多肽虚拟筛选教程 —— 使用 MolAICal 与 LightDock

（同样适用于蛋白质和核酸的虚拟筛选）

Qifeng Bai
邮箱：molaical@yeah.net
主页：https://molaical.github.io 或 https://molaical.gitlab.io

---

1. 引言

MolAICal 可以使用开源的 **LightDock** 软件（https://lightdock.org）并通过 **MM/GBSA** 或 **MM/PBSA** 方法进一步提升生物大分子对接的准确性。LightDock 是一个开源（GPL-3.0 许可）的对接框架，适用于 **蛋白质-蛋白质**、**蛋白质-肽段**、**蛋白质-RNA** 和 **蛋白质-DNA** 复合物的对接。它特别擅长处理柔性及具有挑战性的情况（例如瞬时相互作用、低亲和力复合物，或需要主链/侧链灵活性的体系），并且作为一个可扩展平台，支持测试新型打分函数、约束条件或优化策略。

LightDock 采用 **萤火虫群优化算法**（Glowworm Swarm Optimization, GSO）——一种最初为多峰函数优化设计的仿生群体智能方法。

本教程以 **胰高血糖素样肽-1 受体**（GLP-1R） 与 **艾塞那肽-4**（exendin-4，首个 FDA 批准的 GLP-1R 激动剂）为例进行演示（PDB ID: 7LLL.pdb）。

注意：本教程仅适用于 Linux 操作系统！

本教程展示了一种通过 **MolAICal 与 LightDock 平台集成** 实现肽段虚拟筛选的计算流程。MolAICal 能够高效生成并预测多种肽段候选结构的 3D 构象，包括线性和环状变体以及不同二级结构；而 LightDock 则采用先进的群体优化算法，实现高精度的蛋白质-肽段对接模拟。该联合方法使研究人员能够对成千上万种肽段变体进行靶向筛选，基于结合能和相互作用模式识别出有潜力的结合体。通过多尺度建模方法和并行计算能力，该流程显著加速了治疗性肽段、蛋白质抑制剂和核酸结合物的发现过程（只需将肽段替换为蛋白质或核酸即可）。

更多细节请参阅 **蛋白质-DNA** 和 **蛋白质-RNA** 对接教程，以及 **MolAICal 用户手册**。

---

2. 材料

2.1 软件要求

1. **MolAICal**：https://molaical.github.io

2. **NAMD**（CPU 版本）：https://www.ks.uiuc.edu/Research/namd/
   > **注意**：本教程可使用 NAMD 2.x、3.x 或更高版本。例如，若您使用 NAMD 3.x 版本，则将教程中的命令 “namd2” 替换为 “namd3”。更高版本的 NAMD 亦可采用类似方式替换。

3. **PyMol**：https://github.com/cgohlke/pymol-open-source-wheels , https://github.com/maabuu/pymol-wheels , https://pypi.org/project/pymol-open-source-whl/ , https://github.com/cnpem/PyMOL4Win

4. **VMD**：https://www.ks.uiuc.edu/Research/vmd

2.2 示例文件

1. 所有必需的教程文件可从以下地址下载：
https://gitee.com/molaical/tutorials/tree/master/028-peptide_vs_lm

---

3. 操作流程

3.1 分子准备

为了解决 Linux 中的库依赖问题，Linux 版本的 MolAICal (Windows 版本的 MolAICal 没有采用容器)采用了基于容器的方法。如果需要调用外部程序，建议在 MolAICal 容器内安装这些程序。如果不需要外部程序，可以忽略此步骤。本教程需要外部程序 NAMD 和 VMD，具体步骤如下：

1. 首先，将文件复制到 MolAICal 容器中

```
# 进入容器文件系统（将进入 "/root" 目录）
#> molaical.exe -eset shell in

# 将 VMD 和 NAMD 安装包从本地机器复制到容器中，'cp' 命令的第一部分（源路径）
# 位于本地主机，第二部分（目标路径）在容器内；VMD 和 NAMD 软件包可通过以下命
# 令移入容器。
#> cp /home/user/<本地文件> /root/soft

# 退出容器文件系统
#> exit
```

2. 其次，进入 MolAICal 容器的虚拟环境

```
#> molaical.exe -eset sys run molaical
```

注：molaical 是容器名称。

3. 在 MolAICal 容器虚拟环境中安装软件的方式与在本地主机上安装相同。以下以安装 VMD 和 NAMD 为例：

1) 安装 NAMD：

解压 NAMD 文件（假设解压后的文件夹名为 namdcpu），然后使用以下命令将其路径告知 MolAICal：

```
#> molaical.exe -call set -n NAMD -p "/root/soft/namdcpu/namd3"
```

注：请将上述 VMD 和 NAMD 的路径替换为您系统中的实际路径。-n 后面的 "VMD" 和 "NAMD"（大小写不敏感）是固定的标识符。为确保 MM/GBSA 结果的可重复性，建议使用 NAMD 的 CPU 版本，因为 CUDA 版本中的 seed 参数似乎对结果可重复性无效。

2) 安装 VMD：

按以下步骤操作：

◆ **解压 VMD 文件：**

```
#> tar -xzvf vmd-xxx.tar.gz
```

注：请将上述路径替换为您系统中的实际路径。

◆ **修改 VMD 解压目录中名为 configure 的文件中的安装路径：**

```
# 默认值：
$install_bin_dir="/usr/local/bin";
$install_library_dir="/usr/local/lib/$install_name";

# 修改为：
$install_bin_dir="/root/soft/vmd193/bin";
$install_library_dir="/root/soft/vmd193/lib/$install_name";
```

◆ **安装 VMD：**

```
#> cd vmd-xxx
#> ./configure LINUXAMD64
#> cd src
#> make install
```

注：请运行 ./configure 并根据所用计算机选择正确的类型，此处为 "LINUXAMD64"。

◆ **然后使用以下命令将 VMD 路径告知 MolAICal：**

```
#> molaical.exe -call set -n VMD -p "/root/soft/vmd193/bin/vmd"
```

