通过“StatusFilePath”节点指定研究运行状态文件路径，实时读取当前处理的条款 ID、已完成任务等状态信息；“ResultsFolderPath”节点设定 ImportTXT.dyn 的结果文件夹路径作为输入。“List Create”节点将读取的条款 ID 文件名称转化为列表的形式，并在“Regulations list”节点中展示，为用户提供待匹配的规范条款数据库。

（2）相似性分析与类别过滤：“RegulationCSVViewer”节点用于选择特定条款的文本向量化结果（即相似度矩阵）文件，并在“Similarity Matrix”节点中展示；核心功能节点“Filter Isolated Categories”通过设定阈值 Maximum Threshold，过滤掉相似度低于阈值的类别，仅保留高关联类别。

（3）目标构件隔离：最终通过“Isolate Elements”节点，从 BIM 模型中隔离出与高关联类别对应的构件（如关联度高于 0.890 的类别对应的所有构件），并在 Revit 视图中输出隔离结果，为后续规范审查提供精准的目标构件集。

Dynamo 通过可视化编程充分挖掘运用了 BIM 深度集成的特性，将 AI 规范分析封装为可视化工具，实现规范文档自动化分析与构件智能阈值过滤隔离，极大提升了模型审查效率。

3 案例验证

3.1 基本设计

3.1.1 建筑规范文本

（1）文件选取：《民用建筑设计统一标准》GB 50352-2019

（2）来源：中华人民共和国住房和城乡建设部公告 2019 年第 57 号（PDF 文件下载）

（3）核心章节：3. 基本规定；4. 规划控制；5. 场地设计；6. 建筑物设计；7. 室内环境；8. 建筑设备

（4）说明：总则（第 1 章）、术语（第 2 章）因适用性在预处理中被去除。

3.1.2 BIM 三维模型

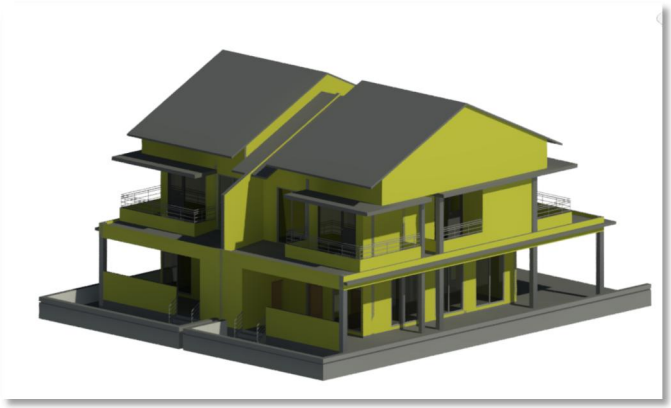

图 3.1 案例三维模型效果图

（1）模型来源：土木与建筑工程 CAE 模型化章节小作业进阶——双拼别墅

（2）模型类型：建筑模型

（3）主要构件类别：栏杆扶手、楼梯、栏杆扶手-顶部扶栏、窗、屋顶、结构柱、墙、楼梯-平台、门、楼板

（4）构件总数：243 个

（5）模型大小：12.3 MB

（6）软件版本：Autodesk Revit 2025

3.1.3 编程环境配置

（1）操作系统：Windows 11

（2）编程语言：Python 3.11.7

（3）开发工具：VS Code

（4）关键库：

表 3.1 本案例中使用的关键 Python 库

关键库	版本	关键库	版本
numpy	2.2.6	matplotlib	3.10.3
pandas	2.2.3	jieba	0.42.1
pytesseract	0.3.13	pdf2image	1.17.0
sentence_transformers	4.1.0	scikit_learn	1.6.1

