UniDesign

Version 1.1

A Program for Computational Protein Design

Xiaoqiang Huang, Ph.D.

June 29, 2023

**Contents**

[Overview and features 3](#_Toc138951352)

[License 3](#_Toc138951353)

[Software availability 4](#_Toc138951354)

[Why UniDesign? 4](#_Toc138951355)

[Platform and installation 5](#_Toc138951356)

[Windows 5](#_Toc138951357)

[Method 1 5](#_Toc138951358)

[Method 2 5](#_Toc138951359)

[Other options 6](#_Toc138951360)

[Linux 6](#_Toc138951361)

[Mac 6](#_Toc138951362)

[Usage 7](#_Toc138951363)

[Basic usage 7](#_Toc138951364)

[Design monomer protein 7](#_Toc138951365)

[Design protein-protein interaction 8](#_Toc138951366)

[Design protein-nucleic acid interaction 8](#_Toc138951367)

[Design protein-ligand interaction 8](#_Toc138951368)

[Design enzyme 10](#_Toc138951369)

[Design protein side-chain packing 13](#_Toc138951370)

[Repair incomplete protein side chain 13](#_Toc138951371)

[Protein structure energy minimization 13](#_Toc138951372)

[Build mutant structure model 13](#_Toc138951373)

[Add polar hydrogen atom 14](#_Toc138951374)

[Optimize hydrogen's position to maximize hydrogen bonding network 14](#_Toc138951375)

[Compute protein stability 14](#_Toc138951376)

[Compute protein interchain binding interaction 15](#_Toc138951377)

[RESFILE 15](#_Toc138951378)

[How to cite UniDesign? 17](#_Toc138951379)

[References 17](#_Toc138951380)

# Overview and features

UniDesign is a computational framework for protein design, targeting a diversity of protein design and engineering tasks, and it can also be used for protein structure modeling and scoring. UniDesign is extended mainly from the [EvoEF2](https://zhanggroup.org/EvoEF/) protein design program with the evolutionary feature taken from [EvoDesign](https://zhanggroup.org/EvoDesign/), but it has many new features. In general, UniDesign applies to but not limited to the following tasks:

* Protein design:
* design monomer proteins
* design protein-protein interactions
* design protein-ligand interactions
* design protein-nucleic acid interactions
* design enzymes
* Protein structure modeling:
* protein side-chain packing
* repair incomplete protein sidechains
* protein minimization to remove sidechain clashes
* build mutant structural models
* add polar hydrogen atoms
* optimize hydrogen atom's position
* Protein scoring:
* compute protein stability
* compute protein interchain binding interaction

# License

The MIT License (MIT)

Copyright (c) Xiaoqiang Huang

xiaoqiah@outlook.com or xiaoqiah@umich.edu

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

# Software availability

UniDesign is fully open-sourced at https://github.com/tommyhuangthu/UniDesign.

# Why UniDesign?

UniDesign is developed based on two previous protein design programs, EvoDesign ([*1*](#_ENREF_1)*,* [*2*](#_ENREF_2)) and EvoEF2 ([*3*](#_ENREF_3)).

EvoDesign is an evolution-based approach to de novo protein sequence design for monomers and protein-protein interactions. The first version of EvoDesign was released in 2013 ([*1*](#_ENREF_1)). At that time, for protein design simulations, EvoDesign utilized an external protein side-chain packing program named SCWRL4 ([*4*](#_ENREF_4)) to construct the structural model for a sequence decoy on a fixed protein backbone and then used a composite scoring function that combined an evolution-based potential (i.e. position-specific scoring matrix, PSSM) and an external empirical scoring function named FoldX ([*5*](#_ENREF_5)) to score the models generated by SCWRL4. A replica exchange Monte Carlo (REMC) simulation was used to accept or reject the sequence decoys through the Metropolis criterion. A large number of low-energy (i.e. the FoldX score) sequence decoys generated in the REMC simulation were then subjected to clustering based on sequence similarity calculated using the BLOSUM62 matrix. The sequences at the largest cluster centers were selected, analyzed, and further screened for wet-lab experimental tests. The first version of EvoDesign was only able to design monomers. In 2019, a newer version of EvoDesign was developed to design protein complexes to enhance protein-protein binding interaction, and the previous packer SCWRL4 was replaced by another much faster one, RASP ([*6*](#_ENREF_6)).