至此，NAMD 和 VMD 在 MolAICal 容器内的安装与配置已完成。为防止 MolAICal 出现问题时丢失已安装的程序（例如 VMD 和 NAMD），请参阅**附录 2** 中的第 **3** 步。

✧ **记得使用 exit 命令退出 MolAICal 虚拟环境，返回本地计算机进行计算（主要是为了省去文件拷贝步骤；在容器内运行也可行，但需手动将数据从本地计算机复制到 MolAICal 容器中）。**

```
#> exit
```

进入教程材料文件夹 “028-peptide_vs_lm”：

```
#> cd 028-peptide_vs_lm
```

将看到以下文件列表：

- **7LLL.pdb**：来自 https://www.rcsb.org 的复合物结构文件（PDB 格式）
- **dock_u.sh**：对接单元脚本（适用于本教程类似的膜蛋白受体）
- **dock_u_nomem.sh**：对接单元脚本（适用于非膜蛋白受体）
- **dock_u_cyc.sh**：环肽与膜蛋白的对接单元脚本
- **dock_u_cyc_nomem.sh**：环肽与非膜蛋白的对接单元脚本
- **mempro.pdb**：本受体对应的膜蛋白受体（制备方法见附录 1 或 MolAICal 蛋白质-肽段对接教程）
- **new.pdb**：受体构建过程中的临时文件
- **P_peptide.pdb**：从 “7LLL.pdb” 中提取的肽段配体（制备方法见附录 1 或 MolAICal 蛋白质-肽段对接教程）
- **restraints.list**：指定活性位点的文件，可减少构象搜索的对接时间（制备方法见附录 1）
- **R_protein.pdb**：从 “7LLL.pdb” 中提取的受体文件（制备方法见附录 1）
- **R_P_complex.pdb**：包含肽段配体和受体的复合物文件（制备方法见附录 1）

选项：LightDock 中的 DFIRE 评分函数需要标准氨基酸残基及其原子名称，因此在使用 DFIRE 评分函数进行对接前，必须检查并修复受体或配体。修复命令如下：

```
#> molaical.exe -call run -c sfile -i lmfix_rec.py -i protein.pdb -o protein_fixed.pdb
```

此处该 PDB 文件正常，无需通过上述选项处理。

3.2 肽段生成

LightDock 的功能实现依赖于打分函数所支持的氨基酸类型。例如：DFIRE 仅支持标准氨基酸及残基名称为"MMB"的氨基酸；此时必须使用"-nc"参数以避免在生成肽段的 N 端和 C 端添加封端基团，因为 DFIRE 无法识别这些修饰结构。

1) 使用 MolAICal 生成线性肽段

#> molaical.exe -call run -c pepgen -i -l 8 -n 2 -3d -nc -o linear_seq.txt -dir linear

参数说明：

- -call：与外部程序或命令交互。值为 'set'（设置环境）或 'run'（运行程序）
- -i：调用外部程序，其后跟随该程序的所有命令行参数
- -c（在 -i 之前）：若值为 "pepgen"，则生成肽段
- -l：肽段中氨基酸残基数
- -n：生成的唯一序列数量
- -3d：使用 pyPept 生成 3D 结构
- -nc：不为序列添加乙酰基（ac）/酰胺基（am）封端
- -o：生成序列的输出文件
- -dir：3D 结构的输出目录
- 更多细节请参阅 MolAICal 手册。

生成的 3D 肽段位于 "linear" 文件夹中，可用于后续虚拟筛选。

2) 使用 MolAICal 生成带有固定残基的线性肽段（用于肽段修饰）

#> molaical.exe -call run -c pepgen -i -l 8 -n 3 -rc 3 -nc -3d -dir mpep -s EEYVPIST -d none -fix 1 7

新增参数说明：

- -dir：3D 结构输出目录
- -rc：使用 RDKit 生成的随机构象数量（默认：1）
- -d：从种子序列扩展的方向（[forward, backward, both, none]）
- -s：起始序列（可选）
- -fix：生成过程中保持固定的残基位置（1 起始索引）

3) 使用 MolAICal 生成环状肽段

MolAICal 还支持生成含二硫键等修饰的肽段（详见 MolAICal 手册）。

#> molaical.exe -call run -c pepgen -i -l 8 -n 3 -rc 3 -cc -3d -c 4 -dir cyclic

新增参数说明：

- -cc：生成环状肽段（仅支持 BILN/HELM 格式）
- -c（在 -i 之后）：用于并行处理的 CPU 核心数

在 “linear”、“mpep” 和 “cyclic” 目录中分别包含了三类肽段，适用于不同目的。由于肽段虚拟筛选计算耗时，本教程仅提取少量肽段用于演示。用户可根据自身计算资源生成足量肽段用于研究。

3.3 肽段虚拟筛选

MolAICal 的虚拟筛选方法包括：创建一个 **对接单元**（Docking Unit）脚本文件，其中包含多个任务；每个单元对接一个分子（见图 1）。在此基础上，通过 **多核并行处理** 可同时筛选多个分子，实现虚拟筛选。

