ChatGPT

SciCapenter 系统复现计划

本项目旨在一个月内完整复现《SciCapenter: Supporting Caption Composition for Scientific Figures with Machine-Generated Captions and Ratings》系统，包括网页前端、PDF上传与内容提取、AI图注生成、评分机制、检查清单生成与交互优化等功能。根据论文描述，SciCapenter为每个图表生成多种候选图注，并针对“有用性、OCR提及、关键信息、视觉属性”等维度给出评分和检查清单 [1] 。下面按天拆解任务，每日投入约4小时进行学习与开发。每个阶段都有具体可操作项，并推荐相关工具和学习资源。

第1天：项目需求梳理与环境准备

- **任务**：阅读SciCapenter论文摘要和相关资料，明确系统模块（PDF处理、AI图注、评分、交互界面等）[1]；梳理项目所需技术栈（Python、Web框架、PyMuPDF、Transformers等）。
- **环境搭建**：安装Python和开发环境（IDE或Jupyter）；使用`pip`安装基础库（如`pymupdf`、`transformers`、`streamlit`或`gradio`等），测试环境是否正常。
- **学习资源**：阅读官方文档和快速入门教程，如PyMuPDF教程、HuggingFace入门指南、Streamlit/Gradio快速入门等。建议先熟悉Python包管理和虚拟环境工具（venv/conda） [2] 。

第2天：PDF内容提取基础

- **重点学习**：使用PyMuPDF（pip名为`PyMuPDF`或`fitz`）进行PDF文本和图像提取。例如，利用`page.getImageList()`列出页内图片标识，并调用`doc.extractImage(xref)`获取图片数据 [2] 。
- **任务**：编写Python脚本，读取示例PDF并提取所有页的嵌入图像。记录提取到的图片数量和格式。
- **深度拓展**：了解AllenAI的PDFFigures2工具，它专注于从学术论文中提取图表、图注和表格 [3] 。如果时间允许，可尝试使用其CLI命令提取图表，或学习其输出格式。
- **参考资源**：PyMuPDF 文档 [2] 、AllenAI PDFFigures2 GitHub说明 [3] 。

第3天：PDF文本提取与处理

- **任务**：继续使用PyMuPDF，从PDF中提取图注文本和周边文字信息。尝试`page.get_text()`方法获取页面文字，并定位图注所在区域。
- **实践**：选取带图注的PDF页，验证提取的文字中包含“Figure 1”、“图1”等关键词。结合OCR技术（下一步学习）准备比对图片和文字。
- **工具学习**：熟悉Python常用的文本处理库（如`re`正则、字符串方法）以清洗提取内容。确保会使用PyMuPDF的其它功能，如`page.get_pixmap()`渲染页面快照 [4] 。

第4天：AI图像描述模型学习

- **重点学习**：HuggingFace Transformers 库中的“image-to-text”或视觉-文本模型。参考文档示例，使用`pipeline("image-to-text")`调用预训练模型生成图像描述 [5] 。例如：

```
from transformers import pipeline
captioner = pipeline("image-to-text", model="Salesforce/blip-image-captioning-base")
caption = captioner("example_figure.png")[0]["generated_text"]
```

- **任务**：安装并测试一个简单的图像描述模型。使用几张科学图表（可手动截图）检验生成的文本效果。重点理解模型输入输出格式。
- **学习资源**：HuggingFace 任务文档 [5]（展示了使用BLIP模型进行图注生成的示例）。可以参考相关教程学习如何加载视觉编码器和文本解码器模型 [6]。

第5天：基础AI图注生成模块开发

- **任务**：基于前一天学习，编写Python函数实现：输入图像文件，输出图像描述文字。可使用HuggingFace管道或`VisionEncoderDecoderModel`搭配ViT/GPT2模型。多尝试不同模型（如BLIP、ViT-GPT2等），记录表现。
- **扩展**：如果GPU可用，可考虑微调小型模型；否则直接使用推理模式。保存生成的候选图注文本供后续展示。
- **资源**：阅读HuggingFace VisionEncoderDecoder示例 [6] 和博客“使用HuggingFace进行图像描述”教程以加深理解。

