Deepseek企业级Agent项目开发实战

Part 6. Microsoft GraphRAG 多模态PDF集成及应用及优化策略

继上一节重点介绍了`Microsoft GraphRAG`中`.csv`文件的默认索引策略，我们采取的优化方式是将自然语言和非自然语言的文本进行分开索引，其中非自然语言类数据通过人工提取节点、关系和属性直接导入`Neo4j`数据库，而自然语言类数据则是需要先做语义增强，再按照`Microsoft GraphRAG`的默认文件加载器进行加载，并自定义集成了`.csv`格式的文档切分器，通过动态`Token`+上下文特殊字符标识的定制化切分策略保证`.csv`中每一行文本的完整性。

因此大家需要明确的是：`.csv`格式文件的优化策略是建立在自然语言类数据的基础之上的，并且沿用了`Microsoft GraphRAG`的默认`.csv`文件加载器，我们对`Microsoft GraphRAG`源码做二次开发并集成的是`.csv`格式的文档切分器。

本节重点介绍的是`PDF`格式文件在`GraphRAG`中的应用。`Microsoft GraphRAG`截止目前最新的`v2.1.0`版本是完全不支持`PDF`格式文件的，本节要做是：在`Microsoft GraphRAG`源码基础上二次开发的将是全新的`PDF`格式文件加载器，以及自定义的`PDF`格式文件切分器，最后再集成到`Microsoft GraphRAG`中。同时，除了给大家详细讲解二次开发的代码逻辑，在这个过程中也将重点给大家介绍`PDF`格式文件的切分难点，以及我们是如何一步一步解决这些切分难点和制定优化策略的。

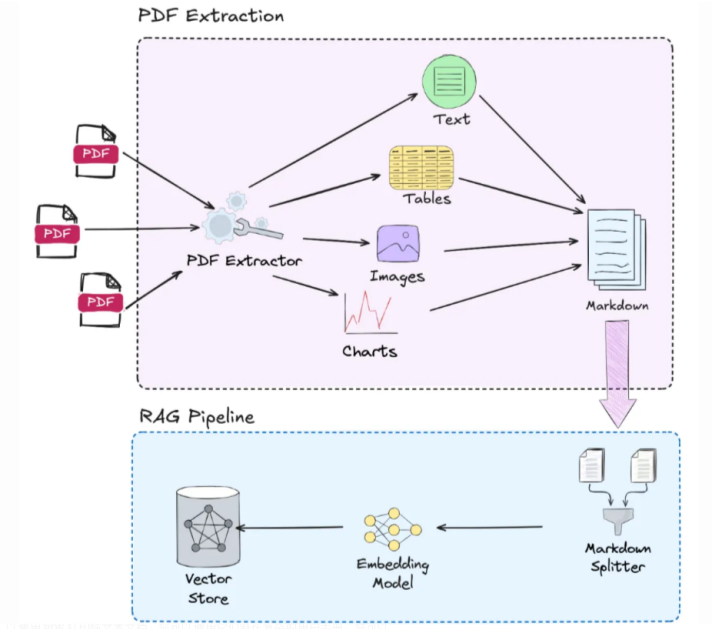
明确了具体的任务目标后，首先，我们就来先了解一下`PDF`格式文件的特点，以及这种格式文件在`GraphRAG`中各个阶段的应用难点。

1. PDF文件难点与解析方法总览

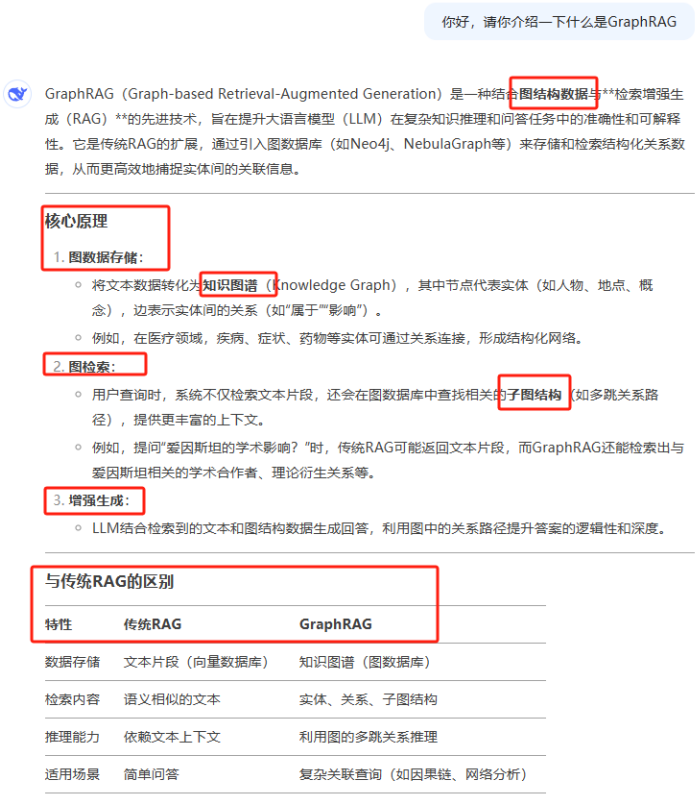
无论是传统`RAG`还是`GraphRAG`，`PDF`格式文件都是最难处理的文件类型，没有之一。主要原因在于`PDF`格式文件的结构非常复杂，包含文本、图像、表格等多种类型的数据，这几种不同类型的每一部分数据对于`RAG`来说都非常重要，因为很多上下文信息都是通过标题、图片、图表和格式等来传达的，我们需要保留这些信息并进行有效的存储，才能够保证在检索阶段正确识别出有效的文本，从而使大模型能够更好地确定如何“思考”给定内容中提供的信息。

我们在实际构建`RAG`的第一步，往往是先做`PDF Parsing`，即`PDF`解析。所谓的`PDF Parsing`，指的是提取`PDF`文件中的内容并将其转换为结构化格式的过程，对于`PDF`格式的这种文件特点，就不能像处理`.txt`、`.csv`、`.json`等文件那样，直接使用`Python`的`open`函数打开文件，然后逐行读取文件内容，然后进行处理。它涉及分析`PDF`文件的结构和内容以提取有意义的信息，即把`PDF`文件中的文本、图像、表格和元数据等正确的识别出来，同时还需要解析出`PDF`文件的结构信息，比如：页眉、页脚、页码、章节、段落、标题等。

`RAG`系统往往是需要依靠高质量的结构化数据来生成准确且与文本相关的输出。`PDF`通常用于官方文件、业务报告和法律合同，包含大量信息，但其布局复杂且数据结构不合理。如果做不到精确的`PDF`解析，关键数据就会丢失，从而导致结果不准确并直接影响`RAG`应用程序的有效性。因此，`PDF`格式文件的索引和检索的流程，在`RAG`框架中的实现过程如下图所示：



在 RAG 处理 PDF 格式文件的解析过程中，将其先转换为 Markdown 文件再做后处理是目前通用的做法。 Markdown 格式文件自 2023 年起就一直是在大模型领域的最流行格式，像 ChatGPT 、 DeepSeek 等聊天机器人格式化其响应的方式都是使用的 Markdown 语法。如下所示： DeepSeek 的响应会以大而粗的字体呈现标题，以及通过使用粗体文本表示关键字。



你好，请你介绍一下什么是GraphRAG

GraphRAG（Graph-based Retrieval-Augmented Generation）是一种结合**图结构数据**与**检索增强生成（RAG）**的先进技术，旨在提升大语言模型（LLM）在复杂知识推理和问答任务中的准确性和可解释性。它是传统RAG的扩展，通过引入图数据库（如Neo4j、NebulaGraph等）来存储和检索结构化关系数据，从而更高效地捕捉实体间的关联信息。

核心原理

1. **图数据存储**：
   - 将文本数据转化为**知识图谱**（Knowledge Graph），其中节点代表实体（如人物、地点、概念），边表示实体间的关系（如"属于""影响"）。
   - 例如，在医疗领域，疾病、症状、药物等实体可通过关系连接，形成结构化网络。
2. **图检索**：
   - 用户查询时，系统不仅检索文本片段，还会在图数据库中查找相关的**子图结构**（如多跳关系路径），提供更丰富的上下文。
   - 例如，提问"爱因斯坦的学术影响？"时，传统RAG可能返回文本片段，而GraphRAG还能检索出与爱因斯坦相关的学术合作者、理论衍生关系等。
3. **增强生成**：
   - LLM结合检索到的文本和图结构数据生成回答，利用图中的关系路径提升答案的逻辑性和深度。

与传统RAG的区别

特性	传统RAG	GraphRAG
数据存储	文本片段（向量数据库）	知识图谱（图数据库）
检索内容	语义相似的文本	实体、关系、子图结构
推理能力	依赖文本上下文	利用图的多跳关系推理
适用场景	简单问答	复杂关联查询（如因果链、网络分析）

如下是 Markdown 语法的一些基本示例：

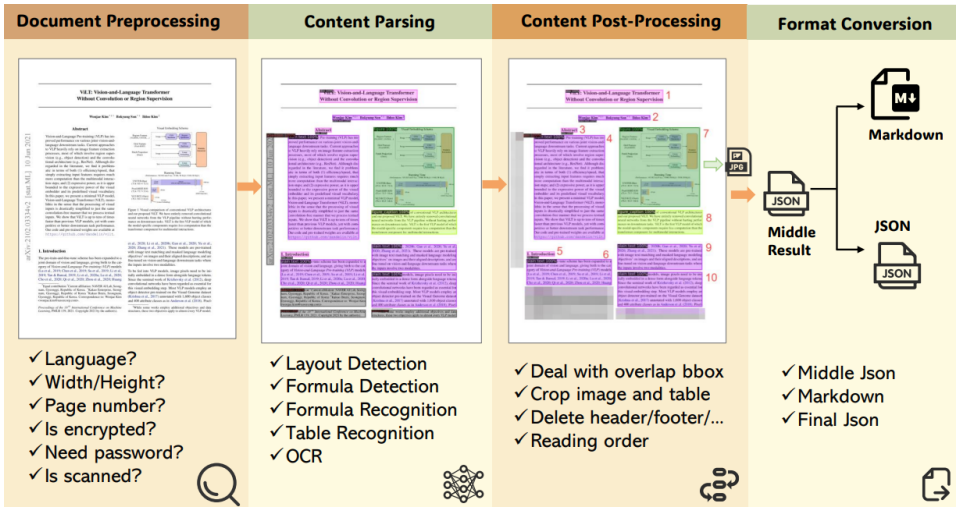
```
# 一级标题
## 二级标题
### 三级标题

**加粗文本**

*斜体文本*

> 引用文本

[链接文本](https://www.example.org)
```

```
代码文本
```

```
| 表头1    | 表头2    |
|----------|----------|
| 表格数据 | 表格数据 |
```

2. MinerU 项目概览与应用入门

`MinerU`是一个非常典型的基于管道的解决方案 (Pipeline-based solution)，并且是一个开源文档解析项目，一共四个核心组件，通过`Pipeline`的设计无缝衔接，实现比较高效、准确的文档解析。如下图所示：（下图来源官方论文：https://arxiv.org/pdf/2409.18839v1）

