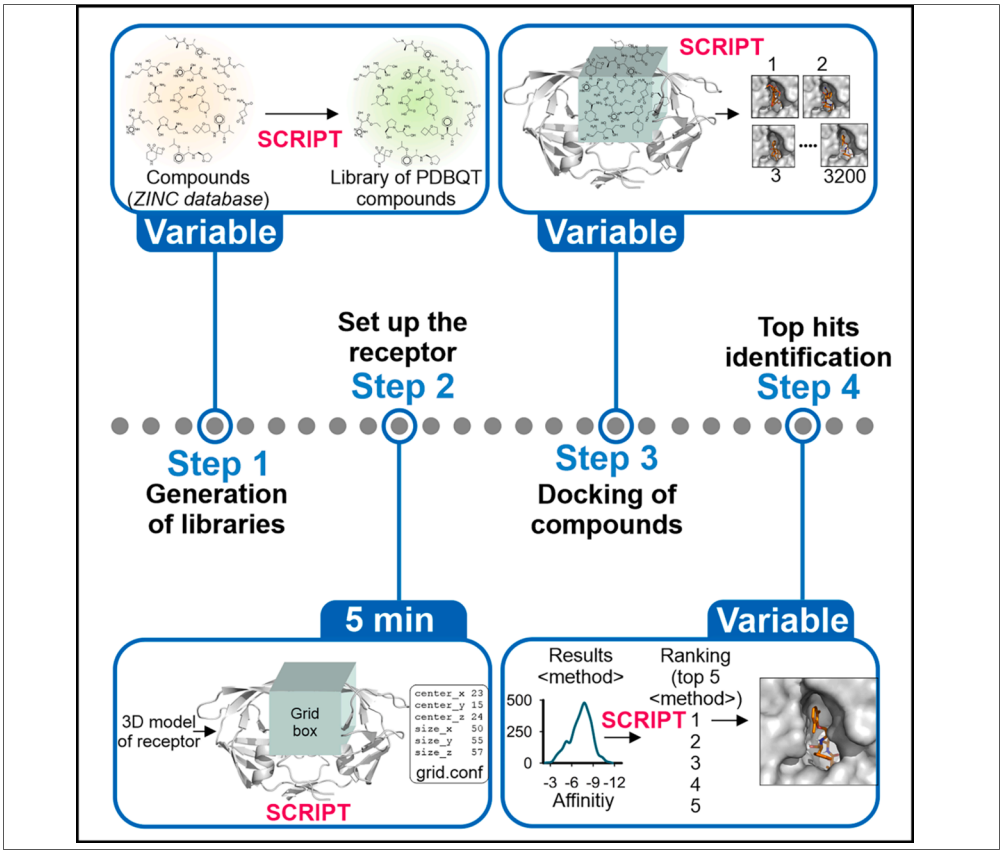
STAR Protocols

CellPress
OPEN ACCESS

Protocol

Protocol for an automated virtual screening pipeline including library generation and docking evaluation



Here, we present a protocol for an automated virtual screening pipeline. We describe steps for generating compound libraries for computational docking including Food and Drug Administration (FDA)-approved drugs, setting up the receptor and grid box, and docking a library of compounds. We then detail procedures for ranking docking results. This protocol offers scripts for Unix-like systems, lowering the access barrier for researchers interested in structure-based drug discovery and supporting more experienced users by improving the efficiency of their studies.

Publisher's note: Undertaking any experimental protocol requires adherence to local institutional guidelines for laboratory safety and ethics.

Pedro José Barbosa Pereira, Jorge Ripoll-Rozada, Sandra Macedo-Ribeiro, José Antonio Manso

joseantonio.manso@unican.es

Highlights

Steps for setting up a fully local virtual screening pipeline using free software

Instructions for generating compound libraries compatible with AutoDock Vina

Guidance on receptor setup, docking execution, and results ranking using scripts

Barbosa Pereira et al., STAR Protocols 6, 104161
December 19, 2025
https://doi.org/10.1016/j.xpro.2025.104161

STAR Protocols

CellPress
OPEN ACCESS

Protocol

Protocol for an automated virtual screening pipeline including library generation and docking evaluation

Pedro José Barbosa Pereira,[1,2] Jorge Ripoll-Rozada,[3,4] Sandra Macedo-Ribeiro,[1,2] and José Antonio Manso[3,4,5,6,*]

[1]IBMC – Instituto de Biologia Molecular e Celular, Universidade do Porto, 4200-135 Porto, Portugal

[2]i3S – Instituto de Investigação e Inovação em Saúde, Universidade do Porto, 4200-135 Porto, Portugal

[3]IBBTEC – Instituto de Biomedicina y Biotecnología de Cantabria, Consejo Superior de Investigaciones Científicas (CSIC) – Universidad de Cantabria, 39011 Santander, Spain

[4]Departamento de Biología Molecular, Universidad de Cantabria, 39011 Santander, Spain

[5]Technical contact

[6]Lead contact

*Correspondence: joseantonio.manso@unican.es
https://doi.org/10.1016/j.xpro.2025.104161

SUMMARY

Here, we present a protocol for an automated virtual screening pipeline. We describe steps for generating compound libraries for computational docking including Food and Drug Administration (FDA)-approved drugs, setting up the receptor and grid box, and docking a library of compounds. We then detail procedures for ranking docking results. This protocol offers scripts for Unix-like systems, lowering the access barrier for researchers interested in structure-based drug discovery and supporting more experienced users by improving the efficiency of their studies.

BEFORE YOU BEGIN

Virtual screening is a powerful computational approach for drug discovery,[1] and its effectiveness has been greatly enhanced by the access to vast compound collections such as ZINC,[2–4] a publicly accessible and free resource that hosts the chemical and structural information of millions of commercially-available compounds. However, the absence of PDBQT-format files in ZINC can hinder the generation of large compound libraries by some of the most popular docking tools in virtual screening like AutoDock Vina (Vina), which requires inputs in this specific format. Vina is widely used due its ease of use, support for ligand flexibility, and relatively accurate binding pose predictions.[5,6] Preparing thousands - or even millions - of PDBQT-format files manually can be an arduous and time-consuming task, particularly for users without extensive experience. Additionally, the arbitrary selection of screening areas on the target can slow down the process and be a source of result variability. Finally, ranking a large number of docking outcomes is often complex, especially when relying solely on local environments, using only free software, and avoiding commercial or cloud-based tools.

Several tools have greatly contributed to making molecular docking more accessible and automated, including MzDOCK,[7] Raccoon2,[8] and DOCK Blaster,[9] among others. Each has been developed with a specific focus. Raccoon2 enables compound library creation through a web interface, well suited for users who prefer graphical user interface (GUI)-based workflows or lack extensive local computing resources. MzDOCK streamlines docking through a GUI but assumes compound libraries are already preprocessed. DOCK Blaster provides automated pipelines built around the DOCK software suite, which are high-performance engines designed for large-scale virtual screening tasks.[10] Its preconfigured workflows simplify the process and reduce the need for

manual setup, especially for users seeking a streamlined solution. Ongoing developments continue to expand its automation capabilities.[11,12]

Innovation

We present a fully local, script-based protocol for Unix-like systems that uses only free and open-source software, without requiring external GUIs. This protocol fulfills some needs in the field automating an entire virtual screening process with Vina —from compound library generation to docking evaluation—all in a fully local environment. Designed for accessibility, the protocol includes step-by-step instructions and is ideal for users with limited experience in molecular docking. The workflow consists of five modular customized computational programs that automate the entire structure-based virtual screening pipeline (Figure 1):

jamlib generates compound libraries ranging from customizable sets of molecules to the USA Food and Drug Administration (FDA)-approved drugs. All molecules are energy-minimized and converted into PDBQT format, addressing the lack of Vina-compatible files in the FDA catalog of the ZINC. **jamreceptor** prepares the receptor by converting PDB files to PDBQT format and analyzing binding sites. Users select target pockets, which are then used to define the docking grid box. **jamqvina** automates docking across the entire compound library. This command-line tool supports local machines, cloud servers, and HPC clusters, offering better scalability than GUI-based tools. **jamresume** enables resuming jobs, ensuring robustness during long-running processes. **jamrank** evaluates and ranks docking results using two scoring methods, helping identify the most promising hits.

