Technical details, methodology

This project was conducted within two overall methods. TBCRalign and ESM-2 on the same dataset as described in **Error! Reference source not found.** section. Below is a description of how data processing and analysis were made, as well as some descriptive explanations of certain codes and files.

TBCRalign

TBCRalign processes TCR sequences by analyzing six key regions of the TCR: three from the alpha chain (A1, A2, A3) and three from the beta chain (B1, B2, B3), corresponding to the CDR regions. The model computes similarity scores between TCR sequences and with the ability to assign different weights to different regions. Usefully, the CDR3 regions (weights A3 and B3) can be given higher weights due to their crucial role in antigen recognition. The model generates pairwise sequence alignment scores between TCRs by comparing their CDR regions. These raw alignment scores represent how well two TCR sequences match based on the similarity of their amino acid sequences.

Distance matrix

By running the script *get_tbcralign_distance_matrix.sh*, which was provided at the start of the project, pairwise distances between TCR sequences were calculated and organized into a distance matrix. By changing the weights argument in the call-line to fit the wanted scheme with the knowledge that -w A1, A2, A3, B1, B2, B3, the script was run with 5 different weighting schemes to investigate the relative importance of different TCR chain regions.

The weights used when calling the script were:

CDR3α	$TBCRALIGN -a -w 0,0,1,0,0,0 $tmppath > $tbcrtmp
CDR3β	$TBCRALIGN -a -w 0,0,0,0,0,1 $tmppath > $tbcrtmp
CDR3α+β	$TBCRALIGN -a -w 0,0,1,0,0,1 $tmppath > $tbcrtmp
Unweighted	$TBCRALIGN -a -w 1,1,1,1,1,1 $tmppath > $tbcrtmp
Weighted	$TBCRALIGN -a -w 1,1,4,1,1,4 $tmppath > $tbcrtmp

Where the change in arguments affect how the distances are calculated, by multiplying each sequence (A1,A2,A3,B1,B2,B3) with their respective value. Thus, the *Weighted* call considers the distances between all 6 sequences but chooses to put more importance to the CDR3 (A3 and B3) since these have been shown to have higher distinguishability. Similarly, the *Unweighted* call weights all 6 chains equally.

The script processes TCR sequences through TBCRalign to generate similarity scores from an all-versus-all comparison. For each TCR pair (A,B) both directional comparisons (A→B and B→A) are

computed to ensure symmetry of the final matrix. When running the data through TBCRalign, each pair will receive a raw similarity score, for which the maximum value depends on the used weights. For the *Weighted* pairwise comparisons, the maximum similarity score would be 12, whereas for e.g., *CDR3α* it would be 1. Following this, the distance between each comparison is calculated by subtracting the normalized score from 1.

$$dist = 1 - \left(\frac{score}{sum_{weight}} \right)$$

Using this method, each pair will get a value ranging from 0 to 1, where 0 indicates a perfect match and 1 a maximal difference.

These calculated distances are then placed into the outputted matrix, for which each row and column represent a TCR sequence. Each cell (i, j) describes the distance between TCR_i and TCR_j, thus creating a zero-diagonal.

Heat Map

With the outputted distance matrix, visualizations of each weight's distance matrix were created using *heat_map_tbcralign.py* to see if any clustering was visible – that is, if the data is showing promise of clustering patterns and distinguishability between peptides. Using the *binders-only* dataset, the distance matrix was preprocessed by reordering both rows and columns by peptide while maintaining that the indexes of rows and columns matched. Subsequently, the core distance matrix was extracted to exclude columns containing metadata (peptide, binder, partition), and the resulting heatmap visualizes the pairwise distances between the TCR sequences. Peptide groupings were added to clearly mark the center of each 'grouping', thus allowing an easier visual assessment of the different similarities.

AUC

To evaluate the different weighting schemes by performance, AUC analysis was performed on the data. Using the provided dataset's partition column (containing values ranging from 1 to 5) to cross-validate, AUC scores were calculated for each peptide and weighting scheme from the matrices generated by the initial TBCRalign analysis. Using the *auc_scores.py* and *auco.1_scores.py*, each combination of peptide and weighting scheme was processed to generate ROC curves. Both AUC and AUC0.1 were calculated for relevant assessment of the model's performance.

The results of the AUC calculation were visualized by a grouped bar plot, showing each peptide's performance across the 5 weighting schemes, to directly compare the weighting schemes with each other. An added grouping with the average score across all weighting schemes was added to provide a more general visualization. By plotting the AUC values on a scale from 0 to 1, higher values nearing 1 will indicate better discrimination between positive and negative cases.

ESM-2

Initializing

As mentioned in the ***Error! Reference source not found.*** section of this report, a new dataset was created by merging new information into the one provided by the start of the project. This was done to retrieve the full-length sequences of the TCR which were needed for the ESM-2 model to run.

