QNIDQPTEMTATEGAIVQINCTYQTSGFNGLFWYQQHAGEAPTFLSYNVLDGLEEKGRFSSFL
SKGYSYLLLKELQMKDSASYLCAVMDSNYQLIWGAGTKLIIKPDIQNPDPAVYQLRDSKSSDKS
LFTDFDSQTNVSQSKDSDVYITDKCVLDMRSMDFKSNSAVAWSNKSDFACANAFNNSIIPEDT
PSPIAGITQAPTSQILAAGRRMTLRCTQDMRHNAMYWYRQDLGLGLRLIHYSNTAGTTGKGEV
GYSVSRANTDDFPLTLASAVPSQTSVYFCASSEAGGNTGELFFGEGSRLTVLEDLKNVFPPEV
FEPSEAEISHTQKATLVCLATGFYPDHVELSWWVNGKEVHSGVCTDPQPLKEQPALNDSRYAL
RLRVSATFWQNPRNHFRCQVQFYGLSENDEWTQDRAKPVTQIVSAEAWGRAMRTHSLRYFR
VSDPIHGVPEFISVGYVDSHPITTYDSVTRQKEPRAPWMAENLAPDHWERYTQLLRGWQQMF
ELKRLQRHYNHSGSHTYQRMIGCELLEDGSTTGFLQYAYDGQDFLIFNKDTLSWLAVDNVAHT
AWEANQHELLYQKNWLEEECIAWLKRFLEYGKDTLQRTEPPLVRVNRKETFPGVTALFCKAHG
PPEIYMTWMKNGEEIVQEIDYGDILPSGDGTYQAWASIELDPQSSNLYSCHVEHSGVHMVLQV
QLVMHVGSHEVHCSYLNSSQPDLEISAWAQYTGDGSPLIDGYDIEQVIEEGNKMWTMYIEPPY
HAKCFLATVGPFTEKRNVRVLPPETRQLTDKGYELFRKLWAICEEELWNKQYLLEHQNAEWAQ
THAVNDVALWSLTDKNFILFDQGDYAYQLFGTTSGDELLECGIMRQYTHSGSHNYHRQLRKLEV
MQQWGRLLQTYREWHDPALNEAMWPARPEKQRTVSDYTTIPHSDVYGVSIFEPVGHIPDSVG
FYRLSHTRMARGWAEASVIQTVPKARDQTWEDNESLGYFQVQCRFHNRPNQWFTASVRLRS
AYRSDNLAPQEKLPQPDTCVGSHVEKGNVWWSLEVHDPYFGTALCVLTAKQTHSIEAESPEFV

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

Austin Seamann & Dario Ghersi

https://github.com/Aseamann/TRain

Month 2022

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.

The `TRain` suite was developed by Austin Seamann in the Bioinformatics lab at 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 provides the accumulation and connection of several in-house and well cited bioinformatic tools to gain further utilization of T-cell receptor sequences and structures. TRain consists of five main components: input, modeling, pairing, docking, and analysis (see Figure 1.1). These components are interchangeable with different modeling and docking tools as the connection between each step is through common file types such as FASTA and PDB files. In the current state, TRain takes advantage of the Rosetta suite of programs found at https://www.rosettacommons.org.

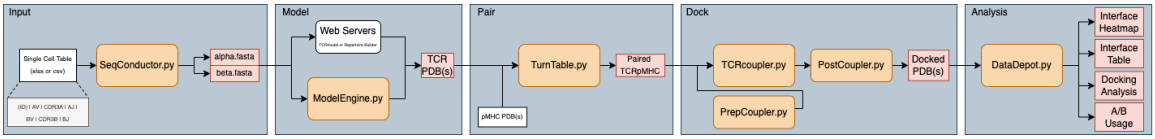

Figure 1.1: **Flowchart for the TRain suite:** General flow of TRain input to analysis. TRain takes input of single cell T-cell receptor sequencing data to produce full TCR sequences. Full TCR sequences are then submitted to modeling webserver of choice. Resulting PDB file containing TCR 3D structure then gets paired with a MHC and bound peptide to be docked. Docking is conducted with an automated pipeline of Rosetta programs. Resulting TCRpMHC complexes then can be further investigated with analytical tools.

1.2 TRain setup & availability

The code for TRain is available through GitHub at: https://github.com/Aseamann/TRain. TRain has several needed Python3 packages and other dependencies. To allow for the flexible of use of TRain , packages and dependencies are laid out for each step and in the quick start guide. Use of a package manager such as Conda (https://www.anaconda.com/products/individual) may be beneficial. With the reliance of several different dependencies, the instructions listed in each section may not be

all-inclusive. Thankfully there are lots of valuable resources to find solutions so if you run into an issue, Google away.