This modular approach offers a flexible and efficient virtual screening tool, ideal for early drug discovery and repurposing, suitable for both beginners and experts.

System setup

Timing: 35 min

Note: This protocol is designed for use on Linux- or Unix-based operating systems. It can also be run on Windows 11 using the Windows Subsystem for Linux (WSL). For macOS users, please see the instructions described in Supplemental Materials (Data S1).

Installing WSL for Windows 11 users

Timing: 5 min

1. Open Windows PowerShell as administrator (right-click and select "Run as administrator").
2. Run the following command:

```
$ wsl --install
```

Note: During the first installation of WSL, the system may require a restart to complete the process. After restarting the system will complete the installation of an Ubuntu distribution.

Note: For more detailed instructions, refer to the official Microsoft guide: https://learn.microsoft.com/en-us/windows/wsl/install.

3. Click on the Ubuntu icon and create a default user account.
4. Continue with step 5 in the section *Installing the software dependencies*.

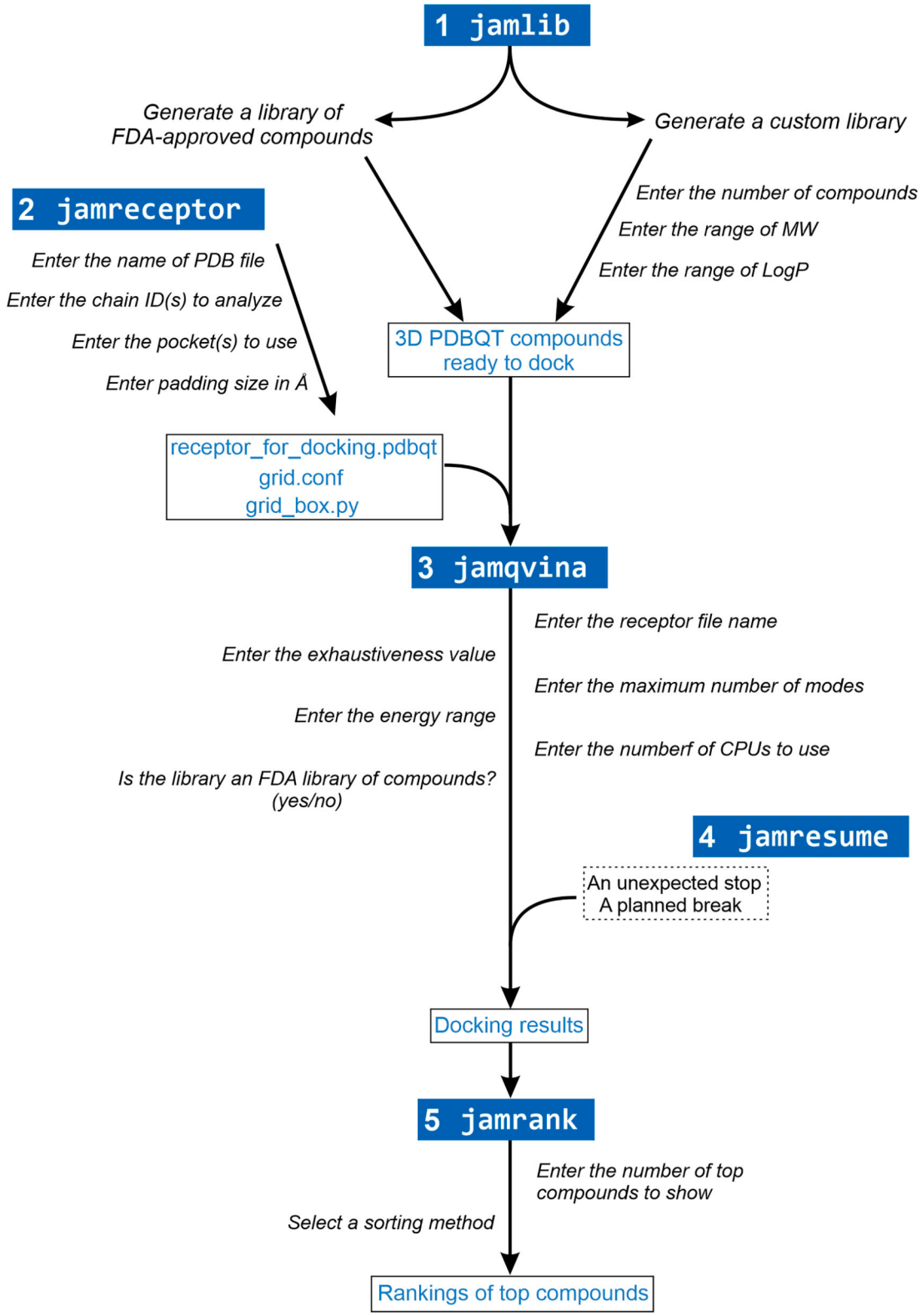


Figure 1. Workflow of the protocol to perform virtual screening indicating the names of the five scripts that automate the process

Installing the software dependencies

Timing: 30 min

5. System update and essential packages installation.
   a. Open a Bash terminal.

   Note: Windows users should run these commands inside a WSL terminal.

   b. Update and upgrade system packages. In the terminal execute the following command:

```
$ sudo apt update && sudo apt upgrade -y
```

   ⚠ CRITICAL: You will be prompted for your superuser password. If you do not have superuser privileges, contact your system administrator.

   c. Install essential packages and software:

```
$ sudo apt install -y build-essential gedit cmake openbabel pymol libxmu6 wget curl bc git \
libboost1.74-all-dev xutils-dev
```

   Note: If you encounter problems running gedit or PyMOL under WSL, please refer to the troubleshooting section.

6. Install AutoDockTools

   Note: AutoDockTools,[13] distributed as part of MGLTools, is required by the jamreceptor script described later in this protocol to generate input files for Vina.

   a. Create a directory for software installation:

```
$ mkdir ~/Programs

$ cd ~/Programs
```

   b. Download and extract MGLTools:

```
$ wget https://ccsb.scripps.edu/mgltools/download/491/mgltools_Linux-x86_64_1.5.7.\
tar.gz

$ tar -zxf mgltools_Linux-x86_64_1.5.7.tar.gz
```

   Note: This command will download version 1.5.7. Users can check the MGLTools download page https://ccsb.scripps.edu/mgltools/downloads/ for newer versions and adapt the commands accordingly.

   c. Install AutoDockTools:

```
$ cd mgltools_x86_64Linux2_1.5.7/

$ ./install.sh
```

d. Configure AutoDockTools alias in your shell:

```
$ echo "alias adt=\"$HOME/Programs/mgltools_x86_64Linux2_1.5.7/bin/adt\"" >> ~/.bashrc
$ source ~/.bashrc
```

7. Install fpocket:

Note: The jamreceptor script uses fpocket, an open-source software for ligand-binding pocket detection and characterization.[14] Fpocket not only identifies potential binding cavities but also provides druggability scores to facilitate selection of relevant docking sites.[15]

a. Clone and build fpocket:

```
$ cd ~/Programs
$ git clone https://github.com/Discngine/fpocket.git
$ cd fpocket
$ make
```

b. Install fpocket system-wide:

```
$ sudo make install
```

8. Install AutoDock Vina (QuickVina 2)

Note: This protocol uses QuickVina 2,[16] a fast and accurate variant of Vina. The jamqvina script relies on this software, though other versions may be used with minor modifications.

a. Clone the QuickVina repository:

```
$ cd ~/Programs
$ git clone https://github.com/QVina/qvina.git
$ cd qvina
$ git checkout qvina2
```

b. Open the Makefile in a text editor and ensure the following lines match your system (e.g., for Ubuntu with Boost 1.74):

```
BOOST_VERSION=1_74
BOOST_INCLUDE = /usr/include
```

c. Build QuickVina 2:

```
$ make
```

d. Configure shell environment for QuickVina 2:

```
$ echo "alias qvina02=$HOME/Programs/qvina/qvina02" >> ~/.bashrc

$ echo "export PATH=$HOME/Programs/qvina:\$PATH" >> ~/.bashrc

$ source ~/.bashrc
```

9. Download the protocol scripts and configure the environment
   a. Clone the jamdock-suite repository:

```
$ cd ~/Programs

$ git clone https://github.com/jamanso/jamdock-suite.git

$ cd jamdock-suite
```

   b. Make the programs executable:

```
$ chmod +x jam*
```

   c. Add jamdock-suite to your shell's path:

```
$ echo "export PATH=\"$HOME/Programs/jamdock-suite:\$PATH\"" >> ~/.bashrc

$ source ~/.bashrc
```

Note: You can now invoke jamlib, jamreceptor, jamqvina, jamresume and jamrank from any terminal window.