EvoDesign calls a few third-party programs (e.g., SCWRL4, RASP, and FoldX) for the two major steps in computational protein design, i.e., model construction and scoring. Since the source code of these external programs is not available, it is difficult to extend them for further research. For example, RASP cannot handle protein-ligand interaction design because a custom ligand cannot be appropriately modeled. For protein design scoring, FoldX has been only optimized for predicting the protein stability change (ΔΔGstability) upon mutations ([*7*](#_ENREF_7)), but its capability for de novo protein sequence design is poor ([*8*](#_ENREF_8)). Besides, FoldX has not been fully optimized for modeling and designing protein-protein interactions ([*2*](#_ENREF_2)). Moreover, the computational speed of FoldX is very slow ([*2*](#_ENREF_2)). To address some of these issues, I developed a physics- and knowledge-based scoring function called EvoEF (EvoDesign Energy Function) to replace FoldX in EvoDesign ([*2*](#_ENREF_2)*,* [*3*](#_ENREF_3)). The first version of EvoEF (EvoEF1) was trained/optimized in the same strategy as FoldX, i.e., to maximize the capability for predicting the thermodynamic change data such as ΔΔGstability and the protein-protein binding affinity change (ΔΔGbind) upon mutations. Based on my benchmark, EvoEF1 runs ~3–5 times faster than FoldX on protein stability and protein-protein binding affinity calculation and slightly outperforms FoldX on the ΔΔG prediction with a higher Pearson correlation coefficient and in the meanwhile a lower root-mean-square error. However, again, I found that EvoEF1 showed relatively poor performance for de novo protein sequence design through native sequence recovery experiments—the designer sequences were not very native-like ([*3*](#_ENREF_3)). Therefore, I developed the second version of EvoEF (EvoEF2) and specifically optimized it for de novo protein sequence design by incorporating more energy terms. Based on my benchmark, EvoEF2 could achieve a similar native sequence recovery rate ([*3*](#_ENREF_3)) compared to Rosetta, a widely used physics-based protein design approach.

EvoEF, EvoEF2, and EvoDesign apply to only proteins consisting of 20 canonical amino acids, significantly limiting their use in designing proteins that interact with non-protein molecules such as small molecules. There are increasing demands from researchers including myself to extend these tools for handling, ideally, all kinds of molecules, such as water molecules, ions, small-molecule ligands, and nucleic acids. Toward this goal, I extended EvoEF2 and EvoDesign into a much more powerful protein design program (UniDesign) as a universal framework for protein design and engineering, including monomer design, protein-protein interaction design, protein-ligand interaction design, protein-nucleic acid interaction design, and enzyme design.

# Platform and installation

## Windows

### Method 1

An executable named UniDesign.exe is provided in the UniDesign package, which is pre-built on a Windows 10 x64 machine using the g++ compiler version 8.2.0. If you use a Windows 7 or higher machine, you should be able to directly run the precompiled UniDesign.exe in the Microsoft Windows command line (cmd.exe).

Open the cmd.exe, locate into the UniDesign directory, and run:

./UniDesign.exe -h

If successful, the UniDesign help message will show on the screen. Then you can run the other commands of UniDesign using commands like:

./UniDesign.exe --command=CommandName [… other options …].

### Method 2

To my knowledge, it is not very convenient to use the Windows command line. In this situation, it is better to install a third-party tool that can run UniDesign in command lines. [Git](https://git-scm.com/) is a good choice and can be used for managing the code. With the g++ compiler installed, one can rebuild the UniDesign.exe, which may lead to higher computational efficiency and fewer bugs.

### Other options

To achieve the highest running speed of UniDesign in Windows, you may want to build the executable UniDesign.exe program by yourself. it is necessary to install a g++ compiler first. You can make the installation through [MinGW](http://www.mingw.org/) or [Cygwin](https://www.cygwin.com/). After you complete the installation of the g++ compiler and set the environmental variable. You can test if the g++ compiler has been successfully installed with g++ -v in the cmd.exe control console. If successful, it will show messages similar to:

Using built-in specs.

COLLECT\_GCC=g++

COLLECT\_LTO\_WRAPPER=c:/mingw/bin/../libexec/gcc/mingw32/8.2.0/lto-wrapper.exe

Target: mingw32

Configured with: ../src/gcc-8.2.0/configure --build=x86\_64-pc-linux-gnu \

--host=mingw32 --target=mingw32 --prefix=/mingw --disable-win32-registry \

--with-arch=i586 --with-tune=generic --enable-languages=c,c++,objc,obj-c++,fortran,ada \

--with-pkgversion='MinGW.org GCC-8

Thread model: win32

gcc version 8.2.0 (MinGW.org GCC-8.2.0-3)

Otherwise, it will say that g++ cannot be found. You should check your installation. If you try many times but still cannot install the program successfully, please contact report problems to the emails listed below.

