TRain: T-cell Receptor Automated Immunoinformatics – User's Guide

Austin Seamann & Dario Ghersi

https://github.com/Aseamann/TRain

Disclaimer and Acknowledgements

These programs are distributed in the hope that they will be useful, but without any warranty; without even the implied warranty of merchantability or fitness for any purpose. The entire risk as to the quality and performance of the program is with the user.

TRain was developed by Austin Seamann in the College of Information Science & Technology, University of Nebraska at Omaha.

Email addresses:
Austin Seamann: aseamann@unomaha.edu
Dario Ghersi: dghersi@unomaha.edu

Contents

1 — Introduction

1.1 Overview of TRain

The TRain suite integrates in-house as well as widely used bioinformatics tools to streamline the creation of T-cell receptor (TCR) models, either in isolation or in complex with peptide-MHCs (pMHCs). TRain consists of five main components: input, modeling, pairing, docking, and analysis (see Figure 1.1). Alternative modeling and docking tools can be utilized, as the connection between each step is through common file types such as FASTA and PDB files. In its current state, TRain takes advantage of the Rosetta suite of programs found at https://www.rosettacommons.org.

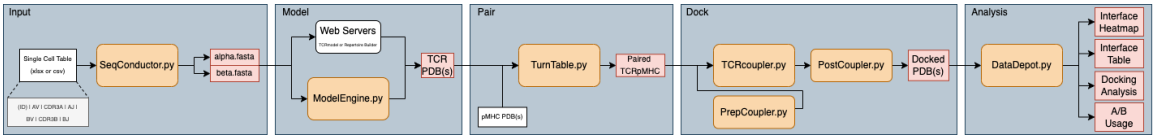

Figure 1.1: **Flowchart for the TRain suite, from input to analysis.** TRain takes as input single cell TCR sequencing data to produce full TCR sequences. Full TCR sequences are then submitted to a modeling web server of choice. Resulting PDB files containing TCR 3D structure are then paired with a pMHC to be docked. Docking is conducted with an automated pipeline based on Rosetta programs. Resulting TCRpMHC complexes then can be further investigated with analytical tools provided in the TRain suite.

1.2 TRain setup & availability

The code for TRain is available through GitHub at: https://github.com/Aseamann/TRain. TRain depends on several Python3 packages and has other dependencies. Packages and dependencies for each step are described in detail in their respective chapters. However, utilizing the pip installation method as described in the Quick Start Guide chapter is the easiest way to get started. Use of a package manager such as Conda (https://www.anaconda.com/products/individual) is recommended.

Although the instructions provided in each section may not account for all possible scenarios, there are lots of valuable online resources to find solutions.

1.3 Input

`TRain` is capable of handling single cell T-cell receptor sequencing data using the in-house Python program `SeqConductor`. `SeqConductor` reads in either a xlsx or csv file containing TCR AV, AJ, BV, and BJ gene segments and CDR3α and CDR3β amino acid sequences, one TCR per line. Using this data, the tool constructs the amino acid sequences for the variable regions of each TCR.

The resulting variable region sequences can then be concatenated with a constant TCR region to produce a full TCR sequence. The primary goal of `TRain` is to gain further understanding of TCRpMHC interfaces. Thus, the constant region is not usually used for analysis but is needed for modeling purposes to avoid alignment errors. The full TCR sequences are then prepared to be passed to the following modeling step. Resulting fasta files are produced and can either be submitted directly to the suggested modeling suites or automated in the Modeling step.

1.4 Modeling

Currently, four modeling pipelines are readily available through web servers to produce 3D structures of TCRs from submitted alpha and beta chain sequences. The four web servers are: `LYRA` [Kla+15], `TCRmodel` [GP18], `Repertoire Builder` [Sch+19], and `TCRBuilder` [20]. `TCRmodel` has the advantage of also being available as a local program through the Rosetta suite.

To increase the efficiency of submission to `TCRmodel`, the next in the pipeline (`ModelEngine`) can utilize multiple CPU cores to rapidly produce models. `TCRmodel` can be run with or without loop refinement, a choice that impacts the time it takes to create a model. `ModelEngine` manages submission and renaming of the resulting output from `TCRmodel`.

Support has now been added for TCRmodel2 [Yin+23]. This modified version of AlphaFold for use on modeling TCRs. Refer to the paper and their GitHub repository for additional instructions beyond what's provided in this manual https://github.com/piercelab/tcrmodel2.

1.5 Pairing

An important step for characterizing the biological function of TCRs is to study their binding properties. To do this, the first step is pairing them with potential antigens. Unlike antibodies, TCRs bind to to peptides displayed on molecules known as the Major Histocompatibility Complex (MHC), also known as human leukocyte antigen (HLA) in human. Since `TRain` can also be utilized for non-human TCRs, we will utilize the general term MHC throughout.

The program `TurnTable` carries out several steps to automate TCR pairing to one or several pMHCs. First, `TurnTable` centers the TCR to the origin and aligns its three principal axes to the x, y, and z axes, ensuring the interaction interface is pointing down with the alpha chain on the left and the beta chain on the right. Subsequently, `TurnTable` positions the TCR's interface in-line but at a distance to the pMHC interface, as recommended when using RosettaDock 4.0 [Mar+18].

Along with repositioning the TCR and pMHC, `TurnTable` trims the TCR chains to just the variable region (if needed), removes the beta-2 microglobulin (which can be present in experimentally derived TCRs), and trims the MHC. This trimming process helps reduce the runtime of docking in the next step of the `TRain` pipeline. Finally, `TurnTable` renumbers the resulting PDB's atoms and

residues as required by RosettaDock 4.0. To return to the original numbering that was found in the crystal structure, the `PostCoupler` program can be used.

1.6 Docking

`TCRcoupler` is the tool that facilitates TCRs docking to the paired pMHC antigen. `TCRcoupler` can be run on a single PDB file or a directory containing several PDB files. Following the RosettaDock 4.0 procedure, flexible side chain and flexible backbone settings are allowed for the TCR chains [Mar+18]. However, less time-intensive (and possibly less accurate) docking can be performed by choosing rigid settings instead [Cha+11a]. In the guided examples provided in this manual, a lower number of docking permutations will be utilized but recommended numbers will be provided.

Resulting docked TCRpMHC complexes can be viewed using any protein visualization program. However, to simplify downstream analyses, a few steps should be carried out after docking: `PostCoupler` takes in resulting PDB files and cleans them of any residual information and most importantly, rematches the numbering of the native PDB files.

To benchmark docking approaches, crystal structures of TCRpMHC complexes are often "re-docked" and the resulting predicted complexes are compared against the experimentally derived structures. To facilitate this step, the program `PrepCoupler` can be used to prepare an experimentally derived PDB file to be submitted for comparison in the following analysis step.