Note: This setup modifies the shell environment only of the current user session and does not apply system-wide.

KEY RESOURCES TABLE

REAGENT or RESOURCE	SOURCE	IDENTIFIER
Deposited data		
Bash scripts for a virtual screening pipeline	This paper; Zenodo: https://doi.org/10.5281/zenodo.15577778	GitHub: https://github.com/jamanso/jamdock-suite
Commercially available compounds for virtual screening	ZINC and files.docking.org databases	https://zinc.docking.org/ https://files.docking.org/ https://files2.docking.org/
Crystal structure of the vitamin D nuclear receptor ligand-binding domain	Tocchini-Valentini et al.[17]	RCSB Protein Data Bank (PDB): 1IE9
Crystal structure of the mineralocorticoid receptor ligand-binding domain	Hasui et al.[18]	PDB: 3VHU
Crystal structure of the light-oxygen-voltage sensor domain	Rivera-Cancel et al.[19]	PDB: 4R38
Crystal structure of the progesterone receptor ligand-binding domain	Madauss et al.[20]	PDB: 1SQN
Software and algorithms		
Windows 11	Microsoft	https://www.microsoft.com/en-us/software-download/windows11

(*Continued on next page*)

Continued

REAGENT or RESOURCE	SOURCE	IDENTIFIER
Windows Subsystem for Linux (WSL) 2.6.1.0	Microsoft	https://learn.microsoft.com/en-us/windows/wsl/install
Ubuntu 24.04 LTS	Ubuntu	https://ubuntu.com/blog/tag/ubuntu-24-04-lts
macOS Sonoma 14.7.6	Apple	https://www.apple.com/app-store/
Python 3	Python	https://www.python.org/downloads/
gedit 46.2	Gedit Technology	https://gedit-text-editor.org/
Cmake 3.28.3	Kitware, Inc.	https://cmake.org/download/
Open Babel 3.1.1	Open Babel	https://openbabel.org/
PyMOL 2.5.0	Schrodinger, LLC	https://www.pymol.org/
GNU Wget 1.21.4	GNU Project	https://www.gnu.org/software/wget/
GNU bc 1.07.1	GNU Project	https://www.gnu.org/software/bc/
curl 8.5.0	curl project	https://curl.se/
git 2.43	Git SCM	https://git-scm.com/
Boost 1.74	Boost C++ Libraries	https://www.boost.org/
xutils-dev 1:7.7.7+6.2	Ubuntu repository	https://launchpad.net/ubuntu/+source/xutils-dev
libxmu6 2:1.1.3-3build2	Ubuntu repository	https://launchpad.net/ubuntu/+source/libxmu
AutoDockTools 1.5.7	Scripps Research Institute, San Diego	https://ccsb.scripps.edu/mgltools/
QuickVina 2.1	QuickVina	https://qvina.github.io//
fpocket 4.0	fpocket	https://github.com/Discngine/fpocket
Xcode Command Line Tools 12.3	Apple	https://developer.apple.com/xcode/resources/
Homebrew 4.6.7	Homebrew Project	https://brew.sh/
Makedepend 1.0.9	Homebrew	https://formulae.brew.sh/formula/makedepend
XQuartz 2.8.5	XQuartz Project	https://www.xquartz.org/
GNU Bash 5.3.3	GNU Project	https://www.gnu.org/software/bash/
Coreutils – GNU core utilities 9.7	GNU Project	https://www.gnu.org/software/coreutils/
Other		
Processor: 12th Gen Intel Core i5-12400, 2.5 GHz, 6 cores. RAM: 32 GB (2 × 16 GB) DDR4 3,200 MHz CL16.	Intel	https://www.intel.la/content/www/xl/es/products/sku/134586/intel-core-i512400-processor-18m-cache-up-to-4-40-ghz/specifications.html
Processor: 1.8 GHz dual-core Intel Core i5. 8 GB 1,600 MHz DDR3	Apple	N/A

STEP-BY-STEP METHOD DETAILS

This section provides detailed, step-by-step instructions for using the programs included in this protocol. The steps cover how to: (1) generate libraries of compounds for computational docking; (2) automatically prepare a receptor and define a grid box; (3) perform docking of a library of compounds; and (4) rank the docking results. As demonstration cases, we applied the pipeline to four study systems using an FDA-approved compound library.

Generating libraries of compounds for computational docking

Timing: Variable

Use the jamlib script (see Data S2) to generate compound libraries compatible with Vina, including QuickVina 2 and other forks. This tool automates compound retrieval, filtering, energy minimization, and file format conversion.

1. Create a working directory:
   a. Open a terminal window.
   b. Create a new directory for your project:

```
$ mkdir ~/<your_project_name>
```

c. Navigate into the directory:

```
$ cd ~/<your_project_name>
```

2. Launch the library generation script:
   a. Run the script in your working directory:

```
$ jamlib
```

Note: This launches an interactive menu (Figure 2) that allows you to chosen between generating a custom library or a library of FDA-approved compounds. Please avoid running multiple instances of this script or making excessive requests. We encourage responsible usage to avoid overloading the database servers.

Optional: 1. Generate a custom library of purchasable compounds

3. Select the "*Generate a custom library*" option from the menu.
4. When prompted, enter the desired filtering parameters:
   a. Molecular weight range.
   b. LogP range.
   c. Total number of compounds.

Note: Compounds are randomly selected from approximately 14 million purchasable molecules available.

5. Wait while the script performs the following preprocessing steps:
   a. Download compounds.

⚠ CRITICAL: The script retrieves compounds from files.docking.org and files2.docking.org. Server maintenance or interruptions may affect access. The program will notify the user in such cases. See troubleshooting for details.

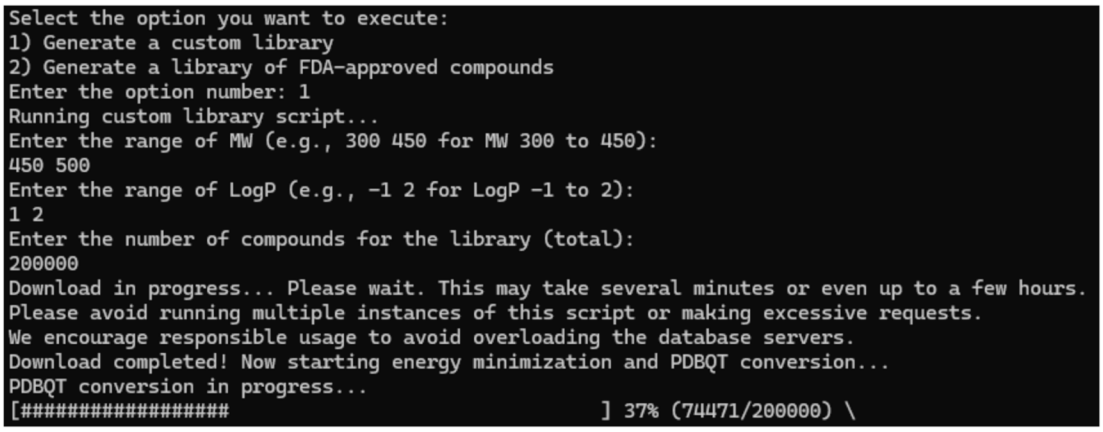


Figure 2. Screenshot of an example of jamlib execution

b. Filter compounds based on your input criteria.
c. Perform energy minimization using the Merck Molecular Force Field (MMFF94).
d. Convert structures to PDBQT format using Open Babel.[21]

Note: Processing time depends on the number of selected compounds. The resulting files are stored in the following directories inside the working directory: "library_sdf_<number of compounds>/" and "library_pdbqt_<number of compounds>/". These files are ready for downstream docking with Vina.

Optional: 2. Generate a library of FDA-approved compounds

6. Select the "*Generate a library of FDA-approved compounds*" option from the menu.
7. Wait while the script performs the following steps:
   a. Retrieve FDA-approved compounds from the FDA catalog of the ZINC database.

