



Python项目执行计划：人源-鼠源抗原结构配对数据集构建

根据要求，我们将把数据处理流程划分为五个阶段，每个阶段说明输入/输出、处理逻辑、使用的Python模块和第三方库、文件命名及目录结构，并阐述如何实现增量构建机制。整个流程均在服务器环境下运行，仅使用公开数据库和已有模型（如 SAbDab、AlphaFold DB），不调用任何额外的AI建模工具。所有结构数据统一使用 mmCIF 格式处理和存储，不使用PDB格式。

阶段一：数据下载与原始数据存储

输入及数据源：无本地输入。通过网络从公开数据库获取人源-抗体复合物数据：
- **SAbDab (Structural Antibody Database)**：提供所有抗原-抗体复合物的信息和结构文件列表。通过SAbDab提供的摘要文件（例如 TSV/CSV）获取复合物清单及元数据，并下载对应的结构文件。
- **下载方式**：使用Python的HTTP请求库（如 `requests`）从SAbDab获取数据。例如：
- **下载 元数据表**：SAbDab提供所有复合物的汇总TSV文件（包含 PDB编号、链信息、抗原名称、物种等）。使用 `requests.get()` 获取该TSV文件并保存本地。
- **下载 结构文件**：根据元数据中的PDB编号列表，逐一请求每个复合物的mmCIF坐标文件。可以直接通过RCSB PDB或PDBe 等提供的API获取mmCIF（例如，RCSB文件下载服务通过URL如 https://files.rcsb.org/download/{PDB_ID}.cif）。使用 `requests` 循环下载，并保存为{PDB_ID}.cif。

处理逻辑： 1. **获取清单**：首先请求SAbDab提供的复合物列表（TSV）。利用Pandas读取该TSV文件方便处理后续信息（如筛选人源抗原、记录链ID等）。 2. **批量下载mmCIF**：遍历清单中的每个条目，提取PDB编号。对于每个PDB，构造请求URL获取mmCIF文件内容。如果成功响应，则将其以二进制写入文件。例如，用 `requests.get(url)` 获取数据并保存为 {PDB_ID}.cif。 3. **保存元数据**：将下载到的原始TSV文件保存为 `sabdab_summary.tsv`（或CSV）。另外，为方便查阅，可从TSV中提取关键信息生成一个**说明表格**（CSV），包含如：PDB编号、抗原名称及链、抗体重链/轻链链ID、抗原物种等列。

输出：下载后的文件保存在原始数据目录：

```
./data/antigen_antibody/SAbDab/raw/
├── sabdab_summary.tsv          # SAbDab提供的原始TSV元数据
├── sabdab_summary_filtered.csv # (可选) 提取主要字段的说明CSV
├── 1XYZ.cif                   # 例如, PDB编号1XYZ的复合物结构mmCIF
├── 1ABC.cif                   # 其它复合物...
└── ...                         # 以PDB编号命名的多个mmCIF文件
```

文件命名规范：原始复合物结构文件直接使用PDB代码作为文件名（大写或小写均可，一般沿用PDB格式，例如 `1XYZ.cif`）。元数据表格文件以清晰名称命名，如 `sabdab_summary.tsv`。

使用模块与库： - 网络请求：`requests`（用于HTTP下载）。 - 数据表处理：`pandas`（读取TSV元数据，方便筛选和写出CSV说明文件）。 - 文件和路径管理：`os / pathlib`（创建目录、检查文件存在等）。

增量构建机制：实现过程中检查文件是否已存在以避免重复下载： - 下载每个mmCIF前，先检查目标路径下是否已有同名文件（例如 `./raw/1XYZ.cif`）。若已存在则跳过下载该项，防止重复。 - 元数据TSV也可根据需要

定期更新（例如SAbDab每周更新时重新获取）。可以记录上次下载日期或哈希，只有在有新更新时才重新下载并处理新条目。 - 可维护一个已处理PDB列表，当SAbDab有新增复合物时，仅下载新增部分。

阶段二：数据清理

输入：阶段一下载的原始mmCIF结构文件（`./SAbDab/raw/` 目录下）以及对应的元数据表（TSV/CSV）。这些输入提供了每个PDB复合物中抗原链和抗体链的识别信息。