1.7 Analysis

Once a complete experimentally derived or modeled docked TCRpMHC structure is produced, `DataDepot` can be used to produce several analytical figures and tables to profile the binding interface, getting quick insights into the docked structures produced. Analysis can be done to look closely at a single interface or compare across several with different tools. All analysis steps are conducted using Python scripts. Thus, users with experience performing data analysis in Python are encouraged to explore the code and expand it or modify it.

2 — Quick Start Guide

2.1 Installs

The code for `TRain` is available through GitHub at: https://github.com/Aseamann/TRain. Download or clone the latest version to your location of choice.

Using git command line:

```
git clone https://github.com/Aseamann/TRain
```

Recommended Setup:

```
# Enter TRain directory
cd TRain
# Setup conda environment with python3.9
conda create –n TRain python=3.9
# Modify "TRain/data/config.ini" to point to Rosetta install (see below)
# Additionally, if intending to use tcrmodel2, provide the paths as well.
# Use preferred text editor (Vi, Nano, etc)
open data/config.ini
# Build python package
python –m pip install –e .
# Activate the conda environment – needed everytime TRain is in use
conda activate TRain
```

For modeling and docking, a local version of Rosetta needs to be installed. For licensing details visit https://www.rosettacommons.org/software/license-and-download. Installation instructions can be found at https://new.rosettacommons.org/docs/latest/getting_started/Getting-Started. Rosetta should be built with MPI executables and instructions can be found at https://new.rosettacommons.org/docs/latest/build_documentation/Build-Documentation (or follow

the instructions below). While there is the option to run without MPI, it is not recommend. For more instructions on how to run `ModelEngine` and for a list of command line options, see chapter 5.

If MPI is not installed on your computer, you can install it using the following command (for a system with an apt installer):

```
# Linux
sudo apt install libopenmpi-dev
# Or visit: https://www.open-mpi.org/faq/?category=building

# MacOS
xcode-select --install
brew install openmpi
# Or visit: https://www-lb.open-mpi.org/software/ompi
```

Please read below for general build options of Rosetta for Linux and MacOS systems. RosettaCommons forums are a good place to resolve compilation issues.

```
# Navigate to rosetta download
cd rosetta_src_xx/main/source
# Recommended build command
./scons.py bin mode=release extras=mpi -j<number_of_processors_to_use>

# MPI not working? Potentially need to point to mpicxx and mpicc
# Print location of mpicxx and mpicc
which mpicxx
which mpicc

# Use preferred text editor to open site.settings
open tools/build/site.settings

# Uncomment cxx and cc override xml options,
# replace with location of your devices executables
```

The configuration file for `TCRcoupler` is inside directory `/TRain/data/config.ini`. Modify the configuration file with your preferred text editor specifying the location of Rosetta installation folder in line 2 inside the quotation marks.

TCRmodel2 can be used alternatively to TCRmodel. However, TCRmodel2 is not included in Rosetta, so an additional installation is necessary. To install TCRmodel2, please visit the GitHub repository https://github.com/piercelab/tcrmodel2. Installation can either be through the singularity installation process or the full installation process (distinguished by using the "-sing" command with ModelEngine). If intending to model unbound TCR chains, utilize the modified `.def` files found in `/TRain/util`. Update `/TRain/data/config.ini` with the paths to TCRmodel2, necessary AlphaFold databases, and if required the singularity `.sif` files.

Each step of this Quick Start guide contains output files, so you can compare what you obtain and make sure each step runs correctly.

2.2 Input

For all the steps, we assume that your working directory is "/TRain". The first step is to construct a full TCR protein sequence using the gene segments and CDR3 regions listed.

```
cd tr01input
ls
SeqConductor Sample_Input/Test_Table_Single.xlsx -f -a
```

The files `alpha.fasta` and `beta.fasta` should appear in your working directory. The contents should be identical to the fasta files location in "tr01input/Sample_Output/Quick_Start/" and "tr01input/Sample_Input/Quick_Start/".

2.3 Model

Now we can submit our fasta files for modeling. For this step we will need a working installation of Rosetta.

```
cd ../tr02model
ls
ModelEngine -a Sample_Input/Quick_Start/alpha_single.fasta
            -b Sample_Input/Quick_Start/beta_single.fasta

# alternative of tcrmodel2 - config must contain paths
ModelEngine -a Sample_Input/Quick_Start/tcrmodel2/alpha_single.fasta
            -b Sample_Input/Quick_Start/tcrmodel2/beta_single.fasta
            --tcrmodel2 -sing
```

The resulting model will be under the directory “Modeled”. You can open this PDB file with your PDB file viewer of choice. Only the variable region of the TCR will be modeled. It should match the model found in "tr02model/Sample_Output/Quick_Start/" and "tr03pair/Sample_Input/Quick_Start/".

2.4 Pairing

The TCR we chose binds to the *Influenza* M1 antigen. Therefore, we will pair it with an M1 antigen and MHC from PDB model `5ISZ`. This step will prepare the two structures for the subsequent docking step.

```
cd ../tr03pair
TurnTable -t Sample_Input/Quick_Start/ES179M1-01.pdb
-p Sample_Input/5isz.pdb
```

The paired TCR and pMHC will now be in the PDB file under the new directory "Paired". The new name of the PDB will start with the TCR name and then the pMHC name.

2.5 Docking

For this Quick Start guide, we will perform a very short docking run to ensure that everything is working properly. There are two sample runs listed so you can determine if the MPI binaries

are working on your system, or else you can use Rosetta without MPI. These examples are not recommended for actual biological interpretation of the interaction, as they are very short. To ensure you're utilizing `TRain` properly, visit chapter 6.

We will begin by creating a new directory to carry out docking. This is done so our output can be organized, as several sub-directories will be created for each docking step.

```
cd ../tr04dock
mkdir QS_Dock
cp Sample_Input/Quick_Start/ES179M1-01_5isz.pdb QS_Dock/
cd QS_Dock

# Example 1 (w/o MPI)
TCRcoupler ES179M1-01_5isz.pdb -q -r -a 10 -d 100 -e 10

# Example 2 (with MPI)
TCRcoupler ES179M1-01_5isz.pdb -r -a 10 -d 100 -e 10
```

Since we are testing with the rigid docking protocol, you will find three folders within the "/output_files" directory. Within the folder there will be the "relax", "dock", and "refine" directories. The best scoring refined PDB will be found in the "refine" directory.