⚠ CRITICAL: The script retrieves compounds from the zinc.docking.org FDA catalog. Server maintenance or interruptions will affect program execution. The program will notify the user if such issues occur. See troubleshooting for more information.

   b. Perform energy minimization and convert all structures to PDBQT format.

Note: The processed compounds are saved in the following directories inside the working directory: "fda_sdf_compounds/" and "fda_pdbqt_compounds/". These files are ready for docking with Vina.

Automatic preparation of receptor and grid box

Timing: 5 min

Use the jamreceptor script (see Data S3) to automate receptor preparation for docking. This tool processes a standard PDB file, performs structural cleaning, and defines the docking grid box based on pocket analysis (Figure 3).

8. Prepare your input file.
   a. Place your receptor structure (e.g., receptor.pdb) in your working directory.

⚠ CRITICAL: The input file must be in a standard PDB format.

9. Launch the jamreceptor script.
   a. In the terminal, go to your working directory and execute the script:

```
$ jamreceptor
```

   b. When prompted, input the name of the PDB file you want to process.

⚠ CRITICAL: It is crucial to execute the script in the same directory where the PDB file is located.

10. Select chain(s) of interest.
    a. Choose one or more chains to retain from the input PDB file.

```
Enter the name of the PDB file to process (e.g., 2q5k.pdb): 4R38.pdb
Enter the chain ID(s) to analyze (e.g., A or A B): A
Cleaning input PDB: keeping chains [A], removing waters, ligands, and other chains...
Cleaned PDB saved as '4R38_for_docking.pdb'.
Searching for prepare_receptor4.py under your home directory...
Found prepare_receptor4.py at: /home/jamanso/Programs/mgltools_x86_64Linux2_1.5.7/MGLToolsPckgs/AutoDockTools/Utilities24/prepare_receptor4.py
Running receptor preparation (PDB to PDBQT)...
setting PYTHONHOME environment
set verbose to  True
read  4R38_for_docking.pdb
setting up RPO with mode= automatic and outputfilename=  4R38_for_docking.pdbqt
charges_to_add= gasteiger
delete_single_nonstd_residues= None
adding gasteiger charges to peptide
Done! Output saved to '4R38_for_docking.pdbqt'. Log written to '4R38_for_docking_prep.log'.
Running Fpocket...
***** POCKET HUNTING BEGINS *****
***** POCKET HUNTING ENDS *****

Detected pockets:
-----------------
Pocket 1
Pocket 2 (could be druggable)
Pocket 3 (could be druggable)
Pocket 4
Pocket 5
Pocket 6

To view the detected pockets in Pymol:
- Press Ctrl + C to exit this script, then open '4R38_for_docking_out/4R38_for_docking.pml' using PyMOL.

Enter the pocket number(s) to use (separated by spaces): 2 3
xmin=2.891, xmax=19.139
ymin=-6.578, ymax=9.758
zmin=18.967, zmax=34.685
Enter padding size in Ångströms (default 10.0):
Using padding: 10.0 Å
Grid box parameters written to 'grid.conf'.
Visualization script 'grid_box.py' generated.
```

Figure 3. Screenshot showing an example execution of the jamreceptor script, including chain selection and pocket analysis

Note: The script will discard all atoms from unselected chains and proceed only with the specified chain(s).

11. Wait a few seconds while the script automatically performs the following steps:
    a. Clean the PDB file by removing non-protein atoms (e.g., water, ligands, ions).
    b. Add Gasteiger charges to the protein.
    c. Convert the structure to PDBQT format ready for use with QuickVina 2 or Vina (e.g., receptor.pdb → receptor_for_docking.pdbqt).
12. Analyze binding pockets with fpocket:
    a. Let the script run fpocket to identify ligand-binding pockets.
    b. Review the pocket analysis summary displayed in the terminal.
        i. Pockets with a druggability score > 0.15 are flagged as "could be druggable".

Note: Pocket annotations are shown in the terminal and saved to output files for review.

Optional: To visually inspect the detected pockets, press Ctrl + C to exit the script and open the receptor_for_docking.pml file located in the receptor_for_docking_out/ directory using PyMOL.

13. Define the docking grid box:
    a. Specify one or more pockets to use for grid preparation.

Note: Selecting multiple pockets may be useful when defining a larger binding region.

    b. Enter the desired padding size (in Å) to control the final grid box dimensions.

Note: The grid will be centered automatically on the selected pocket(s).

14. Locate output files.
    a. Verify that the following files are generated in your working directory:
        i. receptor_for_docking.pdbqt (docking-ready receptor).
        ii. grid.conf (grid box dimensions for docking).
        iii. grid_box.py (PyMOL script for grid box visualization).

Note: The grid.conf file is required for downstream docking with QuickVina 2. To visualize the defined docking region, run:

```
$ pymol grid_box.py
```

Note: You may also load receptor_for_docking.pdbqt along with the grid box to check the final docking region (Figure 4).

Docking a library of compounds

Timing: Variable

Use the jamqvina script (see Data S4) to automate molecular docking of a compound library into a receptor using QuickVina 2. Compound libraries should be prepared using jamlib, and the receptor should be processed with jamreceptor, as described in earlier steps.

Note: Although this protocol uses QuickVina 2, other variants (e.g., CUDA-enabled GPU versions) can also be used. See the note under Step 18 for guidance on adapting the script.

15. Launch the docking script:
    a. In a terminal window and in your working directory, execute the script by running:

```
$ jamqvina
```

Note: This will launch an interactive menu to configure docking parameters (see Figure 5).

⚠ CRITICAL: Ensure the following files and directories are present in your current working directory before launching the script: grid.conf file and receptor in PDBQT format (e.g., receptor_for_docking.pdbqt) generated by jamreceptor, and the compound library directories (e.g., "library_pdbqt_<number of compounds>/" or "fda_pdbqt_compounds/") generated by jamlib).

16. Provide docking parameters when prompted:
    a. Input the name of the receptor file (e.g., receptor_for_docking.pdbqt).
    b. Set the exhaustiveness level (higher values increase accuracy but decrease performance).
    c. Set the maximum number of binding modes to output per ligand.
    d. Specify the energy range (in kcal/mol).
    e. Enter the number of central processing unit (CPU) cores to use for parallel docking.
    f. Chose the type of compound library to dock (custom or FDA-approved).
17. Monitor docking progress:
    a. Let the script run the docking process automatically.

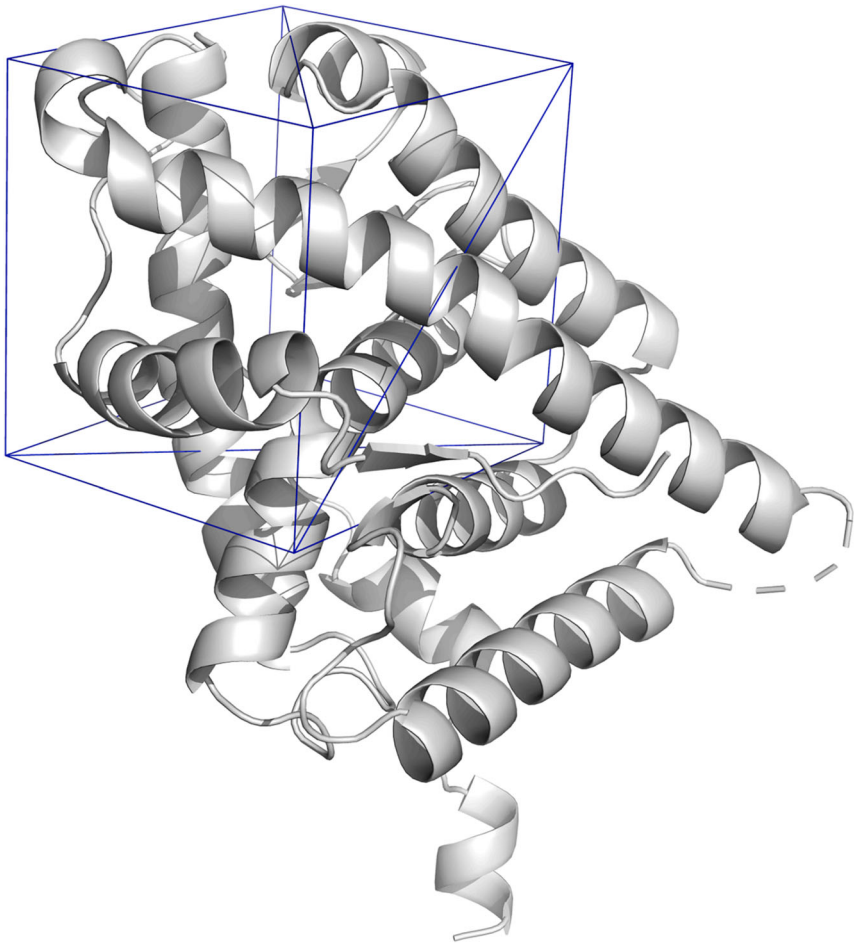


Figure 4. Example of a visual representation of a docking grid box (in blue), as defined in the grid.conf file generated by jamreceptor, which will be used as the docking region

b. Observe live updates printed to the terminal, including:
   i. Number of compounds docked.
   ii. Total number of compounds in the library.
   iii. Estimated time remaining.
   iv. Docking rate (compounds per minute, hour and day).