处理逻辑：清理每个原始复合物结构，移除与抗原/抗体无关的部分：
1. **解析结构：**使用BioPython的PDB模块读取mmCIF文件（例如 `Bio.PDB.MMCIFParser()` 解析）。得到 `Structure` 对象后，可获取其中的模型和链信息。
2. **确定保留链：**从元数据（TSV/CSV）提取该PDB中属于抗原和抗体的链ID列表。例如，根据SAbDab提供的字段识别抗原链（通常标注为Antigen Chain）以及抗体重链和轻链ID。
3. **过滤链：**遍历结构中的所有链，只保留上述所需链：
- 如果结构中存在其它生物分子链（如其他蛋白、核酸）不在保留列表，则从结构对象中剔除。
- 移除水分子、配体、小分子等非主链元素。在BioPython中，可以检查Residue的 `resname` 和 `id`，跳过 `HOH`（水）或 `HETATM` 残基。或者直接在解析时使用SelectiveIO模式，仅构建所需链。
- 确保仅剩 **抗原链** 和 **抗体链**（重链/轻链）相关的蛋白质原子。
4. **保存清理后的结构：**使用BioPython的 `MMCIFIO` 将精简后的结构写回mmCIF文件格式，存储到清理目录。文件命名可与原始PDB相同，以表示内容对应原始PDB但已清理。

输出：清理后的mmCIF文件存放于 `./data/antigen_antibody/SAbDab/cleaned/`：

```
./data/antigen_antibody/SAbDab/cleaned/
├── 1XYZ.cif  # 仅保留目标抗原/抗体链的结构
├── 1ABC.cif  # ...
└── ...
```

每个文件对应原始PDB复合物，但只包含相关链，例如1XYZ包含原先的抗原和抗体链，其它无关链已删除。

使用模块与库： - BioPython的 `Bio.PDB` 模块：使用 `MMCIFParser` 读取mmCIF结构，`Structure` 对象进行链和残基过滤，`MMCIFIO` 保存mmCIF。BioPython提供数据结构方便删除链，例如可以通过 `structure[0].detach_child(chain_id)` 删除某链。 - 或者使用 `Gemmi` 库（一个支持mmCIF的结构处理库）进行过滤，它能方便地加载mmCIF并移除不需要的部分，再写出文件。 - Python内建的文件系统库（`os/pathlib`）检查和创建目录。

文件命名规范：沿用原PDB编号作为文件名，以便溯源。例如 `1XYZ.cif` 清理后仍命名为 `1XYZ.cif`（存放在 `cleaned` 子目录下）。这样日后可以通过相同文件名知道源自哪个PDB。

增量构建机制：对每个原始文件，在清理前检查目标输出是否已经存在：
- 若 `./cleaned/1XYZ.cif` 已存在且假定清理逻辑无变化，则跳过对1XYZ的清理。
- 如果更新了清理规则或原始文件有变化，可设置标志强制重新清理特定PDB，或在比对原始和清理文件时间戳后决定是否重跑。
- 实现上，可扫描 `raw` 目录和 `cleaned` 目录，对比文件清单，找出 `raw` 中存在但 `cleaned` 中缺失的PDB，仅处理这些缺失项。

阶段三：拆分抗原与抗体

输入：阶段二得到的清理后复合物mmCIF文件（每个包含抗原和抗体相关链）。这些文件位于 `./SAbDab/cleaned/` 目录。

处理逻辑：将清理后的每个复合物拆分为独立的抗原结构文件和抗体结构文件： 1. **加载清理结构：**使用 BioPython 解析清理后的 mmCIF (与阶段二类似，用 `MMCIFParser` 生成 `Structure`)。 2. **提取抗原链：**根据元数据或结构内容区分哪几个链属于抗原。例如，在 SAbDab 元数据里可找到该 PDB 的抗原链 ID (有时可能是一个或多个链)。从 `Structure` 中复制这些链组成一个新的结构对象，仅包含抗原部分。 - 注意：若抗原在该复合物中由多个链 (例如抗原是由两条肽链组成的复合物)，则应将所有这些链都包含在抗原结构文件中。 3. **提取抗体链：**类似地，识别抗体部分 (通常包括一条重链和一条轻链，两条链代表一个 Fab 片段；或者可能是单链抗体则只有一条链)。将抗体的所有相关链提取组成抗体结构对象。 4. **保存拆分结果：**使用 `MMCIFIO` 分别将抗原结构和抗体结构写为独立 mmCIF 文件： - 抗原结构保存至 `./SAbDab/antigen/` 目录，文件命名包含原 PDB 编号以示来源，例如 `1XYZ_antigen.cif`。 - 抗体结构保存至 `./SAbDab/antibody/` 目录，文件命名如 `1XYZ_antibody.cif`。对于包含重链和轻链的抗体，这两个链会一起存储在一个 mmCIF 文件中，因为它们共同构成抗体复合物。