We will now take the final structure and renumber the pMHC to what was present in 5isz.pdb. Your input to PostCoupler.py will be unique, so please follow the instructions below.

```
cd ../
PostCoupler QS_Dock/output_files/refine/ES<tab> -c ACDE
-r Sample_input/Quick_Start/5isz.pdb
```

2.6 Analysis

To conduct an analysis of a single docked TCRpMHC complex, we will begin with both the Rosetta Energy Breakdown Table and Heatmap. Energy Breakdown will provide us a summary of which amino acids are interacting between the TCR and the pMHC.

```
cd ../tr05analysis

# Table (output = output.csv)
DataDepot -t
Sample_Input/Quick_Start/ES179M1-01_5isz_Docked.pdb

DataDepot -t
Sample_Input/Quick_Start/ES179M1-01_5isz_Docked.pdb -m

# Heatmap
DataDepot -e
Sample_Input/Quick_Start/ES179M1-01_5isz_Docked.pdb

DataDepot -e
Sample_Input/Quick_Start/ES179M1-01_5isz_Docked.pdb -m
```

ABusage calculates interface scores for alpha chain to pMHC and beta chain to pMHC, utilizing the Rosetta application InterfaceAnalyzer.

```
1 DataDepot -a
2 Sample_Input/Quick_Start/ES179M1-01_5isz_Docked.pdb
```

3 — The Input Step

The input step is handled by the `SeqConductor` program. The `SeqConductor` program takes in input from single cell sequencing data of T-cell receptor gene segment names and CDR3 amino acid sequences. These segments are then aligned with the CDR3 amino acid sequence, giving priority to the CDR3 sequence in case of disagreement. Output can be specified for an information table (saved as .csv) or an alpha and beta fasta file for direct submission to Repertoire Builder or TCRmodel.

3.1 Input Step Installation

`SeqConductor` executable code can be found in the folder `tr01input`. The executable `SeqConductor` is callable after the initial installation (see Quick Start Guide), but can be run individually with by installing the following dependencies:

```
pip install pandas
pip install openpyxl
```

3.2 Using the `SeqConductor` program

3.2.1 Input files

The input to `SeqConductor` is a table of single-cell sequencing data in the form of TRAV, CDR3α sequence, TRAJ, TRBV, CDR3β sequence, TRBJ. By default, TRAV, CDR3α, TRAJ, TRBV, CDR3β, TRBJ are corresponding to columns 1, 2, 3, 4, 5 and 6, respectively, while column 0 is reserved for sequence ID. This can be altered with parameters listed below. These tables can be in the format of xlsx or csv files. The program automatically detects the format based on the file's handle.

For xlsx files, sheet names can also be specified if the information is not contained in the primary sheet. An optional input is the reference fasta file of gene family segments. By default, the fasta file "family_seq.fasta" is utilized. This fasta file is pulling results from IMGT/GENE-DB for the

TCR molecular components of *H. sapiens* [GCL05]. While full support is not guaranteed for other organism, "full_family_seq.fasta" contains the rest of the organisms that IMGT/GENE-DB has data for. Additionally, the user needs to manually select the constant region if a species other than *H. sapiens* is desired.

3.2.2 Parameters

The `SeqConductor` program accepts the following parameters:

Argument	Name	Description	Default
Input			
Single Cell Table		xlsx or csv of single cell t-cell data (Required)	
-s	`--sheet`	Parameter allows for selection of an alternative sheet name from a submitted xlsx file	0
-a	`--append`	Append each chain with constant region	False
-c	`--columns`	List of numbers corresponding to the columns containing each segment. Can exclude Seq ID by providing just 6 columns.	0,1,2,3,4,5,6
-r	`--organism`	Parameter allows you to change organism in use, requires "full_family_seq.fasta" gene family file	*H. sapiens*
-y	`--alpha`	Alpha Chain: Constant region alternative	TRAC*01
-z	`--beta`	Beta Chain: Constant region alternative	TRBC1*01
-g	`--genefamily`	Parameter allows for the selection of an alternative gene family fasta file	family_seq.fasta
Output			
-t	`--information`	Parameter outputs a table containing prominent information along with the resulting $\alpha\&\beta$ chain sequences.	True
-f	–fasta	Parameter provides fasta files with resulting $\alpha\&\beta$ chains separated to two fasta files.	False
-m	`--omission`	Characters to avoid in header ID for fasta file submission | comma sep.	
Other			
-v	`--verbose`	Show alignments being produced	False
-l	`--silent`	Mute poor alignments	False

Examples of typical runs:

```
cd tr01input
# Standard alpha.fasta & beta.fasta output, just variable regions
SeqConductor Sample_Input/Test_Table.xlsx –f

# Information Table output
SeqConductor Sample_Input/Test_Table.xlsx –t

# Modeling input: alpha.fasta & beta.fasta output, with variable and constant
```

```
# region
SeqConductor Sample_Input/Test_Table.xlsx –f –a

# Modeling input: alpha.fasta & beta.fasta output, with variable and constant
# region but adjusting for Rep. Builder naming
SeqConductor Sample_Input/Test_Table.xlsx –f –a ––omission +
```

3.2.3 Output

Sample outputs can be found in the subdirectory `Sample_Output/Chapter_Examples`.

1. alpha.fasta
2. beta.fasta
3. alpha_full.fasta (Appended constant region)
4. beta_full.fsata (Appended constant region)
5. alpha_full_rep.fasta (Removed "+" from Seq. IDs to avoid Repertoire Builder Error)
6. beta_full_rep.fasta (Removed "+" from Seq. IDs to avoid Repertoire Builder Error)
7. Test_Table_Results.csv

3.3 Under the hood

The main steps carried out by `SeqConductor` are:

1. Construct dictionary of gene family segments
2. Construct dictionary containing information from submitted table
3. Create full TCR sequences
4. Output:
   (a) Information Table
   (b) Fasta files: alpha.fasta & beta.fasta

Alignment of V-gene segment + CDR3 + J-gene segment is conducted in step 3. The CDR region, which from cell sequencing is in full, is always retained while the start/end of each gene segment can be trimmed. All alignments are performed with the following scores: *Match* = 2.5; *Mismatch* = -1; *Gap* = -1.5; and *Extend Gap* = -0.5.

First, the program aligns the TRAV segment with CDR3α. To perform the alignment, the V-segment sequence is trimmed to the final amino acids with a length corresponding to the CDR3 sequence minus 5. In the case that a CDR3 sequence is shorter than 7 amino acids, an alignment is performed with all 7 amino acids of the CDR3 sequence to the last 7 amino acids of the V-segment sequence. It is common to see very short overlaps in the V-segment + CDR3 region. To avoid poor alignments where the CDR3 is placed in front of the V-segment, a sliding window approach is then used. In the case of a sliding window alignment, the V-segment is shrunk by one amino acid (from the head of the sequence) until a good alignment is found or the alignment is less than 2 amino acids long. If the alignment ends up being less than 2 amino acids long, the CDR is directly appended to the V-segment and a cautionary message is sent to the console.