## Linux

It is much more convenient to use UniDesign in a Linux system. Download the UniDesign package, go to the main directory of UniDesign, and simply run the command g++ -O3 --fast-math -o UniDesign src/\*.cpp or directly run the build.sh bash file to build the executable UniDesign program into the main directory. If the command does not work, try without the --fast-math option.

## Mac

Building UniDesign on Mac is similar to that in Linux, try the command g++ -O3 -fast-math -o UniDesign src/\*.cpp or simply g++ -O3 -o UniDesign src/\*.cpp in the Mac terminal to build the executable UniDesign program.

# Usage

### Basic usage

In general, UniDesign is run in the following syntax:

<path>/UniDesign --command=CommandName [--options]

One can run UniDesign -h, UniDesign -?, or UniDesign --help to get more information about the supported commands and options.

### Design monomer protein

A typical command for running monomer protein design is as follows:

<path>/UniDesign --command=ProteinDesign --monomer --pdb=XXXX.pdb --design\_chains=A --resfile=RESFILE.txt --ntraj=10

If XXXX.pdb is a monomer protein, the options --monomer and --design\_chains=A can be discarded. The option --ntraj specifies the number of independent simulated annealing Monte Carlo trajectories that is conducted (the number is 1 by default). The option --resfile=RESFILE.txt designates the amino acid positions for mutational design, repacking, or fixed. More information will be introduced in the RESFILE section regarding the format of RESFILE.txt.

XXXX.pdb is the PDB file. UniDesign will parse the XXXX as protein ID and uses it as a prefix of the output files for recording/saving results. If the above command is run successfully, 10 designs will be generated with file names:

|  |  |
| --- | --- |
| File name | Remark |
| **XXXX\_bestseqs.txt** | Collection of the designed lowest-energy sequence for each simulation trajectory |
| **XXXX\_bestsites0001.pdb**  **… …**  **XXXX\_bestsites0010.pdb**  **XXXX\_beststruct0001.pdb**  **… …**  **XXXX\_beststruct0010.pdb** | The design sites (\*bestsites\*) and the full protein structures (\*beststruct\*) for the corresponding lowest-energy designs; for de novo design, the corresponding bestsites file and beststruct file are identical. |
| XXXX\_desrots0001.txt  … …  XXXX\_desrots0010.txt  Xxxx\_desseqs0001.txt  … …  XXXX\_desseqs0010.txt  XXXX\_rotlist.txt  XXXX\_rotlistSEC.txt  XXXX\_selfenergy.txt | These are the log files for recording the design simulation process.  \*desseqs\* record the accepted sequence decoys during simulated annealing Monte Carlo simulation;  \*desrots\* record the rotamer indices for the corresponding accepted sequences. |

The file names highlighted in bold are important result files. A detailed explanation of the designer sequences is provided in the header of XXXX\_bestseqs.txt.

### Design protein-protein interaction

In UniDesign, a typical command for designing one chain (e.g. A) of a dimer protein (e.g. AB.pdb) is as follows:

<path>/UniDesign --command=ProteinDesign --ppint --pdb=AB.pdb --design\_chains=A --resfile=RESFILE.txt --ntraj=10

UniDesign can also design multiple chains (e.g. A and B) of a multichain protein (e.g. ABC.pdb) at the same time:

<path>/UniDesign --command=ProteinDesign --ppint --pdb=ABC.pdb --design\_chains=AB --resfile=RESFILE.txt --ntraj=10

Compared to monomer design, similar outputs will be generated for protein-protein interaction design.

### Design protein-nucleic acid interaction

Currently, protein-nucleic acid interaction design is handled similarly to that for protein-protein interaction design with the option --ppint:

<path>/UniDesign --command=ProteinDesign --ppint --pdb=Prot\_NA.pdb --design\_chains=A --resfile=RESFILE.txt --ntraj=10

In the command line, chain A is a protein chain. Nucleic acids (backbone and sidechains) are kept constant in the design.