第6天：图注评分与检查清单设计

- **任务**：研究图注质量评估方法。SciCapenter使用检查清单评估图注是否包含关键内容 [1]。根据论文描述，设计类似的检查项：是否提及图中文字（OCR提及）、是否描述了主要趋势或结论、是否包含颜色/图例等视觉元素、整体表达有用性。
- **技术实现**：准备使用OCR库（如Tesseract的Python绑定`pytesseract`）检测图像中的文字 [7]。编写代码提取图中真实文字，然后检查生成图注中是否包含这些词汇。
- **评分机制**：可以先采用简单规则打分，例如每满足一项得分，或使用预定义关键词列表。后续可进一步引入BLEU、BERTScore等参考式指标。
- **参考资源**：Tesseract OCR 教程 [7]（说明了`tesseract`与`pytesseract`用于图像文字识别的基本用法）。

第7天：小型端到端工作流验证

- **任务**：结合前几天的工作，实现一个简单命令行流程：接受PDF文件，提取首个图表图片，生成若干候选图注，评估它们并打印检查清单和评分。
- **目标**：验证各模块能串联起来运行，无重大错误。记录性能瓶颈（如模型加载时间）为后续优化做准备。
- **检查点**：确保提取的图片能被图像描述模型正确加载；简单模型生成的文字粗略合理；评分逻辑能给出可读输出。

第8天：Web前端框架选型与学习

- **任务**：比较Streamlit和Gradio等快速Web UI工具。两者都能快速搭建上传图片、展示结果的界面。可以先浏览官方示例，了解如何创建文件上传器、图片显示区、文本输入框等组件。
- **学习资源**：Streamlit 文档和示例（如文件上传`st.file_uploader`、图像显示`st.image`等）；Gradio 文档（其`gr.Image()`组件支持上传或展示图片 [8]）。选择一个框架作为主要开发工具。
- **扩展选项**：如果对前端技术有一定了解，也可考虑Flask/Django搭配HTML/CSS，或学习React/Vue基础。但考虑时间，建议优先掌握高效的现成组件。

第9天：实现基本交互界面

- **任务**：使用Day8选定的框架创建原型界面：包括上传PDF或图片的功能、显示提取出的图表和模型生成的图注。

- **细节**：界面上放置一个文件上传按钮；用户上传PDF后调用后端提取图像，显示在页面上；用户可点击“生成图注”按钮，显示AI生成的图注候选列表。可先实现最简单的单图、单注流程。
- **测试**：用前面提取的样例PDF测试上传功能；确保UI能调用Python后端并刷新显示结果。

第10天：多候选图注与编辑反馈

- **任务**：让系统为每张图生成多个图注候选。例如，使用不同随机种子或beam search策略，或同时调用多个模型。将它们在界面上按序号列表显示。
- **交互**：为用户提供文本编辑框，使其能对任意候选图注进行修改。添加“重新评估”按钮，用户修改后触发对新文本的评分和检查清单更新。
- **细节**：保证编辑后的文本能正确传回后端进行再次处理。可以在界面上直观展示每个步骤：原图、候选一、候选二……以及它们各自的评分和检查项。

第11天：检查清单生成与显示

- **任务**：根据第6天设计的评估维度，生成具体的检查表（检查清单）。比如，对每个图注列出是否满足：“包含图中OCR提取的字词”、“描述了主要趋势”、“提到了颜色/图例”等。
- **实现**：调用OCR提取图中文字（pytesseract），并在表格中标记图注中是否提到了这些词 [7] [1]；其他指标可以基于关键词或简单分类算法判定。
- **输出**：在界面中以表格形式展示各候选图注在每项指标上的得分/勾选情况。确保布局清晰，例如使用Streamlit的`st.table`或Gradio的`gr.Dataframe`。
- **参考**：可借鉴SciCapenter提出的质量检查维度 [1]，虽然具体实现可简化，但至少覆盖OCR提及、关键信息（例如数据点、结论）、视觉元素等方面。

第12天：迭代优化与用户反馈

- **任务**：优化交互流程，例如添加按钮让用户切换不同图表或重置界面。实现用户修改图注后，自动刷新评分和检查表。
- **细节**：考虑使用缓存功能（如Streamlit的`@st.cache`）避免重复计算。保持会话状态（Session State）以支持多次交互。
- **收尾**：测试不同场景，比如用户不满意第一个生成结果，编辑后再评估，检查所有模块仍然稳定工作。