The V-segment/CDR3 + J-segment overlap is handled slightly differently. In this case, larger overlaps are more common. Therefore, a single alignment is carried out between the V-segment/CDR3 + J-segment with no additional sliding window option. For this alignment, the V-segment/CDR3 is trimmed to the length of the J-segment length. In the case of a poor alignment with several gaps in

this method, the J-segment is directly appended to the V-segment/CDR3 sequence and a message is sent to the console again. For additional details on alignments, please refer to Figure 3.1.

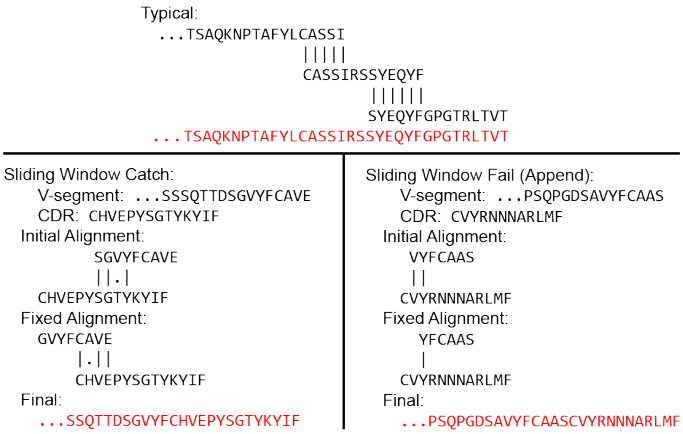

Figure 3.1: `SeqConductor` **Alignment Examples:** The first alignment shows a typical case of V-segment + CDR3 + J-segment. It is common for the V-segment + CDR3 overlap to be very short (2-5 amino acids). The CDR3 + J-segment overlap is typically longer. The sliding window example demonstrates the fallback option if a poor initial alignment is determined. The final example of a sliding window failure happens when an alignment of length 1 is detected and the second segment is simply appended to the first.

4 — The Modeling Step

Many TCR modeling services are already developed and easily available to produce theoretical TCR $\alpha\&\beta$ chain structures. These services include `LYRA` [Kla+15], `TCRmodel` [GP18], `Repertoire Builder` [Sch+19], and `TCRBuilder` [20]. `TCRmodel` and `Repertoire Builder` offer batch submission, which is helpful for generating models of multiple TCR sequences. `ModelEngine` provides the only local modeling option, `TCRmodel`, which is included in distributions of the Rosetta suite. To improve its ease-of-use, `ModelEngine` allows for multi-processing to rapidly produce more models. Support has been extended to `TCRmodel2`. See instructions below and in the Quick Start section if intending to use `TCRmodel2` [Yin+23].

4.1 Modeling Step Installation

`ModelEngine` requires only the installation and building of Rosetta, following the same instructions provided in both the quick start guide and the docking step. Building Rosetta with MPI support (ais not strictly necessary for running `TCRmodel`.

For modeling and docking, a local installation of Rosetta needs to be available. For licensing details visit https://www.rosettacommons.org/software/license-and-download. Installation instructions can be found at https://new.rosettacommons.org/docs/latest/getting_started/Getting-Started. Rosetta should be built with MPI executables and instructions can be found at https://new.rosettacommons.org/docs/latest/build_documentation/Build-Documentation (or follow instructions below).

General build options of Rosetta for Linux and MacOS systems:

```
cd rosetta_src_xx/main/source
./scons.py bin mode=release extras=mpi -j<number_of_processors_to_use>
```

TCRmodel2 can be used alternatively to TCRmodel. However, TCRmodel2 is not included in Rosetta, so an additional installation is necessary. To install TCRmodel2, please visit the GitHub

repository https://github.com/piercelab/tcrmodel2. Installation can either be through the singularity installation process or the full installation process (distinguished by using the "-sing" command with ModelEngine). If intending to model unbound TCR chains, utilize the modified `.def` files found in `/TRain/util`. Update `/TRain/data/config.ini` with the paths to TCRmodel2, necessary AlphaFold databases, and if required the singularity `.sif` files.

4.2 Using the `ModelEngine` program

4.2.1 Configuration File

The `config.ini` file must contain the location of Rosetta installation. The file is found in the directory `/TRain/data/`. If intending to use TCRmodel2, paths must be provided for TCRmodel2, necessary AlphaFold databases, and if required the singularity `.sif` files.

4.2.2 Input files

Two separate fasta files containing alpha chains in one file and beta chains in the other can be submitted. Headers must match up between alpha and beta chains. `SeqConductor` will produce these fasta files.

4.2.3 Parameters

The `ModelEngine` program accepts the following parameters:

Argument	Name	Description	Default
Input			
	`--tcrmodel2`	Use TCRmodel2 instead of TCRmodel	False
-sing	`--singularity`	Use Singularity containter for TCRmodel2	False
-a	`--alpha`	Fasta file containing alpha chain sequences	
-b	`--beta`	Fasta file containing beta chain sequences	
-p	`--peptide`	Fasta file containing peptide chain sequences	
-m	`--mhc`	Fasta file containing MHC chain sequences	
-r	`--refine`	Model with CDR3 loop refinement	False
-s	`--start`	Starting PDB in Fasta file (Alpha)	
Output			
-d	`--dir`	Change standard output directory for resulting modeled TCRs	Modeled
Other			
-v	`--verbose`	Readout output from TCRmodel	False
-c	`--cpus`	Number of CPUs allocated	1
-u	`--cutoff`	Sequence similarity cutoff for templates (int)	100

An example of a typical run:

```
cd tr02model

# Submission to TCRmodel (Some may fail)
# Note, if intending to use MPI support, cpu count should be increased.
```

```
ModelEngine –a Sample_Input/alpha_full.fasta
 –b Sample_Input/beta_full.fasta –m

# alternative of tcrmodel2 – config must contain paths
ModelEngine –a Sample_Input/Quick_Start/tcrmodel2/alpha_single.fasta
                –b Sample_Input/Quick_Start/tcrmodel2/beta_single.fasta
                ––tcrmodel2 –sing
```

4.2.4 Output

Resulting structures will now be available in the output directory "/tr02model/Sample_Output/Chapter_Example/". For sequences that didn't produce a successful model, an error list will be printed after run completion. In this example, as of Rosetta release 362, three models should fail.