1.3 Input

`TRain` is capable of handling single cell T-cell receptor sequencing output from sources such as (company here). This introduces the first 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. This 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 an extremely valuable portion 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 webservers to produce 3D structures of TCRs from submitted alpha and beta chain sequences. These four services being `LYRA`, `TCRmodel`, `Repertoire Builder`, and `TCRBuilder` [GP18; Kla+15; Sch+19; 20]. `TCRmodel` has an advantage that it is also available as a local program through the Rosetta suite.

To increase the efficiency of submission to `TCRmodel`, the next program, `ModelEngine`, can utilize multiple CPU cores to rapidly produce models. `TCRmodel` can be ran with either no loop refinement or with loop refinement, impacting the time it takes per model. `ModelEngine` manages submission and renaming of the resulting output from `TCRmodel`.

1.5 Pairing

Resulting modeled TCRs become beneficial when you are able to analyze their theoretical biological function. To do this, the first step is pairing it with a potential antigen. Unlike antibodies, TCRs are specific to a peptide presenting structure known as the major histocompatibility complex (MHC), also known as human leukocyte antigen (HLA). As this tool is capable of analysis of TCRs non-exclusively to homo sapiens, the term MHC will be utilized.

The program `TurnTable` has several functions to easily pair TCR structure(s) to one or several pMHCs. First, `TurnTable` centers the TCR to coordinates 0,0,0 and zeroes out all three primary axes of inertia ensuring the interaction inface is pointing down with the alpha chain on the left and beta chain on the right. Then `TurnTable` positions the TCR's interface in-line but at a distance to the pMHC interface as needed according to the RosettaDock 4.0 input desires [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, and trims the MHC. This trimming process helps reduce run-time of docking steps in the next component of `TRain`. As a final step, `TurnTable` renumbers the resulting PDB's atoms and residues as desired by RosettaDock 4.0. This renumbering can be corrected for in the `PostCouper` program.

1.6 Docking

`TCRcoupler` is where TCRs are docked to the paired pMHC antigen. `TCRcoupler` can be preformed on a single PDB file or a directory of PDB files. Following RosettaDock 4.0 procedure, flexible side chain and flexible back bone settings are allowed for the TCR chains [Mar+18]. However, less time-demanding but also less refined docking can be done with rigid settings [Cha+11]. In the guided examples 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 make for better analysis, a few steps should be complete before continuing past 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 determine the capabilities of this docking approach, crystal structures of TCRpMHC interactions can be "re-docked". To assist in this step, the program `PrepCoupler` can prepare a native crystal structure PDB to be submitted for comparison in the following analysis step.

1.7 Analysis

With complete crystal/modeled docked TCRpMHC structures `DataDepot` provides several analytical figures and tables to gain insight to the binding/non-binding interface. This step is not all encompassing of analytical features but can provide quick insight 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.

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
```

Pip installs:

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

For modeling and docking, a local installation 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 and 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. 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 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.

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

2.2 Input

For our quick start we'll ensure each step is able to be ran. Assuming your working directory is "/TRain", the rest of the commands will direct you fully. The first step is to construct a full TCR protein sequence using the gene segments and CDR3 regions listed.

```
cd 01_Input
python3 SeqConductor.py 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 "01_Input/Sample_Output/Quick_Start/" and "02_Model/Sample_Input/Quick_Start/".

2.3 Model

Now we can submit our fasta files for modeling. At this point we will need Rosetta installed.

```
cd ../02_Model
python3 ModelEngine.py -a Sample_Input/Quick_Start/alpha_single.fasta
-b Sample_Input/Quick_Start/beta_single.fasta
```

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 "02_Model/Sample_Output/Quick_Start/" and "03_Pair/Sample_Input/Quick_Start/".

2.4 Pairing

Our modeled TCR is predicted to be a M1 antigen specific TCR. So we'll pair it with an M1 antigen from PDB model 5ISZ. This step will prepare the two structures for the following docking step.

```
cd ../03_Pair
python3 TurnTable.py -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 our quick start guide, we'll attempt a very short docking run just to ensure everything is working properly. There are two examples 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 good predictions, to ensure you're utilizing `TRain` properly, visit chapter 6.

We'll begin by updating our config.ini file or copying it from "02_Model". Then we'll create a new directory to conduct docking. This is done so your output can be organized as several new things will be created to perform each step needed for docking.

```
cd ../04_Dock
cp ../02_Model/config.ini .
mkdir QS_Dock
cd QS_Dock
cp ../TCRcoupler.py .
cp ../PDB_Tools_V3.py .
cp ../config.ini .
cp ../Sample_Input/Quick_Start/ES179M1-01_5isz.pdb .