3.2 应用流程

案例验证过程（见图 3.2）以《民用建筑设计统一标准》PDF 文档为对象，构建了从文本预处理到智能分析的全流程技术路线。

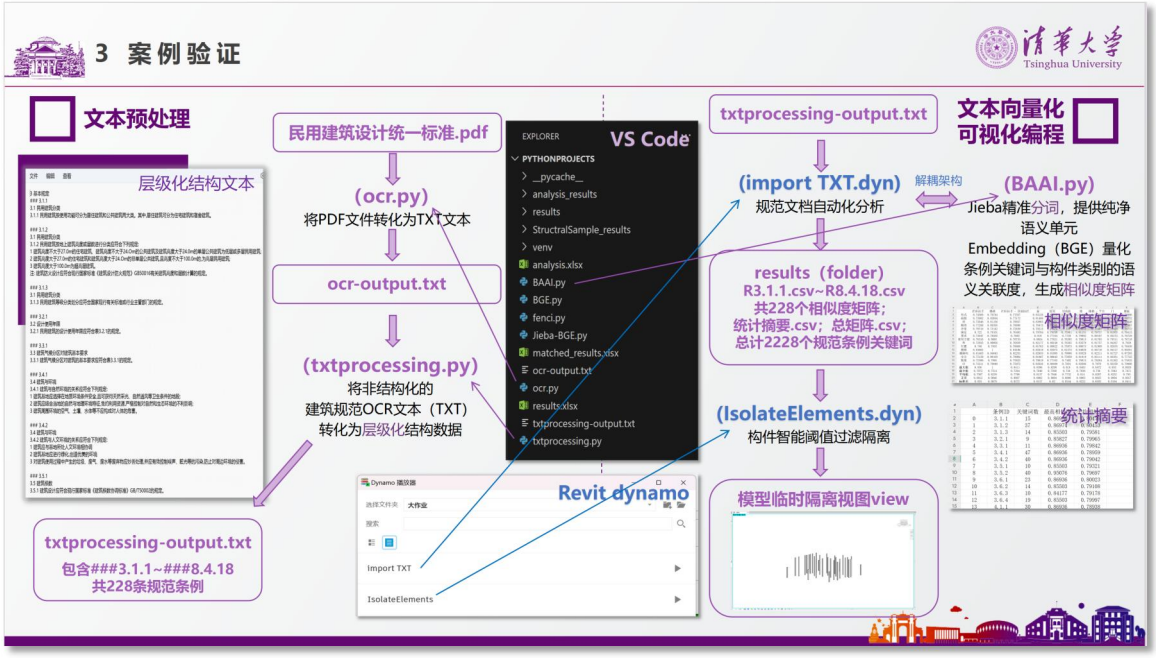

图 3.2 案例验证过程示意图

首先通过自主研发的 ocr.py 脚本实现 PDF 文档的结构化解析，利用 OCR 光学字符识别技术将非结构化图文混合内容转化为纯文本格式，生成包含 3.1.1 至 8.4.18 章节条款的 ocr-output.txt 文件。在此基础上，通过 txtprocessing.py 对 OCR 结果进行语义清洗与层级重构，针对建筑规范文本的条款编号、术语定义等特征，设计规则匹配算法去除冗余符号并修复断句错误，最终输出包含 228 条规范条例的结构化文本 txtprocessing-output.txt，为后续分析奠定数据基础。

文本向量化与可视化编程阶段，采用解耦式架构实现多工具协同。通过 import TXT.dyn 将结构

化文本导入 Dynamo 可视化编程环境，调用 BAAI.py 脚本实现 Jieba 精准分词，结合 BGE 预训练模型对条款文本进行向量化编码，生成包含 228 个规范条例关键词的语义向量矩阵。基于向量相似度计算结果，通过 results 文件夹输出 R3.1.1.csv 至 R8.4.18.csv 共 228 个相似度矩阵，同步生成统计摘要与总矩阵文件。在模型交互环节，利用 Dynamo 平台调用 IsolateElements.dyn，将向量化分析结果映射至建筑模型构件，通过智能阈值过滤实现条款与对应构件的临时隔离视图生成，直观呈现规范条款与审查要素构件的空间关联关系。整个流程形成“语义嵌入-参数匹配-实时反馈”的闭环，实现了规范文本与建筑模型的双向映射。