第13天：数据存储与后台支持

- **任务**：学习使用MongoDB等数据库在后台存储信息。可以将用户上传的PDF元数据、生成的图注和评分结果保存起来，方便统计或后续研究。
- **实践**：参考Streamlit官方示例，使用`pymongo`连接MongoDB并写入文档 [9]。例如，在用户点击“保存”时，将该记录插入数据库集合。
- **资源**：Streamlit 与 MongoDB 集成指南 [9] 提供了基础样例。对于Gradio，可在后端脚本中使用PyMongo。需要先在本地或云端安装并启动MongoDB。

第14天：前端界面美化与功能完善

- **任务**：改进界面布局，使其更清晰易用。比如，将上传区、图像显示、候选列表、评分表分别列为不同的区域或页签。
- **学习**：如果使用Streamlit，可学习其布局组件如`st.columns`、`st.tabs`；Gradio 可使用`Blocks`自定义网格布局。为交互元素加上说明文字和样式。
- **资源**：参考Streamlit官方博客、Gradio示例工程。确保最终界面操作直观，并检查响应性（尤其是在处理大PDF时的加载提示）。

第15天：组件整合与测试

- **任务**：整合前14天的所有模块，形成可运行的完整Demo。进行全面测试：上传不同的论文PDF，测试图表抽取、图注生成、评分、用户编辑等功能是否协同工作。
- **修复**：记录遇到的问题并逐一解决，例如：PDF处理失败、模型生成慢、界面卡顿等。优化模型推理速度（可考虑图像尺寸缩放）、分页加载结果等。
- **总结**：整理复现过程，确认所有核心需求都已实现。根据实际情况调整功能优先级，如可能简化某些复杂指标。

第16–30天：完善与展示准备

- **第16–17天**：深入优化。可尝试引入更强的模型（如CLIP+LM结合）生成图注；完善评分算法，引入辅助模型（如语言模型检查流畅度）。
- **第18–20天**：撰写项目文档，包括系统架构说明、用户手册和代码注释。准备演示用例和结果截图。
- **第21–25天**：进行用户测试（可请同学试用），根据反馈再做改进，确保界面和功能稳定。完善前端细节，如添加Logo、版权信息等。
- **第26–27天**：整合资源，将系统部署在可访问的环境（本地服务器或云平台）。准备演示视频或PPT，突出系统亮点。
- **第28–30天**：最终检查，修复遗留小问题，确保demo完整流畅。以高质量的形式向指导老师或同学展示复现成果。

关键知识与工具：建议掌握PyMuPDF（PDF提取）[2]、pdffigures2（科研图表提取）[3]、HuggingFace Transformers（图像描述模型）[5]、Tesseract OCR[7]、Streamlit/Gradio（快速Web UI）[8]、MongoDB/PyMongo（数据存储）[9]等。上述资源和官方文档将有助于快速上手，每日坚持学习和实践，可在一个月内完成高质量的复现演示。

---

[1] [2403.17784] SciCapenter: Supporting Caption Composition for Scientific Figures with Machine-Generated Captions and Ratings
https://arxiv.org/abs/2403.17784

[2] Extract all Images from PDF in Python | by Ali Aref | Medium
https://aliarefwriorr.medium.com/extract-all-images-from-pdf-in-python-cda3dc195abd

[3] GitHub - allenai/pdffigures2: Given a scholarly PDF, extract figures, tables, captions, and section titles.
https://github.com/allenai/pdffigures2

[4] Images - PyMuPDF documentation
https://pymupdf.readthedocs.io/en/latest/recipes-images.html

[5] What is Image-to-Text? - Hugging Face
https://huggingface.co/tasks/image-to-text

[6] Vision Encoder Decoder Models
https://huggingface.co/docs/transformers/v4.17.0/en/model_doc/vision-encoder-decoder

[7] Python Tesseract OCR: Extract text from images using pytesseract
https://www.nutrient.io/blog/how-to-use-tesseract-ocr-in-python/

[8] Gradio Docs
https://www.gradio.app/docs/gradio/image

[9] Connect Streamlit to MongoDB - Streamlit Docs
https://docs.streamlit.io/develop/tutorials/databases/mongodb