Generation probability of immune receptors and full sequence implanting in adaptive immune receptor repertoires

Olav Nybø

Supervisors: Lonneke Scheffer (main supervisor), Milena Pavlović, Geir Kjetil Sandve

Thesis submitted for the degree of
Master in Informatics: Programming and System Architecture
60 credits

Department of Informatics
The Faculty of Mathematics and Natural Sciences

UNIVERSITY OF OSLO

Spring 2023

Generation probability of immune receptors and full sequence implanting in adaptive immune receptor repertoires

Olav Nybø

Abstract

The adaptive immune system protects the body by remembering previously encountered antigens, so it can react more efficiently when encountering the same antigen in the future. The adaptive immune receptors, collectively called the adaptive immune receptor repertoire, on T-cells and B-cells, play a key role when recognizing antigens. Analyzing these immune repertoires gives us a deeper understanding of them and aids the development of diagnostic technologies. The immune signal is the set of features in the adaptive immune receptor repertoire that are associated with antigen binding or disease status. Simulating these immune signals allows us to have precise control of the ground truth of the immune signal when using the simulated data to assess machine learning models. One approach to simulating immune signals is to assume it will take the form of full sequences. Full sequence implanting simulates the effect of an immune event on the immune repertoire dataset by implanting one or more sequences many times into immune repertoires. Due to biases when generating immune receptors naturally, they have very different probability of generation. This generation probability can be computed. However, if a full sequence that is unlikely to be generated naturally is implanted many times in a dataset, this could make it an easily detectable outlier. This could produce unrealistic simulated data that can give false benchmarking results.

The full sequence implanting in immuneML, an open-source immune repertoire machine learning platform, can produce generation probability outliers. This thesis presents two implementations with different approaches to signal implanting strategy solutions for this generation probability outlier-problem, that will extend the full sequence simulation in immuneML. The distribution of how the generation probability of sequences relate to how often the sequences appear were analyzed in synthetic and experimental datasets to examine how the signal implanting strategies should behave and what parameters should be controlled by the user. Finally, a method that can detect candidates for these generation probability outliers was used to assess the new immune signal implanting strategies.

The new signal implanting strategies both successfully showed that they could implant the signal in such a way that the generation probability outlier-problem could not reliably be exploited. The two strategies have different strengths and weaknesses, and can both be used to simulate full sequence immune signals for different types of machine learning models.

Acknowledgements

I would like to thank my supervisors, Lonneke Scheffer, Geir Kjetil Sandve and Milena Pavlović, for an interesting thesis and for creating a good environment for me and the other master students. I'm especially grateful for my main supervisor Lonneke, for our weekly meetings, her constant availability on Slack and her very thorough feedback on my writing.

I also want to thank someone whose love and support helped keep me sane during these last few months.

Olav Nybø,
May 2023

Contents

List of Figures

List of Abbreviations

P_{gen}	Generation probability of immune receptor
BCR	B-cell receptor
TCR	T-cell receptor
AIR	Adaptive immune receptor
AIRR	Adaptive immune receptor repertoire
ML	Machine learning

Chapter 1

Introduction

1.1 The adaptive immune system

The adaptive immune system is a subsystem of the immune system that protects the body from pathogens. After an encounter with an antigen, an immunological memory is formed, which leads to a more effective response for future encounters with the same antigen [1]. The adaptive immune system consists of two key lymphocytes, B-cells and T-cells [2]. These have adaptive immune receptors (AIRs) which are used to recognize antigens; B-cell receptors (BCRs) for B-cells and T-cell receptors (TCRs) for T-cells. These receptors consist of 2 different protein chains made of sequences of amino acids: α and β chains (or the less common γ and δ chains) for T-cells, and heavy and light chains for B-cells [3] (Figure 1.1). An individual has many of these AIRs, collectively called the AIR repertoire (AIRR), which record past and ongoing adaptive immune responses. Sequencing allows AIRRs to be read at a DNA and amino acid level. The emergence of high-throughput sequencing have allowed the generation of large AIRR datasets much faster and cheaper than before, enabling greater AIRR analysis [4]. There are two major T-cell subtypes: helper T-cells, that activate other cells to start the adaptive immune system process, and cytotoxic T-cells, that recognize and destroy infected cells. B-cells are activated by helper T-cells that match the same germs [5].