3.3 验证结果

通过上述技术流程，案例验证在数据处理精度、模型交互效率及分析可视化效果三方面取得显著成果。在文本处理层面，txtprocessing-output.txt 准确保留了原规范文档的 228 条条款结构，经人工抽检显示，条款编号识别准确率 100%，专业术语分词错误率低，证明了 OCR 与结构化处理算法的有效性。文本向量化结果显示，基于 BGE 模型的语义相似度计算能够有效筛选出特定条款的强关联构件（相似度＞0.9），如条款 6.11.1 为“门窗选用应根据建筑所在地区的气候条件、节能要求等因素综合确定,并应符合国家现行建筑门窗产品标准的规定”，其相似度矩阵如图 3.3 所示，最大值大于 0.9 的构件类别为窗（0.9508）和门（0.9321），验证了向量空间对规范与模型间语义关系的捕捉能力。

	A	B	C	D	E	F	G	H	I	J	K
1		栏杆扶手	楼梯	栏杆扶手	窗	屋顶	结构柱	墙	楼梯	平门	楼板
2	条件	0.73193	0.79008	0.76185	0.81884	0.78816	0.77943	0.80933	0.80537	0.83852	0.77079
3	的	0.75046	0.81156	0.78857	0.83965	0.8165	0.77242	0.84712	0.81401	0.85503	0.79448
4	应	0.72511	0.79848	0.75972	0.82626	0.80489	0.7691	0.82696	0.7979	0.83258	0.79996
5	根据	0.73728	0.78558	0.77229	0.79976	0.80108	0.77507	0.80367	0.80144	0.81315	0.77056
6	选用	0.74791	0.80004	0.78653	0.82183	0.79671	0.76836	0.80918	0.81018	0.81234	0.77788
7	地区	0.7324	0.80015	0.78185	0.83743	0.80657	0.7679	0.82166	0.82218	0.84864	0.79746
8	规定	0.77671	0.80639	0.80467	0.82094	0.78658	0.79185	0.81534	0.81604	0.83019	0.79785
9	确定	0.74652	0.79054	0.78117	0.8168	0.80315	0.80011	0.82256	0.80865	0.82525	0.78397
10	气候	0.70115	0.75843	0.73099	0.80736	0.80586	0.74682	0.78868	0.76712	0.79926	0.75642
11	因素	0.71671	0.77735	0.7461	0.80099	0.78412	0.77426	0.79984	0.7851	0.82085	0.77341
12	所在	0.74703	0.8147	0.79163	0.84251	0.83067	0.79687	0.83553	0.84463	0.85442	0.81234
13	节能	0.73964	0.7945	0.77782	0.82701	0.81455	0.78799	0.81086	0.80504	0.81361	0.77666
14	符合国家	0.70566	0.75738	0.74877	0.77563	0.76138	0.74845	0.77879	0.78399	0.79708	0.74417
15	等	0.74192	0.80533	0.78121	0.83024	0.79936	0.75116	0.82061	0.81191	0.83034	0.78783
16	综合	0.7667	0.82288	0.80418	0.85063	0.8221	0.79868	0.85149	0.83098	0.84622	0.82894
17	并	0.74288	0.82877	0.77597	0.8326	0.82164	0.78954	0.85198	0.83004	0.8516	0.81774
18	标准	0.73973	0.78892	0.77847	0.81297	0.7852	0.77959	0.8091	0.808	0.82852	0.77508
19	建筑	0.78375	0.84598	0.81175	0.84969	0.86349	0.85331	0.86936	0.84927	0.85745	0.84498
20	产品	0.75103	0.80465	0.78749	0.81493	0.77704	0.78015	0.80151	0.82175	0.834	0.78076
21	要求	0.73047	0.78265	0.7665	0.818	0.77164	0.7728	0.79892	0.80107	0.83272	0.76729
22	门窗	0.78635	0.81857	0.80257	0.95076	0.85306	0.80388	0.87854	0.82026	0.92307	0.82947
23	现行	0.74236	0.78111	0.7799	0.80947	0.7991	0.78259	0.8071	0.80722	0.82955	0.78661
24	最大值	0.7864	0.846	0.8118	0.9508	0.8635	0.8533	0.8785	0.8493	0.9231	0.845
25	最小值	0.7012	0.7574	0.731	0.7756	0.7614	0.7468	0.7788	0.7671	0.7971	0.7442
26	平均值	0.7429	0.7984	0.7782	0.8275	0.8042	0.7814	0.8208	0.811	0.8352	0.7898
27	方差	0.0005	0.0005	0.0004	0.0011	0.0006	0.0005	0.0007	0.0004	0.0007	0.0006
28	标准差	0.0218	0.0212	0.0201	0.0326	0.0245	0.0226	0.0255	0.0189	0.026	0.0249
29											