Calling *full_tcr_retrive.py* solved this issue, by merging based on the the *raw_index* column of both datafiles. This added new columns to the data, from which the most relevant were TCRa, TCRb, CDR1a, CDR2a, CDR3a, CDR1b, CDR2b, and CDR3b. A final column was made by concatenating TCRa and TCRb together to make *full_tcr*.

Once the useful data was available, the ESM-2 model could be loaded onto the server by cloning its GitHub repository. Following this, the data could be passed through by using the command,

results = model(batch_tokens, repr_layers=[33][1], return_contacts=True)

, and the embedding saved to the specified directory.

Two different methods for passing data through the model were attempted. One by running individual sequences of each CDR through the MLM, and a second by running TCRα and TCRβ separately, to later be further evaluated:

Individual CDR sequences through ESM-2

Calling the two files *run_esm_binders.py* and *run_esm_swaps.py* would parse the peptide_x column and a specified weight, e.g., CDR3a, through the ESM-2 model in batches of 25 entries for memory capacity. Since only the specified sequences were put individually through the model, the resulting matrices will have the dimension:

$$\text{SequenceLength}_{\{\text{weight}\}} \times 1280$$

Thus, to obtain the embedding vector, a simple computation to sum along the columns is made to maintain the embedding size of 1280 while acquiring a vector to use for similarity computation.

```
for i, tokens_len in enumerate(batch_lens):

    sequence_representations.append(token_representations[i, 1 : tokens_len -1].sum(0))
```

Following the retrieval of embedding data, the following calls were made:

python3 get_cosine_sim.py "$peptide"

[1] The variable 33 is predetermined by which version of ESM-2 is used. For this project, the pretrained model 'ESM-2_t33_650M_UR50D' was used.

Which was made by the bash script *call_get_cosine_sim.sh*. In the *get_cosine_sim.py* code, the program calculates cosine similarities between the test (swaps) and training (binders) sequences by using different weightings of the CDR regions, similarly to how it was done for the TBCRalign model.

Initially, the code obtains binder-values by matching the raw_index and current peptide with the raw_index and peptide_x columns respectively, in the full-sequence dataset.

In the code, each peptide is separately processed and thus multiple files are created based on different CDR combinations. The results from the ESM-executing programs were loaded, and the embedding vectors are used in three distinct ways to calculate the similarity:

Firstly, an individual CDR3 analysis is made, where CDR3α and CDR3β are processed separately, calculating the maximum cosine similarity between each test sequence and all training sequences.

Secondly, a combined CDR3 analysis is made, in which the training sequence that results in the highest combined similarity score when considering CDR3α and CDR3β together is found, and its score saved. In this step it is imperative to look at the CDR3α and CDR3β as a combined set, and to not simply add together the results from the highest individual analysis.

Finally, a full CDR analysis is executed, for which all six CDR regions (CDR1α, CDR1β, CDR2α, CDR2β, CDR3α, CDR3β) are considered. Both a Weighted and Unweighted analysis was made. For the Unweighted analysis, the similarity scores from all six CDR regions contribute equally to the final similarity score. However, for the Weighted analysis, the CDR3 regions are given four times more importance by multiplying their similarity scores by four before summing, while CDR1 and CDR2 regions maintain their original weights. In both approaches, the program locates the training sequence for which the similarity score is the highest across all regions to ensure that mixing across sequences to obtain the highest possible score does not occur.

For each analysis, the program generates two output files: one containing binder-only sequences and another containing all sequences. Each output file maintains the connection to the original sequences through the raw_index column and includes the binder status obtained from the full sequence dataset.

TCRa and TCRb sequences through ESM-2

Calling the file *optimized_single_chains.py* runs ESM-2 based on firstly the alpha chain sequence, specified by the column *TCRa* in the dataset. This is done by the specifying "TCRa" in the bash file *call_optimized_single_chains.sh.* Similarly, the same is done for the beta chain, just using "TCRa" with "TCRb", to let the program know which column to process. Additionally, the bash script loops through the 6 peptides that are defined at the beginning of the file and referenced in the sbatch commands:

python3 optimized_single_chains.py "$peptide" "TCRa"

python3 optimized_single_chains.py "$peptide" "TCRb"

The code saves the embedding-matrix with a fitting name in the parent directory:

final_esm_embedding_matrix_{processed_sequence}_data_{peptide}.npz

Following the ESM-2-embedding matrix being outputted, the needed rows for each sequence (CDR1, CDR2, CDR3) from both alpha and beta chains of TCRs should be extracted. This is done in the *call_sequence_extraction_single.sh* script, which executes the python script *sequence_extraction_single.py*, which has the main role of locating the start and end positions of CDR sequences within the TCR chain. By loading the embedding-matrices, the program filters the data to only be for the argument-specified peptide and then moves on to extract embeddings of the specific CDR regions of both the alpha- and beta chain. The program uses the knowledge that the original datafile contained columns indicating the sequences of each CDR regions. By matching e.g., the sequence of CDR3α within the full TCRα sequence, the location of those few amino acid-embeddings in the ESM-2 embedding-matrix is suddenly known, and only the specified rows are extracted and saved for each peptide for each weight.