When the adaptive lymphocytes encounter and bind to an antigen, these cells are activated and then multiply, where each new cell will inherit the antigen-specific immune receptor sequence. This process is called clonal expansion. Some of these activated cells will then mature to be used as long-term memory against antigen re-exposure, which allows the immune system to react more efficiently after having already encountered an antigen [1]. This creates an ever-evolving immune system that continuously adapt throughout a person's life according to their immune event history.

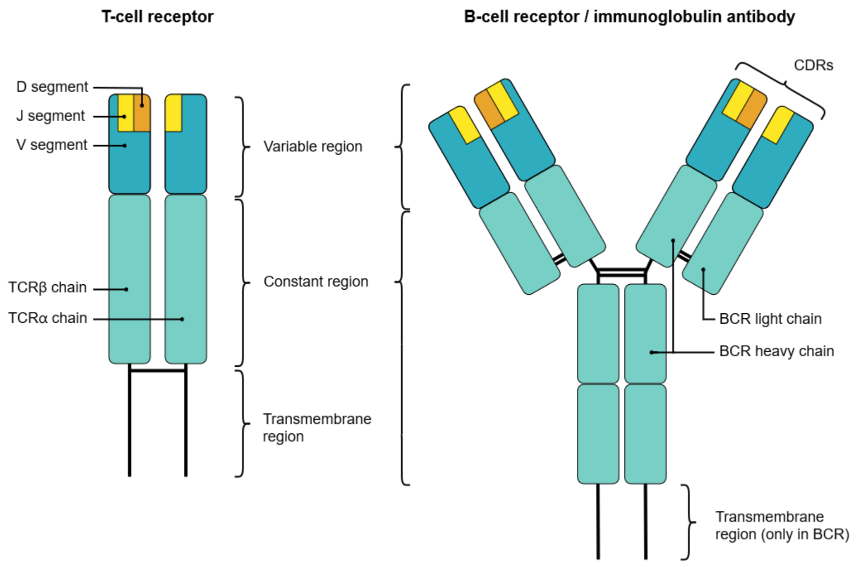


Figure 1.1: Structure of adaptive immune receptors. The variable region is created by a stochastic process called V(D)J-recombination. T-cell receptors have 2 chains, and B-cell receptors have 4 chains. The complementarity determining regions (CDRs) are where these receptors bind to their specific antigen. With courtesy: figure by Lonneke Scheffer [6], modified from: Backhaus [7], and Calis and Rosenberg [8].

1.2 The generation and variability of immune receptors

An individual human repertoire maintains $\sim 10^8$–10^{10} distinct AIRs of each type [9, 10]. A highly diverse range of AIRs play a vital role in an immune system with efficient control of a wide breadth of pathogens [9, 11]. The diversity of immune receptors is largely due to the stochastic process called V(D)J recombination. The immune receptors are created using the gene segments variable (V), diversity (D), joining (J) and constant (C) [12]. There exists multiple different versions of these gene segments in the germline DNA. Only the TCR β and BCR heavy chains contain a D segment [13], making them more variable than the other chains. For this reason, often TCR β and BCR heavy chains are chosen over the other chains in sequencing studies that use single chained data. This small set of genes that encode AIRs can produce 10^{15}-10^{20} possible sequences [9, 10]. First, the D and J gene segments are joined together, which is then joined with the V gene. The resulting sequence, along with the C gene, is the resulting DNA sequence of the adaptive immune receptor. Random insertions and deletions are formed in the junctions when the genes are spliced together, resulting in high diversity in the junctional regions [11, 14]. Another source of diversity comes from the pairing between the immune receptor chains (TCR α and β, BCR heavy and light) [11].

The binding site on immune receptors consists of six complementarity determining regions (CDRs). These are hypervariable regions located on the V region of each chain. There are three CDRs in each immune receptor's chain: CDR1, CDR2 and CDR3. CDR3 is the key determinant for antigen recognition, as it has by far the most sequence diversity of the CDRs [15]. The diversity of the CDR3 region is due to multiple separate factors. CDR1 and CDR2 regions are encoded just on the V region, but CDR3 straddles the V(D)J junction, leaving it more stochastic and variable [9]. It contains some of the V- and J-gene segments and, in the TCR β and BCR heavy chains, all the D-gene segment. The CDR3 region is also quite short, only about 15 amino acids long, making it effective for storage and computational reasons. CDR3 binds more, a majority of the amino acids that are in close contact with the antigen are located within the CDR3 [2, 15]. For these reasons, CDR3 is of most interest for sequencing studies [2].