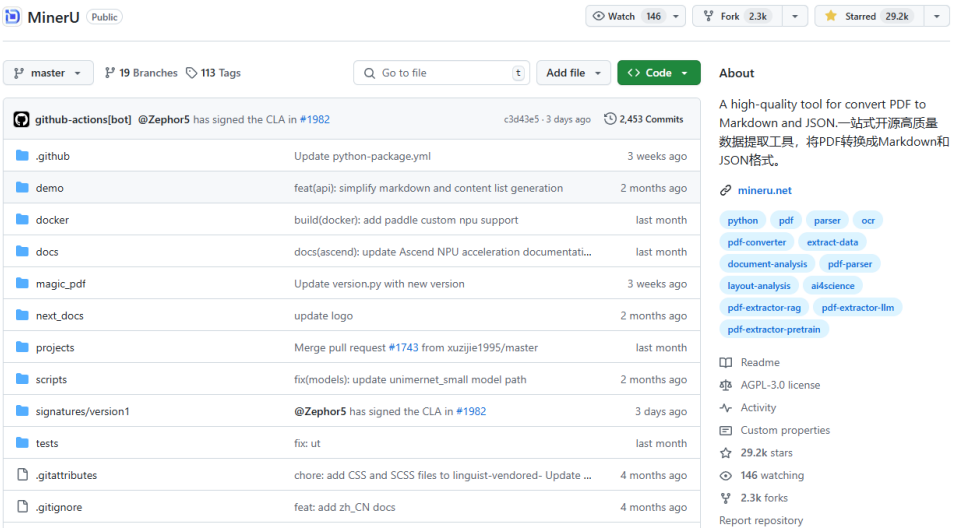


`MinerU`的主要工作流程分为以下几个阶段：

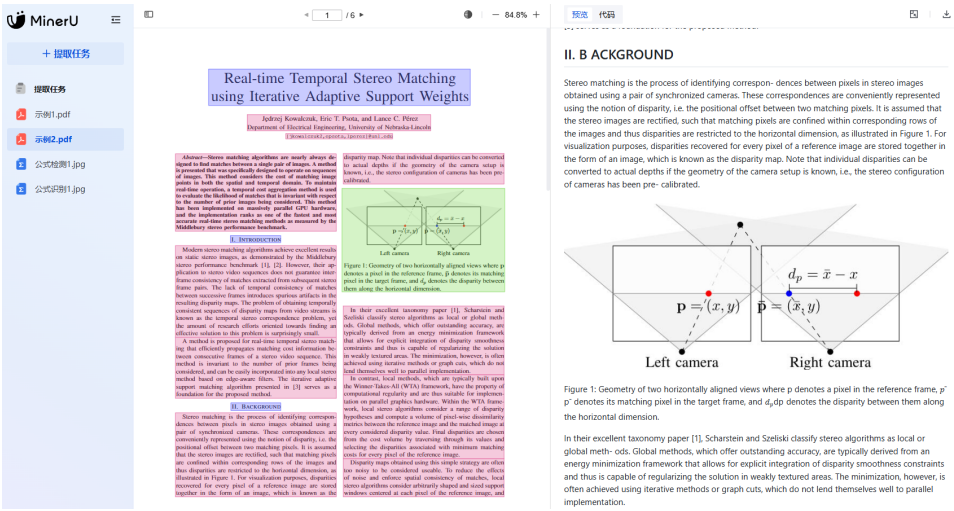
1. **输入**：接收`PDF`格式文本，可以是简单的纯文本，也可以是包含双列文本、公式、表格或图等多模态`PDF`文件;
2. **文档预处理（Document Preprocessing）**：检查语言、页面大小、文件是否被扫描以及加密状态;
3. **内容解析（Content Parsing）**：
    - 局分析：区分文本、表格和图像。
    - 公式检测和识别：识别公式类型（内联、显示或忽略）并将其转换为`LaTeX`格式。
    - 表格识别：以`HTML/LaTeX`格式输出表格。
    - OCR：对扫描的`PDF`执行文本识别。

4. **内容后处理（Content Post-processing）**：修复文档解析后可能出现的问题。比如解决文本、图像、表格和公式块之间的重叠，并根据人类阅读模式重新排序内容，确保最终输出遵循自然的阅读顺序。
5. **格式转换（Format Conversion）**：以 `Markdown` 或 `JSON` 格式生成输出。
6. **输出（Output）**：高质量、结构良好的解析文档。

目前在`Github`上，`MinerU`的`Star`数为`29.2K`，`Fork`数为`2.3K`，拥有非常良好的社区支持和活跃的贡献者，且一直处于`active development`状态。



`MinerU`提供了在线`Demo`页面，我们可以直接线进行测试。试用地址：https://opendatalab.com/OpenSourceTools/Extractor/PDF/



同时，`MinerU`项目于`2024年07月05日`首次开源，底层主要是集成`PDF-Extract-Kit`开源项目做`PDF`的内容提取，`PDF-Extract-Kit`同样是一个开源项目：https://github.com/opendatalab/PDF-Extract-Kit

`PDF-Extract-Kit`这个项目主要针对的是`PDF`文档的内容提取，通过集成众多`SOTA`模型对`PDF`文件实现高质量的内容提取，其中应用到的模型主要包括：

PDF-Extract-Kit 应用的模型类型

模型类型	模型名称	GitHub链接	模型下载链接	任务描述
布局检测模型	LayoutLMv3 / DocLayout-YOLO	GitHub / GitHub	模型下载 / 模型下载	定位文档中不同元素位置：包含图像、表格、文本、标题、公式等
公式检测模型	YOLO	GitHub	模型下载	定位文档中公式位置：包含行内公式和行间公式
公式识别模型	UniMERNet	GitHub	模型下载	识别公式图像为latex源码
表格识别模型	StructEqTable	GitHub	模型下载	识别表格图像为对应源码（Latex/HTML/Markdown）
OCR模型	PaddleOCR	GitHub	模型下载	提取图像中的文本内容（包括定位和识别）

`MinerU`与`PDF-Extract-Kit`的关系是：`MinerU`结合`PDF-Extract-Kit`输出的高质量预测结果，进行了专门的工程优化，使得文档内容提取更加便捷高效，处理底层原理的优化细节外，主要提升点在以下几点：

- 加入了自研的`doclayout_yolo(2501)`模型做布局检测，在相近解析效果情况下比原方案提速10倍以上，可以通过配置文件与`layoutlmv3`自由切换使用；
- 加入了自研的`unimernet(2501)`模型做公式识别，针对真实场景下多样性公式识别的算法，可以对复杂长公式、手写公式、含噪声的截图公式均有不错的识别效果；

- 增加 OCR 的多语言支持，支持 84 种语言的检测与识别，支持列表：https://paddlepaddle.github.io/PaddleOCR/latest/ppocr/blog/multi_languages.html#5
- 重构排序模块代码，使用 layoutreader 进行阅读顺序排序，确保在各种排版下都能实现极高准确率：https://github.com/ppaanngggg/layoutreader
- 表格识别功能接入了 StructTable-InternVL2-1B 模型，大幅提升表格识别效果，模型下载地址:https://huggingface.co/U4R/StructTable-InternVL2-1B

在部署和使用方面，MinerU 支持 Linux、Windows、MacOS 多平台部署的本地部署，并且其中用的到布局识别模型、OCR 模型、公式识别模型、表格识别模型都是开源的，我们可以直接下载到本地进行使用。而且，MinerU 项目是完全支持华为昇腾系列芯片的，可适用性非常广且符合国内用户的使用习惯。

因此，接下来我们将重点介绍如何在 Linux 系统下部署 MinerU 项目并进行 PDF 文档解析流程实战。注意：这里强烈建议大家使用 Linux 系统进行部署，其支持性和兼容性远高于 Windows 系统。如果大家想选择其他操作系统，可以参考如下链接进行自行实践：Windows 10/11 + GPU，Docker 部署，

2.1 Linux本地部署MinerU

MinerU 官方给出了本地部署的推荐配置，如下图所示：

操作系统		Linux after 2019	Windows 10 / 11	macOS 11+
CPU		x86_64 / arm64	x86_64(暂不支持ARM Windows)	x86_64 / arm64
内存		大于等于16GB，推荐32G以上		
存储空间		大于等于20GB，推荐使用SSD以获得最佳性能		
python版本		3.10 (请务必通过conda创建3.10虚拟环境)		
Nvidia Driver 版本		latest(专有驱动)	latest	None
CUDA环境		自动安装[12.1(pytorch)+11.8(paddle)]	11.8(手动安装)+cuDNN v8.7.0(手动安装)	None
CANN环境(NPU支持)		8.0+(Ascend 910b)	None	None
GPU硬件支持列表	显存8G以上	2080~2080Ti / 3060Ti~3090Ti / 4060~4090 8G显存及以上可开启全部加速功能		None

其中对操作系统的版本做了限定，同时这里最需要关注的一个点是 Python 版本：务必通过 conda 创建 Python 3.10 版本的虚拟环境。（我们实测其他版本也确实存在一些兼容问题），因此，我们接下来就严格按照官方的版本要求，逐步的进行 MinerU 的本地部署。

- Step 1. 确认系统版本

我们使用的是 Ubuntu 22.04 系统，可以通过 cat /etc/os-release 命令查看系统版本，如下图所示：

```
(base) root@4U:07_minerU# cat /etc/os-release
PRETTY_NAME="Ubuntu 22.04.2 LTS"
NAME="Ubuntu"
VERSION_ID="22.04"
VERSION="22.04.2 LTS (Jammy Jellyfish)"
VERSION_CODENAME=jammy
ID=ubuntu
ID_LIKE=debian
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
UBUNTU_CODENAME=jammy
```

- Step 2. 确认 CUDA 版本

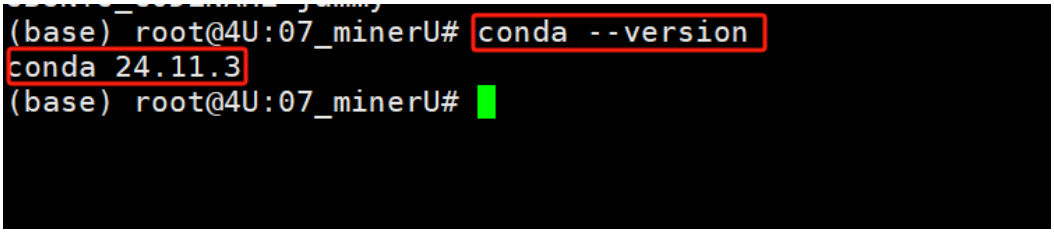
在 Linux 系统下，可以通过 nvidia-smi 命令查看 CUDA 版本，这里的服务器配置是四卡的 RTX 3090 显卡，如下图所示：

```
(base) root@4U:07_minerU# nvidia-smi
Thu Mar 27 15:41:29 2025
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 530.30.02              Driver Version: 530.30.02    CUDA Version: 12.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf           Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA GeForce RTX 3090         On | 00000000:18:00.0 Off |                  N/A |
| 31%   37C    P8              11W / 350W|     19MiB / 24576MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   1  NVIDIA GeForce RTX 3090         On | 00000000:3B:00.0 Off |                  N/A |
| 33%   37C    P8              14W / 350W|      8MiB / 24576MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   2  NVIDIA GeForce RTX 3090         On | 00000000:86:00.0 Off |                  N/A |
| 30%   37C    P8              12W / 350W|      8MiB / 24576MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   3  NVIDIA GeForce RTX 3090         On | 00000000:AF:00.0 Off |                  N/A |
| 34%   37C    P8              10W / 350W|      8MiB / 24576MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
```