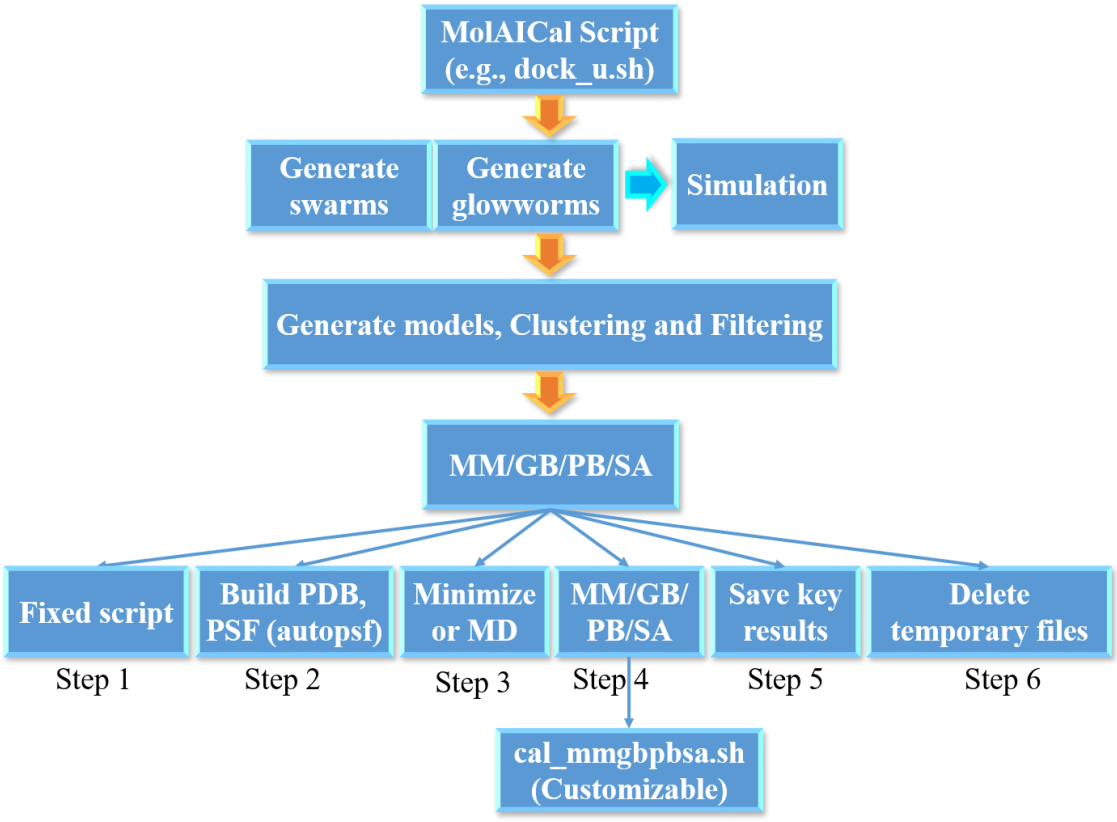

图 1：对接单元示意图

“linear” 和 “mpep” 文件夹中的肽段可使用相同流程筛选。对于 “cyclic” 文件夹中的环状肽段，需进行**小幅修改**：在 MM/GB/PB/SA 步骤中，末端残基必须使用 ‘LINK’ 进行打补丁。该方法也适用于肽段中二硫键等残基修饰。

1) 对 “linear” 和 “mpep” 文件夹中的线性肽段进行筛选：

```
#> molaical.exe -call run -c sfile -i vs_ml.sh 1::=linear 2::=mempro.pdb 3::=R 4::=Z 5::=8 6::=1 9::=dock_u.sh
```

参数说明：

- '-call'：与外部程序交互（‘set’/‘run’）
- '::= '：用于按指定顺序为参数赋值，后面的变量必须紧跟"::="，不能有空格。
- '-c'：若值为 “sfile”，则运行 MolAICal 脚本
- '1::= '：待筛选分子（如肽段）的工作目录（默认：‘linear’）
- '2::= '：受体文件（默认：‘mempro.pdb’）
- '3::= '：第一个分子（即受体）的链名（默认：‘R’）
- '4::= '：第二个分子（即配体）的链名（默认：‘Z’）
- '5::= '：每个子任务（即每次对接）使用的 CPU 核心数（默认：2）
- '6::= '：虚拟筛选任务使用的 CPU 核心总数。默认值为 1。此参数决定启动的并行管道数量，**多个并行管道会显著增加内存消耗。建议保持默认值 1，**即仅激活一个管道。除非可用内存资源非常充足，方可考虑增加并行管道数量。 若要进一步提升计算效率，可增加单个管道内子任务的并行核心数，例如通过"5::=8"格式配置（表示该子任务使用 8 个核心）。
- '8::= '：调试模式（0 或 1）。1 表示测试任务以检查错误；0 表示直接运行（默认：0）
- '9::= '：MolAICal 脚本格式的对接脚本文件（默认：‘dock_unit.sh’）
- '11::= '：结果输出目录（默认：‘dresults’）

更多细节请参阅 MolAICal 手册。

2) 对 “cyclic” 文件夹中的环状肽段进行筛选：

```
#> molaical.exe -call run -c sfile -i vs_ml.sh 1::=cyclic 2::=mempro.pdb 3::=R 4::=Z 5::=8 6::=1 8::=0 9::=dock_u_cyc.sh 11::=cyc_result
```

3.4 结果排序

此处以 “cyc_result” 文件夹中的结果为例进行说明，用户可参考此案例用于自身研究。

1) 仅处理自定义百分比（此处为 20%）的筛选候选分子：

```
#> molaical.exe -call run -c sfile -i rank_ppn.py cyc_result -m filtered -p 20 -fo filtered_results
```

或

2) 仅处理 MMGBPBSA，并自定义结果输出目录：

```
#> molaical.exe -call run -c sfile -i rank_ppn.py cyc_result -m mmgbpbsa -mo
mmgbpbsa_results -p 20
```

或

3) 同时处理筛选与 MMGBPBSA，使用默认设置并获取前 20% 的结果：

```
#> molaical.exe -call run -c sfile -i rank_ppn.py cyc_result -p 20 -fo filtered_results -mo
mmgbpbsa_results
```

参数说明：

- -m {both,filtered,mmgbpbsa}: 处理模式（默认：both）
- -p: 提取顶部分子的百分比（默认：5.0）
- -fo: 筛选结果输出目录（默认：vs_filtered_results）
- -mo: MMGBPBSA 结果输出目录（默认：vs_mmgbpbsa_results）
- "cyc_result"是包含用于虚拟筛选的候选肽段分子的输入目录。
- 更多帮助信息，请查阅 MolAICal 用户手册

任务续跑：如果任务被中断，请运行以下命令进行恢复：

```
#> molaical.exe -call run -c sfile -i vs_ml.sh 1::=linear 5::=8 7::=1 12::=continuevs
```