1.3 Analyzing AIRR datasets with machine learning

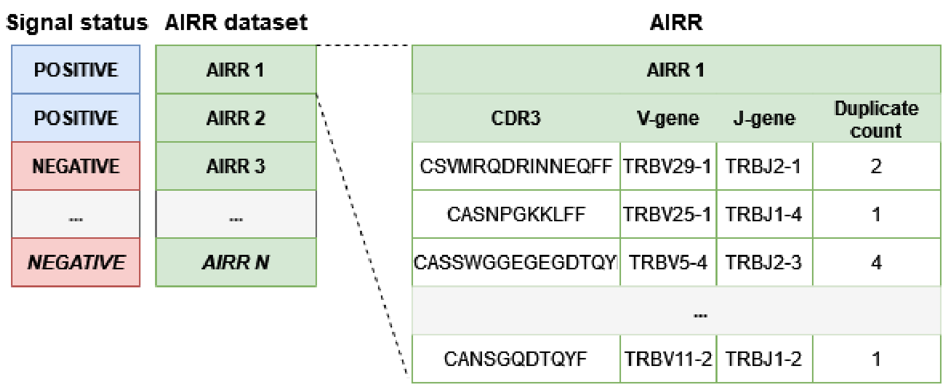
The immune signal (or just signal for brevity) is the set of AIRR features that are associated with antigen binding or disease status [16]. It is not known what this signal will look like, but here we will assume that immune signals could take the form of sequence patterns. There is little consensus on whether the signal takes the form of full CDR3 sequences or shorter sub-sequences [17]. For a specific antigen, these relevant signal patterns can occur in a very small portion of the AIR sequences, and may be as rare as one antigen-binding AIR per million lymphocytes in a repertoire [16]. This makes machine learning (ML) the ideal approach for classification of high-complexity AIRR datasets [18]. Machine learning is a field that uses algorithms that are able to learn discriminating patterns in large datasets. ML models train on data and use this experience to generalize to new similar data. An ML model can for example be trained with AIRR data where the disease status for each repertoire is known, and then classify new unlabeled repertoires as either sick or healthy. Classification means that the model categorizes the input into classes. The data used for creating a machine learning model is split into three parts: training data, validation data and testing data. First, the training data is used to fit the parameters of the model. The parameters of a model are the internal weights that are estimated when learning from the data. Then, the validation data is used to tune the hyperparameters of the model. The hyperparameters are tunable variables that change the behavior of the model and are not affected by training the model. This process is often repeated multiple times to test different hyperparameters. The same fixed data splits can be used each iteration, or one can use cross-validation, where different portions of the dataset are used each time. Finally, the test data is used to give an unbiased assessment of the model.

immuneML is an open-source, collaborative software ecosystem that implements each step of the AIRR ML process [18]. The main immuneML applications are sequence- and repertoire-based classification. Sequence-based classification is classification with respect to sequence-level labels, most commonly antigen specificity, but one could also learn other properties such as the extent to which the receptor sequence is observed in multiple individuals [16]. Repertoire-based classification focuses mostly on prediction of donor’s immune status (such as having or not having a disease, or past exposure to specific antigen) [16]. The main applications of these classification types are therapeutics discovery and immunodiagnostics respectively[16, 18]. For this thesis, only repertoire-based classification, as our focus is on distributions of sequences within repertoires.

The data typically used in AIRR ML is a huge oversimplification of biology. Immune receptors are in reality very complex 3D structures, but, for our ML purposes, we may only be interested in parts of this information. Often it may only be necessary to have the CDR3, ignoring TCRα and BCR light chains, and the V- and J-genes. The AIRR data that is relevant for this thesis is illustrated in Figure 1.2. The D-gene is fully present within the CDR3 sequence, but is often not explicitly annotated, as it is difficult to acquire from the sequence. The D-gene segment is often quite short and deformed on both ends by the random insertions and deletions in the junctions.