`CUDA Version` 显示的版本号必须 >= 12.1，如显示的版本号小于12.1，需要自行升级 `CUDA` 版本。

- Step 3. 确认 `Conda` 版本

我们使用的是 `Anaconda` 安装的 `Conda` 环境，可以通过 `conda --version` 命令查看 `Conda` 版本，如下图所示：



如果出现 `Conda not found` 等报错，需要先安装 `Conda` 环境，再执行接下来的步骤。

- Step 4. 使用 `Conda` 创建 `Python 3.10` 版本的虚拟环境

使用 `Conda` 创建 `Python 3.10` 版本的虚拟环境，可以通过 `conda create -n mineru python==3.10` 命令创建，如下图所示：

```
(base) root@4U:07_minerU# conda create -n mineru python==3.10
/root/anaconda3/lib/python3.12/site-packages/conda/base/context.py:201: FutureWarning: Add
 and will be removed in 25.3.

To remove this warning, please choose a default channel explicitly with conda's regular co
st of channels:

  conda config --add channels defaults

For more information see https://docs.conda.io/projects/conda/en/stable/user-guide/configu

  deprecated.topic(
Retrieving notices: done
WARNING: A conda environment already exists at '/root/anaconda3/envs/mineru'

Remove existing environment?
This will remove ALL directories contained within this specified prefix directory, includi

 (y/[n])? y

/root/anaconda3/lib/python3.12/site-packages/conda/base/context.py:201: FutureWarning: Add
 and will be removed in 25.3.
```

- Step 5. 激活虚拟环境

创建完虚拟环境后，使用 `Conda` 激活虚拟环境，通过 `conda activate mineru` 命令激活，如下图所示：

```
# To activate this environment, use
#
#     $ conda activate mineru
#
# To deactivate an active environment, use
#
#     $ conda deactivate

(base) root@4U:07_minerU# conda activate mineru
(mineru) root@4U:07_minerU#
```

- Step 7. 安装 `MinerU` 项目依赖

使用 `Conda` 安装 `MinerU` 项目依赖，需要通过如下命令在新建的 `mineru` 虚拟环境中安装运行 `MinerU` 项目程序的所有依赖，命令如下：

```
    pip install -U magic-pdf[full] --extra-index-url https://wheels.myhloli.com -i https://mirrors.aliyun.com/pypi/simple
```

```
(mineru) root@4U:07_minerU# pip install -U magic-pdf[full] --extra-index-url https://wheels.myhloli.com -i https://mirrors.aliyun.com/pypi/simple

Looking in indexes: https://mirrors.aliyun.com/pypi/simple, https://wheels.myhloli.com
Collecting magic-pdf[full]
  Downloading https://mirrors.aliyun.com/pypi/packages/b3/9c/ffb90814ea62f6f566a2b9365ec46b791c4cbec6d28aeb2ad44eb0846add/magic_pdf-1.2.2-py3-none-any.whl (3.9 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.9/3.9 MB 821.2 kB/s eta 0:00:00
Collecting boto3>=1.28.43 (from magic-pdf[full])
  Downloading https://mirrors.aliyun.com/pypi/packages/19/6d/a5be3e0ced4cf4f6b48b549d61955f86ad5d9710d1c4189ffed1ec5e5b80/boto3-1.37.21-py3-none-any.whl (139 kB)
Collecting Brotli>=1.1.0 (from magic-pdf[full])
  Downloading https://mirrors.aliyun.com/pypi/packages/d5/00/40f760cc27007912b327fe15bf6bfd8eaecbe451687f72a8abc587d503b3/Brotli-1.1.0-cp310-cp310-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_12_x86_64.manylinux2010_x86_64.whl (3.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.8/3.0 MB 824.0 kB/s eta 0:00:02
```

至此，基础的 `MinerU` 项目依赖就安装完成了，接下来我们需要下载 `MinerU` 项目中用到的模型文件，并进行项目配置。

- Step 8. 下载 `MinerU` 项目中用到的模型文件

`MinerU` 项目中用到的模型文件包括：

1. 布局检测模型：LayoutLMv3 、doclayout_yolo(2501)
2. 公式识别模型：unimernet_small_2501
3. 表格识别模型：TableMaster、StructEqTable

所有的模型文件全部在 `Hugging Face` 或者 `ModelScope` 开源，所以我们可以直接下载到本地进行使用。但因为网络原因，建议国内的用户使用 `ModelScope` 下载模型文件。官方提供了 `MinerU` 项目中用到的所有模型文件的下载脚本，如果当前的服务器环境可以连接外网，则可以通过如下命令下载脚本文件：

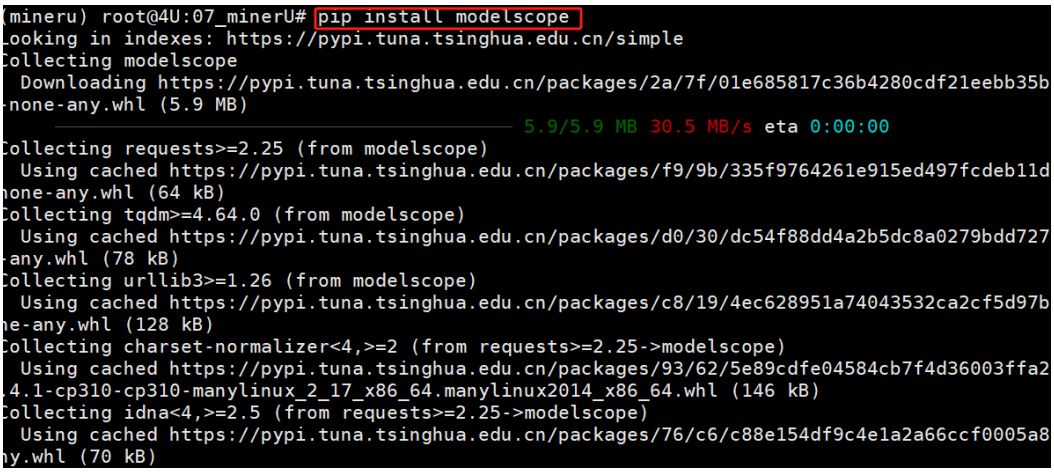
```
    wget https://gcore.jsdelivr.net/gh/opendatalab/MinerU@master/scripts/download_models.py -O download_models.py
```

这条命令的意思是：从 `gcore.jsdelivr.net` 下载 `MinerU` 项目中用到的所有模型文件的下载脚本文件，存储在当前目录下并命名为 `download_models.py`。

```
(mineru) root@4U:07_minerU# wget https://gcore.jsdelivr.net/gh/opendatalab/MinerU@master/scripts/download_models.py -O download_models.py
--2025-03-27 16:02:05--  https://gcore.jsdelivr.net/gh/opendatalab/MinerU@master/scripts/download_models.py
Resolving gcore.jsdelivr.net (gcore.jsdelivr.net)... 104.18.187.31, 104.18.186.31, 2606:4700::6812:ba1f, ...
Connecting to gcore.jsdelivr.net (gcore.jsdelivr.net)|104.18.187.31|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1938 (1.9K) [application/octet-stream]
Saving to: 'download_models.py'

download_models.py                  100%[=================================================================>]   1.89K  --.-KB/s    in 0s

2025-03-27 16:02:06 (11.6 MB/s) - 'download_models.py' saved [1938/1938]

(mineru) root@4U:07_minerU# ll
total 11
drwxr-xr-x  2 root root    3  3月 27 16:02 ./
drwxr-xr-x 25 root root   31  3月 27 15:56 ../
-rw-r--r--  1 root root 1938  3月 27 16:02 download_models.py
(mineru) root@4U:07_minerU#
```

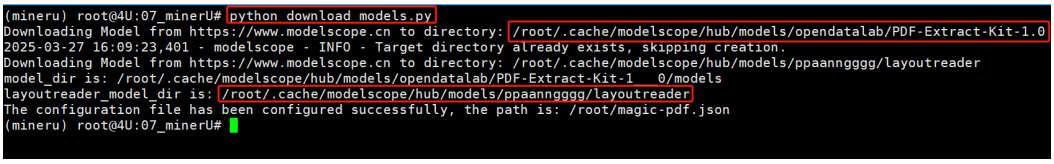
然后安装`modelscope`的`pip`包，安装命令如下：

```
pip install modelscope
```

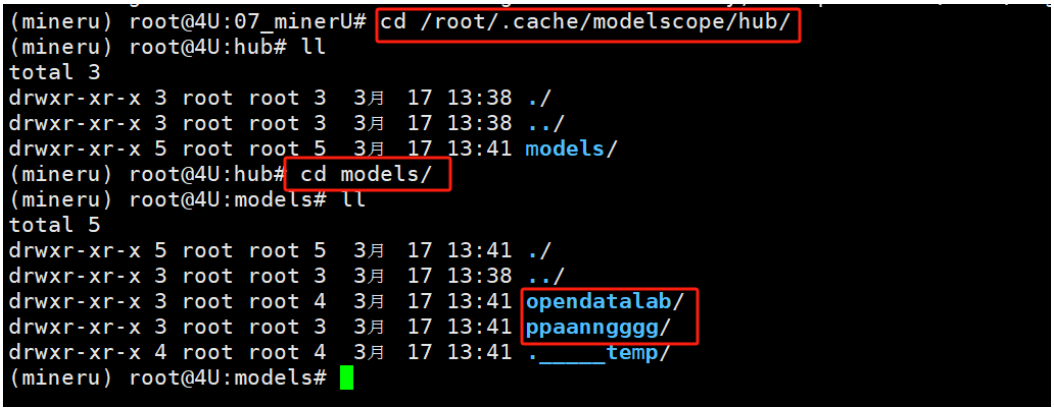


安装完成后，我们就可以开始下载`Mineru`项目中用到的模型文件了。执行命令：

```
python download_models.py
```



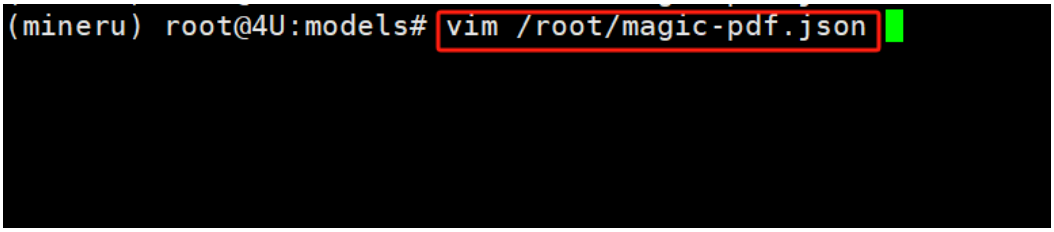
注意：上述截图是因为当前的机器已经执行过下载脚本，所以显示的是`Downloading models...`，如果大家是第一次执行，会有实际的下载过程。等待下载完成后，所有的模型权重文件都会存储在`/root/.cache/modelscope/hub/models/`下面。



下载完模型权重，并在开始执行项目之前，需要先了解一下项目的配置文件。

- Step 9. 项目配置文件

默认情况下，在执行上一步的下载脚本后，会自动生成用户目录下的`magic-pdf.json`文件，并自动配置默认模型路径，其存储的文件路径在：`/root/magic-pdf.json`。如下所示：