# Example 1 (w/o MPI)
python3 TCRcoupler.py ES179M1-01_5isz.pdb -q -r -a 10 -d 100 -e 10

# Example 2 (with MPI)
python3 TCRcoupler.py ES179M1-01_5isz.pdb -r -a 10 -d 100 -e 10
```

Since we're testing with the rigid docking protocol, you'll find three folders with 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'll now take the final structure and renumber the pMHC to what was present in 5isz.pdb. While entering input for PostCoupler.py with the example below, your input will be unique so follow instructions below.

```
cd ../
python3 PostCoupler.py 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'll begin with both the Rosetta Energy Breakdown Table and Heatmap. Energy Breakdown will provide us with insight of what amino acids are interacting between the TCR and the pMHC.

```
cd ../05_Analysis

```

```
# Table (output = output.csv)
python3 DataDepot.py –t
Sample_Input/Quick_Start/ES179M1–01_5isz_Docked.pdb
python3 DataDepot.py –t
Sample_Input/Quick_Start/ES179M1–01_5isz_Docked.pdb –m

# Heatmap
python3 DataDepot.py –e
Sample_Input/Quick_Start/ES179M1–01_5isz_Docked.pdb
python3 DataDepot.py –e
Sample_Input/Quick_Start/ES179M1–01_5isz_Docked.pdb –m
```

ABusage

```
python3 DataDepot.py –a
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 segments names and CDR3 amino acid sequences. The inputted segments are then aligned with the CDR3 amino acid sequence taking priority in final output. Output can be specified for an information table or an alpha and beta fasta file for direct submission to Repertoire Builder or TCRmodel. If other modeling suite required, refer to chapter 4, modeling (LYRA & TCRBuilder).

3.1 Input Step Installs

`SeqConductor` executable code can be found in the folder `01_Input`. Python packages required for use can be installed as followed:

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

3.2 Using the `SeqConductor` program

3.2.1 Input files

Required input for `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 respective to columns 1, 2, 3, 4, 5 and 6. 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 based on the file's handle. Regarding xlsx files, sheets names can also be specified if not 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 pulled results from IMGT/GENE-DB specified for the TR molecular components of Homo sapiens

[GCL05]. While full support isn't guaranteed for other organism, "full_family_seq.fasta" contains the rest of the organisms that IMGT/GENE-DB has data for.

3.2.2 Parameters

The `SeqConductor` program accepts the following parameters:

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

`-a --append` Append each chain with constant region

`-c --columns` List of numbers correlating to the columns containing each segment. Can exclude Seq ID by providing just 6 columns. eg. Seq ID, TRAV, CDR3α, TRAJ, TRBV, CDR3β, TRBJ = 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

`-g --genefamily` Parameter allows for the selection an alternative gene family fasta file

Output:

`-t --information` (Default) Parameter outputs a table containing prominent information along with the resulting $\alpha\&\beta$ chain sequences.

`-f --fasta` Parameter provides fasta files with resulting $\alpha\&\beta$ chains separated to two fasta files.

`--omission` Characters to avoid in header ID for fasta file submission | comma sep.

Other:

`-v -verbose` Show alignments being produced

`-l --silent` Silent: Mute even poor alignments

Examples of typical runs:

```
cd 01_Input
# Standard alpha.fasta & beta.fasta output, just variable regions
python3 SeqConductor.py Sample_Input/Test_Table.xlsx –f

# Information Table output
python3 SeqConductor.py Sample_Files/Test_Table.xlsx –i

# Modeling input: alpha.fasta & beta.fasta output, with variable and constant
# region
python3 SeqConductor.py Sample_Files/Test_Table.xlsx –f –a