图 3.3 条款 6.11.1 相似度矩阵

在模型交互验证中，Revit Dynamo 生成的临时隔离视图实现了条款与构件的动态关联，视图响应时间控制在 2 秒左右，极大减少了人工筛选排查构件的时间，满足工程应用效率需求。此外，统计摘要文件显示，全规范文档累计提取 2228 个有效关键词，构建了覆盖功能分区、防火安全、无障碍设计等核心领域的术语体系，为后续规范自动检索与智能校验提供了标准化数据支撑。

4 总结展望

4.1 总结

本研究面向建筑行业数字化转型需求，提出了一种基于 BGE 嵌入与 Dynamo 集成的建筑规范智能审查方法。通过构建"语义嵌入-参数匹配-实时反馈"闭环机制，实现了规范条款与 BIM 构件的

自动化关联。主要完成的工作包括以下三个方面：

（1）首创技术框架：提出“BGE 嵌入+ Dynamo 集成”的智能审查框架，实现规范语义与 BIM 构件的自动化关联；

（2）突破流程瓶颈：通过结构化处理、BGE 向量化、Dynamo 可视化编程，打通“规范文本→向量分析→模型审查”全流程；

（3）验证应用价值：案例中成功解析 228 条规范，实现构件智能隔离，提升人工审查效率。验证表明该方法在文本处理精度、模型交互效率及分析可视化效果等方面达到工程实用水平，为建筑工程智能审查提供了新的技术路径。

4.2 展望

未来研究可从以下方向深化，

（1）数据驱动扩展：建立跨地域、多类型的建筑规范知识图谱，增强模型对地方性技术规程的适应性；

（2）深度学习融合：探索 BERT 等预训练模型与 BGE 的协同机制，提升复杂句式和专业术语的语义表征能力；

（3）实时性优化：研究轻量化模型压缩技术，开发云端-本地协同计算框架，满足大规模工程项目的实时审查需求；

（4）多规范兼容：构建支持 IFC 标准的中性文件接口，实现不同 BIM 平台与规范体系的互联互通。

本研究为建筑工程智能审查领域提供了可复用的技术框架，后续工作可以重点突破跨规范复合审查、动态施工过程合规性监控等关键方向。

参考文献

[1] 住房和城乡建设部. 住房和城乡建设部关于印发“十四五”住房和城乡建设科技发展规划的通知. [EB/OL](2022-03-01). https://www.gov.cn/zhengce/zhengceku/2022-03/12/content_5678693.htm

[2] 张吉松,于泽涵,李海江. 基于语义网的 BIM 结构模型合规性审查方法[J]. 图学学报,2023,44(2):368-379.

[3] 何清华,钱丽丽,段运峰,等. BIM 在国内外应用的现状及障碍研究[J]. 工程管理学报,2012,26(1):12-16.

[4] 林佳瑞,郭建锋. 基于 BIM 的合规性自动审查[J]. 清华大学学报（自然科学版）,2020,60(10):873-879.

[5] 邢雪娇,钟波涛,骆汉宾,等. 基于 BIM 的建筑专业设计合规性自动审查系统及其关键技术[J]. 土木工程与管理学报,2019,36(5):129-136.