配置文件中会自动配置默认模型路径，如果大家想使用其他模型，可以自行修改`magic-pdf.json`文件中的模型路径，但刚开始的时候，我们建议使用默认模型：

如果服务器上有`GPU`资源的话，可以把`device-mode`参数改成`CUDA`，这样就可以使用GPU进行加速了。除此以外，其他的相关的参数解释如下：

```
    {
        // other config
        "layout-config": {
            "model": "doclayout_yolo" // 使用layoutlmv3请修改为“layoutlmv3"
        },
        "formula-config": {
            "mfd_model": "yolo_v8_mfd",
            "mfr_model": "unimernet_small",
            "enable": true  // 公式识别功能默认是开启的，如果需要关闭请修改此处的值为"false"
        },
        "table-config": {
            "model": "rapid_table",  // 默认使用"rapid_table",可以切换为"tablemaster"和"struct_eqtable"
            "sub_model": "slanet_plus",  // 当model为"rapid_table"时，可以自选sub_model，可选项为"slanet_plus"和"unitable"
            "enable": true, // 表格识别功能默认是开启的，如果需要关闭请修改此处的值为"false"
            "max_time": 400
        }
    }
```

至此，`Mineru`项目的本地配置就全部完成了，接下来我们可以尝试运行`Mineru`项目并进行`PDF`文档解析测试。

2.2 MinerU 工具使用方法

`Mineru`项目目前主要支持两种使用方式，其一是通过命令行启动，类似`MicroSoft GraphRAG`项目中我们介绍的`CLI`工具，其二是通过`API`调用。两种方法在使用上都较为简单，这里我们先通过命令行启动`Mineru`项目的方式进行快速体验。

`Mineru`项目的核心文件是`magic-pdf`，所以其`CLI`工具是以`magic-pdf`作为命令行启动文件，可以通过`magic-pdf --help`查看`magic-pdf`命令行工具的帮助信息，如下图所示：

```
magic-pdf --help
```

```
(mineru) root@4U:07_minerU# magic-pdf --help
/root/anaconda3/envs/mineru/lib/python3.10/site-packages/paddle/utils/cpp_extensio
ll source files may be required. You can download and install ccache from: https:/
  warnings.warn(warning_message)
import tensorrt_llm failed, if do not use tensorrt, ignore this message
import lmdeploy failed, if do not use lmdeploy, ignore this message
Usage: magic-pdf [OPTIONS]

Options:
  -v, --version                   display the version and exit
  -p, --path PATH                 local filepath or directory. support PDF, PPT,
                                  PPTX, DOC, DOCX, PNG, JPG files  [required]
  -o, --output-dir PATH           output local directory  [required]
  -m, --method [ocr|txt|auto]     the method for parsing pdf. ocr: using ocr
                                  technique to extract information from pdf. txt:
                                  suitable for the text-based pdf only and
                                  outperform ocr. auto: automatically choose the
                                  best method for parsing pdf from ocr and txt.
                                  without method specified, auto will be used by
                                  default.
  -l, --lang TEXT                 Input the languages in the pdf (if known) to
                                  improve OCR accuracy.  Optional. You should
                                  input "Abbreviation" with language form url: ht
                                  tps://paddlepaddle.github.io/PaddleOCR/latest/e
                                  n/ppocr/blog/multi_languages.html#5-support-
                                  languages-and-abbreviations
  -d, --debug BOOLEAN             Enables detailed debugging information during
                                  the execution of the CLI commands.
  -s, --start INTEGER             The starting page for PDF parsing, beginning
                                  from 0.
  -e, --end INTEGER               The ending page for PDF parsing, beginning from
                                  0.
  --help                          Show this message and exit.
```

从上图可以看到，`magic-pdf` 命令行工具支持的参数如下：

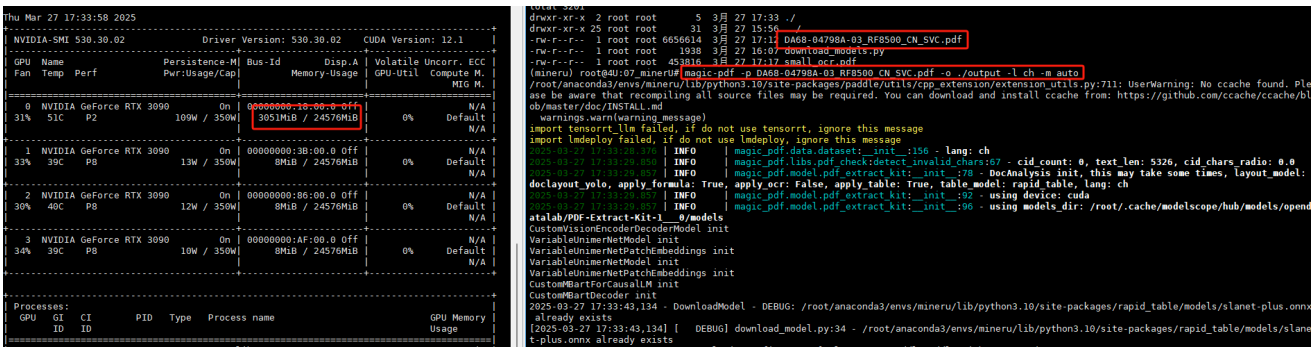
magic-pdf 命令行工具支持的参数

选项	描述
`-v, --version`	显示版本并退出
`-p, --path PATH`	本地文件路径或目录。支持 PDF、PPT、PPTX、DOC、DOCX、PNG、JPG 文件 [必需]
`-o, --output-dir PATH`	输出本地目录 [必需]
`-m, --method ocr、txt、auto`	解析 PDF 的方法。ocr：使用 OCR 技术从 PDF 中提取信息。txt：适用于仅基于文本的 PDF，性能优于 OCR。auto：自动选择最佳解析方法。未指定方法时，默认使用 auto。
`-l, --lang TEXT`	输入 PDF 中的语言（如果已知）以提高 OCR 准确性。可选。您应该输入“缩写”，语言形式请参见 PaddleOCR 多语言支持
`-d, --debug BOOLEAN`	在执行 CLI 命令期间启用详细调试信息。
`-s, --start INTEGER`	PDF 解析的起始页，从 0 开始。
`-e, --end INTEGER`	PDF 解析的结束页，从 0 开始。

| `--help` | 显示此帮助信息并退出。

有了`Microsoft GraphRAG`项目中`CLI`工具的经验，这里再看`MinerU`项目的`CLI`工具就会觉得非常熟悉了。按照其参数的说明进行使用即可。这里我们上传一个`PDF`文档进行快速测试。这里我们使用一个智能客服电商领域的产品手册进行测试，测试的命令如下：

```
magic-pdf -p DA68-04798A-03_RF8500_CN_SVC.pdf -o ./output -l ch -m auto
```



这里我们测试其`GPU`的使用峰值是`3051`，也就是仅有`3G`多的显存，同时，右侧也会实时显示`PDF`中的每一页解析所花费的时间，如下图所示：

```
2025-03-27 17:33:44.490 | INFO     | magic_pdf.model.pdf_extract_kit:__init__:174 - DocAnalysis init done!
2025-03-27 17:33:44.491 | INFO     | magic_pdf.model.doc_analyze_by_custom_model:custom_model_init:128 - model init cost: 14.635640144348145
2025-03-27 17:33:44.491 | INFO     | magic_pdf.model.doc_analyze_by_custom_model:doc_analyze:180 - gpu_memory: 24 GB, batch_ratio: 8
2025-03-27 17:33:50.912 | INFO     | magic_pdf.model.batch_analyze:__call__:74 - layout time: 4.73, image num: 48
2025-03-27 17:33:55.352 | INFO     | magic_pdf.model.batch_analyze:__call__:85 - mfd time: 4.44, image num: 48
2025-03-27 17:33:57.027 | INFO     | magic_pdf.model.batch_analyze:__call__:100 - mfr time: 1.67, image num: 30
2025-03-27 17:35:31.742 | INFO     | magic_pdf.model.batch_analyze:__call__:195 - det time: 68.78, image num: 740
2025-03-27 17:35:31.743 | INFO     | magic_pdf.model.batch_analyze:__call__:197 - table time: 25.31, image num: 21
2025-03-27 17:35:32.363 | INFO     | magic_pdf.model.doc_analyze_by_custom_model:doc_analyze:235 - gc time: 0.59
2025-03-27 17:35:32.364 | INFO     | magic_pdf.model.doc_analyze_by_custom_model:doc_analyze:239 - doc analyze time: 107.87, speed: 0.44 page
s/second
2025-03-27 17:35:32.909 | INFO     | magic_pdf.pdf_parse_union_core_v2:pdf_parse_union:946 - page_id: 0, last_page_cost_time: 0.0
2025-03-27 17:35:33.707 | INFO     | magic_pdf.pdf_parse_union_core_v2:pdf_parse_union:946 - page_id: 1, last_page_cost_time: 0.8
2025-03-27 17:35:33.818 | INFO     | magic_pdf.pdf_parse_union_core_v2:pdf_parse_union:946 - page_id: 2, last_page_cost_time: 0.11
2025-03-27 17:35:33.934 | INFO     | magic_pdf.pdf_parse_union_core_v2:pdf_parse_union:946 - page_id: 3, last_page_cost_time: 0.12
2025-03-27 17:35:34.027 | INFO     | magic_pdf.pdf_parse_union_core_v2:pdf_parse_union:946 - page_id: 4, last_page_cost_time: 0.09
2025-03-27 17:35:34.134 | INFO     | magic_pdf.pdf_parse_union_core_v2:pdf_parse_union:946 - page_id: 5, last_page_cost_time: 0.11
2025-03-27 17:35:34.226 | INFO     | magic_pdf.pdf_parse_union_core_v2:pdf_parse_union:946 - page_id: 6, last_page_cost_time: 0.09
2025-03-27 17:35:34.328 | INFO     | magic_pdf.pdf_parse_union_core_v2:pdf_parse_union:946 - page_id: 7, last_page_cost_time: 0.1
2025-03-27 17:35:34.436 | INFO     | magic_pdf.pdf_parse_union_core_v2:pdf_parse_union:946 - page_id: 8, last_page_cost_time: 0.11
2025-03-27 17:35:34.536 | INFO     | magic_pdf.pdf_parse_union_core_v2:pdf_parse_union:946 - page_id: 9, last_page_cost_time: 0.1
2025-03-27 17:35:34.645 | INFO     | magic_pdf.pdf_parse_union_core_v2:pdf_parse_union:946 - page_id: 10, last_page_cost_time: 0.11
2025-03-27 17:35:34.754 | INFO     | magic_pdf.pdf_parse_union_core_v2:pdf_parse_union:946 - page_id: 11, last_page_cost_time: 0.11
2025-03-27 17:35:34.859 | INFO     | magic_pdf.pdf_parse_union_core_v2:pdf_parse_union:946 - page_id: 12, last_page_cost_time: 0.11
2025-03-27 17:35:34.971 | INFO     | magic_pdf.pdf_parse_union_core_v2:pdf_parse_union:946 - page_id: 13, last_page_cost_time: 0.11
2025-03-27 17:35:35.079 | INFO     | magic_pdf.pdf_parse_union_core_v2:pdf_parse_union:946 - page_id: 14, last_page_cost_time: 0.11
2025-03-27 17:35:35.191 | INFO     | magic_pdf.pdf_parse_union_core_v2:pdf_parse_union:946 - page_id: 15, last_page_cost_time: 0.11
2025-03-27 17:35:35.642 | INFO     | magic_pdf.pdf_parse_union_core_v2:pdf_parse_union:946 - page_id: 16, last_page_cost_time: 0.45
2025-03-27 17:35:35.919 | INFO     | magic_pdf.pdf_parse_union_core_v2:pdf_parse_union:946 - page_id: 17, last_page_cost_time: 0.28
2025-03-27 17:35:36.108 | INFO     | magic_pdf.pdf_parse_union_core_v2:pdf_parse_union:946 - page_id: 18, last_page_cost_time: 0.19
2025-03-27 17:35:36.268 | INFO     | magic_pdf.pdf_parse_union_core_v2:pdf_parse_union:946 - page_id: 19, last_page_cost_time: 0.16
2025-03-27 17:35:36.458 | INFO     | magic_pdf.pdf_parse_union_core_v2:pdf_parse_union:946 - page_id: 20, last_page_cost_time: 0.19
2025-03-27 17:35:36.680 | INFO     | magic_pdf.pdf_parse_union_core_v2:pdf_parse_union:946 - page_id: 21, last_page_cost_time: 0.22
2025-03-27 17:35:36.871 | INFO     | magic_pdf.pdf_parse_union_core_v2:pdf_parse_union:946 - page_id: 22, last_page_cost_time: 0.19
2025-03-27 17:35:37.170 | INFO     | magic_pdf.pdf_parse_union_core_v2:pdf_parse_union:946 - page_id: 23, last_page_cost_time: 0.3
2025-03-27 17:35:37.375 | INFO     | magic_pdf.pdf_parse_union_core_v2:pdf_parse_union:946 - page_id: 24, last_page_cost_time: 0.21
2025-03-27 17:35:37.545 | INFO     | magic_pdf.pdf_parse_union_core_v2:pdf_parse_union:946 - page_id: 25, last_page_cost_time: 0.17
```