### Design protein-ligand interaction

For protein-ligand interaction design, the atom parameter and residue topology files for the ligand must be first generated before the design. The command line for making these two files is as follows:

<path>/UniDesign --command=MakeLigParamAndTopo --mol2=LIG.mol2 --lig\_param=LIG\_PARAM.prm --lig\_topo=LIG\_TOPO.inp

The LIG.mol2 file must contain all the ligand atoms including hydrogen atoms. One needs to ensure that ligand atomic partial charges have been appropriately assigned (the last column in the @<TRIPOS>ATOM section in the mol2 file) before running the above command. Once the command is run successfully, the atom parameters will be saved to LIG\_PARMA.prm and residue topology to LIG\_TOPO.inp, respectively.

To redesign protein residues to enhance binding affinity to the ligand, one can run the ProteinDesign command with the --protlig option, e.g.:

<path>/UniDesign --command=ProteinDesign --protlig --pdb=prot.pdb --mol2=LIG.mol2 --read\_lig\_poses=LIG\_POSES.pdb --lig\_param=LIG\_PARAM.prm --lig\_topo=LIG\_TOPO.inp --resfile=RESFILE.txt --ntraj=10

Depending on one’s needs for the design, a file LIG\_POSES.pdb should be prepared that contains the necessary ligand poses. Let’s suppose the experimental protein-ligand complex structure is available, then one can directly divide the complex into prot.pdb and LIG\_POSES.pdb, where the latter contains only one pose (i.e. the experimental pose). In case where an ensemble of ligand conformations (rather than a single deterministic conformation) is necessary for design, molecular dynamics simulations or similar protocols can be used to prepare this ensemble. In another scenario, e.g. the experimental protein-ligand complex is not available, one can use molecular docking tools such as AutoDock, Vina, or DOCK to dock the ligand to the protein to prepare the required poses (and save them to LIG\_POSES.pdb). Unlike the LIG.mol2 file, the ligand hydrogen atoms can be discarded in LIG\_POSES.pdb and they will be added automatically by UniDesign when the poses are read in.

In LIG\_POSES.pdb, each ligand pose is shown in the field between PDB keywords MODEL and ENDMDL, e.g.:

MODEL 1

[The atomic coordinates of model1 here]

ENDMDL

MODEL 2

[The atomic coordinates of model2 here]

ENDMDL

….

MODEL 10

[The atomic coordinates of model10 here]

ENDMDL

For protein-ligand interaction design (and enzyme design with small-molecule ligand involved), the optimal ligand pose selected in each design simulation trajectory will be saved to an individual mol2 file, e.g.:

|  |  |
| --- | --- |
| **XXXX\_bestlig0001.mol2**  **… …**  **XXXX\_bestlig0010.mol2** | The corresponding ligand pose mol2 files for 10 designs. |

### Design enzyme

Compared to protein-ligand interaction design, enzyme design (especially the design for activity) is a little bit more complicated since one has to take into account appropriately the chemistry/catalysis. According to the transition state theory, a commonly used strategy for enzyme activity design is to redesign the active-site residues to make the enzyme bind strongest to the high-energy transition state of a reaction (i.e., to maximize the affinity between the enzyme and the transition state), thus lowering the activation energy barrier and accelerating reaction velocity.

The first problem to be addressed in computational enzyme design is to model the transition state of the reaction, which may be obtained from transition-state mimics, quantum mechanics calculation, or chemical knowledge. Once the transition state is obtained, UniDesign uses catalytic geometric constraints to constrain the catalytic relationship between the enzyme catalytic residues and the substrate (ligand). The catalytic constraints, saved in the LIG\_CATACONS.txt file, will be utilized for the generation of catalytic ligand poses and the subsequent enzyme design process.

The whole enzyme design process consists of a few steps:

**Step 1: protein and ligand preparation**. In this step, one should prepare the enzyme scaffold in the PDB file format (e.g. XXXX.pdb) and the ligand in the mol2 file format (e.g. LIG.mol2). It should be noted that the ligand atomic partial charges must be appropriately assigned.

**Step 2: Generation of the ligand atomic parameters and residue topology**. The residue topology of a small-molecule ligand that describes bond connections and internal coordinates was generated from the atomic Cartesian coordinates, starting from three heavy atoms that are closely involved in the catalytic constraints (see Step 4) with a protein or cofactor residue. UniDesign can generate the ligand atomic parameter and topology files from a mol2 file, e.g.:

<path>/UniDesign --command=MakeLigParamAndTopo --mol2=LIG.mol2 --lig\_param=LIG\_PARAM.prm --lig\_topo=LIG\_TOPO.inp --init\_3atoms=AN1,AN2,AN3

where LIG\_PARAM.prm and LIG\_TOPO.inp are the files to save atomic parameters and residue topology, respectively. AN1, AN2, and AN3 are the names of the three aforementioned atoms.