[6] 甘晨. 基于 IFC 和本体的建筑施工图合规性审查研究[D]. 湖北:华中科技大学,2018.

[7] 魏然,舒赛,余宏亮,等. 自然语言建筑设计规范条文的规则表达式自动提取方法[J]. 土木工程与管理学报,2019,36(1):109-114,122.

[8] 孙澄宇,柯勋. 建筑设计中 BIM 模型的自动规范检查方法研究[J]. 建筑科学,2016,32(4):140-145.

附录 A　ocr.py 脚本

```python
# -*- coding: utf-8 -*-
import pytesseract
import cv2
import numpy as np
from pdf2image import convert_from_path
import os

# 配置 Tesseract 路径（需根据实际安装路径修改）
pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe'

def ocr_pdf(pdf_path, output_dir='.'):
    """对 PDF 文件进行 OCR 识别并保存为 TXT 文本"""
    try:
        # 创建输出目录（如果不存在）
        os.makedirs(output_dir, exist_ok=True)

        # 将 PDF 转换为图像列表（设置 DPI 为 400 保证清晰度）
        pages = convert_from_path(pdf_path, dpi=400)

        all_text = []
        for i, page in enumerate(pages):
            # 图像预处理流程
            img = cv2.cvtColor(np.array(page), cv2.COLOR_RGB2BGR)  # 转换为 OpenCV 格式
            gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)           # 灰度化处理
            blur = cv2.GaussianBlur(gray, (5,5), 0)                # 高斯模糊降噪

            # 使用 Tesseract 进行 OCR 识别（中英文混合+优化参数）
            text = pytesseract.image_to_string(
                blur,
                lang='chi_sim+eng',              # 中文简体+英文语言包
                config='--psm 6'                 # 页面分段模式 6（假设文本为统一块）
            )
            all_text.append(text)
            print(f"已完成第 {i+1}/{len(pages)} 页处理")

        # 保存所有识别结果
        output_path = os.path.join(output_dir, 'ocr-output.txt')
        with open(output_path, 'w', encoding='utf-8') as f:
            f.write('\n\n'.join(all_text))  # 用双换行分隔不同页面
        return True, output_path
    except Exception as e:
        return False, str(e)

def main():
    """程序主入口"""

    while True:
        # 获取用户输入
        pdf_path = input("\nEnter full PDF path (or 'q' to quit): ").strip()
        if pdf_path.lower() == 'q':
            print("Exiting program...")
            break

        # 验证文件是否存在
        if not os.path.isfile(pdf_path):
            print(f"Error: File not found - {pdf_path}")
            continue

        # 执行 OCR 处理
        print(f"\nProcessing: {pdf_path}...")
        success, result = ocr_pdf(pdf_path)

        # 显示处理结果
        if success:
            print(f"\nConversion successful!")
            print(f"Output saved to: {result}")
        else:
            print(f"\nConversion failed:")
            print(f"Error details: {result}")

        print("Press any key to continue...")
        input()

if __name__ == "__main__":
    main()
```