除了关注`MineRU`解析过程的耗时，最关键的是在解析完成后，会生成一个`output`目录，里面会包含`PDF`文档的解析结果，

```
(mineru) root@4U:07_minerU# ll
total 3202
drwxr-xr-x  3 root root       6  3月 27 17:33 ./
drwxr-xr-x 25 root root      31  3月 27 15:56 ../
-rw-r--r--  1 root root 6656614  3月 27 17:12 DA68-04798A-03_RF8500_CN_SVC.pdf
-rw-r--r--  1 root root    1938  3月 27 16:07 download_models.py
drwxr-xr-x  3 root root       3  3月 27 17:33 output/
-rw-r--r--  1 root root  453816  3月 27 17:17 small_ocr.pdf
(mineru) root@4U:07_minerU# cd output/
(mineru) root@4U:output# ll
total 11
drwxr-xr-x 3 root root 3  3月 27 17:33 ./
drwxr-xr-x 3 root root 6  3月 27 17:33 ../
drwxr-xr-x 3 root root 3  3月 27 17:33 DA68-04798A-03_RF8500_CN_SVC/
(mineru) root@4U:output# cd DA68-04798A-03_RF8500_CN_SVC/
(mineru) root@4U:DA68-04798A-03_RF8500_CN_SVC# ll
total 11
drwxr-xr-x 3 root root  3  3月 27 17:33 ./
drwxr-xr-x 3 root root  3  3月 27 17:33 ../
drwxr-xr-x 3 root root 10  3月 27 17:35 auto/
(mineru) root@4U:DA68-04798A-03_RF8500_CN_SVC# cd auto/
(mineru) root@4U:auto# ll
total 8878
drwxr-xr-x 3 root root      10  3月 27 17:35 ./
drwxr-xr-x 3 root root       3  3月 27 17:33 ../
-rw-r--r-- 1 root root  100788  3月 27 17:35 DA68-04798A-03_RF8500_CN_SVC_content_list.json
-rw-r--r-- 1 root root 6752709  3月 27 17:35 DA68-04798A-03_RF8500_CN_SVC_layout.pdf
-rw-r--r-- 1 root root   64544  3月 27 17:35 DA68-04798A-03_RF8500_CN_SVC.md
-rw-r--r-- 1 root root 3176082  3月 27 17:35 DA68-04798A-03_RF8500_CN_SVC_middle.json
-rw-r--r-- 1 root root  916192  3月 27 17:35 DA68-04798A-03_RF8500_CN_SVC_model.json
-rw-r--r-- 1 root root 6606157  3月 27 17:35 DA68-04798A-03_RF8500_CN_SVC_origin.pdf
-rw-r--r-- 1 root root 6734070  3月 27 17:35 DA68-04798A-03_RF8500_CN_SVC_spans.pdf
drwxr-xr-x 2 root root      73  3月 27 17:35 images/
(mineru) root@4U:auto#
```

这里得到的解析文件，就是将用于我们后续构建`GraphRAG`时用来作为分块的原始文件，即`xxx.md`，同时搭配其他的`.json`文件，可以用来构建丰富的上下文语义信息及语义增强。因此，我们需要了解每个文件中存储的核心结构及内容，分别如下：

1. images:这是一个文件夹，里面存储了`PDF`文档中出现的所有图片；
2. .origin.pdf：进行解析的原始`PDF`文件；
3. .markdown：`PDF`文档的解析结果，输出为`xxx.md`的`Markdown`格式；
4. .layout.pdf：这个文件用不同的背景色块圈定不同的内容块，以此来区分整体的布局识别结果；
5. .model.json：这个文件的主要作用是将`PDF`从像素级别的展示转换为结构化的数据，使计算机能够"理解"文档的组成部分。主要存储的信息是：

- 文档布局识别：存储文档中的各种元素（标题、正文、图表等）及其在页面上的精确位置

- 内容分类：将识别出的元素分类为不同类型（如标题、表格、公式等）
- 特殊内容解析：对于公式、表格等特殊内容，提供其结构化表示（如LaTeX、HTML格式）

其示例数据如下：


```
[
    {
        "layout_dets": [
            {
                "category_id": 2,
                "poly": [
                    99.1906967163086,
                    100.3119125366211,
                    730.3707885742188,
                    100.3119125366211,
                    730.3707885742188,
                    245.81326293945312,
                    99.1906967163086,
                    245.81326293945312
                ],
                "score": 0.9999997615814209
            }
        ],
        "page_info": {
            "page_no": 0,
            "height": 2339,
            "width": 1654
        }
    },
    {
        "layout_dets": [
            {
                "category_id": 5,
                "poly": [
                    99.13092803955078,
                    2210.680419921875,
                    497.3183898925781,
                    2210.680419921875,
                    497.3183898925781,
                    2264.78076171875,
                    99.13092803955078,
                    2264.78076171875
                ],
                "score": 0.9999997019767761
            }
        ],
        "page_info": {
            "page_no": 1,
            "height": 2339,
            "width": 1654
        }
    }
]
```

其中每一个`layout_dets`中存储了`PDF`文档中每一页的布局识别结果，`poly`是四边形坐标，分别是 左上，右上，右下，左下 四点的坐标，形式为：`[x0, y0, x1, y1, x2, y2, x3, y3]`，分别表示左上、右上、右下、左下四点的坐标，`page_info`是`PDF`文档中每一页元数据，包含页码、高度、宽度信息，而`category_id`为`PDF`文档中每一页的布局识别结果的类型，分别为：

category_id 的类别

类别 ID	类别名称	描述
0	title	标题
1	plain_text	文本
2	abandon	包括页眉页脚页码和页面注释
3	figure	图片
4	figure_caption	图片描述
5	table	表格
6	table_caption	表格描述
7	table_footnote	表格注释
8	isolate_formula	行间公式
9	formula_caption	行间公式的标号
13	embedding	行内公式
14	isolated	行间公式
15	text	OCR 识别结果

这个`.json`文件主要可以内容提取，比如精确提取特定类型的内容（如所有表格或公式），搜索，比如实现对文档内容的结构化搜索，文档分析，比如分析文档的结构特征（如有多少图表、公式密度等）等需求场景。

6. _middle.json：这个文件的核心作用是存储`PDF`文档解析过程中多层次结构化数据，它比`model.json`提供了更详细的层次结构和内容信息，其示例数据是这样的：


```
{
    "pdf_info": [
        {
            "preproc_blocks": [
                {
                    "type": "text",
                    "bbox": [
                        52,
                        61.956024169921875,
                        294,
                        82.99800872802734
                    ],
                    "lines": [
                        {
                            "bbox": [
                                52,
```

```
                                61.956024169921875,
                                294,
                                72.0000228881836
                            ],
                            "spans": [
                                {
                                    "bbox": [
                                        54.0,
                                        61.956024169921875,
                                        296.2261657714844,
                                        72.0000228881836
                                    ],
                                    "content": "dependent on the service headway and
the reliability of the departure ",
                                    "type": "text",
                                    "score": 1.0
                                }
                            ]
                        }
                    ]
                }
            ],
            "layout_bboxes": [
                {
                    "layout_bbox": [
                        52,
                        61,
                        294,
                        731
                    ],
                    "layout_label": "V",
                    "sub_layout": []
                }
            ],
            "page_idx": 0,
            "page_size": [
                612.0,
                792.0
            ],
            "_layout_tree": [],
            "images": [],
            "tables": [],
            "interline_equations": [],
            "discarded_blocks": [],
            "para_blocks": [
                {
                    "type": "text",
                    "bbox": [
                        52,
                        61.956024169921875,
                        294,
                        82.99800872802734
                    ],
                    "lines": [
                        {
                            "bbox": [
                                52,
                                61.956024169921875,
```

```
                                294,
                                72.0000228881836
                            ],
                            "spans": [
                                {
                                    "bbox": [
                                        54.0,
                                        61.956024169921875,
                                        296.2261657714844,
                                        72.0000228881836
                                    ],
                                    "content": "dependent on the service headway and
the reliability of the departure ",
                                    "type": "text",
                                    "score": 1.0
                                }
                            ]
                        }
                    ]
                }
            ]
        }
    ],
    "_parse_type": "txt",
    "_version_name": "0.6.1"
}
```


解析结果是一个多层嵌套，从大到小依次为：PDF 文档 → 包含多个页面 → 包含多个区块(blocks) → 分为一级区块和二级区块 → 每个区块包含多行 → 每行包含多个最小单元， 其中`para_blocks`内存储的元素为区块信息，`span`是所有元素的最小存储单元。大家理解了这个结构后，可以结合下图对每个字段进行理解，我们这里不再赘述。

一级

字段名	解释
pdf_info	list，每个元素都是一个 dict，这个 dict 是每一页 PDF 的解析结果，详见下表
_parse_type	ocr
_version_name	string，表示本次解析使用的 magic-pdf 的版本号

二级

字段名	解释
preproc_blocks	PDF 预处理后，未分段的中间结果
layout_bboxes	布局分割的结果，含有布局的方向（垂直、水平），和 bbox，按阅读顺序排序
page_idx	页码，从 0 开始
page_size	页面宽度和高度
_layout_tree	布局树状结构
images	list，每个元素是一个 dict，每个 dict 表示一个 img_block
tables	list，每个元素是一个 dict，每个 dict 表示一个 table_block
interline_equations	list，每个元素是一个 dict，每个 dict 表示一个 interline_equation_block
discarded_blocks	list，模型返回的需要 drop 的 block 信息
para_blocks	将 preproc_blocks 进行分段之后的结果

三级（最外层block）

字段名	解释
type	block 类型（table 或 image）
bbox	block 矩形框坐标
blocks	list，里面的每个元素都是一个 dict 格式的二级 block

四级（内层block）

字段名	解释
type	block 类型
bbox	block 矩形框坐标
lines	list，每个元素都是一个 dict 表示的 line，用来描述一行信息的构成
----------	----------
其中 type 类型	**描述**
image_body	图像的本体
image_caption	图像的描述文本
image_footnote	图像的脚注
table_body	表格本体
table_caption	表格的描述文本
table_footnote	表格的脚注
text	文本块
title	标题块
index	目录块
list	列表块
interline_equation	行间公式块