**Step 3: Design site selection**. Given the enzyme structure (ideally with ligand), an appropriate number of sites can be chosen for the design based on the distances between amino acids and the ligand. These sites are defined in a residue-constraining file (RESFILE) that will be input to UniDesign to control designs.

**Step 4: Defining catalytic constraints and ligand placement workflow to generate ligand poses**. Figuring out the ligand binding pose that is ready for catalysis can be helpful for accurate enzyme design. However, in many cases, one may not know this catalytic pose, which not only depends on the structure and flexibility of the ligand itself but also the context of protein active-site residues. Though it can be possible to use molecular docking methods to search ligand binding poses given the context of an enzyme variant, it is not practical to do so when there are a huge number of variants to be evaluated. I propose to generate a representative ensemble of ligand poses that can potentially cover the catalytic poses even before knowing the context of active-site residues (i.e. before a design process is complete) and let the design algorithm itself decide which ligand pose will best fit an enzyme variant of interest.

“Catalytic constraints”, a set of geometric parameters (distances, angles, and dihedral angles), are defined to constrain the necessary contacts between ligand and catalytic residues or cofactors to ensure catalysis. The catalytic constraints can be derived from experimentally solved protein complex structure(s), quantum mechanics calculation, expert knowledge, or even chemical intuition.

Next, a “grow-and-check” strategy is used to generate ligand poses in the enzyme active-site pocket. Given the enzyme scaffold, UniDesign first truncates all amino acids into glycines and then starts to “grow” ligand atoms one by one from the catalytic anchor residue by varying the catalytic constraints between the ligand and anchoring residue and the dihedral angles of rotatable bonds within the ligand. By progressively growing the ligand atoms, UniDesign can “check” all the catalytic constraints (if applicable) as well as the pose’s internal clash energy (termed “Internal” energy) and external clash interaction with the enzyme backbone (termed “Backbone” energy). Both Internal and Backbone energy should be sufficiently low to make the pose physically feasible and meaningful. The repulsive van der Waals term of the UniDesign energy function (UniEF) is used to calculate the two energies. Ligand poses that satisfy all catalytic constraints with both energies below the given thresholds are saved.

A ligand placement workflow (LPW) file is used to control the ligand-placing process, which describes in detail the rules for ligand pose generation, e.g. how fine a catalytic constraint parameter is sampled, how fine a ligand dihedral angle is varied, the order of calculating atom coordinates, checking catalytic constraints and energy thresholds, etc.

Below is an example command to use UniDesign to generate the ligand poses:

<path>/UniDesign --command=MakeLigPoses --pdb=enzyme.pdb --mol2=LIG.mol2 --resfile=RESFILE.txt --lig\_param=LIG\_PARAM.prm --lig\_topo=LIG\_TOPO.inp --lig\_catacons=LIG\_CATACONS.txt --lig\_placing=LIG\_PLACING.txt --write\_lig\_poses=LIG\_POSES.pdb

where LIG\_POSES.pdb is the file to save the generated ligand poses.

**Step 5: Screening ligand poses**. Of note, not all poses grow in the desired orientation in the active-site pocket. Removal of such “inappropriate” poses can facilitate the convergence of design simulations. As the number of initially generated poses can be huge, it is impossible to remove the “bad” poses manually. The following command is used to screen poses with a more promising orientation:

<path>/UniDesign --command=ScreenLigPoses --pdb=enzyme.pdb --mol2=LIG.mol2 --resfile=RESFILE.txt --lig\_param=LIG\_PARAM.prm --lig\_topo=LIG\_TOPO.inp --read\_lig\_poses=LIG\_POSES.pdb --write\_lig\_poses=LIG\_POSES\_ORNT1.pdb --screen\_by\_ornt=SCREEN\_LIG\_POSES\_BY\_ORIENTATION\_Rule1.txt

The distance rule can be applied repeatedly. After screening by the first rule (e.g. Rule1.txt), the retained poses can be rescreened by a second rule, e.g.:

<path>/UniDesign --command=ScreenLigPoses --pdb=enzyme.pdb --mol2=LIG.mol2 --resfile=RESFILE.txt --lig\_param=LIG\_PARAM.prm --lig\_topo=LIG\_TOPO.inp --read\_lig\_poses=LIG\_POSES\_ORNT1.pdb --write\_lig\_poses=LIG\_POSES\_ORNT12.pdb --screen\_by\_ornt=SCREEN\_LIG\_POSES\_BY\_ORIENTATION\_Rule2.txt