# Modeling input: alpha.fasta & beta.fasta output, with variable and constant
# region but adjusting for Rep. Builder naming
python3 SeqConductor.py Sample_Files/Test_Table.xlsx –f –a ––omission +
```

3.2.3 Output

Sample outputs can be found in the sub-directory `Sample_Output/`.

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
8. Test_Table_Results_Full.csv

3.3 Under the hood

The main steps carried out by `SeqConductor`:

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 single cell sequencing is in full, is always retained while the start/end of each gene segment can be trimmed. All alignments are done with scores of 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 of the CDR3 sequence minus 5. In the case that a CDR3 sequence is shorter than 7 amino acids, the last amino acids from the V-segments are pulled equal to the length of the CDR3 sequence. It is common to see very short overlaps in the V-segment + CDR3 overlap. 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 seq.) until a good alignment is found or the alignment is less than 2 amino acids long. In the case the alignment is 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 slightly different. In this case, larger overlap is more common. So a single alignment is completed 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 append to the V-segment/CDR3 sequence and a message is sent to the console again. For additional details on performed alignments refer to Figure 3.1

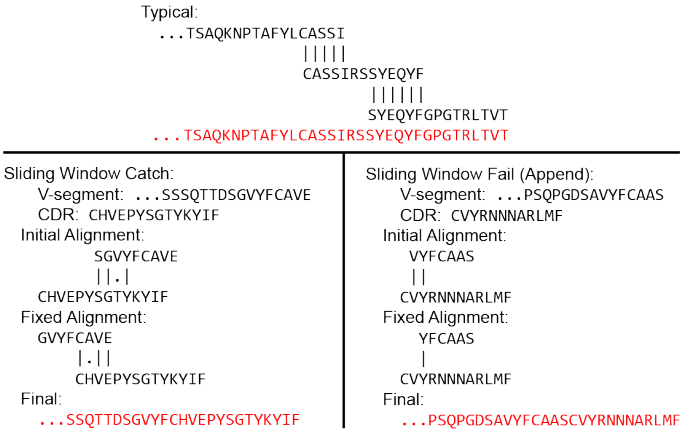

Figure 3.1: `SeqConductor` **Alignment Examples:** The first alignment example is of a typical alignment of V-segment + CDR3 + J-segment. It is common for the V-segment + CDR3 overlap to be very short (2-5 amino acids). The CDR3 + J-segment overlap is typically longer. The sliding window catch example demonstrates the fall back option if a poor initial alignment is determined. Then the final example of a sliding window fail is where an alignment of length 1 is detected and the second segment is just 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 pairs. These services include `LYRA`, `TCRmodel`, and `Repertoire Builder`, `TCRBuilder` [GP18; Kla+15; Sch+19; 20]. `TCRmodel` and `Repertoire Builder` offer batch submission which is helpful for generating models of TCR sequences collected. `ModelEngine` utilizes the only locally available modeling option, `TCRmodel`. `TCRmodel` is included in Rosetta distribution and to improve its ease-of-use, `ModelEngine` allows for multi-processing to rapidly produce more models.

4.1 Modeling Step Installs

`ModelEngine` only relies on Rosetta to be installed and built. Instructions are the same for what we find in the quick start guide and in the docking step. However, building with MPI is not necessary for running `TCRmodel`.

For modeling and docking, a local installation 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 and 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.

Follow below for general build options of Rosetta for Linux and MacOS systems.

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

4.2 Using the `ModelEngine` program

4.2.1 Configuration File

Config.ini file must be prepared. Point to the location of chromedriver installation. The file is found in the primary directory of TRain.

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.

To provide a blacklist, in which homologs are restricted to not include provided PDB id, follow the same practice as `Repertoire Builder`. An example is provided below. Be sure that both chain's headers contain the identical list.

```
>4jff ((3vxt,3pwp,4jff))
KQEVEQNSGPLSVPEGAIASLNCTYSFLGSQSFFWYRQYSGKSPELIMFTYREGDKEDGRF
TAQLNKASQHVSLLIRDSQPSDSATYLCAVNDGGRLTFGDGTTLTVKPNIQNPDPAVYQLR
DSKSSDKSVCLFTDFDSQTNVSQSKDSDVYITDKCVLDMRSMDFKSNSAVAWSNKSDFAC
```

4.2.3 Parameters

The `ModelEngine` program accepts the following parameters:

`-a --alpha` Fasta file containing alpha chain sequences
`-b --beta` Fasta file containing alpha chain sequences
`-l --lyra` Automate LYRA submission
`-m --tcrmodel` Automate TCRmodel submission
`-t --tcrbuilder` Automate TCRBuilder submission
`-s --start` Optional starting position in fasta file. Provide header.
`-d --download` Download directory folder name in reference to program location default=Models
`-g --long` Scalar for wait times for elements to appear and page to load

An example of a typical run:

```
cd 02_Model
# Submission to LYRA
python3 ModelEngine.py –a Sample_Input/alpha_full.fasta
 –b Sample_Input/beta_full.fasta –l