输出：拆分后的文件存储于两个目录：

```

./data/antigen_antibody/SAbDab/antigen/
├── 1XYZ_antigen.cif    # 来源PDB 1XYZ的抗原链结构
├── 1ABC_antigen.cif    # ...
└── ...

./data/antigen_antibody/SAbDab/antibody/
├── 1XYZ_antibody.cif   # 来源PDB 1XYZ的抗体链结构 (含重链+轻链)
├── 1ABC_antibody.cif   # ...
└── ...

```

文件命名规范： `{PDB_ID}_antigen.cif` 和 `{PDB_ID}_antibody.cif`。这种命名直观表明文件来源以及内容类型。例如 `1XYZ_antigen.cif` 表示来自 PDB 1XYZ 的抗原部分。

使用模块与库： - `BioPython Bio.PDB`：重复利用解析和写文件功能。可以新建一个 `Structure` 对象，将所需链的 `Chain` 对象添加进去再保存。也可通过 `BioPython` 提供的 `Select` 类机制，让 `PDBIO/MMCIFIO` 只输出选定链。 - 也可考虑 `PyMOL` 的 API 或 `MDAnalysis` 库：例如，用 `MDAnalysis` 读取 mmCIF 并通过选择原子链然后写出 PDB (但需要 mmCIF 输出支持) 或者 `PyMOL` 加载后保存所选链 (`PyMOL` 也支持输出 mmCIF)。

增量构建机制：在拆分前检查输出是否已存在： - 若某个 PDB 的抗原/抗体文件已经存在，则默认跳过该 PDB 的拆分，避免重复工作。 - 可以细化检查：如果抗原文件和抗体文件都存在，则跳过；如果有缺，则重新生成缺失部分。 - 例如，遍历 `cleaned` 目录，对于每个 `X.cif` 检查 `X_antigen.cif` 是否存在于 `antigen` 目录，若不存在才执行提取保存。抗体文件同理。

阶段四：人源-鼠源抗原对齐

输入：本阶段需要两部分输入： - **人源抗原链结构：**来自阶段三抗原目录下的 mmCIF 文件，但仅选择其中物种为人 (Human) 的抗原。可通过元数据筛选抗原物种字段为 Human，对应的文件即为人源抗原结构。 - **对应的鼠源抗原结构：**需要为每个人源抗原找到对应的鼠源版本的结构模型。这里利用公开数据库 `AlphaFold DB` 或 PDB： - **AlphaFold DB：**提供几乎所有已知蛋白的预测结构 (mmCIF 格式)。通常根据蛋白的 UniProt ID 获取。对于人源抗原，首先确定其在小鼠中的同源蛋白 UniProt ID，然后通过 AlphaFold DB API 下载其预测结构 mmCIF。 - **PDB 数据库 (可选)：**若存在该抗原的鼠源晶体结构 (例如某些常见小鼠蛋白也有解析结构)，可以通过 PDB ID 获取。不过一般 AlphaFold 预测会更普遍可得，且满足“已有模型”要求。

处理逻辑：对于每一个人源抗原，执行以下步骤： 1. **识别对应的鼠源抗原：**利用元数据确定人源抗原的标识 (比如名称或 UniProt)。常用方法： - 使用抗原名称/基因符号：如人源 IL-2，对应鼠源 IL-2 (有时名称上