4.3 Under the hood

`ModelEngine` utilizes TCRmodel, which is a component of the Rosetta suite. `ModelEngine` determines what script is available to run based on the user's installation of Rosetta. TCR sequences are pulled from the submitted fasta files and paired with their partnered chains. Sequences are then submitted to the TCRmodel script and outputted to the desired output folder. Multi-processing is handled by the Python package `concurrent.futures.ProcessPoolExecutor`. After models are produced, the files are renamed according to the header in the fasta files. Models that fail to build are reported to the user.

5 — The Pairing Step

The modeled TCR structure must be paired to a pMHC prior to proceeding with the docking steps. `TurnTable` allows for submission of single or multiple TCR and pMHC PDB files. It then pairs each each TCR with each pMHC that is submitted. To facilitate the comparison of crystal structures against modeled TCRs, full TCRpMHC complexes can be submitted and the TCR or pMHC automatically extracted. This approach also enables the submission of complete TCRpMHC crystal structures for native structure docking comparisons, as well as the exchange of each TCR with each pMHC in crystal structures.

To help with efficient docking, TCR chains are trimmed if coming from a crystal structure, the MHC is trimmed, and the beta-2 microglobulin is removed. Following trimming, the PDB(s) are then renumbered and relabeled as needed by `TCRcoupler` and this can be adjusted after docking with `PostCoupler.py` if the user wants to return to the original numbering. Returning to the original numbering can simplify comparisons between native crystal structures and re-docked structures for further analysis. As a final step, TCR chains are centered to coordinates 0,0,0 with the pMHC positioned below the TCR.

5.1 Pairing Step Installs

The only non-standard dependencies of `TurnTable` are BioPython, Numpy, SciPy, and Sklearn. These are needed by `PDB_Tools_V3.py`, which `TurnTable` utilizes, and these dependencies are taken care of during the initial installation. Thus, nothing additional needs to be installed for the pairing step.

```
pip3 install biopython
pip install scipy
pip install -U scikit-learn
```

5.2 Using the `TurnTable` program

5.2.1 Input files

There are several input methods for `TurnTable` . The standard is to supply one TCR and one pMHC. Either can be extracted from a complete complex, and pairing will occur between those two structures. Submission of multiple TCRs or pMHC is possible with a folder containing the PDB files, as this is checked for automatically. A single directory of TCRpMHC structures can also be submitted. In this case, every TCR will be paired with every pMHC. If submitting TCR crystal structure, please ensure that you utilize `--trimA` and `--trimB` command line options, as this is not done by default.

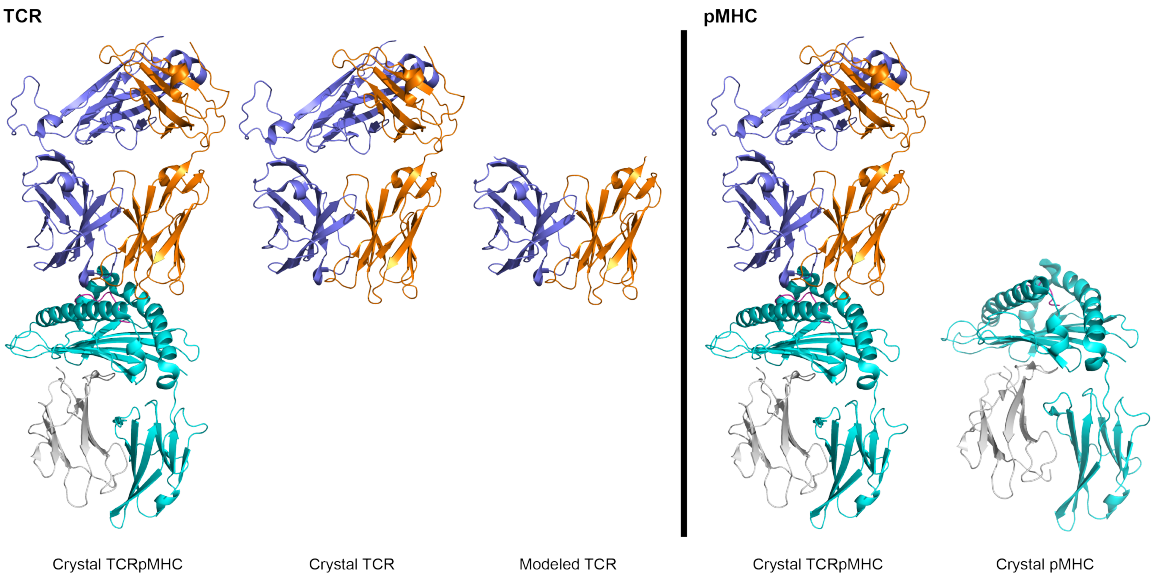

Figure 5.1: `TurnTable` **Input Options:** Above is a visualization of the potential structures that can be submitted to `TurnTable` as PDBs. Additionally, with the command line option "-p", a single or multiple full TCRpMHC crystal structures can be submitted.

5.2.2 Parameters

The `TurnTable` program accepts the following parameters:

Argument	Name	Description	Default
Input			
-t	`--tcr`	TCR pdb file or folder of TCR PDBs	
-p	`--pmhc`	pMHC pdb file or folder of pMHC PDBs	
-d	`--pdbs`	Folder of PDBs and all TCRs and pMHCs will be swapped	
Output			
-a	`--trimA`	Trim TCR alpha, needed if full crystal structure	False
-b	`--trimB`	Trim TCR beta, needed if full crystal structure	False
-m	`--trimM`	Prevent trimming of MHC	False

Examples of typical runs:

```
cd tr03pair
# Run_1:
TurnTable –t Sample_Input/1qrn.pdb –p Sample_Input/1qse.pdb
––trimA ––trimB

# Run_2:
TurnTable –t Sample_Input/1qse.pdb –p Sample_Input ––trimA
––trimB

# Run_3:
TurnTable –t Sample_Input/Modelled –p Sample_Input/1qrn.pdb

# Run_4:
TurnTable –d Sample_Input ––trimA ––trimB
```

5.2.3 Output

Resulting output will create a `Paired/` sub-directory containing updated TCRpMHC PDB files. PDB files are named as "TCR_pMHC.pdb". These files are now ready for docking (see Chapter 6).

5.3 Under the hood

The main steps carried out by `TurnTable` are:

1. Determine files versus folders (for multi-TCR/pMHC support).
2. Create temporary TCR PDB.
3. If crystal structure, clean PDB and extract TCR.
4. Update TCR labels to D and E for alpha and beta chains, respectively.
5. If crystal structure, trim TCR.
6. Align TCR principal axes of inertia to the x, y, and z axes.
7. Create temporary pMHC PDB.
8. Clean crystal structure and extract pMHC.
9. Superimpose reference to centered temporary TCR file.
10. Superimpose pMHC to reference pMHC.
11. Removed reference structure and save as "TCR Name_pMHC Name".pdb

Most of the tasks carried out by `TurnTable` are handled by methods from `PDB_Tools_V3.py` .