With both rules applied, a large proportion of poses with “inappropriate” binding orientation can be removed. However, the number of poses can be still sufficiently large (e.g. 20K), making it difficult to search the global energy minimum of the enzyme–ligand system. Poses with both Internal and Backbone energy ranked in the top (e.g., 20%; the lower the better) are reserved, e.g.:

<path>/UniDesign --command=ScreenLigPoses --pdb=enzyme.pdb --mol2=LIG.mol2 --lig\_param=LIG\_PARAM.prm --lig\_topo=LIG\_TOPO.inp --read\_lig\_poses=LIG\_POSES\_ORNT12.pdb --write\_lig\_poses=LIG\_POSES\_VDW20.pdb --screen\_by\_vdwpct=0.2

**Step 6: Enzyme sequence/variant design and selection**. The ligand poses generated and screened as above are then used for enzyme sequence design or enzyme variant evaluation (the latter is the case in this study). Below is a design example:

<path>/UniDesign --command=ProteinDesign --enzyme --pdb=enzyme.pdb --mol2=LIG.mol2 --resfile=RESFILE.txt --lig\_param=LIG\_PARAM.prm --lig\_topo=LIG\_TOPO.inp --lig\_catacons=LIG\_CATACONS.txt --read\_lig\_poses=LIG\_POSES\_VDW20.txt --ntraj=5 --wbind=1.0

where --ntraj=5 specifies that five independent design simulation trajectories will be carried out. The --wbind=1.0 option specifies that the interaction energy between enzyme and ligand is scaled by a factor of one (the default value).

The following result files will be generated by UniDesign: XXXX\_bestseqs.txt, XXXX\_beststruct000[1–5].pdb, XXXX\_bestlig000[1–5].mol2, XXXX\_bestsites000[1–5].pdb, and XXXX\_bestmutsites000[1–5].pdb. XXXX\_bestseqs.txt records overall the five designed sequences with total and interaction energy (in UniDesign energy units), whereas the XXXX\_beststruct000[1–5].pdb, XXXX\_bestlig000[1–5].mol2, XXXX\_bestsites000[1–5].pdb, and XXXX\_bestmutsites000[1–5].pdb record the structure models for protein, ligand, all design sites, and mutable sites, respectively. These data files were used for analysis, enzyme variant selection, and data visualization. Among the five independent designs, the one with the lowest total energy was taken as the best design and its interaction score was considered as the enzyme–ligand interaction energy.

### Design protein side-chain packing

<path>/UniDesign --command=ProteinDesign --pdb=xxxx.pdb --wildtype\_only

UniDesign employs simulated annealing Monte Carlo simulation, a stochastic algorithm, to do protein design and protein side-chain packing, and thus different models (i.e., different side-chain configurations) may be generated in distinct runs. This feature is useful when it is necessary to investigate an ensemble of models for highly dynamic proteins such as intrinsically disordered proteins (IDPs).

On the other hand, one can run my [FASPR](https://zhanggroup.org/FASPR/) program([*9*](#_ENREF_9)) for protein side-chain packing when a deterministic model is preferred. FASPR is the most effective and efficient packer among many others.

## Repair incomplete protein side chain

<path>/UniDesign --command=RepairStructure --pdb=XXXX.pdb

UniDesign can be used to repair and optimize the sidechain conformations of amino acids with missing heavy atoms, without disturbing other amino acids.

## Protein structure energy minimization

<path>/UniDesign --command=Minimize --pdb=XXXX.pdb

UniDesign can be used to optimize the sidechain conformations of all amino acids to remove/reduce steric clashes, based on the fixed protein backbone. One can run the RepairStructure command first if the structure has incomplete residues, and then run Minimize to reduce the probable clashes in the whole protein.

## Build mutant structure model

<path>/UniDesign --command=BuildMutant --pdb=protein.pdb --mutant\_file=muts.txt

The mutants are listed in muts.txt, in which each line records one mutant. A mutation is written in the format of {1-letter native amino acid}{chain ID}{amino acid position in the chain}{1-letter mutated amino acid}, e.g. QA22D. A multi-point mutant is the combination of multiple single-point mutations joined with the “,” symbols. Each line is ended with a “;”. This mutant file format is also used by FoldX. A typical UniDesign mutant file is as follows:

QA22D;

HA18F,QA22D;

HA18F,MB20A;

In this example, the first line represents a single mutation on Chain A where Glu22 is mutated to Asp. The second line represents a double mutation on the same chain: His18 on chain A is mutated to Phe and Glu22 on chain A is mutated to Asp. The third line represents a double mutation on different chains: His18 on chain A is mutated to Phe and Met20 on chain B is mutated to Ala.