Note: Live updates are helpful to track performance and estimated completion. The performance of jamqvina depends mainly on CPU cores, and to a lesser extent on random access memory (RAM) and storage speed. For small to medium libraries, a modern multi-core desktop or laptop with at least 8 CPU cores and 16 GB RAM is generally sufficient. Runtimes vary: small libraries (a few thousand compounds) can be processed in 6–12 h, while larger libraries (tens of thousands) may take several days. Solid-state drives (SSDs) help avoid file-handling bottlenecks, and graphics processing unit (GPU)-accelerated versions of Vina can provide several-fold speedups. Hardware choice affects runtime but does not limit the ability to run the protocol.

18. Customize the script for GPU acceleration or other docking engines.

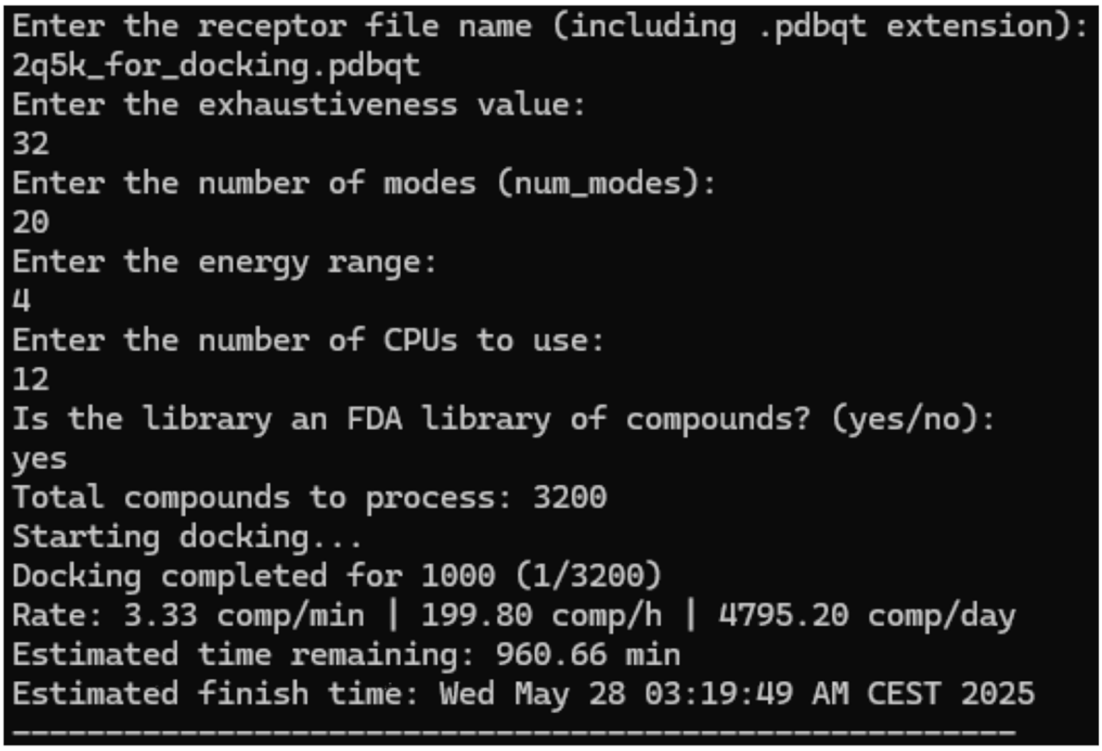


Figure 5. Screenshot of an example of jamqvina execution interface, showing docking configuration and real-time progress updates

Note: This step is not required if you are using CPU-only docking. When available, GPU acceleration is recommended, especially for large libraries (>50,000 compounds), since docking speed can increase substantially. For example, comparable docking runs have been observed to complete up to 3.7 times faster with Vina-GPU 2.1 (tested on an NVIDIA GeForce RTX 4090 GPU, driver 550.144.03, CUDA 12.4, Persistence-M mode enabled) using --search_depth 64 and --threads 8192, compared to QuickVina2 on CPUs (tested on a 12th Gen Intel Core i5-12400) using --exhaustiveness 32 on 10 cores, with similar docking performance.

⚠ CRITICAL: If using a different docking binary (e.g., CUDA-enabled GPU versions), modify the script accordingly:

a. Open the jamqvina script (Data S4) in a text editor.
b. Replace the qvina02 command with the appropriate binary name.
c. Remove --exhaustiveness and --cpu parameters.
d. Add GPU-appropriate flags, such as --threads and --search_depth.

Note: Always test modified versions on a small compound set before large-scale docking.

19. Resume an interrupted docking job:
Use the jamresume script (see Data S5) to continue from where a previous docking session left off.
a. Run the script in the same directory:

```
$ jamresume
```

b. When prompted, re-enter the docking parameters.
c. The script will automatically:
   i. Identify which compounds have already been docked.
   ii. Skip those compounds.
   iii. Resume docking for the remaining entries.

Note: This feature is especially useful after planned interruptions or system crashes.

Ranking docking results

Timing: Variable

Use the jamrank script (see Data S6) to rank docked compounds based on their binding affinities. The program also generates other relevant parameters such as SimScore, number of docking modes, and a link to the ZINC database, among others. This step helps identify the most promising candidates for further analysis.

20. Launch the ranking script executing the following command in a terminal window in the directory where your docking results are located:

```
$ jamrank
```

⚠ CRITICAL: The program must be executed in the directory containing both the docking_results/ and fda_sdf_compounds/ directories.

21. When prompted, enter the number of top compounds to display and include in the summary output.
22. Select one of the two available ranking options (see Figure 6).

Optional: 1. Simplified Output

23. Select the "*Top compounds based on affinity of first mode*" option from the menu.

Note: This option is recommended for an initial, and faster exploration of top hits.

24. Wait until the program displays a summary table including:
    a. Binding affinity (from the first docking mode).
    b. A direct link to the ZINC database entry for the compound.

Note: This is particularly useful for quickly accessing vendor information for the compound with a single click.

    c. Local file name.

Note: The docked file is in PDBQT format (e.g. 1432_docking.pdbqt) and is located in the docking_results/ directory. It can be opened in PyMOL together with receptor_for_docking.pdbqt for visual inspection of interactions.

```
Enter the number of top compounds to display: 20
Select an option:
1) Top compounds based on affinity of first mode
2) Top compounds based on affinity + SimScore + TotalModes + MW (only if using a FDA library)
3) Exit
Enter your choice: 2
Executing option 2: please wait, this might take a few minutes or even hours, depending on the number of results...
```

Figure 6. Screenshot of an example of jamrank execution

25. After completion, check the two output files which are generated automatically:
    a. Results_affinity_<date>.txt, which contains all results (unsorted).
    b. Top_<N>_hits_affinity_<date>.txt, which contains only the top *N* ranked compounds.

    Note: These files are saved in the same directory where the script is executed, and the date field will be automatically formatted with the day and hour of table generation.

    Optional: 2. Extended Output

26. Select the "*Top compounds based on affinity of first mode + SimScore + TotalModes + MW (only if using a FDA library)*" option from the menu.

    Note: This extended method takes longer to run but provides a more comprehensive evaluation of each compound (see Figure 7).