- **'7::=':** 检查对接任务是否完成。其值为 0 或 1。若值为 1，将检查 MM/GB/PB/SA 步骤的结果；若值为 0，则检查第一步虚拟筛选步骤的结果。默认值为 1。
- **'12::=':** 为对接脚本文件 '9::=' 提供更多选项。默认值为空字符串，用 "" 表示。若此输入包含多个参数，请使用逗号字符 ',' 分隔。MolAICal 会自动将每个逗号 ',' 转换为空格 " "。它有一些特殊值：

- 若为 "createdir"，则仅创建虚拟筛选的文件夹；
- 若为 "startvs"，则仅运行虚拟筛选任务；
- 若为 "continuevs"，则会处理剩余的未完成虚拟筛选任务；
- 若为 "sn_{num}"，则会设置 ${num} 个并行管道，这对应于 '6::=' 的设置；
- 否则，运行所有任务。

在指定文件夹 ${work_dir}（例如 "linear" 文件夹）中完成虚拟筛选的 PDB 文件，将被移动到目录 ${work_dir}_complete_folder 中。然后，直接运行之前的虚拟筛选程序将从 ${work_dir} 文件夹恢复任务，从而有效实现任务续跑。

参考文献

1. Jiménez-García, B.; Roel-Touris, J.; Romero-Durana, M.; Vidal, M.; Jiménez-González, D.; Fernández-Recio, J., LightDock: a new multi-scale approach to protein-protein docking. *Bioinformatics* **2018**, *34* (1), 49–55.

2. Krishnanand, K. N.; Ghose, D., Glowworm swarm optimization for simultaneous capture of multiple local optima of multimodal functions. *Swarm Intelligence* **2009**, *3* (2), 87–124.

附录 1

以下附录 1 描述了本教程所用材料的制备方法，其流程与 **蛋白质-肽段分子对接** 相同；若您已熟悉该流程，可跳过本节。

1. 分子文件处理

在 PyMol 中加载 PDB 文件 **7LLL.pdb**，并按照 **图 a1** 的步骤分别保存 “R_P_complex.pdb”、“R_protein.pdb” 和 “P_peptide.pdb”。

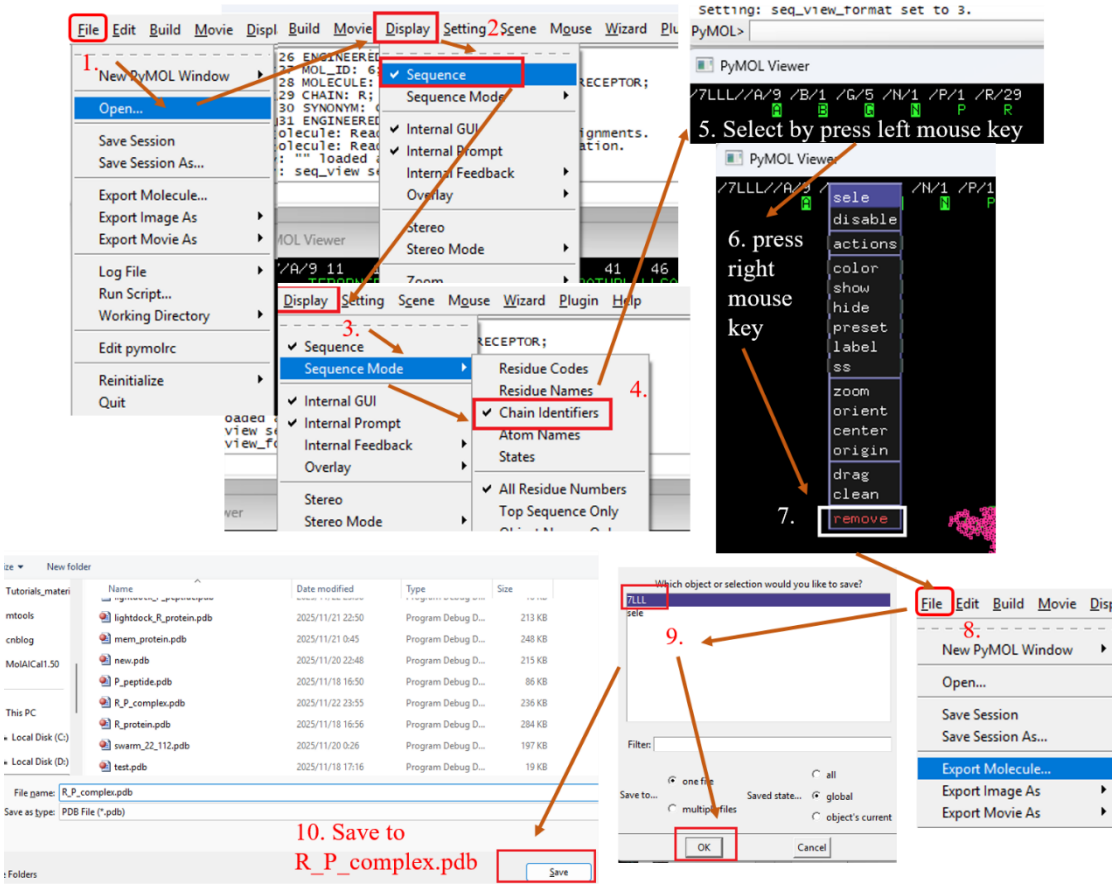

图 a1

上述文件分别从 **GLP-1R 与艾塞那肽-4**（首个 FDA 批准的 GLP-1R 激动剂，PDB ID: 7LLL.pdb） 的复合物中提取： - “R_P_complex.pdb”（链名为 “P” 和 “R”） - “R_protein.pdb”（链名为 “R”） - “P_peptide.pdb”（链名为 “P”）

艾塞那肽-4 是一种肽段。若用户希望进行 **蛋白质-蛋白质** 或 **蛋白质-核酸** 的虚拟筛选，只需将肽段文件替换为蛋白质或核酸文件即可（肽段可视为小型蛋白质）。