If the command is run successfully, mutant models will be saved to protein\_Model\_0001.pdb, protein\_Model\_0002.pdb, and protein\_Model\_0003.pdb. Each file corresponds to a mutant in the order listed in the mutant file.

## Add polar hydrogen atom

<path>/UniDesign --command=AddPolarHydrogen --pdb=XXXX.pdb --show\_hydrogen

The option --show\_hydrogen must be added to save the coordinates of polar hydrogen atoms. If the command is run successfully, the model will be saved to XXXX\_PolH.pdb.

## Optimize hydrogen's position to maximize hydrogen bonding network

<path>/UniDesign --command=OptimizeHydrogen --pdb=XXXX.pdb --show\_hydrogen

The option --show\_hydrogen should be added for writing hydrogen atoms in the file. If successful, the model will be saved to XXXX\_OptH.pdb.

## Compute protein stability

<path>/UniDesign --command=ComputeStability --pdb=xxxx.pdb

If successful, the weighted total energy score (stability) and the individual energy terms will be shown on the screen.

Note that the energy value of a single protein is meaningless. Therefore, one should never compare the stability scores of two irrelevant proteins. Instead, it is meaningful to calculate the ΔΔGstability caused by mutations on a protein. For example, one can run the BuildMutant command to generate a mutant model for a wildtype protein, and then run the ComputeStability command for the wildtype and mutant protein to compute two stability values; the difference can be taken as ΔΔGstability (mutant minus wildtype). The more negative of ΔΔGstability, the more stable the mutant.

## Compute protein interchain binding interaction

<path>/UniDesign --command=ComputeBinding --pdb=AB.pdb

If the protein has more than 2 chains, one needs to split them into two parts and evaluate their binding interaction, e.g.:

<path>/UniDesign --command=ComputeBinding --pdb=ABC.pdb --split\_chains=AB,C

In the second command, UniDesign computes the binding interaction energy between chains AB and chain C. If the chains are not split for this multichain system, UniDesign will compute the pairwise interaction energy between A and B, A and C, and B and C separately.

Depending on the cases studied, one may be able to compare the binding interaction scores for two protein-ligand systems (see ref([*10*](#_ENREF_10))).

It is also useful to compute the protein-protein binding affinity change (ΔΔGbind) caused by amino acid mutations. For example, one can run the BuildMutant command on a wildtype protein to create the mutant model, and then run the ComputeBinding command for the wildtype and mutant to get two binding scores; their difference can be interpreted as ΔΔGbind. Sometimes, the mutation may take place at the protein-protein interface, and for these mutations, a specific approach named [SSIPe](https://zhanggroup.org/SSIPe/)([*11*](#_ENREF_11)) can provide more accurate ΔΔGbind prediction than UniDesign.

# RESFILE

Most of the time, one is not interested in redesigning a whole protein (chain) but just a few positions while keeping others unchanged. For example, one may want to redesign just a few interface residues on chain A of a huge dimer protein (e.g. AB.pdb) to enhance the affinity for binding chain B. In such a case, one can use a residue-constraining file (RESFILE) to specify the residues to be designed (mutable), repacked, or fixed.

In UniDesign, the residues for a designer protein can be divided into four groups: designable, repacked, fixed, and catalytic. The former three groups apply to all design tasks, while catalytic is just for enzyme design. The designable residues can be substituted into other amino acid types (and conformations). The repacked residues can have their conformations changed only. The fixed residues are kept constant. The catalytic residues indicate which residues are subjected to catalytic constraints in enzyme design and can have their conformations either fixed as the native or changed (which is similar to repacked).