加“Mouse”以示区别）。对于规范操作，可以使用**UniProt映射**：元数据若给出人源抗原链的UniProt ID，查询UniProt数据库以找到对应物种为Mouse的小鼠同源蛋白的UniProt ID（例如通过基因名称匹配，或UniProt提供的ID mapping服务）。 - 如果缺乏直接映射，可根据蛋白名称手动定义映射表或利用生物信息库（如**bioservices**或**requests**调用UniProt REST API）搜索小鼠蛋白。 - **输出命名约定**：确定鼠源抗原的名字用于文件夹命名。例如采用人类抗原通用名称+前缀“Mouse”，如 human抗原为“IL2”，则鼠源命名“MouseIL2”。（精确命名可依据基因符号或UniProt提供的蛋白名称字段。） 2. **获取鼠源抗原结构模型**：利用上一步找到的鼠源抗原UniProt ID，从AlphaFold DB获取预测结构： - 使用**requests**调用AlphaFold数据库的公开API。例如AlphaFold提供下载链接，如：https://alphafold.ebi.ac.uk/files/AF-{UniProtID}-F1-model_v{version}.cif。发起GET请求下载mmCIF文件。 - 或使用BioPython内置功能：Bio.PDB有AlphaFold接口，可以用**Bio.PDB.alphafold_db.get_structural_models_for(uniprot_id)**直接下载并返回Structure对象。 - 将下载的鼠源抗原结构暂存在内存或本地（可在临时目录或专门目录，如`./data/antigen_antibody/MouseModels/`保存原始AlphaFold模型，方便重复使用）。 3. **结构对齐**：将鼠源抗原结构对齐到对应的人源抗原结构的构象上，以便两者在同一坐标系中比较。对齐以**人源抗原链**为参考，旋转/平移鼠源结构： - **加载结构**：用BioPython解析人源抗原CIF（得到Structure对象，仅一条链）和鼠源抗原CIF。 - **序列比对**：可先获取两者氨基酸序列（BioPython `Structure/Chain.get_sequence()`）。使用BioPython的序列比对（`Bio.pairwise2`或`Bio.Align`模块）找出匹配的残基对应关系。这样确定对齐的残基索引对。 - **计算超位**：利用BioPython的**SVDSuperimposer**或**Superimposer**，输入人源链和鼠源链对应残基坐标（通常使用Ca原子坐标）计算最佳旋转平移矩阵。 - **应用变换**：将计算得到的变换应用到**鼠源抗原结构**的所有坐标上（BioPython中`Superimposer.apply()`可以直接对结构原子坐标进行变换）。现在鼠源结构已经在人源结构坐标系下对齐。 - **（可选）PyMOL API方法**：另一种方式是调用PyMOL的Python接口（`pymol.cmd`模块）。将人源和鼠源结构分别加载为对象，使用`cmd.align()`或`cmd.super()`执行结构对齐，PyMOL内部会完成序列匹配和坐标超定位。然后用`cmd.save`导出对齐后的鼠源结构为mmCIF。此方法封装了对齐算法，简化实现，但需要在服务器安装PyMOL并运行无图形界面模式。 4. **存储配对结果**：为每一对人源-鼠源抗原创建独立的子目录，保存对齐后的结构文件： - **子文件夹命名规则**：`{human_ag_name}_{chain_id}_{mouse_ag_name}`。其中**human_ag_name**是人源抗原名称或标识，**chain_id**是该抗原在原PDB中的链标号，**mouse_ag_name**是鼠源抗原名称。例如：`IL2_A_MouseIL2`表示人源IL-2抗原（链A）及对应的小鼠IL-2。 - 在每个子文件夹中保存两个文件： - **人源抗原链**（`*_human_ag_chain.cif`）：即人源抗原的mmCIF结构。可直接从阶段三的抗原文件复制过来，或者读取后重新保存（确保格式统一）。文件命名例如`IL2_A_MouseIL2_human_ag_chain.cif`，也可以简化为放在文件夹内命名`human_ag_chain.cif`（通过所在目录已隐含身份）。 - **对齐后的鼠源抗原链**（`*_mouse_ag_chain_aligned.cif`）：包含鼠源抗原结构经过对齐变换后的坐标。命名类似`IL2_A_MouseIL2_mouse_ag_chain_aligned.cif`（或文件夹内简名`mouse_ag_chain_aligned.cif`）。 - **目录结构示例**：

```
./data/antigen_antibody/HumanMouse/IL2_A_MouseIL2/
├── human_ag_chain.cif      # 人源抗原链（例如IL-2，链A）
└── mouse_ag_chain_aligned.cif # 鼠源抗原链（IL-2鼠源，同人源对齐）
```

- **记录配对信息**：在输出同时，将该人-鼠抗原对的信息添加到一个CSV说明表中（结构配对说明）。例如CSV包含列：人源抗原名称、人源PDB及链、鼠源抗原名称、鼠源结构来源（AlphaFold或PDB）、输出文件夹路径等。逐条记录便于汇总最终的配对列表。

输出：所有对齐配对文件组织在`./data/antigen_antibody/HumanMouse/`目录下，每对一个子文件夹，内部有上述两条链的CIF文件。并有汇总的CSV文件描述所有配对情况，例如`human_mouse_pairs.csv`。