可选操作：MolAICal 默认生成的肽段链名为 “Z”。若两个分子的链名相同（例如均为 “Z”），用户可通过 MolAICal 调用 VMD 对受体链重命名（如改为 “A”，但不能为 “Z”）。如以下命令：

```
#> molaical.exe -call run -c vmdargs -i -pdb R_protein.pdb -args none set sel [atomselect top all], \\\$sel set chain A, \\\$sel writepdb R_protein_A.pdb
```

注意：

- 它包括外部程序的所有命令行参数，但'**-i**'参数必须是最后指定的，而所有其他标识符(例如，'-call'、'-c'等)必须在'-i'之前。
- '**-pdb**'表示将 pdb 文件加载到 VMD 中。
- **-c:** 如果其值为"vmdargs"，它将在 Tk 控制台中运行 VMD 命令一次。
- '-args'后面的字符串表示可以在 VMD 的 Tk 控制台中运行的命令。
  - '-args'后的第一个参数可以是 VMD 的包名称或'none'。如果'-args'后第一个参数是'none'，它将省略加载包。例如，如果'-args'后第一个参数是'autopsf'，它将在 VMD 的 Tk 控制台中运行命令"package require autopsf"。如果'-args'后第一个参数是'none'，它将不会运行命令"package"。
  - '-args'后面的字符串包含一个可以在 VMD 的 Tk 控制台中执行的命令，每个逗号','之前；换句话说，逗号字符代替了在 VMD 的 Tk 控制台中输入每个命令并按 Enter 键的过程。这使得只需一个命令就可以执行多行命令。
  - 一些特殊字符，例如字符"$"，需要使用转义字符""。对于特殊字符，MolAICal 使用三个转义字符""(即"\")。

在本教程中未使用该可选操作，因 “R_protein.pdb” 的链名为 “R”，与默认肽段链名 “Z” 不冲突。

---

2. 制备残基约束文件

约束文件用于指定活性口袋中的关键残基，分子将在这些残基周围进行对接。用户可根据文献或经验自定义这些关键残基。

约束文件中残基的表示格式为：

```
Chain.Residue_Name.Residue_Number[.Residue_Insertion]
```

注意：“Residue_Insertion” 是可选的单字符标识符，用于 PDB 文件中在不重新编号整个序列的情况下区分插入残基。在生物信息学中，插入码对于准确追踪和描述特定残基至关重要。

若省略 “Residue_Insertion”，通用格式为：

```
Chain.Residue_Name.Residue_Number
```

或

```
Chain.Residue_Name.Residue_Number P
```

注意：标签 “P” 表示该约束为**被动约束**（passive restraint），与无标签的**主动约束**（active restraint）相对。除非完全理解其含义，否则不建议使用被动约束。

此处使用 MolAICal 在受体（蛋白质）中生成距离配体（肽段）**4.0 Å 以内**的残基列表文件（命名为 ‘rec.dat’）：

```
#> molaical.exe -call run -c sfile -i get_res.tcl -pdb 7LLL.pdb -args R P 4.0 rec.dat R
```

注意: 以下 1st、2nd、3rd 和 4th 分别表示第一个参数、第二个参数、第三个参数和第四个参数，依此类推。

- '-call': 与外部程序或命令交互。其值可以是'set'或'run'。'set'表示设置外部程序的环境，包括名称和路径。'run'表示调用外部程序，可以从设置的环境文件中搜索程序的路径。
- -c: 如果其值为"sfile"，它将运行 MolAICal 的 VMD 脚本，这里调用脚本文件'get_res.tcl'。
- '-pdb'表示将 pdb 文件加载到 VMD 中。
- 1st: 所选分子的链名；
- 2nd: 用于选择的参考分子的链名；
- 3rd: 在指定距离内从参考分子选择的所选分子的残基；
- 4th: 保存的文件名；
- 5th: 指定分子类型，'R'代表受体，'L'代表配体。

类似地，MolAICal 用于生成配体(肽)在受体(蛋白质)4.0Å 范围内的残基列表文件(命名为'lig.dat')作为关键残基，如下所示:

```
#> molaical.exe -call run -c sfile -i get_res.tcl -pdb 7LLL.pdb -args P R 4.0 lig.dat L
```

将'rec.dat'和'lig.dat'合并到名为'restraints.list'的约束文件中，如下所示:

```
#> molaical.exe -call run -c ecommand -i cat rec.dat lig.dat > restraints.list
```

在许多情况下，配体的具体细节是未知的。因此，本教程仅使用受体(第一个分子)的关键残基进行约束(并根据经验手动删除不必要的氨基酸)，命令如下:

```
#> molaical.exe -call run -c ecommand -i cp rec.dat restraints.list
```

3. 分子对接设置

首先，运行群体(swarm)生成的设置，如下所示:

```
#> molaical.exe -call run -c ldock -i lightdock3_setup.py R_protein.pdb P_peptide.pdb --noxt
--noh --now -sp -rst restraints.list
```

- '-call': 与外部程序或命令交互。其值可以是'set'或'run'。'set'表示设置外部程序的环境，包括名称和路径。'run'表示调用外部程序，可以从设置的环境文件中搜索程序的路径。
- -c: 如果其值为"ldock"，它将通过调用"LightDock"运行分子对接(蛋白质-蛋白质、蛋白质-肽、蛋白质-DNA 或蛋白质-RNA)。
- "lightdock3_setup.py"后的第一个和第二个参数分别是受体和配体文件。
- --noxt: 如果启用此选项，LightDock 将忽略 OXT 原子。这对于几种不理解这种特殊类型原子的评分函数很有用。
- --noh: 如果启用此选项，LightDock 将忽略氢原子。这对于 dfire、fastdfire 或 dfire2 评分函数(以及其他)相关。仅删除以 H 字符开头的原子，因此不识别 1H 等类型。
- --now: 如果启用此选项，晶体水分子将被删除。
- --anm: 如果启用此选项，将激活 ANM 模式，并使用 ANM(通过 ProDy)建模主链柔性。该参数通过 ANM（使用 ProDy）启用骨架柔性建模，可能导致对接结果的结构畸变。**除非实施更优构象采样（如能量最小化），否则应避免使用。**替代方案：预生成多重构象以规避对接过程中的骨架柔性问题。
- -membrane: 启用时，此标志考虑受体伙伴沿 Z 轴对齐，并过滤掉与跨膜域不兼容的群体。为此，它使用提供的受体约束残基。
- -transmembrane: 参数"-transmembrane"具有与"-membrane"相反的功能。启用时，此标志考虑受体伙伴沿 Z 轴对齐，并过滤掉不在膜内的群体。
- -sp: 如果启用(默认为 False)，将在 init 文件夹中写入调试额外文件。
- -r, -rst restraints_file: 如果提供了 restraints_file，在设置和模拟过程中将考虑残基约束。如果没有提供 restraints_file，设置和模拟将集中在整个受体上。