The RESFILE has the following format:

SITES\_DESIGN\_START

A 782 ACDEFGHIKLMNPQRSTVWY

A 785 ACDEFGHIKLMNPQRSTVWY

… …

A 991 ACDEFGHIKLMNPQRSTVWY

A 993 ACDEFGHIKLMNPQRSTVWY

SITES\_DESIGN\_END

SITES\_REPACK\_START

A 783 C

… …

A 1022 D

A 1024 E

SITES\_REPACK\_END

The designable residues are specified between the keywords SITES\_DESIGN\_START and SITES\_DESIGN\_END; one line stands for one design site. For instance, line A 782 ACDEFGHIKLMNPQRSTVWY means that position 782 of chain A is designable and can be substituted into all 20 amino acid types. One can change the content of the string ACDEFGHIKLMNPQRSTVWY to restrict the alternative types. For instance, one can set it to ACFGILMPVW if the site needs to be a hydrophobic residue. The repacked residues are specified between the keywords SITES\_REPACK\_START and SITES\_REPACK\_END. Similarly, the catalytic residues are listed between SITES\_CATALYTIC\_START and SITES\_CATALYTIC\_END. The fixed residues can be ignored as all residues not listed in the RESFILE are treated as fixed.

There are some scenarios for defining which group a residue should be classified into. For example, for the complete redesign of a whole protein sequence, all the residues are designable. For protein-ligand interaction design, the residues that are in direct contact with the ligand (the first shell) can be chosen as designable sites, and the residues in the second shell (in direct contact with the first-shell residues but no contact with the ligand) can be taken as repacked residues; all others are considered fixed. For enzyme activity design, the first-shell residues involved in catalysis are catalytic residues, the first-shell non-catalytic residues are designable, the second-shell residues are repacked, and the others are fixed. For protein-protein interaction design to enhance binding, the interface sites are designable, the second-shell residues are repacked, and the others are fixed. Dividing residues into these groups appropriately is quite helpful for one’s design, especially when the protein is quite large.

# How to cite UniDesign?

Huang X, Zhou J, Yang D, Zhang J, Xia X, Chen YE, Xu J. Decoding CRISPR–Cas PAM recognition with UniDesign. Briefings in Bioinformatics 2023, 24 (3): ddad133. doi: 10.1093/bib/bbad133.

Huang X, Sun Y, Osawa Y, Chen YE, Zhang H. Computational Redesign of Cytochrome P450 CYP102A1 for Highly Stereoselective Omeprazole Hydroxylation by UniDesign. 2023. Submitted.

# References

1. P. Mitra, D. Shultis, Y. Zhang, EvoDesign: de novo protein design based on structural and evolutionary profiles. *Nucleic Acids Res.* **41**, W273-280 (2013).

2. R. Pearce, X. Huang, D. Setiawan, Y. Zhang, EvoDesign: Designing protein-protein binding interactions using evolutionary interface profiles in conjunction with an optimized physical energy function. *J. Mol. Biol.* **431**, 2467-2476 (2019).

3. X. Huang, R. Pearce, Y. Zhang, EvoEF2: accurate and fast energy function for computational protein design. *Bioinformatics* **36**, 1135-1142 (2020).

4. G. G. Krivov, M. V. Shapovalov, R. L. Dunbrack, Jr., Improved prediction of protein side-chain conformations with SCWRL4. *Proteins* **77**, 778-795 (2009).

5. J. Schymkowitz *et al.*, The FoldX web server: an online force field. *Nucleic Acids Res.* **33**, W382-388 (2005).

6. Z. Miao, Y. Cao, T. Jiang, RASP: rapid modeling of protein side chain conformations. *Bioinformatics* **27**, 3117-3122 (2011).

7. R. Guerois, J. E. Nielsen, L. Serrano, Predicting Changes in the Stability of Proteins and Protein Complexes: A Study of More Than 1000 Mutations. *J. Mol. Biol.* **320**, 369-387 (2002).

8. A. Leaver-Fay *et al.*, Scientific benchmarks for guiding macromolecular energy function improvement. *Methods Enzymol.* **523**, 109-143 (2013).

9. X. Huang, R. Pearce, Y. Zhang, FASPR: an open-source tool for fast and accurate protein side-chain packing. *Bioinformatics* **36**, 3758-3765 (2020).

10. X. Huang, C. Zhang, R. Pearce, G. S. Omenn, Y. Zhang, Identifying the Zoonotic Origin of SARS-CoV-2 by Modeling the Binding Affinity between the Spike Receptor-Binding Domain and Host ACE2. *J. Proteome Res.* **19**, 4844-4856 (2020).

11. X. Huang, W. Zheng, R. Pearce, Y. Zhang, SSIPe: accurately estimating protein-protein binding affinity change upon mutations using evolutionary profiles in combination with an optimized physical energy function. *Bioinformatics* **36**, 2429-2437 (2020).