27. Wait until the program displays a quick summary table with:
    a. Binding affinity (from the first docking mode).
    b. Similarity score (SimScore).

    Note: SimScore evaluates both convergence and diversity of docking poses. It is calculated as:

$$\text{SimScore} = \frac{1}{2}\left(P_{\text{RMSD l.b. } <1.6} + P_{\text{RMSD u.b. } <3.2}\right)$$

where $P_{\text{RMSD l.b. } <1.6}$ is the percentage of modes with an root mean square deviation (RMSD) lower bound < 1.6 Å relative to the most favorable (lowest-energy) mode and $P_{\text{RMSD u.b. } <3.2}$ the percentage of modes with an RMSD upper bound < 3.2 Å relative to the most favorable (lowest-energy) mode.

    Note: A higher SimScore reflects better convergence among predicted poses, increasing confidence in the result. However, values of 0 may indicate that only a few docking modes were

Affinity	SimScore	TotalModes	MW	ZINC_Link	Filename
-13.2	25	8	396	https://zinc.docking.org/substances/ZINC000004629876/	1460_docking.pdbqt
-13.2	25	8	492	https://zinc.docking.org/substances/ZINC000003927822/	1308_docking.pdbqt
-12.8	27	11	452	https://zinc.docking.org/substances/ZINC000640033452/	2203_docking.pdbqt
-12.8	40	5	412	https://zinc.docking.org/substances/ZINC000095915244/	2279_docking.pdbqt
-12.6	31	16	442	https://zinc.docking.org/substances/ZINC000011679756/	1695_docking.pdbqt
-12.6	56	9	416	https://zinc.docking.org/substances/ZINC000100015048/	2320_docking.pdbqt
-12.4	10	15	376	https://zinc.docking.org/substances/ZINC000003830767/	709_docking.pdbqt
-12.4	12	4	366	https://zinc.docking.org/substances/ZINC000003927200/	1307_docking.pdbqt
-12.4	20	15	376	https://zinc.docking.org/substances/ZINC000003830769/	711_docking.pdbqt
-12.4	28	20	410	https://zinc.docking.org/substances/ZINC000000538312/	2736_docking.pdbqt

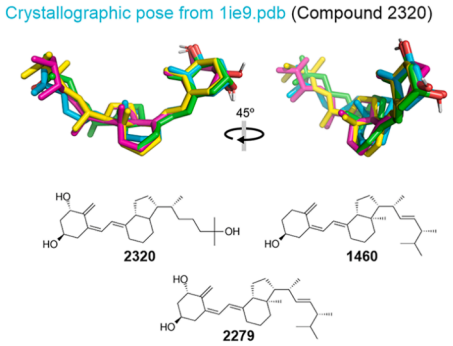
Figure 7. Screenshot of a ranking table showing the top 10 compounds sorted by binding affinity after execution of optional 2 of the jamrank program

A

Case study #1: Nuclear receptor for vitamin D ligand binding domain

top_10_hits_affinity_and_simscore_<date>.txt

Affinity	SimScore	TotalModes	ZINC ID	Filename
-13.2	25	8	ZINC000004629876	1460_docking.pdbqt
-13.2	25	8	ZINC000003927822	1308_docking.pdbqt
-12.8	27	11	ZINC000064033452	2203_docking.pdbqt
-12.8	40	5	ZINC000095915244	2279_docking.pdbqt
-12.6	31	16	ZINC000011679756	1695_docking.pdbqt
-12.6	56	9	ZINC000100015048	2320_docking.pdbqt
-12.4	10	15	ZINC000003830767	709_docking.pdbqt
-12.4	12	4	ZINC000003927200	1307_docking.pdbqt
-12.4	20	15	ZINC000003830769	711_docking.pdbqt
-12.4	28	20	ZINC000000538312	2736_docking.pdbqt

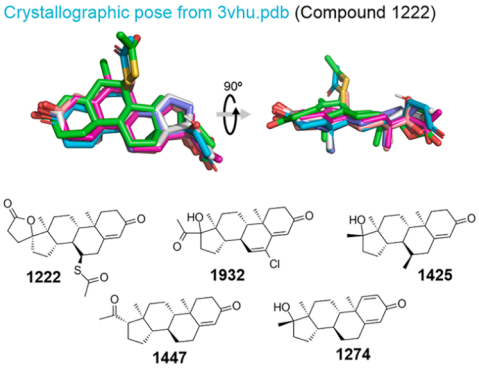


B

Case study #2: Mineralocorticoid receptor ligand-binding domain

top_16_hits_affinity_and_simscore_<date>.txt

Affinity	SimScore	TotalModes	ZINC ID	Filename
-12.1	0	3	ZINC000030691600	1932_docking.pdbqt
-11.7	0	1	ZINC000003861599	1222_docking.pdbqt
-11.7	25	2	ZINC000004215039	1425_docking.pdbqt
-11.5	0	1	ZINC000003927200	1307_docking.pdbqt
-11.5	25	2	ZINC000034962060	1969_docking.pdbqt
-11.4	50	4	ZINC000004428529	1447_docking.pdbqt
-11.4	50	7	ZINC000003807917	354_docking.pdbqt
-11.2	25	2	ZINC000022116662	1898_docking.pdbqt
-11.2	25	2	ZINC000004083557	1381_docking.pdbqt
-11.2	33	3	ZINC000034962062	1970_docking.pdbqt
-11.2	57	7	ZINC000004343508	1442_docking.pdbqt
-11.1	0	3	ZINC000118912522	2438_docking.pdbqt
-11.1	50	5	ZINC000003813047	391_docking.pdbqt
-11.0	12	4	ZINC000003938750	1325_docking.pdbqt
-11.0	50	4	ZINC000005167143	1490_docking.pdbqt
-11.0	70	5	ZINC000003875469	1274_docking.pdbqt

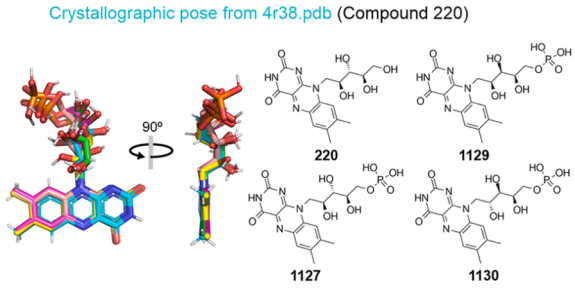


C

Case study #3: Light–oxygen–voltage sensor domain

top_10_hits_affinity_and_simscore_<date>.txt

Affinity	SimScore	TotalModes	ZINC ID	Filename
-10.5	11	9	ZINC000003831427	1129_docking.pdbqt
-10.4	0	9	ZINC000003831422	1124_docking.pdbqt
-10.3	0	6	ZINC000002036848	220_docking.pdbqt
-10.3	0	7	ZINC000003831424	1126_docking.pdbqt
-10.3	5	20	ZINC000006716957	1525_docking.pdbqt
-10.2	14	7	ZINC000003831425	1127_docking.pdbqt
-10.1	10	10	ZINC000003831428	1130_docking.pdbqt
-10.1	4	13	ZINC000003831423	1125_docking.pdbqt
-10.1	8	20	ZINC000012503187	1714_docking.pdbqt
-10.0	5	20	ZINC000003831453	1141_docking.pdbqt



D

Case study #4: Progesterone receptor ligand-binding domain

top_10_hits_affinity_and_simscore_<date>.txt

Affinity	SimScore	TotalModes	ZINC ID	Filename
-12.0	33	3	ZINC000003830791	735_docking.pdbqt
-11.2	33	6	ZINC000003831512	1166_docking.pdbqt
-11.1	50	6	ZINC000003830763	705_docking.pdbqt
-11.0	25	2	ZINC000013540519	1730_docking.pdbqt
-11.0	42	6	ZINC000003831515	1170_docking.pdbqt
-11.0	44	8	ZINC000003831249	972_docking.pdbqt
-10.9	33	3	ZINC000085205451	2229_docking.pdbqt
-10.8	38	4	ZINC000004428529	1447_docking.pdbqt
-10.8	56	9	ZINC000004343508	1442_docking.pdbqt
-10.8	58	6	ZINC000034962062	1970_docking.pdbqt