附录 B　txtprocessing.py 脚本

```
import re

def process_regulations(input_path, output_path):
    """处理规范文档，提取结构化内容并格式化输出"""

    # 1. 文件读取和初始处理
    with open(input_path, 'r', encoding='utf-8') as f:
        lines = f.readlines()

    target_index = -1
    end_index = -1

    # 定位起始行（匹配"3 基本规定"及其变体）
    for i, line in enumerate(lines):
        line_stripped = line.strip()
        if re.match(r'^3[\.\s\u3000]*基 本 规 定', line_stripped):
            target_index = i
            Break

    # 定位结束行（匹配"本标准用词说明"）
    if target_index != -1:
        for i in range(target_index, len(lines)):
            line_stripped = lines[i].strip()
            if re.match(r'^本 标 准 用 词 说 明', line_stripped):
                end_index = i
                Break

    # 截取目标内容区间
    content_lines = []
    if target_index != -1:
        content_lines = lines[target_index:end_index] if end_index != -1 else lines[target_index:]
    else:
        content_lines = lines
    content = ''.join(content_lines)

    # 处理中文字符间多余空格（保留中英文之间的空格）
    content = re.sub(r'([\u4e00-\u9fa5])\s+([\u4e00-\u9fa5])', r'\1\2', content)

    # 2. 文档结构解析
    processed = []
    stack = []  # 用于跟踪标题层级
    current_indent = 0
    in_table = False  # 表格内容标记

    # 文档分段处理
    current_level2 = ""  # 当前二级标题缓存
    sections = []        # 存储所有段落
    current_section = [] # 当前处理的段落

    for line in content.split('\n'):
        line = line.strip()
        # 跳过空行和纯数字行
        if not line or re.fullmatch(r'^\d+$', line):
            Continue

        # 表格处理逻辑
        if line.startswith('表'):
            in_table = True
            continue
        if in_table and (not line or re.match(r'^(\d+(?:[\.\s]+\d+)+)[\s  ]+', line)):
            in_table = False
        if in_table:
            Continue

        # 标题行处理（匹配数字编号开头的行）
        if match := re.match(r'^(\d+(?:[\.\s]+\d+)+)[\s  ]+', line):
            # 规范化编号格式（统一为X.X.X格式）
            number_part = re.sub(r'[\.\s]+', '.', match.group(1)).strip('.')
            levels = number_part.split('.')

            # 二级标题处理（X.X）
            if len(levels) == 2:
                current_level2 = line
                # 分段处理：遇到新二级标题时保存当前段落
                if current_section:
                    sections.append(current_section)
                    current_section = []
                current_section.append(("level2", current_level2))
```

```
            # 三级标题处理（X.X.X）
            elif len(levels) == 3:
                # 遇到新三级标题时保存当前段落
                if current_section and any(t[0] == "level3" for t in current_section):
                    sections.append(current_section)
                    current_section = []
                    # 保留关联的二级标题
                    if current_level2:
                        current_section.append(("level2", current_level2))
                current_section.append(("level3", line))

            # 其他层级标题
            else:
                current_section.append(("other", line))

            # 更新缩进级别（用于后续格式化）
            while stack and len(stack) >= len(levels):
                stack.pop()
            stack.append(number_part)
            current_indent = (len(levels) - 1) * 4

        else:
            # 普通文本行处理
            if current_section:
                current_section.append(("text", line))
            else:
                current_section.append(("text", line))

    # 添加最后一个段落