- 它将生成名为'**init**'的文件夹，其中包含代表群体萤火虫的 PDB 格式分子，用于进一步对接。
- 文件'lightdock_R_protein.pdb'和'lightdock_P_peptide.pdb'是由 LightDock 从'R_protein.pdb'和'P_peptide.pdb'重新生成的

在 PyMol 中打开 ‘lightdock_R_protein.pdb’ 和 ‘init’ 文件夹中所有以 “starting_positions*” 开头的 PDB 文件（见图 a2）。

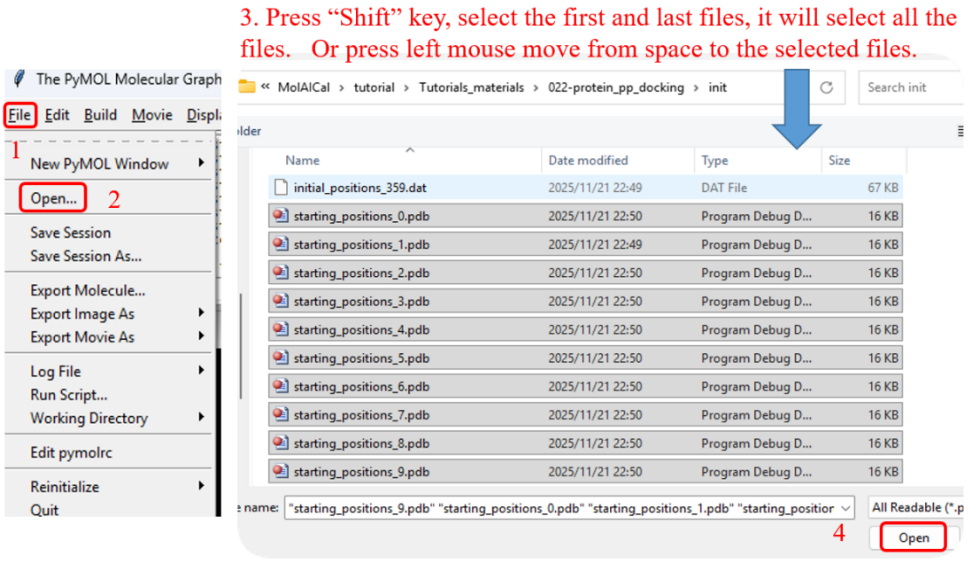

图 a2

结果如 **图 a3** 所示：许多 glowworm 位于跨膜区域内，而配体肽段位于膜蛋白 7 次跨膜螺旋中央的胞外侧口袋中。这些跨膜区内的 glowworm 不仅不合理，还会消耗大量计算资源和时间，可能影响对接结果。

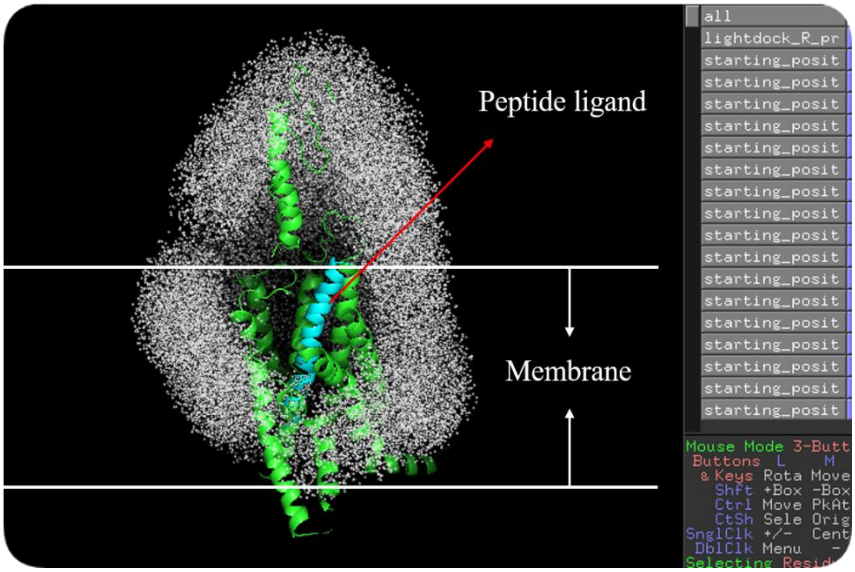

图 a3

因此，MolAICal 提供了一种简单的膜生成方法，以尽量减少不合理跨膜 glowworm 的生成。

3.1（可选）在受体中设置膜结构

若目标受体不是膜蛋白，请跳过此步！

3.1.1（可选）首先在 VMD 中打开 “lightdock_R_protein.pdb”，目视检查受体在 x、y、z 方向的取向。若跨膜区（朝向胞外侧）已大致沿 z 轴排列，则无需操作；否则使用 MolAICal 调整其方向：

```
#> molaical.exe -call run -c sfile -i align_axis.tcl -pdb R_protein.pdb -args protein 2 new.pdb
```