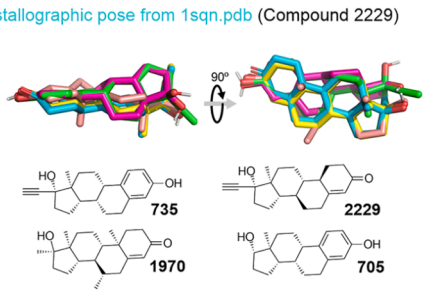


Figure 8. Application of the virtual screening protocol on four protein targets

(A–D) (A) Vitamin D nuclear receptor ligand-binding domain (PDB: 1IE9, chain A, pocket #5), (B) Mineralocorticoid receptor ligand-binding domain (PDB: 3VHU, chain A, pocket #1), (C) Light–oxygen–voltage (LOV) sensor domain (PDB: 4R38, chain A, pockets #2 and #3), and (D) Progesterone receptor ligand-binding domain (PDB: 1SQN, chain A, pocket #2). Each panel includes a table listing the top-ranked compounds with predicted binding affinity

Figure 8. *Continued*
(kcal/mol), SimScore (pose similarity to the crystallographic ligand), total number of docking modes, ZINC IDs, and filename. To the right of the tables, two structural views compare the crystallographic ligand and top compounds in stick representation (carbon atoms color-coded per compound). Colored arrows, matching the carbon color scheme of the structural representations, indicate the position of the compounds in the affinity ranking table. Note that SimScore = 0 should be interpreted with caution, particularly in cases where only a few docking modes were generated (e.g., in panel B, the top two compounds have SimScore = 0 due to the generation of only 3 and 1 modes out of the 20 allowed).

generated. SimScore should always be interpreted in combination with TotalModes (e.g., see Figure 8B).

c. Total docking modes.
d. Molecular weight (only when using FDA-approved libraries).
e. A direct link to the ZINC database entry for the compound.

Note: This is particularly useful for quickly accessing vendor information for the compound with a single click.

f. Local file name.

Note: The docked file is in PDBQT format (e.g. 1460_docking.pdbqt) and is located in the docking_results/ directory. It can be opened in PyMOL together with receptor_for_docking.pdbqt for visual inspection of interactions.

28. After completion, check the two output files which are generated automatically:
    a. Results_affinity_and_simscore_<date>.txt, which contains all results (unsorted).
    b. Top_<N>_hits_affinity_and_simscore_<date>.txt, which contains only the top *N* ranked compounds.

Note: These files are saved in the same directory where the script is executed, and the date will be automatically formatted with the day and time when table was generated.

EXPECTED OUTCOMES

By following the step-by-step installation procedures outlined in this protocol, users can transform a standard computer into a fully functional virtual screening workstation in under 60-minutes, using only free and open-source software. This workflow allows any user to find potential small molecule binders to any desired target protein in PDB format by molecular docking.

To validate the pipeline proposed in this study, we selected four protein targets with available crystallographic structures co-crystallized with FDA-approved ligands (PDB: 1IE9,[17] 3VHU,[18] 4R38,[19] 1SQN[20]). The full pipeline was applied to each target, including system setup, generation of an FDA-approved compound library containing 3,200 molecules, receptor preparation, and virtual screening using the following docking parameters: an exhaustiveness of 32, a maximum number of 20 binding modes, and an energy range of 0-4 kcal/mol above the best binding affinity. The entire process was completed within in 8 to 12 hours per target, on a system equipped with a 12th Gen Intel Core i5-12400 processor (2.5 GHz, 32 GB RAM) utilizing 10 CPU cores.

The top-ranked compounds identified by the jamrank program included the known co-crystallized ligands (Figure 8), as well as closely related analogs with binding poses highly similar to those experimentally determined by X-ray crystallography. These results highlight the utility of the protocol for rapidly and easily identifying potential drug repurposing candidates using FDA-approved compounds. Moreover, the same workflow can be extended to custom libraries of compounds, offering great value for hit identification in early-stage drug discovery.

LIMITATIONS

One key limitation of this protocol is its reliance on external servers provided by the free ZINC database (https://zinc.docking.org/) and associated file servers (https://files.docking.org/ and https://files2.docking.org/) for compound library generation via the jamlib script. Any downtime or inaccessibility of these servers can directly impact the successful execution of this step. We have observed instances where temporary outages have affected performance. For example, when working with the FDA catalog available in ZINC, we have encountered difficulties downloading the complete set of compounds. The "download all files" service often fails to execute properly, and some associated pages are intermittently unavailable, being accessible at certain times and not at others. As a result, the size and completeness of the generated compound library may vary depending on the availability of specific pages within the FDA catalog on the ZINC platform. Recommendations for addressing these issues are provided in the troubleshooting section below.

In addition, the protocol currently employs only rigid receptor docking, which does not account for protein flexibility, an important factor in ligand recognition and binding affinity. The absence of ensemble docking or flexible receptor modeling may reduce predictive accuracy, particularly for targets with highly dynamic binding sites.

Furthermore, compound ranking is currently based solely on predicted binding affinity, with supplementary information provided on pose similarity, the number of generated docking modes, and molecular weight. However, the workflow does not yet incorporate additional pharmacological or drug-likeness filters, such as absorption, distribution, metabolism, excretion and toxicity (ADMET) properties, which could improve compound prioritization for downstream validation.

TROUBLESHOOTING

Problem 1

The jamlib script downloads compound structures from the ZINC database (https://zinc.docking.org), as well as from associated file repositories (https://files.docking.org and https://files2.docking.org). Therefore, any server maintenance, downtime, or structural changes in these resources may interfere with the proper functioning of the program.

Potential solution

To mitigate this dependency on external servers, we have implemented several fault-tolerance strategies within our scripts. For example, during FDA compound library generation, the script first verifies the reachability of the FDA catalog in the ZINC server. Additionally, a timer mechanism is included to handle delays or unresponsiveness. If the required web pages are unavailable, the program attempts to re-download failed compounds in a second cycle of tries (a process that may take up to approximately two hours, but in our experience substantially increases the number of successfully retrieved compounds). Alternatively, users are offered the option to stop the program and proceed with the protocol using the FDA compounds already downloaded at that point.

In addition, in the custom compound library module, where compounds are typically retrieved from https://files.docking.org, we have implemented an automatic fallback to a mirror server (https://files2.docking.org) in case the primary server is inaccessible.

Users encountering persistent issues are advised to.

- Retry after some time in case of temporary server maintenance;
- Verify whether the uniform resource locators (URLs) or server structure has changed by checking the official ZINC website;
- Update any hardcoded URLs in the script accordingly.

Maintaining flexibility in the code to accommodate any potential updates in server's architecture or URLs is crucial for the long-term functionality of the program.

Problem 2

If you are installing programs in a fresh WSL installation and use gedit as your text editor, you might experience a very slow opening time.

Potential solution

Use dbus-launch to run gedit. You can set this easily in the .bashrc file by adding the following alias:

```
$ echo "alias gedit=\"dbus-launch gedit --standalone\"" >> ~/.bashrc
```

Then activate the changes with:

```
$ source ~/.bashrc
```

Problem 3

PyMOL does not launch properly under a WSL distribution on Ubuntu.

Potential solution

Install the zombie-imp Python module using:

```
$ sudo apt install python3-zombie-imp
```

Problem 4

A common issue when writing or copying scripts is the unintentional use of an en dash (–) or em dash (—) instead of a hyphen (-). This often happens when text is copied from word processors (e.g., Microsoft Word or Google Docs) or PDF files that automatically format hyphens into typographic dashes. These visually similar characters are treated differently by the shell, leading to syntax errors that can be difficult to trace.

Potential solution

Always ensure that your script uses standard hyphens (-) for command-line options and flags. To detect and fix hidden dash issues.

- Use a plain text editor (e.g., nedit, gedit or emacs) that preserves ASCII characters and doesn't auto-format hyphens.
- Search for and replace en dashes (–) and em dashes (—) with regular hyphens (-). Most editors support find-and-replace with special characters.
- Use a command-line check to spot non-ASCII characters in your script. Run the following in your terminal:

```
$ grep -P -n "[^\x00-\x7F]" script
```

This will print the line numbers where non-ASCII characters (including en/em dashes) are detected.

Problem 5

We encountered recognition issues with MGLTools installations outside the home directory, especially with the location of the prepare_receptor4.py tool.