# Submission to TCRmodel
python3 ModelEngine.py –a Sample_Input/alpha_full.fasta
 –b Sample_Input/beta_full.fasta –m

# Submission to TCRBuilder
python3 ModelEngine.py –a Sample_Input/alpha_full.fasta
 –b Sample_Input/beta_full.fasta –t
```

4.2.4 Output

Resulting structures that were successful will now be in the download directory `Models` or in the download directory file name provided. For sequences that didn't produce a successful model, an error list will be printed after completion of run.

4.3 Under the hood

`ModelEngine` utilizes the python package Selenium to automate the webtools. This allows for a test environment in a Chrome browser window. This tool works by pointing to HTML and CSS elements and providing input. For `LYRA`, this is straight forward as it submits the chain sequences and accompanying blacklist, then hitting submit. The tool waits for the results page to show and then downloads the file to the desired directory and then followed by renaming the file to the header provided in the fasta file. This is an identical procedure for `TCRmodel` and `TCRBuilder`. However, it is considerably faster to do batch submission for `TCRmodel` if no blacklist is needed. This method is highly reliant on the future modifications made to these webservers. If ever a tool is non-functional, please inform the contact references provided.

Timing:
Each web page has slightly different wait times for elements to appear on screen. For `LYRA`, we initially wait 10 seconds for the web page to load after it's first called. Then we wait until the chain 1 text entry box to appear for a maximum of 10 seconds. Then we input the blacklist, alpha chain, beta chain, and then hit submit. Then we wait for the results page to appear based on it's "jsmol" element for a maximum of 200 seconds. This is typically well within the time it takes for `LYRA` to return a result. We then wait a few seconds before clicking the download button for the PDB to ensure it's loaded. Once downloaded, `ModelEngine` renames the PDB file to the corresponding header. If the TCR model failed to produce, then it's appended to an error list and presented once the program is finished running. This is identical for `TCRmodel`. However, for `TCRBuilder` a maximum wait time for a result is 1800 seconds (20 minutes) as it typically takes longer for a result.

If for any reason that the timings are too short and resulting with errors the command line option "--long" can be included with your scalar. A scalar of 1.5 or 2 may be appropriate to initially try.

5 — The Pairing Step

After constructing a modeled TCR structure, it then needs to get paired to a pMHC before the docking steps. `TurnTable` allows for submission of single or multiple TCR and pMHC PDB files. It then pairs each from the collection. To assist in the ability of utilizing crystal structures versus the modelled TCRs, full TCRpMHC complexes can be submitted and the TCR or pMHC extracted. This process also facilitates the submission of full TCRpMHC crystal structures to be submitted for native structure docking comparison along with swapping of each TCR for each pMHC of crystal structures. To help with efficient docking, TCR chains are trimmed if 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 for `TCRcoupler` and this can be corrected for after docking with `PostCoupler.py` if you want to return to the native numbering. Returning to native numbering can help assist with ensuring consistency between native crystal structures and the re-docked structures for further analysis. As a final step, the center of the TCR chains are centered to coordinates 0,0,0 with the pMHC below the TCR.

5.1 Pairing Step Installs

The only non-standard dependencies of `TurnTable` are BioPython, numpy, SciPy, and sklearn. These are needed for the use of `PDB_Tools_V3.py`, which `TurnTable` utilizes and are requested to be installed in the universal installations. Thus, nothing additional needs to be installed for the pair 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 stripped 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, this is checked for automatically. A single directory of TCRpMHC structures can also be submitted and for every TCR and every pMHC to be swapped around resulting with the squared number of inputs. If submitting TCR crystal structure, ensure you utilize --trimA and --trimB command line options as this is not default.

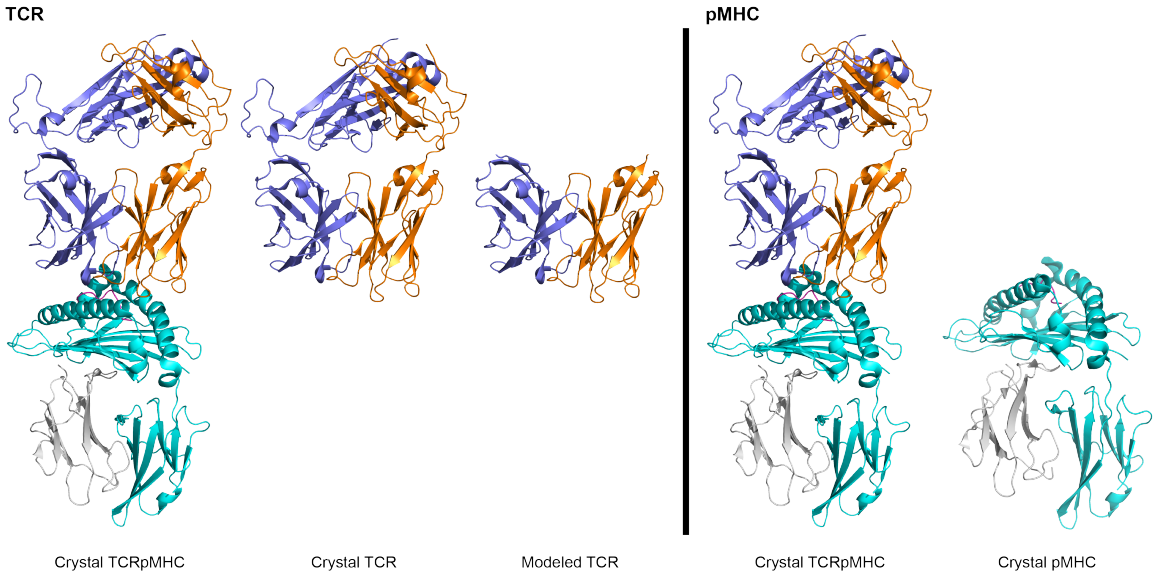

Figure 5.1: TurnTable **Input Options:** Above is a visualization of the potential structures that can be submitted to TurnTable as PDBs. Additionally, with the command line option "-p", a single or multiple full TCRpMHC crystal structures can be submitted.

5.2.2 Parameters

The TurnTable program accepts the following parameters:

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

An example of a typical run:

```
1 cd 03_Pair
```

```
# Run_1:
python3 TurnTable.py –t Sample_Input/1qrn.pdb –h Sample_Input/1qse.pdb
––trimA ––trimB

# Run_2:
python3 TurnTable.py –t Sample_Input/1qse.pdb –h Sample_Input ––trimA
––trimB

# Run_3:
python3 TurnTable.py –t Sample_Input/Modelled –h Sample_Input/1qrn.pdb

# Run_4:
python3 TurnTable.py –p Sample_Input
```

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 file are now in a state prepared for docking, see chapter 6.

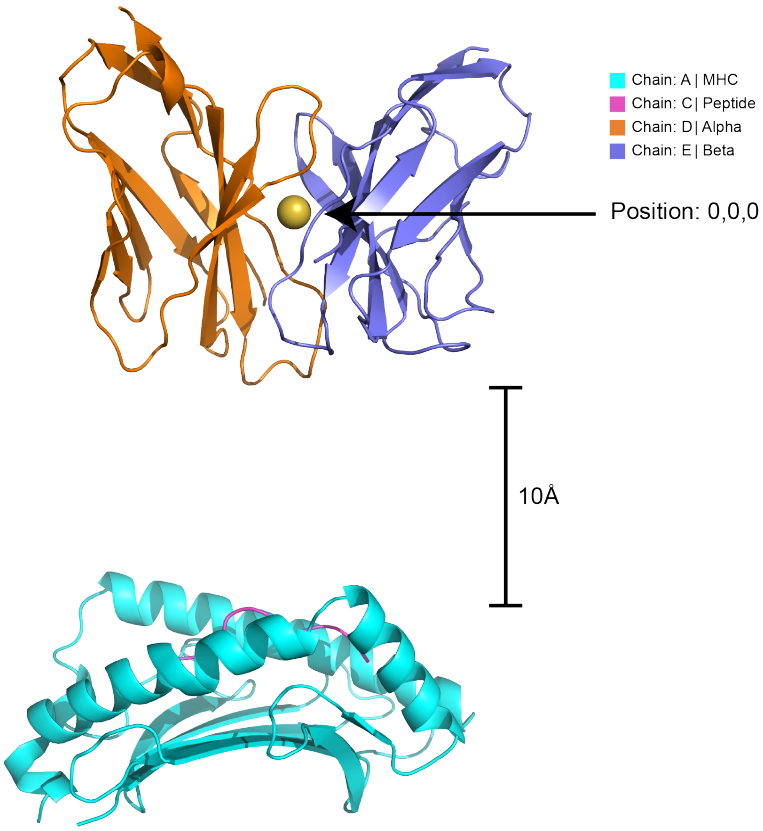

Figure 5.2: **Paired TCRpMHC:** Add new description here.

5.3 Under the hood

The main steps carried out by `TurnTable`:

1. Determine files versus folders.
2. Create temporary TCR PDB.
3. If crystal structure, clean PDB and split off TCR.
4. Update TCR labels to D and E for alpha and beta respectively.
5. If crystal structure, trim TCR.
6. Center TCR utilizing primary component axis (PCA).
7. Create temporary pMHC PDB.
8. Clean crystal structure and split off 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

The majority of `TurnTable` is handled by methods from `PDB_Tools_V3.py` .

6 — The Docking Step

To gain a structural prediction a TCRpMHC interface, two Rosetta docking protocols are part of the `TRain` suite. One or several PDBs can be provided for computational docking conducted following the RosettaDock 4.0 protocol [Mar+18] or alternatively a simplified rigid docking protocol following RosettaDock 3.2. Flexible docking allows for movement of sidechains and slight movement of the backbone of the TCR, this allows for potentially better predictions. Docking is a computationally taxing process; If a heavily multi-core computer is not available, restraining to rigid docking may be necessary. Optional parameters and alternative starting positions can allow for additional customization.

6.1 Docking Step Installs

Installed on server under "/opt/rosetta_src_2021.16.61629_bundle".
For docking, a local installation 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 and Rosetta must be built with MPI executable and instructions can be found at https://new.rosettacommons.org/docs/latest/build_documentation/Build-Documentation or follow 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 the computer you can install using the following (apt installer):

```
# Linux
sudo apt install libopenmpi-dev

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

```
8
9   # Both
10  pip3 install biopython
11  pip install scipy
12  pip install -U scikit-learn
```

or following instructions at https://www.open-mpi.org/faq/?category=building.