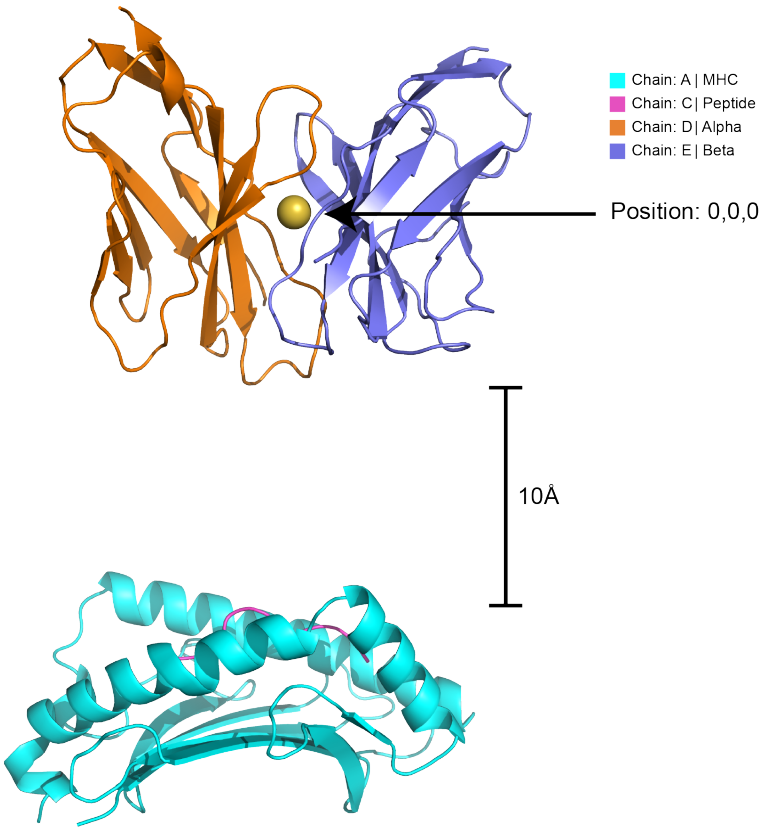

Figure 5.2: **Paired TCRpMHC:** Structure after alignment step. TCR chains' principal axes of inertia are aligned to the x, y, and z axes. The pMHC is placed 10Å away from the interface of the TCR, with the peptide aligned to the x-axis of the TCR

6 — The Docking Step

One or several PDBs can be provided for docking, using the RosettaDock 4.0 protocol [Mar+18] or alternatively a simplified rigid docking protocol from RosettaDock 3.2 [Cha+11b]. Flexible docking allows for movement of side chains and slight movement of the TCR backbone. This flexibility allows for potentially better predictions. Docking is a computationally taxing process: if a multi-core workstation or high-performance computing environment is not available, it may be necessary to stick to rigid docking. Optional parameters and alternative starting positions allow for additional customization.

6.1 Docking Step Installation

For docking, a local installation of Rosetta needs to be available (see Chapter 2).

If MPI is not installed on the computer you can install using the following (apt installer):

```
# Linux
sudo apt install libopenmpi-dev
# Or visit: https://www.open-mpi.org/faq/?category=building

# MacOS
xcode-select --install
brew install openmpi
# Or visit: https://www-lb.open-mpi.org/software/ompi
```

Follow below for general build options of Rosetta for Linux and MacOS systems. RosettaCommons forums are a good place to resolve compilation issues.

```
# Navigate to rosetta download
cd rosetta_src_xx/main/source
# Recommended build command
./scons.py bin mode=release extras=mpi –j<number_of_processors_to_use>

# MPI not working? Potentially need to point to mpicxx and mpicc
# Print location of mpicxx and mpicc
which mpicxx
which mpicc

# Use preferred text editor to open site.settings
open tools/build/site.settings

# Uncomment cxx and cc override xml options,
# replace with location of your devices executables
```

The configuration file for `TCRcoupler` is inside the directory `/TRain/data/config.ini`. Modify the configuration file with your preferred text editor specifying the location of Rosetta install in line 2, inside the quotations.

6.2 Using the `PrepCoupler` program

`PrepCoupler` can come in handy if `TurnTable` was not used, either because the user performed the previous step manually, or in the case of crystal structures being re-docked without separating the individual components. `PrepCoupler` prepares the PDB(s) for Rosetta. To do so, several modifications are made. The PDB is renumbered sequentially as opposed to restarting the sequence for each chain. Further, the chain labels are standardized as follows: A (MHC), C (Peptide), D (Alpha), E (Beta). The beta-2 microglobulin (chain: B) is removed from experimental structures for more efficient docking, as it is not a component of the interface. To further enhance efficiency, the MHC chain is trimmed to just the two alpha helices and the N-termini beta pleated sheet. The $\alpha\&\beta$ chains are also trimmed to contain only the variable region. However, this tool does not separate the TCR from the pMHC.

6.2.1 Input files

Provide any TCR PDB file or folder of TCR PDB files.

6.2.2 Output

TCR PDB file with modifications, to comply with Rosetta applications including: proper order of chains, updated numbering, updated chain labels, removal of unneeded data.

6.2.3 Example Run

```
cd 04_Dock
# Run 1:
```

```
PrepCoupler Sample_Files/1ao7.pdb

# Run 2:
PrepCoupler Sample_Files/PDB_Dir/
```

6.3 Using the TCRcoupler program

TCRcoupler automates several Rosetta applications to generate a docked TCRpMHC complex. Depending on the computational resources available, use of optional parameters may be necessary. To reduce runtime, consider the following options: changing from flexible to rigid docking; decreasing the number of relaxation steps; decreasing the number of docking permutations; decreasing the number of refinement runs; and increase core count if possible.

To guarantee the smooth operation of TCRcoupler several key aspects should be verified. TCRcoupler relies on the mpi complied version of Rosetta. This allows TCRcoupler to utilize multiple cores to decrease runtime. By default, TCRcoupler will request all cores available on the machine, but this can be changed with command line argument (see section 6.1.2). Rosetta requires a specific numbering scheme for the structure and chain order within the PDB. To avoid numbering and chain order issues, utilize TurnTable (refer to Chapter 5).

6.3.1 Configuration File

The config.ini file must be prepared before the installation process in the Quick Start Guide, pointing to the location of the local Rosetta installation (please refer to Chapter 2).