使用模块与库： - **BioPython**：用于解析mmCIF、序列提取、序列比对（`Bio.Align`或`Bio.pairwise2`）、坐标超定位（`Bio.PDB.Superimposer`）以及文件保存（MMCIFIO）。 - **PyMOL API**：用于结构对齐的替代方案，尤其当BioPython对齐需要较多手动步骤时。可通过`pymol2`库在脚本中启动PyMOL实例执行对齐。 - **Requests**：获取AlphaFold模型mmCIF数据。或者使用BioPython内置AlphaFold接口（`Bio.PDB.alphafold_db`模块）简化下载。 - **os/pathlib**：创建子目录，文件读写。 - **pandas**：用于逐步构建最终CSV说明表（每处理一对追加一行）。

增量构建机制：- **跳过已存在配对**：在对齐前检查目标子文件夹是否已存在。如果某个人源抗原已经对齐过对应的鼠源结构且文件存在，则跳过，防止重复计算。这样反复运行时，不会重新对已处理的抗原对进行对齐。 - **重复利用下载模型**：对于鼠源抗原结构，如果已在本地下载（例如在前一次运行中保存在临时目录或 HumanMouse子目录下），则不再重新下载AlphaFold模型。可通过文件存在或保存下载记录（比如已知 UniProtID对应文件路径）来判断。 - **新增数据处理**：当SAbDab源有新增人源抗原时（或之前未处理的人源抗原），脚本会检测到尚无对应HumanMouse文件夹，于是执行对齐流程新建之。CSV说明表也在末尾添加新记录。

阶段五：构建完整的鼠源抗原结构

输入：来自阶段四的**对齐结果**和可能的其他鼠源抗原链结构： - 阶段四已经对每个人源抗原链找到并对齐了鼠源对应链。如果某抗原在复合物中由多条链组成（例如一个抗原可能是多个亚基的复合体），那么会存在该抗原的多个子文件夹（每条链各对应一个对齐结果）。需要将同一个抗原的所有鼠源链组合起来。 - 另外，也可直接利用AlphaFold获取的**完整鼠源抗原结构模型**（若AlphaFold已预测了全长多链结构，然而AlphaFold DB主要提供单链预测，因此多亚基复合体需逐链组合）。

处理逻辑：为每个抗原（尤其是多链抗原）构建完整的鼠源版本： 1. **识别需要拼接的链**：浏览阶段四输出的 HumanMouse子文件夹。按照抗原名称分组（如根据子文件夹名中的human_ag_name或 mouse_ag_name）。对于某一抗原，如果只存在单链，对应鼠源结构本身就是完整的（无需拼接）。如果发现同一抗原名称有多条链分别对齐，比如抗原由链A和链B组成的人源复合物，那么会有两个文件夹如 `AntigenName_A_MouseAntigenName` 和 `AntigenName_B_MouseAntigenName`。 2. **拼接鼠源结构**：将上述同一抗原的多个鼠源链结构合并到一起： - 读取每个子文件夹中的 `mouse_ag_chain_aligned.cif`（这些都是该抗原不同链的鼠源结构，已经各自对齐到人源对应链的位置）。 - 创建一个新的Structure对象来容纳完整抗原。在 BioPython中，可以新建一个空的Structure或取第一条链的Structure为基础。 - 将后续链的原子坐标添加为新的链到该结构中。需要注意保留链标识，避免冲突（可用原链ID或重新编号）。 - **检查连接性**：如果这些链本应是一个连续的多肽（比如原结构中因为不连续拆成两段，但实际上序列上连贯），则可能需要检查末端原子间距离以确定是否需要加连接。另外，如果抗原由独立亚基构成，则无需连接，只需保证它们都包括在同一结构中。 - 验证组合后结构的完整性：确保没有重叠原子或严重空间冲突（通常不会有，因为它们源自对齐在正确位置）。如果发现无法直接拼接（例如找不到对应的另一个链，或者两段结构相距过远无法确定相对关系），记录问题并可能跳过该抗原的完整结构构建。 3. **保存完整结构**：使用BioPython的 `MMCIFIO` 将组合后的结构保存为单个mmCIF文件，文件名为 `{mouse_ag_name}.cif`，存放在MouseAntigen目录下。例如，对于MouseIL2，输出文件 `MouseIL2.cif` 代表小鼠IL-2完整结构。 - 如果抗原只有单链，那么可直接将阶段四所得 `mouse_ag_chain_aligned.cif` 重命名或复制到MouseAntigen目录作为完整结构文件。 - 如果抗原有多链，则现在的文件包含所有链，各链保留独立Chain ID。