五级（lines）

字段名	解释
bbox	line 的矩形框坐标
spans	list，每个元素都是一个 dict 表示的 span，用来描述一个最小组成单元的构成

六级（spans）

字段名	解释
bbox	span 的矩形框坐标
type	span 的类型
content	img_path
----------	----------

| **其中 type 类型** | **描述**
type	描述
image	图片
table	表格
text	文本
inline_equation	行内公式
interline_equation	行间公式

该`.json`文件除了能像`model.json`一样提取特定类型的内容，因为其内容的丰富程度，可以做更精细化的文本分析，比如段落级别的文本分析，建立更精确的搜索索引，支持按内容类型搜索等。

7. span.pdf：根据`span`类型的不同，采用不同颜色线框绘制页面上所有`span`。该文件可以用于质检，可以快速排查出文本丢失、行间公式未识别等问题。

最后一个`.content_list.json`文件，是`PDF`文档中所有内容的列表，使用 JSON 格式存储数据，每个元素都是一个字典，包含了不同类型的内容（文本、图像等）。

```
[
    {
        // 文本内容块
        "type": "text",
        "text": "...",
        "text_level": N,  // 可选字段
        "page_idx": N
    },
    {
        // 图像内容块
        "type": "image",
        "img_path": "...",
        "img_caption": [],
        "img_footnote": [],
        "page_idx": N
    },
    {
        // 表格内容块
        "type": "table",
        "img_path": "...",
        "table_caption": [],
        "table_footnote": [],
        "table_body": "...",
        "page_idx": N
    }
]
```

字段说明

字段名	出现位置	数据类型	说明
type	所有内容块	字符串	内容块的类型，可能的值有："text"、"image"、"table"
page_idx	所有内容块	整数	内容所在的页码，从0开始计数
text	文本类型块	字符串	文本内容
text_level	部分文本类型块	整数	文本的层级，通常用于标题，1表示一级标题
img_path	图像和表格类型块	字符串	图像或表格截图的文件路径
img_caption	图像类型块	数组	图像的说明文字，通常为空数组
img_footnote	图像类型块	数组	图像的脚注，通常为空数组
table_caption	表格类型块	数组	表格的说明文字，通常为空数组
table_footnote	表格类型块	数组	表格的脚注，通常为空数组
table_body	表格类型块	字符串	表格的HTML内容，包含完整的表格结构

以上是对`Mineru`解析结果的说明，通过这些文件，我们基本上可以获取到`PDF`文档中所有内容的位置、类型、内容等信息，理解各个文件的结构，也是我们后续进行自定义处理的关键，所以这里大家务必明确文件及其内部结构的组成。除此以外，除了`PDF`格式，该流程也可以支持图像（.jpg及.png）、PDF、Word（.doc及.docx）、以及PowerPoint（.ppt及.pptx）在内的多种文档格式的解析，大家可以进行尝试。

实践到这里，我们通过`CLI`工具快速实现了`Mineru`对`PDF`文档的解析过程，这个过程并不是很复杂。除此之外，如果想更灵活的控制解析过程，一种更主流的方式是通过`Python` API 接口来实现，这可以在官方的`API Docs`中找到详细的说明：https://mineru.readthedocs.io/en/latest/user_guide/usage/api.html， 大家可以在`Python IDE`环境中根据官方的示例进行接口级别的解析测试。

`MineRU`的`API`接口内容比较多，我们这里重点在于`Microsoft GraphRAG`项目中`MineRU`的`API`接口使用，所以无法花费特别多的时间依次展开介绍。接下来主要给大家介绍的是`MineRU`的工程化应用形式及接入`Microsoft GraphRAG`项目中的具体方法。

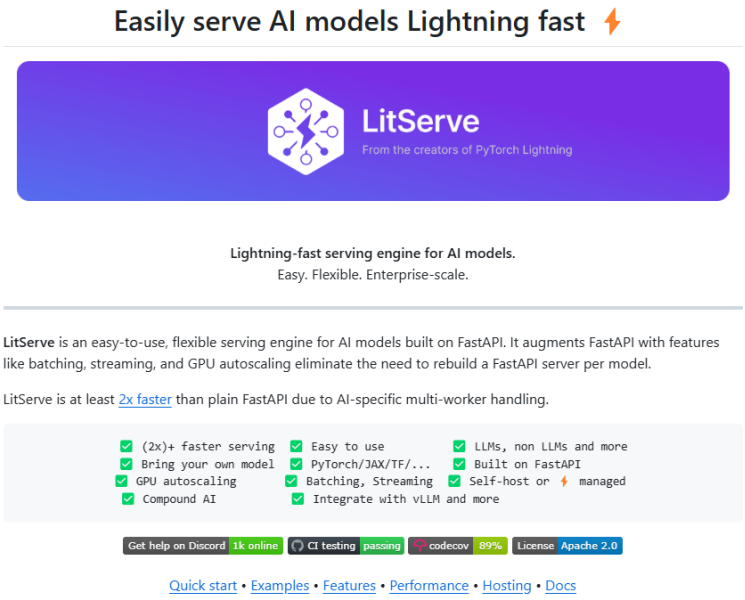
3. MinerU 的工程化应用形式

`MineRU`作为一个比较重的工具，其解析过程涉及调用多个模型，所以其解析速度相对较慢，所以需要在解析效果和响应速度之间进行平衡。除此以外，在`Microsoft GraphRAG`项目中，我们主要将其作为`PDF`文档解析工具，用于将`PDF`文档解析为`Markdown`格式，然后作为`GraphRAG`的输入，进行知识图谱的构建。而如果集成到`Microsoft GraphRAG`项目中，则会产生项目上的冗余，不利于后续的维护，所以这里我们需要采用服务化的方式，将`MineRU`作为一个独立的服务，在`Microsoft GraphRAG`项目中只保留`MineRU`的调用接口。

因此，我们需要将`MineRU`封装成一个服务，并提供`RESTful` API 接口，以便`Microsoft GraphRAG`项目中直接进行调用。在这个封装的过程中，我们再去结合具体使用的框架综合考虑效率优化等问题。

这里我们采用的是一个`LitServe`框架去做多`GPU`的并行解析，并基于该框架提供`RESTful` API 接口。

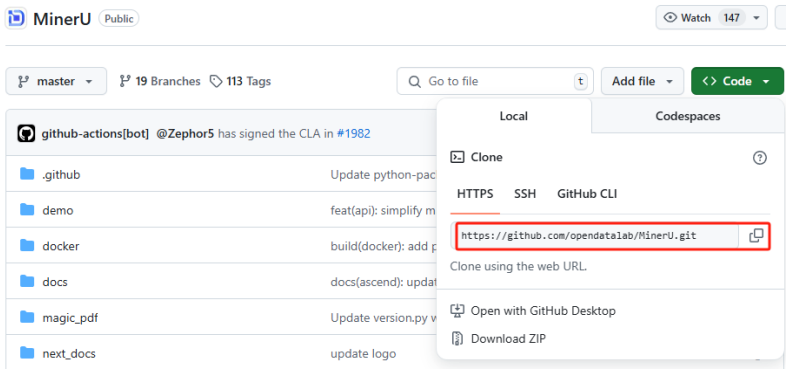
我们之前的项目经常使用`FastAPI`框架去封装后端服务的`RESTful` API 接口，而`LitServe`框架适用于基于`FastAPI`构建的模型后端服务，主要通过批处理、流式处理和`GPU`自动扩缩等功能增强了`FastAPI`，并且其评测结果显示会比普通的`FastAPI`快至少2倍。其开源地址如下：https://github.com/Lightning-AI/LitServe



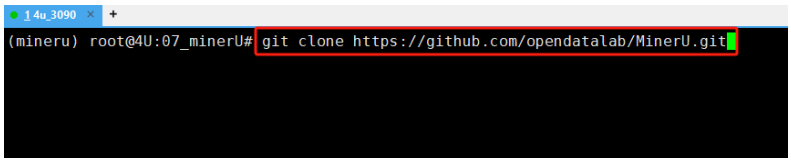
构建 `MinerU` 的 `RESTful` API 接口逻辑是这样的：首先下载 `MinerU` 的源码，然后根据源码中特定函数的函数，通过 `LitServe` 的 `Service` 类进行封装，然后通过 `LitServe` 的 `RESTful` API 接口进行调用。

因此，首先第一步要做的事，就是先把 `MinerU` 的源码下载到当前的服务器上。操作方法如下：

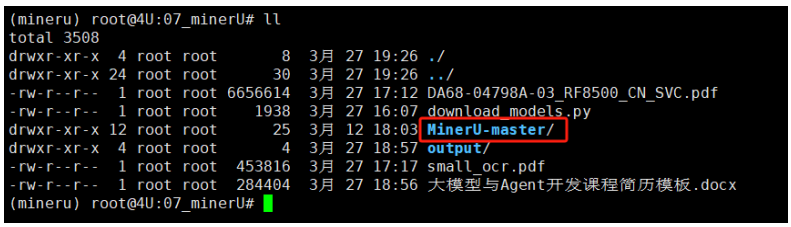
- step 1. 在 `Github` 上找到 `MinerU` 的 `https` 地址



- Step 2. 在服务上执行 `git clone https://github.com/mineru-ai/mineru.git` 命令，将 `MinerU` 的源码下载到当前的服务器上



如果服务器没有网络，可以先下载到本地，然后通过 `scp` 命令上传到服务器上，或者借助一些 `Xftp` 工具上传到服务器上。无论通过哪种方式，我们现在保证在服务上存在 `mineru` 的源码目录。如下所示：



接下来，我们就可以开始进行 `MinerU` 的 `RESTful` API 接口的封装了。首先， `MinerU` 的源码中提供了一个 `LitServe` 的基础服务，其源码存放位置为 `MinerU-master/projects` 中的 `multi_gpu` 文件夹。

```
cd MinerU-master/projects/multi_gpu
```

```
(mineru) root@4U:07_minerU# cd MinerU-master/projects/
(mineru) root@4U:projects# ll
total 74
drwxr-xr-x  8 root root  10  3月 12 18:03 ./
drwxr-xr-x 12 root root  25  3月 12 18:03 ../
drwxr-xr-x  3 root root   8  3月 12 18:03 gradio_app/
drwxr-xr-x  3 root root   9  3月 12 18:03 llama_index_rag/
drwxr-xr-x  2 root root   5  3月 17 15:36 multi_gpu/
-rw-r--r--  1 root root 516  3月 12 18:03 README.md
-rw-r--r--  1 root root 507  3月 12 18:03 README_zh-CN.md
drwxr-xr-x  4 root root  18  3月 12 18:03 web/
drwxr-xr-x  2 root root   9  3月 17 14:20 web_api/
drwxr-xr-x  5 root root  10  3月 12 18:03 web_demo/
(mineru) root@4U:projects#
```

我们进入`multi_gpu`文件夹，可以看到`LitServe`的基础服务，其提供了三个文件，分别是`server.py`、`client.py`和`README.md`。其中`server.py`文件是`LitServe`的基础服务，`client.py`文件是`LitServe`的客户端，`README.md`文件是`LitServe`的说明文档。 而我们需要调整的是`server.py`文件，其内部会定义`LitServe`的`RESTful` API 接口，并提供`LitServe`的`RESTful` API 接口的调用方法。

```
(mineru) root@4U:projects# cd multi_gpu/
(mineru) root@4U:multi_gpu# ll
total 29
drwxr-xr-x 2 root root     5  3月 17 15:36 ./
drwxr-xr-x 8 root root    10  3月 12 18:03 ../
-rw-r--r-- 1 root root  1062  3月 12 18:03 client.py
-rw-r--r-- 1 root root  1711  3月 12 18:03 README.md
-rw-r--r-- 1 root root 17725  3月 25 13:13 server.py
(mineru) root@4U:multi_gpu#
```