6.3.2 Parameters

The `TCRcoupler` program accepts the following parameters:

Argument	Name	Description	Default
		Input	
pdb		PDB file or folder of PDBs for docking	
-n	`--native`	Native structure folder or file to run optional comparisons to crystal structure, pdbs have to be in folder and match numbering of submitted pdbs for batch submission.	
		Standard Options	
-c	`--cores`	Max number of CPU cores provided	All
-q	`--nompi`	Run Rosetta without MPI, not recommended for large runs	False
		Docking Options	
-f	`--flexible`	Initializes flexible docking	True
-r	`--rigid`	Initializes rigid docking	False
-p	`--pmhc`	(Flex) Number of pmhc relax runs	100
-x	`--xml`	(Flex) Number of xml relax runs for TCR	40
-b	`--bb`	(Flex) Number of back bone rub runs for TCR	30
-s	`--fast`	(Flex) Number of fast relax runs performed	30
-a	`--relax`	(Rigid) Number of relax runs performed	100
-d	`--docking`	(Both) Number of docking runs performed	10000
-e	`--refine`	(Both) Number of refinement runs, post dock	100
-t	`--rotation_check`	(Both - Select to not perform) Ensure alpha chain is over N-terminus of peptide	False
-w	`--cluster`	(Both - Select to not perform) Pairwise spectral clustering of top 200 interface scores (I_sc) after docking permutations, suggested for runs with >1000 docking runs	False
-u	`--num_clusters`	(Both) Desired number of clusters for spectral clustering (Default: 0, which uses an automatic estimate with the eigengap approach	0
-C	`--clear_pdbs`	(Both) Remove PDB files to reduce storage demand after runs. Prevents being able to rerun refine with clustering selection of PDBs and rotation check.	False

6.3.3 Flexible Docking

Input files

PDB(s) submitted for flexible docking must be numbered properly and the alpha and beta chain must come after the MHC and peptide or resutling errors may occur. If PDBs submitted have not be processed through `TurnTable` then `PrepCoupler` should be run. PDBs from experimental structures

should also be run through PrepCoupler . Flexible docking also requires PDB_Tools_V3.py to be located in the same directory as TCRcoupler .

Output

Docking generates several output files, including: several flag files, split TCR and pMHC PDB files, input_files and output_files directories and sub-directories, several intermediate PDB files, and ensemble list files. To help reduce storage utilization, you can automatically delete PDBs produced that are not the highest scoring ones by using the argument "-C". Doing so prevents the option to cluster PDBs for structural comparison. For results, the highest scoring structure will be placed in the primary directory of the docking run.

Example Run

Setup:
Before we start the run, use your preferred text editor and modify config.ini to point to your Rosetta installation.

```
(Driver)
rosetta_loc="/opt/rosetta_src_2021.16.61629_bundle"
```

For an example run, create a new directory in tr04dock for our run. Copy in the structures that are having the docking protocol performed on them.

```
cd tr04dock
mkdir Run_1
cp Sample_Files/2nx5_2nx5.pdb Run_1/
cd Run_1
```

Now let's do a flexible docking run. For this run, we're going to allocate 32 CPU cores.

```
TCRcoupler 2nx5_2nx5.pdb –f –c 100
```

As an alternative, let's submit a folder of PDBs. Here's the entire process. Nohup and allow us to run in background

```
cd tr04dock
mkdir Run_2
cp –r Sample_Files/Multi_Run Run_2/
cd Run_2
nohup TCRcoupler ../Multi_Run –f –c 100 &
```

6.3.4 Rigid Docking

Input files

PDB(s) submitted for rigid docking must be numbered properly and the alpha and beta chain must come after the MHC and peptide or resulting errors may occur. If PDBs submitted have not be processed through TurnTable then PrepCoupler should be run. PDBs from experimental structures should also be run through PrepCoupler .

Output

Rigid docking will produce less resulting files and directories than flexible docking. The produced item will include: three flag files, a time.txt file, several intermediate PDB files, and `output_files` directory. Results can also be found in the `output_files` directory and within `/Refine/`. The best refined structure will be the only pdb in the `output_files` directory.

Example Run

Rigid docking is similar to flexible docking, differing only in the substitution of the -f parameter with the -r parameter. Please notice that a directory of PDBs can also be submitted for rigid docking.

```
cd tr04dock
mkdir Run_3
cp –r Sample_Files/2nx5_2nx5.pdb Run_3/
cd Run_3

TCRcoupler ../Multi_Run –r –c 100
```

6.4 Using the `PostCoupler` program

6.4.1 Parameters

The `PostCoupler` program accepts the following parameters:

Argument	Name	Description	Default
Input			
pdb		PDB file being renumbered	
-r	`--reference`	Reference PDB file being used to match numbering	
-c	`--chains`	Chains that will be renumbered from reference PDB	All
-u	`--custom`	Custom numbering for smallest chain | e.g. 9,11,2,4,5 | Has to match number of amino acids to numbers provided	
Output			
-n	`--name`	(Optional) Customized name for output file	

6.4.2 Input files

PDB being submitted would be the docked structure. This was renumbered to the required scheme for Rosetta applications. Assumes chain labeling is identical.

6.4.3 Output

Resulting PDB with either "_orinum.pdb" appending PDB file provided or with custom name provided.

6.4.4 Example Run

Let's try two examples, the first one, with a redocked structure where both the TCR and pMHC can use the same reference. In the second, we'll used a TCR and pMHC from two separate structures and only renumber from their respective reference.

```
cd tr04dock
# Run 1:
PostCoupler Sample_Files/1qrn_1qrn_docked.pdb
 -r Sample_Files/1qrn.pdb -n 1qrn_1qrn_updated.pdb

# Run 2:
PostCoupler Sample_Files/1qrn_1qse.pdb -r 1qse.pdb -c AC

# Run 3:
PostCoupler Sample_Files/1qrn_1qse.pdb -r 1qrn.pdb -c DE
```

7 — The Analysis Step

The analysis step is a collection of Python methods that allow for quick visual insights. All of these methods are contained in `DataDepot` . Some of these methods still rely on Rosetta to collect further information from the structures produced. Methods that require a Rosetta installation will be highlighted (see below).

7.1 Analysis Step Installation

Several dependencies are needed for `DataDepot` and can be installed using pip if the user did not follow the initial installation described in the Quick Start Guide.

```
pip3 install biopython
pip install scipy
pip install -U scikit-learn
pip install pandas
pip install matplotlib
pip install seaborn
```

7.2 Using the `DataDepot` program

`DataDepot` assumes that TCRs are labeled A = MHC, C = Peptide, D = Alpha, and E = Beta. To relabel chains correctly, `PrepCoupler` can be used (see Chapter 6).