    if current_section:
        sections.append(current_section)

    # 3. 内容重组与格式化输出
    final_content = []
    for section in sections:
        # 检查是否包含三级标题
        has_level3 = any(t[0] == "level3" for t in section)

        if has_level3:
            level3_items = [t for t in section if t[0] == "level3"]
            level2_items = [t for t in section if t[0] == "level2"]

            # 处理每个三级标题块
            for level3_item in level3_items:
                # 三级标题解析（支持带空格的编号如"3. 1.1"）
                title_match = re.match(r'^(\d+)\s*[\.\s]+\s*(\d+)\s*[\.\s]+\s*(\d+)\s*(.*)',
level3_item[1].strip())
                if title_match:
                    # 规范化编号（统一为X.X.X格式）
                    level3_num = f"{title_match.group(1)}.{title_match.group(2)}.{title_match.group(3)}"
                    level3_text = title_match.group(4)
                    # 生成Markdown三级标题（### X.X.X）
                    final_content.append(f"\n### {level3_num}")
                else:
                    # 备用匹配模式（处理其他编号格式）
                    alt_match = re.match(r'^(\d+(?:[\.\s]+\d+)+)\s*(.*)', level3_item[1].strip())
                    if alt_match:
                        level3_num = re.sub(r'[\.\s]+', '.', alt_match.group(1)).strip('.')
                        level3_text = alt_match.group(2)
                        final_content.append(f"\n### {level3_num}")
                    else:
                        final_content.append("\n###")

                # 添加关联的二级标题（如果有）
                if level2_items:
                    final_content.append(level2_items[0][1])

                # 添加所有相关内容（包括三级标题本身）
                for item in section:
                    if item[0] == "text" or item[0] == "level3":
                        final_content.append(item[1])
        else:
            # 无三级标题的内容直接输出
            for item in section:
                final_content.append(item[1])

    # 4. 最终格式清理
    cleaned_content = []
    # 处理数字编号格式（如"3 . 1" → "3.1"）
```

```
    number_pattern =
r'(?P<number>(\b\d+)(\s*[\.\s\u3000]+\s*)(\d+(?:\s*[\.\s\u3000]+\s*\d+)*\b))(?P<other>.*)'
    for line in final_content:

        # 处理标准编号格式（X.X.X）
        if re.match(number_pattern,line) is not None:
            _l = re.match(number_pattern,line)
            number = _l.group("number").replace(" ","")
            other = _l.group("other").replace(" ","")
            line = number + " " + other
            cleaned_content.append(line)

        # 处理简单数字编号（如"1)"）
        elif re.match(r"(?P<number>\d{1}\)?)(?P<other>[^\d\.]{1}.*)",line):
            _l = re.match(r"(?P<number>\d{1}\)?)(?P<other>[^\d\.]{1}.*)",line)
            number = _l.group("number").replace(" ","")
            other = _l.group("other").replace(" ","")
            line = number + " " + other
            cleaned_content.append(line)

        # 保留"注"开头的行（不处理空格）
        elif re.match(r"注.*",line):
            cleaned_content.append(line.replace(" ",""))

        # 保留换行符
        elif re.match(r"\n",line):
            cleaned_content.append(line)

        # 其他内容合并处理
        else:
            line = line.replace(" ","")
            cleaned_content[-1] += line

    # 5. 结果输出
    with open(output_path, 'w', encoding='utf-8') as f:
        f.write('\n'.join(cleaned_content))

if __name__ == "__main__":
    process_regulations(input("输入文件路径："), "txtprocessing_output.txt")
```