其中，如下文件是`litserve`内置提供的接口服务，我们不需要调整：

```
class MinerUAPI(ls.LitAPI):
    def __init__(self, output_dir='/home/07_minerU/tmp'):
        self.output_dir = Path(output_dir)

    def setup(self, device):···

    def decode_request(self, request):···

    def predict(self, inputs):···

    def encode_response(self, response):···

    def clean_memory(self):···

    def cvt2pdf(self, file_base64):···
```

结合我们的需求，即需要将其作为一个独立服务运行，`Microsoft GraphRAG`项目中仅调用`MinerU`的`RESTful` API 接口获取解析的结果。所以我这里二次开发了一个服务，即`download_output_files`服务，其主要作用是将在`Linux`服务中生成的`PDF`文档解析结果文件下载到当前的运行环境中。因为我们需要将得到的解析结果做进一步格式化处理后，才会放到`Microsoft GraphRAG`的索引构建流程中。

我们已经将配置好的`server.py`文件上传到了`MinerU-master/projects/multi_gpu`文件夹中，大家可以直接下载使用，这里需要灵活配置的参数如下所示：

LitServe 配置参数说明

字段名	解释	示例用法
accelerator	要使用的硬件类型（cpu、GPU、mps）。	
	自动检测硬件（如果可用，NVIDIA GPU 或 Apple (mps)）。	`LitServer(accelerator='auto')`
	仅使用 CPU。	`LitServer(accelerator='cpu')`
	使用任何可用的 GPU。	`LitServer(accelerator='cuda')`
	使用任何可用的 Apple GPU (mps) 芯片。	`LitServer(accelerator='mps')`
devices	服务器使用的 (CPU、GPU) 数量。	
	自动检测可用设备的数量。	`LitServer(devices='auto')`
	使用 2 个 CPU。	`LitServer(accelerator='cpu', devices=2)`
	使用 1 个 GPU。	`LitServer(accelerator='cuda', devices=1)`
	使用 8 个 GPU。	`LitServer(accelerator='cuda', devices=8)`
workers_per_device	每个设备的工作者（进程）数量。例如，如果有 1 个 GPU 和 2 个工作者，则该 GPU 上将有 2 个服务器副本。	`LitServer(workers_per_device=1)`
	默认（每个设备 1 个工作者）。	`LitServer(workers_per_device=2)`
	每个设备使用 2 个工作者。	

> 更多的LitServe 配置参数说明，可以参考：https://lightning.ai/docs/litserve/api-reference/litserver

然后，在`server.run`中配置`ip`地址和`port`端口，即可通过`http://ip:port`访问`MinerU`的`RESTful` API 接口，同时`output_dir`参数用于指定`MinerU`的输出目录，即`MinerU`解析`PDF`文档后生成的文件存放路径。

```python
class MinerUAPI(ls.LitAPI):
    def __init__(self, output_dir='/home/07_minerU/tmp'):...

    def setup(self, device):...

    def decode_request(self, request):...

    def predict(self, inputs):...

    def encode_response(self, response):...

    def clean_memory(self):...

    def cvt2pdf(self, file_base64):...


if __name__ == '__main__':
    server = ls.LitServer(
        MinerUAPI(output_dir='/home/07_minerU/tmp'),
        accelerator='cuda',
        devices='auto',
        workers_per_device=1,
        timeout=False
    )

    # 获取FastAPI应用实例
    app = server.app


    @app.get("/download_output_files")
    async def download_output_files(output_dir: str):...


    # 启动服务器
    server.run(ip='0.0.0.0', port=8000)
```

完成以上自定义配置后，回到服务器终端，依次执行如下三条命令安装`LitServe`服务运行所需要的依赖环境：

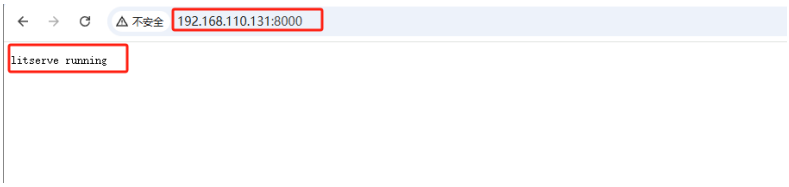
```
    pip install -U litserve python-multipart filetype
    pip install -U magic-pdf[full] --extra-index-url https://wheels.myhloli.com
    pip install paddlepaddle-gpu==3.0.0b1 -i
https://www.paddlepaddle.org.cn/packages/stable/cu118
```

```
(mineru) root@4U:multi_gpu# pip install -U litserve python-multipart filetype
Looking in indexes: https://pypi.tuna.tsinghua.edu.cn/simple
Collecting litserve
  Using cached https://pypi.tuna.tsinghua.edu.cn/packages/81/75/796f6ff8e668a8d12f9127a19e4ecf4b8ab
f784a711e5b/litserve-0.2.7-py3-none-any.whl (63 kB)
Collecting python-multipart
  Using cached https://pypi.tuna.tsinghua.edu.cn/packages/45/58/38b5afbc1a800eeea951b9285d3912613f2
f4bd7f405fc/python_multipart-0.0.20-py3-none-any.whl (24 kB)
Collecting filetype
  Using cached https://pypi.tuna.tsinghua.edu.cn/packages/18/79/1b8fa1bb3568781e84c9200f951c735f3f1
da55894d620/filetype-1.2.0-py2.py3-none-any.whl (19 kB)
Collecting fastapi>=0.100 (from litserve)
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/50/b3/b51f09c2ba432a576fe63758bddc81f78f0c
4313bf021e/fastapi-0.115.12-py3-none-any.whl (95 kB)
Collecting uvicorn>=0.29.0 (from uvicorn[standard]>=0.29.0->litserve)
  Using cached https://pypi.tuna.tsinghua.edu.cn/packages/61/14/33a3a1352cfa71812a3a21e8c9bfb83f60b
99d51928209/uvicorn-0.34.0-py3-none-any.whl (62 kB)
Collecting pyzmq>=22.0.0 (from litserve)
  Using cached https://pypi.tuna.tsinghua.edu.cn/packages/97/d4/4dd152dbbaac35d4e1fe8e8fd26d73640fc
5692df1ffb7/pyzmq-26.3.0-cp310-cp310-manylinux_2_28_x86_64.whl (867 kB)
```

安装依赖后，直接通过`python server.py`命令运行`MinerU`的`RESTful` API 接口服务，如下所示：

```
(mineru) root@4U:multi_gpu# python server.py
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
import tensorrt_llm failed, if do not use tensorrt, ignore this message
import tensorrt_llm failed, if do not use tensorrt, ignore this message
import tensorrt_llm failed, if do not use tensorrt, ignore this message
import lmdeploy failed, if do not use lmdeploy, ignore this message
import lmdeploy failed, if do not use lmdeploy, ignore this message
import lmdeploy failed, if do not use lmdeploy, ignore this message
import tensorrt_llm failed, if do not use tensorrt, ignore this message
import lmdeploy failed, if do not use lmdeploy, ignore this message
2025-03-27 20:01:01.991 | INFO     | magic_pdf.model.pdf_extract_kit:__init__:78 - DocAnalysis init, this may
e some times, layout_model: doclayout_yolo, apply_formula: True, apply_ocr: True, apply_table: True, table_mo
 rapid_table, lang: None
2025-03-27 20:01:01.991 | INFO     | magic_pdf.model.pdf_extract_kit:__init__:92 - using device: cuda
2025-03-27 20:01:01.991 | INFO     | magic_pdf.model.pdf_extract_kit:__init__:96 - using models_dir: /root/.c
/modelscope/hub/models/opendatalab/PDF-Extract-Kit-1___0/models
2025-03-27 20:01:02.002 | INFO     | magic_pdf.model.pdf_extract_kit:__init__:78 - DocAnalysis init, this may
e some times, layout_model: doclayout_yolo, apply_formula: True, apply_ocr: True, apply_table: True, table_mo
 rapid_table, lang: None
2025-03-27 20:01:02.002 | INFO     | magic_pdf.model.pdf_extract_kit:__init__:92 - using device: cuda
2025-03-27 20:01:02.002 | INFO     | magic_pdf.model.pdf_extract_kit:__init__:96 - using models_dir: /root/.c
/modelscope/hub/models/opendatalab/PDF-Extract-Kit-1___0/models
2025-03-27 20:01:02.065 | INFO     | magic_pdf.model.pdf_extract_kit:__init__:78 - DocAnalysis init, this may
e some times, layout_model: doclayout_yolo, apply_formula: True, apply_ocr: True, apply_table: True, table_mo
 rapid_table, lang: None
2025-03-27 20:01:02.065 | INFO     | magic_pdf.model.pdf_extract_kit:__init__:92 - using device: cuda
2025-03-27 20:01:02.065 | INFO     | magic_pdf.model.pdf_extract_kit:__init__:96 - using models_dir: /root/.c
/modelscope/hub/models/opendatalab/PDF-Extract-Kit-1___0/models
2025-03-27 20:01:02.075 | INFO     | magic_pdf.model.pdf_extract_kit:__init__:78 - DocAnalysis init, this may
e some times, layout_model: doclayout_yolo, apply_formula: True, apply_ocr: True, apply_table: True, table_mo
 rapid_table, lang: None
2025-03-27 20:01:02.075 | INFO     | magic_pdf.model.pdf_extract_kit:__init__:92 - using device: cuda
2025-03-27 20:01:02.075 | INFO     | magic_pdf.model.pdf_extract_kit:__init__:96 - using models_dir: /root/.c
/modelscope/hub/models/opendatalab/PDF-Extract-Kit-1___0/models
```

服务正常启动后，我们可以在浏览器通过`http://ip:port`快速验证`MinerU`的`RESTful` API 接口是否正常工作，如下所示：（因为我的服务器地址是`192.168.110.131`，所以这里我访问的是`http://192.168.110.131:8000`,大家需要根据实际的`ip`地址和`port`端口进行访问）



如果浏览器网页中出现`litserver running`的提示，则说明`MinerU`的`RESTful` API 接口服务已经正常启动，并监听在`http://ip:port`端口上，否则需要检查`server.py`文件中的配置是否正确，以及网络连接是否正常。