输出：完整的鼠源抗原结构文件集，保存在 `./data/antigen_antibody/MouseAntigen/` 目录：

```
./data/antigen_antibody/MouseAntigen/
├── MouseIL2.cif      # 例如，小鼠IL-2抗原完整结构（单链）
├── MouseAntigenX.cif  # 如果MouseAntigenX由多链组成，则包含所有链
└── ...
```

每个文件以鼠源抗原名称命名，内容是该抗原的全长度/全亚基结构。

使用模块与库： - **BioPython** `Bio.PDB`：用于加载对齐后的链结构并合并。可以直接操作Structure/Model/Chain层级：例如，用 `structure[0].add(chain_obj)` 将新的链加入现有结构的模型0下。也可逐原子复制，但直接加链更方便。 - 或使用**Gemmi**库：Gemmi的Model对象可包含多个Chain，也支持合并模型并写出mmCIF。 - Python基础 `itertools` 或集合：帮助分组同名抗原； `os` 遍历文件系统找出哪些鼠源抗原已存在。 - 日志/错误处理：记录哪些抗原成功构建完整结构，哪些失败（及原因，如缺链）。

增量构建机制： - 在生成每个鼠源抗原完整结构前，检查目标文件是否已存在于MouseAntigen目录。如果存在且之前构建成功，则跳过，避免重复组合。 - 如果发现某抗原新增了以前没有的链（例如因为新的人源PDB补充了另一亚基），脚本应检测到MouseAntigen目录下该抗原文件过期或不完整，进而重新构建。实现上，可通过比较HumanMouse子文件夹的数量与已合成文件包含的链数，或维护一个记录（如在CSV说明表中记录每个抗原有几条链）。 - 每次运行可重新扫描HumanMouse目录，以防有新对齐结果出现，然后更新或新增完整结构文件，仅对尚未生成或需要更新的抗原执行组合。

文件目录结构总览

经过以上各阶段处理，项目的数据目录结构如下所示：

```
./data/antigen_antibody/
├── SAbDab/
│   ├── raw/          # 原始下载的数据 (mmCIF结构+元数据TSV)
│   ├── cleaned/      # 清理后的复合物mmCIF
│   ├── antigen/      # 抗原链独立mmCIF文件
│   └── antibody/     # 抗体链独立mmCIF文件
└── HumanMouse/      # 人源-鼠源抗原对齐结果
    ├── {human_ag}_{chain}_{mouse_ag}/ # 每对抗原的子文件夹
    │   ├── human_ag_chain.cif        # 人源抗原链结构
    │   └── mouse_ag_chain_aligned.cif # 对齐后的鼠源抗原链结构
    └── MouseAntigen/      # 完整的鼠源抗原结构
        └── {mouse_ag_name}.cif       # 每个鼠源抗原的完整结构
```

此外，会有说明文档： - `sabdab_summary.tsv` / `sabdab_summary.csv`：SAbDab原始数据表及筛选后的信息表（位于raw目录）。 - `human_mouse_pairs.csv`：人-鼠抗原配对清单，包含每对的关键描述（位于主目录或指定输出位置）。

增量构建机制汇总

整个管道各阶段都设计了**幂等性**和**增量更新能力**。重复运行时已完成的部分不会重复处理，从而支持批量迭代构建：

- **阶段1：**下载时检查已有文件，避免重复。未来如果SAbDab更新新的结构，只下载新增PDB的数据。
- **阶段2：**清理时仅处理新的或更新的结构。可通过文件存在与否或时间戳控制。
- **阶段3：**拆分同样跳过已拆分出的文件。
- **阶段4：**对齐过程按抗原对进行，如果某人源抗原已经有对应鼠源对齐结果文件夹，则不再重复执行。此外，对于已经下载的小鼠模型，不再重新获取。
- **阶段5：**完整结构构建按抗原名称检查终结果文件是否存在。如已存在则不重复组合；如检测到需要更新则重新生成。

通过上述机制，实现数据集的**增量构建**：可以定期运行管道脚本，新增的数据自动处理，而已处理的数据保持不变，保证效率和一致性。最终将得到符合要求的mmCIF结构文件集以及配对说明CSV表格，方便后续分析使用。