附录 C　BAAI.py 脚本

```
import re
import jieba
import pandas as pd
import numpy as np
import os
os.environ['HF_ENDPOINT'] = 'https://hf-mirror.com'
import json
import time  # 添加缺失的 time 模块导入
import argparse
from pathlib import Path
from sentence_transformers import SentenceTransformer
from sklearn.metrics.pairwise import cosine_similarity

def load_document(file_path):
    """增强版文件加载"""
    try:
        with open(file_path, 'r', encoding='utf-8') as f:
            content = f.read()
            if not content:
                raise ValueError(f"文件 {file_path} 为空")
            return content
    except Exception as e:
        raise IOError(f"文件加载失败: {str(e)}")

def process_regulation_content(doc, rule_id):
    """优化分词逻辑"""
    pattern = rf'### {rule_id}\n(.*?)(?=\n###|\Z)'
    match = re.search(pattern, doc, re.DOTALL)
    if not match:
        print(f"警告: 未找到条例 {rule_id}")
        return []
    content = match.group(1).strip()
    cleaned = re.sub(r'^\d+(\.\d+)*\s*', '', content, flags=re.MULTILINE)
    words = [w for w in jieba.lcut(cleaned)
            if len(w) >= 1 and re.match(r'^[\u4e00-\u9fa5]+$', w)]
    return list(set(words))

class EnhancedBatchProcessor:
    def __init__(self, categories, model_name='BAAI/bge-base-zh'):
        self.model = SentenceTransformer(model_name) # BGE-base-zh 是北京智源研究院开发的中文优化模型
        self.categories = [c.replace('[', '').replace(']', '').strip()
                           for c in categories.split(',') if c.strip()] # 处理输入的类别字符串:
        self.summary = [] # 存储每个条例的统计摘要
        self.master_df = pd.DataFrame() # 存储所有条例的相似度矩阵（合并后）
        self.generated_files = [] # 记录生成的所有 CSV 文件名

    def _sorted_rules(self, rule_ids):
        return sorted(set(rule_ids),
                      key=lambda x: tuple(map(int, x.split('.'))))

    def _calculate_stats(self, df):
        """计算统计指标"""
        stats = pd.concat([
            df.max().round(4).rename('最大值'),
            df.min().round(4).rename('最小值'),
            df.mean().round(4).rename('平均值'),
            df.var().round(4).rename('方差'),
            df.std().round(4).rename('标准差')
        ], axis=1).T
        return pd.concat([df, stats], axis=0)

    def _save_csv(self, df, filename, output_folder):
        """统一保存 CSV 并记录"""
        path = Path(output_folder) / filename
        df.to_csv(path, encoding='utf-8-sig')
        self.generated_files.append(filename)
        return path

    def process_all(self, doc_path, output_folder):
        """增强版批量处理"""
        os.makedirs(output_folder, exist_ok=True)
        doc_content = load_document(doc_path)

        rule_ids = re.findall(r'### (\d+\.\d+\.\d+)\n', doc_content)
        for rid in self._sorted_rules(rule_ids):
            try:
                keywords = process_regulation_content(doc_content, rid)
                if not keywords:
```

```
                    continue

                cat_emb = self.model.encode(self.categories)
                key_emb = self.model.encode(keywords)
                sim_matrix = cosine_similarity(cat_emb, key_emb)

                df = pd.DataFrame(sim_matrix.T,
                                  index=keywords,
                                  columns=self.categories).round(5)
                df_with_stats = self._calculate_stats(df)

                # 保存单个条例 CSV
                csv_name = f"R{rid}.csv"
                self._save_csv(df_with_stats, csv_name, output_folder)

                # 更新总矩阵
                self.master_df = pd.concat([self.master_df, df_with_stats.iloc[:-5]])

                # 更新摘要
                self.summary.append({
                    '条例 ID': rid,
                    '关键词数': len(keywords),
                    '最高相似度': sim_matrix.max().round(5),
                    '平均相似度': sim_matrix.mean().round(5)
                })

            except Exception as e:
                print(f"处理条例 {rid} 出错: {str(e)}")
                continue

        # 保存汇总文件
        if not self.master_df.empty:
            self.master_df = self.master_df[~self.master_df.index.duplicated(keep='first')]
            self._save_csv(self._calculate_stats(self.master_df), "总矩阵.csv", output_folder)
            self._save_csv(pd.DataFrame(self.summary), "统计摘要.csv", output_folder)

        return self.generated_files

def main():
    parser = argparse.ArgumentParser()
    parser.add_argument("--config", type=str, required=True)
    args = parser.parse_args()

    with open(args.config, 'r', encoding='utf-8') as f:
        config = json.load(f)

    status = {
        "status": "processing",
        "progress": 0,
        "result_files": [],
        "output_dir": config["output_dir"],
        "start_time": time.strftime("%Y-%m-%d %H:%M:%S")  # 添加时间戳
    }

    try:
        # 写入初始状态
        with open(config["status_file"], 'w', encoding='utf-8') as f:
            json.dump(status, f, ensure_ascii=False)

        # 执行处理
        processor = EnhancedBatchProcessor(config["categories"])
        result_files = processor.process_all(config["txt_path"], config["output_dir"])

        # 更新完成状态
        status.update({
            "status": "completed",
            "result_files": result_files,
            "completion_time": time.strftime("%Y-%m-%d %H:%M:%S"),
            "processing_duration": f"{time.time() - time.mktime(time.strptime(status['start_time'],
'%Y-%m-%d %H:%M:%S')):.2f}秒"
        })

    except Exception as e:
        status.update({
            "status": f"failed: {str(e)}",
            "error_details": str(e),
            "completion_time": time.strftime("%Y-%m-%d %H:%M:%S")
        })

    finally:
        with open(config["status_file"], 'w', encoding='utf-8') as f:
```

```
            json.dump(status, f, ensure_ascii=False, indent=4)

if __name__ == "__main__":
    main()
```