- '-call': 与外部程序或命令交互。其值可以是'set'或'run'。'set'表示设置外部程序的环境，包括名称和路径。'run'表示调用外部程序，可以从设置的环境文件中搜索程序的路径。
- -c: 如果其值为"sfile"，它将运行 MolAICal 的 VMD 脚本，这里调用脚本文件'align_axis.tcl'。
- '-pdb'表示将 pdb 文件加载到 VMD 中。
- 1st: 原子选择(与 VMD 的 Tk 控制台中"atomselect"命令相同)。如果有一个以上的字母(例如，protein 和 chain B)，它将使用逗号","表示空格" "(例如，protein,and,chain,B)；
- 2nd: 如果蛋白质水平对齐(平躺)，将 0 更改为 2；
- 3rd: 输出文件名；
- 4th: VMD 的"transaxis"所选轴，可以是 x、y、z 或""。""可以等于无字符，包括用于输入的空格。

文件"new.pdb"沿 z 轴对齐，将用于以下步骤。

3.1.2 确认受体大致沿 z 轴后，识别用于膜生成的两个受体位置：

- ✧ 首先，在 VMD 中打开"R_protein.pdb"
- ✧ 其次，按住数字键'1'，然后用鼠标大致点击膜的内边缘和外边缘附近。这些位置可以根据实验要求进行调整，主要是限制膜之间萤火虫的生成。
- ✧ 获取这两个位置的 z 轴值，并将它们用作范围[min, max]的最小值和最大值(如**图 a4** 所示)

这里，[min, max]=[-29.0, 10.0]。这些只是粗略的数字。用户可以根据研究目的进行调整。然后，运行以下命令进行膜生成:

```
#> molaical.exe -call run -c sfile -i gen_mem.tcl -pdb new.pdb -args protein -29.0 10.0 mempro.pdb
```

- '-call': 与外部程序或命令交互。其值可以是'set'或'run'。'set'表示设置外部程序的环境，包括名称和路径。'run'表示调用外部程序，可以从设置的环境文件中搜索程序的路径。
- -c: 如果其值为"sfile"，它将运行 MolAICal 的 VMD 脚本，这里调用脚本文件'gen_mem.tcl'。
- '-pdb'表示将 pdb 文件加载到 VMD 中。
- 1st: 原子选择(与 VMD 的 Tk 控制台中"atomselect"命令相同)。如果有一个以上的字母(例如，protein 和 chain B)，它将使用逗号","表示空格" "(例如，protein,and,chain,B)；
- 2nd: 膜生成的最小 z 轴；
- 3rd: 膜生成的最大 z 轴；
- 4th: 保存的输出文件名；
- 5th: delta 值用于[($max - $delta), $max]，表示用于确定沿 z 轴区域的最大和最小 x,y 值的范围值。这确保膜横截面中的膜颗粒不会位于膜蛋白内部。默认值: 3.0；
- 6th: 边界框的安全缓冲区(设置 0 以严格使用蛋白质限制)。默认值: -2.0；
- 7th: 膜填充(埃)。默认值: 15.0；

- 8th: 颗粒之间的最小间距。默认值: 3.5；
- 9th: 颗粒之间的最大间距。默认值: 6.0；
- 10th: 与蛋白质原子保持的距离(埃)。默认值: 4.0。

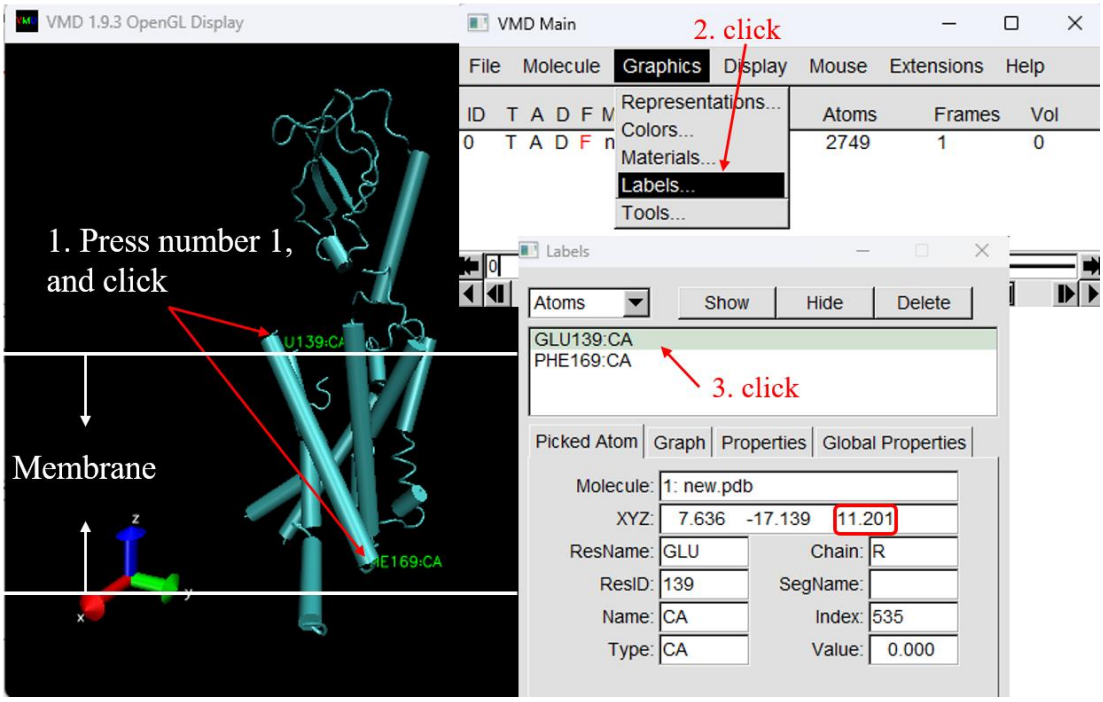

图 a4

3.1.3 生成包含受体和膜分子的文件 “mempro.pdb”

- **选项:** 删除先前生成的文件(如果不删除，将导致一些错误)。如果先前生成的文件不存在，请跳过此步骤。

```
#> molaical.exe molaical.exe -call run -c ecommand -i rm -rf init swarm_* lightdock_*
```

重新运行群体（swarm）生成的设置(添加参数: -membrane):

```
#> molaical.exe -call run -c ldock -i lightdock3_setup.py mempro.pdb P_peptide.pdb
-membrane --noxt --noh --now -sp -rst restraints.list
```