Follow below for general build options of Rosetta for most operating systems.

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

The configuration file for `TCRcoupler` is inside directory `04_Dock`. Modify the configuration file with your preferred text editor writing the location of Rosetta install in line 2 inside the quotations.

6.2 Using the `PrepCoupler` program

`PrepCoupler` is a useful program if `TurnTable` is not necessary for any reason. `PrepCoupler` prepares the PDB(s) for Rosetta. To do so, several modifications are made. The PDB is renumbered to a continuous count of amino acids versus a new count for each chain. As well as, the chains are labeled to standard TCR PDB chain labeling of MHC: A, Peptide: C, Alpha:D, and Beta:E. With the removal of the beta-2 microglobulin (chain: B) for more efficient docking as it is not a component of the interface. Then to further enhance efficiency, the MHC chain is trimmed to just the two alpha helices and the back beta pleated sheet. The $\alpha\&\beta$ chains are also trimmed to contain only the variable region. This program however does not separate the TCR from the pMHC. This program should also be ran on native structure being submitted for docking analysis.

6.2.1 Input files

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

6.2.2 Output

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

6.2.3 Example Run

```
1   cd 04_Dock
2   # Run 1:
3   python3 PrepCoupler.py Sample_Files/1ao7.pdb
4
5   # Run 2:
6   python3 PrepCoupler.py Sample_Files/PDB_Dir/
```

6.3 Using the TCRcoupler program

TCRcoupler is an automation of several Rosetta applications to generate a docked TCRpMHC complex. Depending on the computational resources available, use of optional parameters may be necessary. Things to reduce run time include: changing from flexible to rigid docking, decrease number of relaxation steps, decrease number of docking permutations, decrease number of refinement runs, or increase core count.

To ensure TCRcoupler run smoothly a few things should be ensured. TCRcoupler relies on the mpi complied version of Rosetta. This allows TCRcoupler to unitize multiple cores to increase efficiency of runs. By default, TCRcoupler will request all cores available on the machine but 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

Config.ini file must be prepared. Point to the location of Rosetta installation (refer to chapter 2).

6.3.2 Parameters

The TCRcoupler program accepts the following parameters:

pdb PDB file or folder of PDBs for docking
-l --linux Changes to Linux runnable program default
-m --mac Changes to MacOS runnable program
-n --nompi Run Rosetta without mpi, not recommended for large runs default=False
-f --flexible Initializes flexible docking default
-r --rigid Initializes rigid docking
-c --cores Max number of cpu cores provided default=all
-h --pmhc (Flex) Number of pmhc relax runs default=100
-x --xml (Flex) Number of xml relax runs for TCR default=40
-b --bb (Flex) Number of back bone rub runs for TCR default=30
-s --fast (Flex) Number of fast relax runs for TCR default=30
-a --relax (Rigid) Number of relax runs performed default=100
-d --docking (Both) Number of docking runs performed default=20000
-e --refine (Both) Number of refinement runs, post dock default=100
-n --native Native structure folder or file to run optional comparison to crystal structure, pdbs have to be in folder and match numbering of submitted pdbs.

6.3.3 Flexible Docking

Input files

PDB(s) submitted for flexible docking must be number 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 ran. PDBs provided as native structures should also be ran 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, `TCRcoupler` automatically deletes PDBs produced that are not the highest scoring. Automatic removal of PDBs can be inhibited by command line parameter. For results, enter into the `output_files` directory and within `/Refine/` the remaining PDB will be the best scoring docked structure.

Example Run

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

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

For an example run, create a new directory in 04_Dock for our run. Then copy `TCRcoupler.py`, PDB_Tools_V3.py, config.ini, and Sample_Files/2nx5_2nx5.pdb to our new folder.

```
cd 04_Dock
mkdir Run_1
cp TCRcoupler.py Run_1/
cp PDB_Tools_V3.py Run_1/
cp config.ini 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 allocated 100 CPU cores.

```
python3 TCRcoupler.py 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 04_Dock
mkdir Run_1
cp TCRcoupler.py Run_2/
cp PDB_Tools_V3.py Run_2/
cp config.ini Run_2/
cp -r Sample_Files/Multi_Run Run_2/
cd Run_2
nohup python3 TCRcoupler.py Multi_Run -f -c 100 &
```

6.3.4 Rigid Docking

Input files

PDB(s) submitted for rigid docking must be number 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 ran. PDBs provided as native structures should also be ran through `PrepCoupler` .

Output

Rigid docking will produce less resulting files and directories. The produced item will include: three flag files, a time.txt file, several intermediate PDB files, and `output_files` directory. Just with flexible docking, intermediate PDB files will be removed unless requested by user not to. Results can also be found in the `output_files` directory and within `/Refine/` the remaining PDB will be the best scoring docked structure.

Example Run

Rigid docking follows in line with flexible docking but with just the replacement parameter of -f with -r. Keep in mind, a directory of PDBs can also be submitted for rigid docking.