Potential solution

In such cases, the automatic search inside jamreceptor might fail. By default, it includes the line.

```
PREP_SCRIPT=$(find ~ -path "*AutoDockTools*/Utilities24/prepare_receptor4.py" 2>/dev/null
| head -n 1)
```

If the tool is installed elsewhere, replace it with the specific path where prepare_receptor4.py is located.

```
PREP_SCRIPT=/path/to/your/prepare_receptor4.py
```

Save the modified script and try running it again.

RESOURCE AVAILABILITY

Lead contact

Requests for further information and resources should be directed to and will be fulfilled by the lead contact, José Antonio Manso (joseantonio.manso@unican.es).

Technical contact

Questions about the technical specifics of performing the protocol should be directed to the technical contact, José Antonio Manso (joseantonio.manso@unican.es).

Materials availability

This study describes a computational protocol and did not generate new physical materials.

Data and code availability

The data used and generated during this study are available from the lead contact upon request. The source code is publicly available at GitHub (https://github.com/jamanso/jamdock-suite) and Zenodo (https://doi.org/10.5281/zenodo.15577778).

ACKNOWLEDGMENTS

This work was funded by Portuguese funds through Fundação para a Ciência e a Tecnologia (FCT) in the framework of project 2023.13395.PEX (digital object identifier https://doi.org/10.54499/2023.13395.PEX.

We thank Dr. José María de Pereda, Dr. Arturo Carabias, and Dr. Andreia M. Silva for their valuable discussions and critical feedback on the development of this protocol. We also extend our acknowledgments to Dr. Luis Miguel Lozano Gordillo.

AUTHOR CONTRIBUTIONS

Conceptualization, software, formal analysis, funding acquisition, and writing – original draft, J.A.M.; Writing – review and editing, P.J.B.P., J.R.-R., and S.M.-R.

DECLARATION OF INTERESTS

The authors declare no competing interests.

SUPPLEMENTAL INFORMATION

Supplemental information can be found online at https://doi.org/10.1016/j.xpro.2025.104161.

REFERENCES

1. Sadybekov, A.V., and Katritch, V. (2023). Computational approaches streamlining drug discovery. Nature *616*, 673–685. https://doi.org/10.1038/s41586-023-05905-z.

2. Irwin, J.J., Tang, K.G., Young, J., Dandarchuluun, C., Wong, B.R., Khurelbaatar, M., Moroz, Y.S., Mayfield, J., and Sayle, R.A. (2020). ZINC20—A Free Ultralarge-Scale Chemical Database for Ligand Discovery. J. Chem. Inf. Model. *60*, 6065–6073. https://doi.org/10.1021/acs.jcim.0c00675.

3. Sterling, T., and Irwin, J.J. (2015). ZINC 15 – Ligand Discovery for Everyone. J. Chem. Inf. Model. *55*, 2324–2337. https://doi.org/10.1021/acs.jcim.5b00559.

4. Irwin, J.J., and Shoichet, B.K. (2005). ZINC – A Free Database of Commercially Available Compounds for Virtual Screening. J. Chem. Inf. Model. *45*, 177–182. https://doi.org/10.1021/ci049714+.

5. Trott, O., and Olson, A.J. (2010). AutoDock Vina: Improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading. J. Comput. Chem. *31*, 455–461. https://doi.org/10.1002/jcc.21334.

6. Eberhardt, J., Santos-Martins, D., Tillack, A.F., and Forli, S. (2021). AutoDock Vina 1.2.0: New Docking Methods, Expanded Force Field, and Python Bindings. J. Chem. Inf. Model. *61*, 3891–3898. https://doi.org/10.1021/acs.jcim.1c00203.

7. Kabier, M., Gambacorta, N., Trisciuzzi, D., Kumar, S., Nicolotti, O., and Mathew, B. (2024). MzDOCK: A free ready-to-use GUI-based pipeline for molecular docking simulations. J. Comput. Chem. *45*, 1980–1986. https://doi.org/10.1002/jcc.27390.

8. Forli, S., Huey, R., Pique, M.E., Sanner, M.F., Goodsell, D.S., and Olson, A.J. (2016). Computational protein–ligand docking and virtual drug screening with the AutoDock suite. Nat. Protoc. *11*, 905–919. https://doi.org/10.1038/nprot.2016.051.

9. Irwin, J.J., Shoichet, B.K., Mysinger, M.M., Huang, N., Colizzi, F., Wassam, P., and Cao, Y. (2009). Automated Docking Screens: A Feasibility Study. J. Med. Chem. *52*, 5712–5720. https://doi.org/10.1021/jm9006966.

10. Bender, B.J., Gahbauer, S., Luttens, A., Lyu, J., Webb, C.M., Stein, R.M., Fink, E.A., Balius, T.E., Carlsson, J., Irwin, J.J., and Shoichet, B.K. (2021). A practical guide to large-scale docking. Nat. Protoc. *16*, 4799–4832. https://doi.org/10.1038/s41596-021-00597-z.

11. Knight, I., Tang, K., and Irwin, J. (2023). DOCK Blaster 2.0 - An Investigation of Automated Docking. Preprint at ChemRxiv. https://doi.org/10.26434/chemrxiv-2023-6h2c9.

12. Knight, I., Tang, K., Mailhot, O., and Irwin, J. (2023). DockOpt: A Tool for Automatic Optimization of Docking Models. Preprint at ChemRxiv. https://doi.org/10.26434/chemrxiv-2023-6h2c9-v3.

13. Morris, G.M., Huey, R., Lindstrom, W., Sanner, M.F., Belew, R.K., Goodsell, D.S., and Olson, A. J. (2009). AutoDock4 and AutoDockTools4: Automated Docking with Selective Receptor Flexibility. J. Comput. Chem. *30*, 2785–2791. https://doi.org/10.1002/jcc.21256.

14. Le Guilloux, V., Schmidtke, P., and Tuffery, P. (2009). Fpocket: An open source platform for ligand pocket detection. BMC Bioinf. *10*, 168. https://doi.org/10.1186/1471-2105-10-168.

15. Schmidtke, P., and Barril, X. (2010). Understanding and predicting druggability. A high-throughput method for detection of drug binding sites. J. Med. Chem. *53*, 5858–5867. https://doi.org/10.1021/jm100574m.

16. Alhossary, A., Handoko, S.D., Mu, Y., and Kwoh, C.-K. (2015). Fast, accurate, and reliable molecular docking with QuickVina 2. Bioinformatics *31*, 2214–2216. https://doi.org/10.1093/bioinformatics/btv082.

17. Tocchini-Valentini, G., Rochel, N., Wurtz, J.M., Mitschler, A., and Moras, D. (2001). Crystal structures of the vitamin D receptor complexed to superagonist 20-epi ligands. Proc. Natl. Acad. Sci. USA *98*, 5491–5496. https://doi.org/10.1073/pnas.091018698.

18. Hasui, T., Matsunaga, N., Ora, T., Ohyabu, N., Nishigaki, N., Imura, Y., Igata, Y., Matsui, H., Motoyaji, T., Tanaka, T., et al. (2011). Identification of benzoxazin-3-one derivatives as novel, potent, and selective nonsteroidal mineralocorticoid receptor antagonists. J. Med. Chem. *54*, 8616–8631. https://doi.org/10.1021/jm2011645.

19. Rivera-Cancel, G., Ko, W.h., Tomchick, D.R., Correa, F., and Gardner, K.H. (2014). Full-length structure of a monomeric histidine kinase reveals basis for sensory regulation. Proc. Natl. Acad. Sci. USA *111*, 17839–17844. https://doi.org/10.1073/pnas.1413983111.

20. Madauss, K.P., Deng, S.-J., Austin, R.J.H., Lambert, M.H., McLay, I., Pritchard, J., Short, S. A., Stewart, E.L., Uings, I.J., and Williams, S.P. (2004). Progesterone receptor ligand binding pocket flexibility: Crystal structures of the norethindrone and mometasone furoate complexes. J. Med. Chem. *47*, 3381–3387. https://doi.org/10.1021/jm030640n.

21. O'Boyle, N.M., Banck, M., James, C.A., Morley, C., Vandermeersch, T., and Hutchison, G.R. (2011). Open Babel: An open chemical toolbox. J. Cheminform. *3*, 33. https://doi.org/10.1186/1758-2946-3-33.