注意: **-transmembrane**: 参数"–transmembrane"具有与"–membrane"相反的功能。启用时，此标志考虑受体伙伴沿 Z 轴对齐，并过滤掉不在膜内的群体。

很明显，生成的群体数量比以前少。通过 PyMol 打开'lightdock_mempro.pdb'和'init'文件夹中所有前缀为"starting_positions*"的 PDB 文件(如**图 a2** 所示的方法)。重新生成的群体萤火虫如**图 a5** 所示；值得注意的是，膜之间没有生成群体，这是合理的。

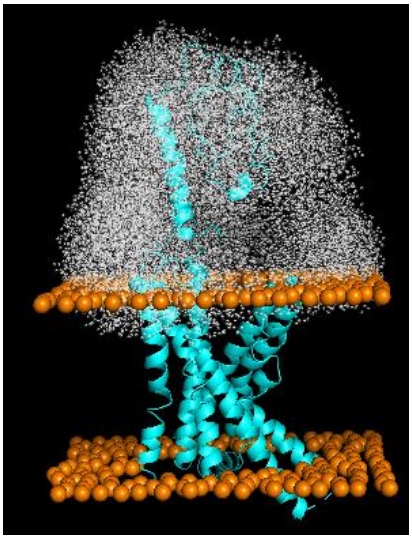

图 a5

至此，群体和受体已处理完毕。

附录 2

在 MolAICal 容器内安装外部程序（推荐，适用于 Linux 版本的 MolAICal）

请注意，Windows 版与 Linux 版的 MolAICal 配置存在差异：Linux 版本采用基于 udocker 容器的技术方案，需在容器内部完成设置，而 Windows 版本则无需此步骤。

为解决 Linux 环境中的库依赖问题，Linux 版 MolAICal 采用容器化部署方案。如需调用外部程序，建议在 MolAICal 容器内进行安装；若无外部程序调用需求，可忽略此步骤。本教程涉及外部程序 NAMD 和 VMD 的调用，具体操作流程如下：

1. 首先，将文件复制到 MolAICal 容器中

```
# 进入容器文件系统（将进入 "/root" 目录）
#> molaical.exe -eset shell in

# 将 VMD 和 NAMD 安装包从本地机器复制到容器中，'cp' 命令的第一部分（源路径）
# 位于本地主机，第二部分（目标路径）在容器内；VMD 和 NAMD 软件包可通过以下命
# 令移入容器。
#> cp /home/user/<本地文件> /root/soft

# 退出容器文件系统
#> exit
```

2. 其次，进入 MolAICal 容器的虚拟环境

```
#> molaical.exe -eset sys run molaical
```

注：molaical 是容器名称。

3. 在 MolAICal 容器虚拟环境中安装软件的方式与在本地主机上安装相同。以下以安装 VMD 和 NAMD 为例：

1) 安装 NAMD：

解压 NAMD 文件（假设解压后的文件夹名为 namdcpu），然后使用以下命令将其路径告知 MolAICal：

```
#> molaical.exe -call set -n NAMD -p "/root/soft/namdcpu/namd3"
```

注：请将上述 VMD 和 NAMD 的路径替换为您系统中的实际路径。-n 后面的 "VMD" 和 "NAMD"（大小写不敏感）是固定的标识符。为确保 MM/GBSA 结果的可重复性，建议使用 NAMD 的 CPU 版本，因为 CUDA 版本中的 seed 参数似乎对结果可重复性无效。

2) 安装 VMD：

按以下步骤操作：

◆ **解压 VMD 文件：**

```
#> tar -xzvf vmd-xxx.tar.gz
```

注：请将上述路径替换为您系统中的实际路径。

◆ **修改 VMD 解压目录中名为 configure 的文件中的安装路径：**

```
# 默认值：
$install_bin_dir="/usr/local/bin";
$install_library_dir="/usr/local/lib/$install_name";

# 修改为：
$install_bin_dir="/root/soft/vmd193/bin";
$install_library_dir="/root/soft/vmd193/lib/$install_name";
```

◆ **安装 VMD：**

```
#> cd vmd-xxx
#> ./configure LINUXAMD64
#> cd src
#> make install
```

注：请运行 ./configure 并根据所用计算机选择正确的类型，此处为 "LINUXAMD64"。

◆ **然后使用以下命令将 VMD 路径告知 MolAICal：**

```
#> molaical.exe -call set -n VMD -p "/root/soft/vmd193/bin/vmd"
```

至此，NAMD 和 VMD 在 MolAICal 容器内的安装与配置已完成。

✧ **记得使用 exit 命令退出 MolAICal 虚拟环境，返回本地计算机进行计算（主要是为了省去文件拷贝步骤；在容器内运行也可行，但需手动将数据从本地计算机复制到 MolAICal 容器中）。**

```
#> exit
```

3) 保存并加载已修改的容器（可选）

为保留容器内所做的更改，并避免日后重新安装软件（因为一旦容器被删除，所有数据都会丢失），可执行以下操作：

✧ **获取容器 ID 和名称：**

```
#> molaical.exe -eset sys ps
```

✧ **克隆该容器：**

```
#> molaical.exe -eset sys export --clone -o molaicalv2.tar molaical
```

注：`molaical` 是容器名称，也可使用容器 ID。如果报错，那可能是因为挂载的文件系统的权限问题，可以忽略。

✧ **假如需要，导入克隆的容器：**

```
#> molaical.exe -eset sys import --clone --name molaicalv2 molaicalv2.tar
```

✧ **将新导入的容器名称更改为默认容器名 "molaical"，原容器重命名为 "bakmolaical"：**

```
#> molaical.exe -eset sys rename molaical bakmolaical
#> molaical.exe -eset sys rename molaicalv2 molaical
```

注意：容器（动态）是从镜像（静态）生成的。软件安装等操作必须在动态容器中进行；如果克隆了该容器，恢复时仍基于原始底层镜像。

更多详情请参阅 MolAICal 用户手册，或运行以下命令获取帮助：

```
#> molaical.exe -eset sys --help
```