```
cd 04_Dock
mkdir Run_3
cp TCRcoupler.py Run_3/
cp config.ini Run_3/
cp -r Sample_Files/2nx5_2nx5.pdb
cd Run_3

python3 TCRcoupler.py Multi_Run -r -c 100
```

6.4 Using the `PostCoupler` program

6.4.1 Parameters

The `PostCoupler` program accepts the following parameters:

`pdb` PDB file being renumbered
`-r --reference` Reference PDB file being used to match numbering
`-n --name` (Optional) Customized name for output file
`-c --chains` Chains that will be renumbered from reference PDB default=all
`-u --custom` Custom numbering for smallest chain | e.g. 9,11,2,4,5 | Has to match num of aa to numbers provided

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 to examples. 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 seperate structures and only renumber from their respective reference.

```
cd 04_Dock
# Run 1:
python3 PostCoupler.py Sample_Files/1qrn_1qrn_docked.pdb
  –r Sample_Files/1qrn.pdb –n 1qrn_1qrn_updated.pdb

# Run 2:
python3 PostCoupler.py Sample_Files/1qrn_1qse.pdb –r 1qse.pdb –c AC

# Run 3:
python3 PostCoupler.py Sample_Files/1qrn_1qse.pdb –r 1qrn.pdb –c DE
```

7 — The Analysis Step

The analysis step is an accumulation of Python methods that allow for quick visual insight. All of these methods are encompassed in `DataDepot` . Some of these methods still rely on Rosetta to collect even further information from the structures produced. Methods that require a Rosetta installation will be signified and the configuration file must be updated, see more information below.

7.1 Analysis Step Installs

Several decencies are needed for `DataDepot` and can be installed using pip. For more instructions on how to run `ModelEngine`and for a list of command line options, see chapter 7.

```
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. If labeling does not match, `PrepCoupler` can be used from chapter 6.

7.2.1 Configuration File

Config.ini file must be prepared if conducting Energy Breakdown and Native Structure Comparison. Point to the location of Rosetta installation (refer to chapter 2).

7.2.2 Energy Breakdown

Rosetta's application Residue Energy Breakdown provides details of residue-residue interactions occurring within a PDB file. This information can provide more insight than just distance value cutoffs for interactions occurring at the interface of a TCRpMHC complex. `DataDepot` provides two forms to view the interactions happening at the interface of the TCRpMHC complex. These two forms are in a simple table and in 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 ran and the resulting .out file is converted to a .csv. 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 either produce the table or heatmap.

Parameters

Energy Breakdown parameters:

`-t --table` Energy breakdown csv input for table
`-e --heatmap` Energy breakdown csv input for heatmap
`-m --mac` Changes rosetta from Linux to MacOS version default=False

Table

The Energy 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.

Example Run

```
05_Analysis
# Run 1:
python3 DataDepot.py --table Sample_Input/sample.pdb

# Run 2:
python3 DataDepot.py --table Sample_Input/energy_breakdown_sample.out
```

Heatmap

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

Example Run

```
05_Analysis
# Run 1:
python3 DataDepot.py --heatmap Sample_Input/sample.pdb

# Run 2:
python3 DataDepot.py --heatmap Sample_Input/energy_breakdown_sample.out
```

7.2.3 ABusage

ABusage calculates interface scores for Alpha to pMHC and Beta to pMHC utilizing the Rosetta application InterfaceAnalyzer. The program is ran 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 to the pMHC and Beta to the MHC. The output is AB_Usage.csv, which is 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
`-m --mac`	Changes rosetta from Linux to MacOS version default=False

Example Run

```
05_Analysis
# Run 1:
python3 DataDepot.py --ab Sample_Input/sample.pdb

# Run 2:
python3 DatDepot.py --ab Sample_Input/Multi_PDBs
```

7.2.4 Native Structure Comparison

`TCRcoupler` allows you to submit a native crystal structure along with a PDB being submitted for docking. This can be useful to compare a crystal structure of a TCRpMHC complex to a re-docking attempt or compare against slightly mutated submitted chains. Submitting a native structures adds additional columns to the score files produced during docking and refinement.

Parameters

Native 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

```
05_Analysis
# Run 1:
python3 DataDepot.py --sc Sample_Input/1qrn_1qrn.sc -x I_sc -y Fnat

# Run 2:
python3 DataDepot.py --sc Sample_Input/Score_Files -x motif_dock -y Fnat

# Run 3:
python3 DataDepot.py --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+11] 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).

[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 16).

[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, 19).

[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, 19).

[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, 27).

[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, 19).

[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, 19).