7.2.1 Configuration File

Config.ini file must point to the location of Rosetta installation if conducting Energy Breakdown and Native Structure Comparison (see Chapter 2).

7.2.2 Energy/Distance Breakdown

Rosetta's application Residue Energy Breakdown provides details of residue-residue interactions occurring within a PDB file. This information can provide more insights than just distance value cutoffs for interactions occurring at the interface of a TCRpMHC complex. `DataDepot` provides two ways to view the interactions happening at the interface of the TCRpMHC complex: a simple table and a heatmap.

For both, you can submit either a PDB file or an already complete energy breakdown table. If a PDB is provided, Energy Breakdown is run and the resulting .out file is converted to a .csv file. If a energy breakdown table is submitted, .out and .tsv files are converted to a .csv. The resulting .csv file of either is then processed to produce the table or the heatmap.

Parameters

Energy/Distance Breakdown parameters:

`-t --table` Input pdb for energy breakdown, generates output table
`-e --heatmap_energy` Input pdb for energy breakdown, generates heatmap
`-p --heatmap_dist` Input pdb for distance breakdown, generates heatmap
`-f --fontsize` Adjust font size | Default 12

Table

The Energy/Distance Breakdown table provides a list of interactions occurring between the alpha and beta chains of the TCR to the MHC or peptide. Both attractive and repulsive energy values are shown. Alternatively, distance (angstrom) between residues can be calculated.

Example Run

```
tr05Analysis
DataDepot --table Sample_Input/sample.pdb
```

Heatmap

The Energy/Distance Breakdown heatmap provides similar information as the table. However, the heatmap makes it easier to get a visual summary of interactions occurring at the interface. Only attractive energy values are displayed in the heatmap.

Example Run

```
tr05Analysis
# Run 1:
DataDepot --heatmap_energy Sample_Input/sample.pdb

# Run 2:
DataDepot --heatmap_dist Sample_Input/sample.pdb
```

7.2.3 ABusage

ABusage calculates interface scores for alpha chain to pMHC and beta chain to pMHC, utilizing the Rosetta application InterfaceAnalyzer. The program is run twice, once for the alpha chain and

another for the beta chain. From the output of InterfaceAnalyzer for both, the delta G separated value is pulled. This provides us with an energy based value for the interactions occurring between alpha chain and the pMHC and beta chain and the pMHC. The output is AB_Usage.csv, a table containing all delta G separated scores for each alpha and beta chain.

Parameters

ABusage parameters:

`-a --ab` Submit PDB for ab_usage interface scores

Example Run

```
tr05nalysis
# Run 1:
DataDepot --ab Sample_Input/sample.pdb

# Run 2:
DataDepot --ab Sample_Input/Multi_PDBs
```

7.2.4 Native Structure Comparison

The `TCRcoupler` tool enables the submission of an experimental crystal structure in conjunction with a PDB file for docking purposes. This feature is particularly beneficial for comparing the crystal structure of a TCRpMHC complex with a re-docking attempt, or for contrasting it against submissions that include slightly mutated chains. Submitting a native structures adds additional columns to the score files produced during docking and refinement.

Parameters

Experimental structure comparison parameters:

`-s --sc` Score file produced from docking or refinement (or dir of .sc)
`-x --xaxis` X axis for native structure comparison
`-y --yaxis` Y axis for native structure comparison

Example Run

```
tr05Analysis
# Run 1:
DataDepot --sc Sample_Input/1qrn_1qrn.sc –x I_sc –y Fnat

# Run 2:
DataDepot --sc Sample_Input/Score_Files –x motif_dock –y Fnat

# Run 3:
DataDepot --sc Sample_Input/Score_w_RMSD.sc –x total_score
–y all_rmsd
```

Axis Options:

1. I_sc
2. motif_dock
3. total_score
4. Fnat
5. ca_rmsd
6. all_rmsd

Bibliography

[Cha+11a] Sidhartha Chaudhury et al. “Benchmarking and analysis of protein docking performance in Rosetta v3.2”. In: *PloS one* 6.8 (2011). ISSN: 1932-6203. DOI: 10.1371/JOURNAL.PONE.0022477. URL: https://pubmed.ncbi.nlm.nih.gov/21829626/ (cited on page 9).

[Cha+11b] Sidhartha Chaudhury et al. “Benchmarking and analysis of protein docking performance in Rosetta v3.2”. In: *PloS One* 6.8 (2011), e22477. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0022477 (cited on page 29).

[GCL05] Véronique Giudicelli, Denys Chaume, and Marie Paule Lefranc. “IMGT/GENE-DB: A comprehensive database for human and mouse immunoglobulin and T cell receptor genes”. In: *Nucleic Acids Research* 33.DATABASE ISS. (2005). ISSN: 03051048. DOI: 10.1093/nar/gki010 (cited on page 18).

[GP18] Ragul Gowthaman and Brian G. Pierce. “TCRmodel: High resolution modeling of T cell receptors from sequence”. In: *Nucleic Acids Research* 46.W1 (2018). ISSN: 13624962. DOI: 10.1093/nar/gky432 (cited on pages 8, 21).

[Kla+15] Michael Schantz Klausen et al. “LYRA, a webserver for lymphocyte receptor structural modeling”. In: *Nucleic Acids Research* 43.W1 (2015). ISSN: 13624962. DOI: 10.1093/nar/gkv535 (cited on pages 8, 21).

[Mar+18] Nicholas A. Marze et al. “Efficient flexible backbone protein-protein docking for challenging targets”. In: *Bioinformatics* 34.20 (2018). ISSN: 14602059. DOI: 10.1093/bioinformatics/bty355 (cited on pages 8, 9, 29).

[Sch+19] Dimitri Schritt et al. “Repertoire Builder: High-throughput structural modeling of B and T cell receptors”. In: *Molecular Systems Design and Engineering* 4.4 (2019). ISSN: 20589689. DOI: 10.1039/c9me00020h (cited on pages 8, 21).

[20] "TCRBuilder: multi-state T-cell receptor structure prediction". In: *Bioinformatics* 36.11 (June 2020), pages 3580–3581. ISSN: 1367-4803. DOI: 10.1093/BIOINFORMATICS/BTAA194. URL: https://academic.oup.com/bioinformatics/article/36/11/3580/5809140 (cited on pages 8, 21).

[Yin+23] Rui Yin et al. "TCRmodel2: high-resolution modeling of T cell receptor recognition using deep learning". In: *Nucleic Acids Research* 51 (W1 July 5, 2023), W569–W576. ISSN: 0305-1048. DOI: 10.1093/nar/gkad356. URL: https://doi.org/10.1093/nar/gkad356 (visited on 11/17/2024) (cited on pages 8, 21).