After retrieving the needed embeddings, the *call_binder_split.sh* file is executed, to add a binder-column to the dataset, such that it is possible to distinguish binders from non-binders. To do this, the code takes the raw_index value of each element in the datafile and finds its match in the original datafile. For further ensuring the correct identifier, the program matches both peptide_x and raw_index in both files, and only takes the binder-value for the entry that matches both. This code creates two files:

{peptide}_{cdr_region}_binder.npz

{peptide}_{cdr_region}_full.npz

In which _binder.npz is a binder-only dataset, consisting only of true positives, and became the training data. Consequently, _full.npz is the swap-dataset, containing both binders and non-binders, thus becoming testdata.

Finally in the data processing, the cosine similarities between each entry in the testing set and every entry in the training set are calculated in the *call_cosine_similarity.sh.* It should be noted that if the entry has the same raw_index its value will be skipped. Three different methods of calculating the similarities are implemented to indicate the same weights that were used in the TBCRalign analysis.

Individual CDR3 analysis

In this section of the program, CDR3α and CDR3β were analyzed separately. In the extraction-code, a file for each peptide were made for CDR3α and CDR3β, which is now utilized. The program runs two approaches seperately: one that sums the embedding-matrix to get a single 1280-dimension vector for each entry, and one that takes the mean but proceeds with the same method. Taking advantage of the fact that every vector will now have the same length, the cosine similarity is

calculated between each test vector and all training vectors. The maximum similarity found for each training sequence is recorded and stored in a new file with a binder and raw_index column.

For an easier way of distinguishing the different weights in later analysis, the program saves its outputs with the names:

{peptide}_{CDR3a}_similarities.csv

{peptide}_{CDR3b}_similarities.csv

Combined CDR3 analysis

In the combined CDR3 analysis, the initial steps as in the individual analysis are the same. Data is loaded, summed, and the cosine similarity is calculated both for CDR3α and CDR3β, and these are added together. From then, the comparison is based on the highest combined score, rather than the individual highest scores of CDR3α and CDR3β across the test- and training data.

{peptide}_CDR3_combined_similarities.csv

Combined CDR analysis

This step of the program includes all 6 sequences, and thus uses all 6 embedding-files for each peptide. By creating two different scoring schemes, it is possible to create a weighted and unweighted similarity similarly to how it was done with TBCRalign. In the unweighted scheme, all regions contribute equally to the final score, whereas the CDR3 regions are weighted 4x higher in the Weighted scheme.

```
# Calculate unweighted sum
unweighted_sum =
    (current_similarities['CDR1a'] +
    current_similarities['CDR1b'] +
    current_similarities['CDR2a'] +
    current_similarities['CDR2b'] +
    current_similarities['CDR3a'] +
    current_similarities['CDR3b']
)

# Calculate weighted sum
weighted_sum =
    (current_similarities['CDR1a'] +
    current_similarities['CDR1b'] +
    current_similarities['CDR2a'] +
    current_similarities['CDR2b'] +
    4 * current_similarities['CDR3a'] +
    4 * current_similarities['CDR3b']
)
```

For each CDR region for each peptide, the embeddings are summed along their axes to create a single vector representation, both for the training and test data. Then, for each test sequence, the similarities for all six regions are calculated, and both a weighted and unweighted sum is determined. Finally, the highest overall score is stored, and the resulting output has ten columns:

Raw_index	Binder	Weighted_sum	Unweighted_sum	Max_sim_CDR1a
Max_sim_CDR1b	Max_sim_CDR2a	Max_sim_CDR2b	Max_sim_CDR3a	Max_sim_CDR3b

These datafiles are stored with the name {peptide}_all_CDR_combined_similarities.csv.

AUC

Finally, once all cosine similarities are calculated, AUC can be calculated and visualized on the data. The script *call_auc_analysis.sh* calculated AUC from all the weights for its specified peptide. Once the final peptide in the list has been computed, the program executes the visualization to create a plot with the same structure as the one created in the TBCRalign analysis.

Lastly, to wrap up the comparative element, the script *call_model_comparison.sh* was executed to create a comparison plot by taking the AUC values for each peptide for the best performing model for TBCRalign and ESM-2 and comparing them side-by-side and once again adding an averaged comparison as well. In this case, the weighted data was used for the comparison.