准备好了`MinerU`的`RESTful` API 接口服务后，接下来的工作就是在`Microsoft GraphRAG`项目中进行集成。

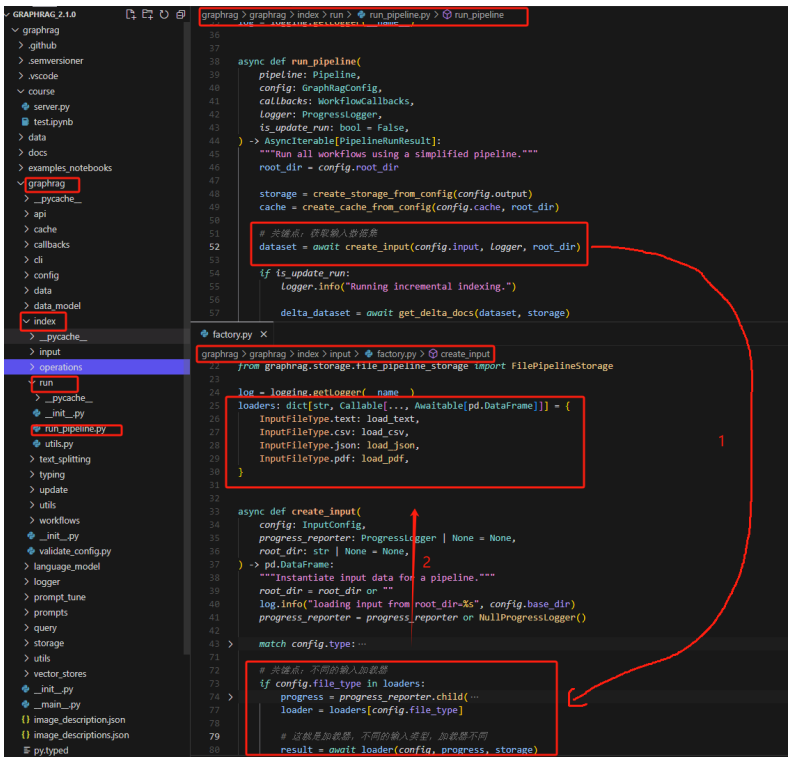
4. Microsoft GraphRAG 项目中集成 MinerU

回顾一下`Microsoft GraphRAG`项目的索引构建流程：我们首先需要通过`init`命令初始化项目文件，生成`.env`文件和`settings.ymal`文件。接下来手动创建一个`input`文件夹，把我们要执行索引的原始文件放在这个文件夹中，最后通过`index`命令执行一系列的`Workflow`任务，包括文档加载、分块、提取实体、关系、生成社区报告、摘要等等一系列步骤。

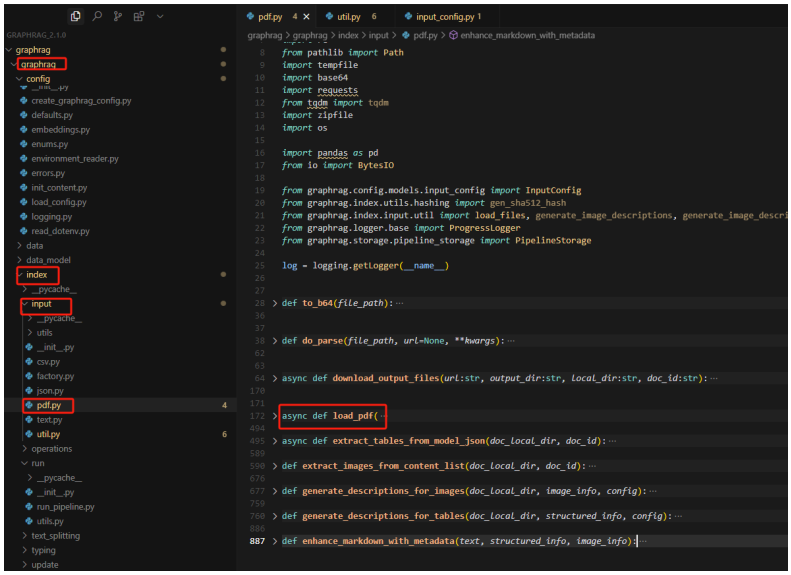
而默认情况下，`Microsoft GraphRAG`项目仅仅支持了`.txt`、`.csv`和`.json`三种文件格式，所以这就意味着在`input`文件夹中，我们只能放置这三种格式的文件，如果想要支持`PDF`文件，首先我们需要做的就是：**手动构建`PDF`文档加载器，并按照`Microsoft GraphRAG`的输入输出格式进行格式化处理，而`MinerU`的`RESTful` API 接口服务，则需要在这个文档加载器的实现过程中被调用。**

接下来，我们就带着大家一步一步完成这个基于源码的二次开发`PDF`文档加载器的过程，并在实现过程中给大家重点介绍`PDF`和`MinerU`的集成优化策略。

4.1 手动构建`PDF`文档加载器



如上所示，我们自定义接入了`load_pdf`的接口，并且在`input`路径下的`pdf.py`文件中实现了完整的接入逻辑和规范：



我们重点介绍一下 `PDF` 文档加载器的具体实现及优化方法。 完整的思路如下：

1. 调用 `MinerU` 的 `do_parse` RestFul API 接口，将本地的文件上传 `PDF` 依次传到 `MinerU` 服务上进行解析；
2. 调用 `MinerU` 的 `download_output_files` RestFul API 接口，将解析后的所有文件拉取到项目代码的运行环境（一般企业场景需要搭配专业的文件服务器）；
3. 读取 `xxx.md` 文件，获取 `PDF` 文件的文本内容，这里面包含：纯文本、图片、表格、公式等；
4. 在进行文本分块之前，先进行语义增强和文本优化，主要包括以下几点：
    - 对于表格数据：
        1. 首先提取结构化数据，从 `model.json` 文件中，根据 `category_id` 是否等于 5， 提取出都有哪些表格；
        2. 调用大模型接口，对表格进行总结和摘要，作为额外描述添加到元数据中；
    - 对于图片数据：
        1. 从 `content_list.json` 文件中，根据 `type` 找到图片，并提取其前后文，作为图片的背景信息（因为按照一般的逻辑，当出现图片的时候，前文或者后文会对图片有相关的描述或解释性文字）；
        2. 从本地对应的 `PDF` 解析文件中找到 `Images` 文件夹，获取到所有的图片路径；
        3. 调用视觉模型对图片生成详细的描述，包括图片中的内容、对象、场景、布局等关键信息，作为额外描述添加的元数据中；
5. 将表格、图片的元数据以注释形式插入到第3步读取的 `Markdown` 文本中，组成最终要用于后续做 `text_units` 切分的原始文本，形式为：

```
<!-- METADATA
type: table
description: "下表展示了2023年各季度销售额对比..."
source_page: 5
parent_headings: ["## 实验结果"]
-->
该表格统计了2023年各季度的销售额（见下表）：
| 季度 | 销售额（万元） |
|------|----------------|
| Q1   | 120            |
```

因此，我们在自定义 `PDF` 文档加载器及根据特定的优化策略实现语义增强的流程中，需要额外在 `settings.yaml` 文件中的 `input` 参数下添加如下参数：

LitServe 配置参数说明

参数名	说明	默认值/示例
`type`	配置类型，表示这是文件处理器配置	`file`
`file_type`	要处理的文件类型	`pdf`
`base_dir`	存放PDF文件的基础目录	`input`
`file_encoding`	文件编码格式	`utf-8`
`file_pattern`	用于匹配PDF文件的正则表达式	`.*\\.pdf$$`
`local_output_dir`	本地存储解析后PDF输出文件的目录	示例：`./data/pdf_outputs`
`mineru_api_url`	MinerU服务的API访问地址	示例：`http://192.168.110.131:8000/`
`mineru_output_dir`	MinerU服务器上存储处理结果的目录	示例：`/home/07_minerU/tmp/`
`table_description_api_key`	表格描述处理的API密钥	示例：`sk-ad49340f6dd9a97d0a2db`
`table_description_model`	用于生成表格描述的模型名称	示例：`deepseek-chat`
`base_url`	表格描述API的基础URL	示例：`https://api.deepseek.com`
`image_description_api_key`	图像描述处理的API密钥	示例：`sk-proj-CUT84SGX34nP...`
`image_description_model`	用于生成图像描述的模型名称	示例：`gpt-4o`
`image_description_base_url`	图像描述API的基础URL	示例：`https://ai.devtool.tech/proxy/v1`

其中元数据的格式是：

```
{
  "metadata": {
    "file_path": "原始文件路径",
    "output_dir": "解析输出目录",
    "parse_time": "解析时间",
    "doc_id": "文档ID",
    "local_output_dir": "本地输出目录",
    "content_elements": [
      {
        "type": "table",
        "page": 4,
        "element_idx": 0,
        "html": "<table>...</table>",
```

```
        "description": "这是一个包含命令行参数的表格..."
      },
      {
        "type": "image",
        "page": 2,
        "element_idx": 0,
        "path": "images/example.jpg",
        "description": "这张图片展示了系统架构图..."
      }
    ]
  }
}
```

最后，我们再来看PDF格式文件的切分策略。

3.2 手动构建PDF文档切分器

PDF包含文本、图像和表格，所以在分块时也常被称多模式分块，涉及处理单个文档中的多种类型的数据（例如文本、图像和表格）。比如有一些常见的切分方法：

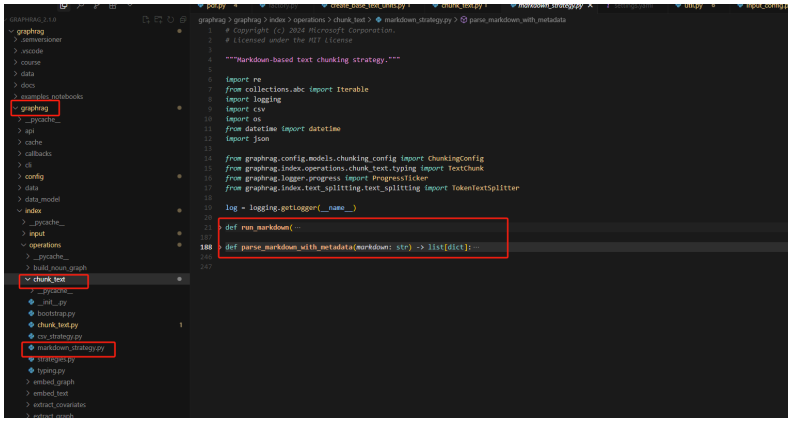
- 基本策略：将文档的连续部分组合起来以填充每个块，确保它们不超过指定的大小。其中表格被单独处理，如果太大，进行拆分。
- 按标题策略：保留章节边界，每当新章节或页面开始时，就从新区块开始。合并较小的章节，以避免区块过小。
- 按页面策略：这种方法可确保来自不同页面的内容不会出现在同一个块中。每个新页面都会开始一个新的块。
- 按相似度策略：使用大模型来识别和分组相似内容。所需的相似度级别可以调整。

不同的切分方法其实都涉及两个主要问题： 如果处理表格，如何处理图片。传统RAG常用固定大小的分块，比如512个token，但可能破坏表格结构，其次GraphRAG需要更语义化的分块，比如按章节或段落。同时，是否需要重叠块来保持上下文联系，最后，表格的处理是关键，因为表格的结构信息一旦破坏，模型可能无法正确理解。图片的话，转成Markdown后可能只是图片链接，需要OCR提取文字或添加描述。

而上面三个核心的问题，我们在PDF文档加载器中已经全部解决，因此，这里我们实现的是核心策略：结构感知分块（Structure-aware Chunking），具体来看：

- 首先按标题层级划分主块：将文档按一级标题（#）分割为顶级块，再按二级标题（##）划分子块，同时保持标题下的所有内容（文本、表格、图片）在同一个块中。
- 动态块大小调整：对于文本块，使用滑动窗口分割，并允许窗口间重叠，表格/图片块：单独成块，不分割，直接关联描述和上下文。

其实现的核心逻辑及源码的接入位置，如下图所示：



同时，在setting.yaml文件中的chunk配置下，将strategy设置为markdown，并结合input下的配置，即可正常在Microsoft GraphRAG源码中接入PDF、Docx等文件格式进行自动化的索引构建。如下图所示：

最后，还是通过`poetry run poe index`命令即可开启索引构建流程。