The count of each clonotype is affected by the type of sequencing. In this thesis, we will define a clonotype as all identical CDR3 with the same V-, D and J-genes. When sequencing individual cells (single-cell sequencing), the clonotypes can be counted as the number of cells with the same clonotype. Bulk-sequencing reads a population of cells, instead of sequencing cells individually [19]. In bulk-sequencing, the clonotype counts are the total number of specific clonotype reads in the sequenced cell population. This read count can be assumed to be approximately proportionate to cell count. However, bulk-sequencing is error-prone and only allow single chains to be sequenced; often only TCR β and BCR heavy, as these are the most variable. Two cells could have the exact same clonotype in one chain, but be completely different in the other chain. Sequencing only one chain means this information is lost [20]. While single-cell sequencing is more accurate, bulk-sequencing is much cheaper and more high-throughput, and is often adequate for large AIRR data. The distribution for sequence counts follow a heavy-tailed distribution, meaning that there is a large number of low-abundant clonotypes and some high-abundant clonotypes [21]. Some have suggested that the clonotype count in AIRRs follow a power-law distribution [21].

In this thesis, we separate between two ways of counting clonotypes in a dataset: count occurrences and clone occurrences. Count occurrences sum how many times the clonotype appears in each repertoire, and clone occurrences simply count how many repertoires the clonotype appears in. An illustrated practical example of these counting methods can be found in Figure 3.1 in section 3.3. These counting methods can produce different sequence frequency distribution behavior. A clonotype that appears in few repertoires, but has a very high duplicate count within these repertoires will likely have a high count occurrence, but a low clone occurrence. Counting clone occurrences can be useful when shared clonotypes between repertoires is of interest. Counting count occurrences could be useful for spotting high-abundant sequences that occur in few repertoires.



AIRR 1			
CDR3	V-gene	J-gene	Duplicate count
CSVMRQDRINNEQFF	TRBV29-1	TRBJ2-1	2
CASNPGKKLFF	TRBV25-1	TRBJ1-4	1
CASSWGGEGEGDTQY	TRBV5-4	TRBJ2-3	4
...			
CANSGQDTQYF	TRBV11-2	TRBJ1-2	1

Figure 1.2: Structure of AIRR data in immuneML. An AIRR dataset has a metadata file that contains immune signal status for each AIRR. Positive signal status means that the repertoire contains the specific immune signal that is in interest for the task the AIRR data is used in. However, AIRRs with negative signal status can also contain the patterns from the immune signal by chance. AIRRs can contain many more feature columns, but CDR3, V- and J-gene, and duplicate count (number of same clonotype in the repertoire) are the most relevant for this thesis.

1.4 Simulation of immune signals

Machine learning models vary greatly in performance, and therefore it is necessary to assess these models to provide a reference benchmark for novel models and provides realistic expectations of their performance. Large and varied benchmarking datasets are needed to assess these models, but there is a very limited amount of such datasets with adequate size [22]. An even bigger challenge is that signals occur chaotically and there is little understanding regarding their distribution, shape, size, or diversity [16]. An approach to solve these issues is to use simulated datasets, which include known ground-truth simulated disease signals [17]. Some argue that simulated and experimental data should be considered complementary and of equal importance for method assessment [22]. AIRR data immune signal simulation allows the ground truth immune signal to be known [22]. This means that we can be clear about our assumptions of the immune signal, e.g., whether the signal is a k-mer (sequence substring of length *k*) or a full sequence, as well as the occurrence rate of the signal. Knowing the ground truth in simulated data also allows testing whether the ML model has actually learned the implanted signal, for example by checking if the signal matches the sequences or k-mers that the model deem impactful. Recovery of a ground truth signal is an important method for testing whether the models are learning the intended signal pattern or some other unexpected pattern that coincidentally line up better with the provided data. One can simulate datasets with different assumptions of the immune signal, and compare ML models under these different conditions. This could give a better understanding for which models will work for what type of signal. In the future, we could have a better idea of what a realistic immune signal will look like. Knowing what models work under different assumptions will make us more prepared for this eventuality.

It is worth noting that it is not a problem if the machine learning model learns an unexpected pattern that does not 100% correspond to the immune signal, if it just unexpectedly lines up better. An example of this not being a problem could be if a k-mer that is deemed important by the model lines up with multiple different immune signals. However, it may be a problem if there is an unwanted and completely separate factor that influence the data. Confounders are variables that causally affect the cause and effect variables (in AIRR ML, these variables would be immune state and AIRR). It has been shown that age, sex, and genetic background may act as confounders in AIRR data [23]. Age has been found to be a confounder in Type 1 Diabetes (T1D) data [24]. For predictive purposes, confounders are not always an issue if the confounder distribution does not change from source to target population. This means that similar performance is expected if confounder distribution remains the same in the training and test data [23].

immuneSIM [25] and immuneML [18] are both programs that allow the simulation of immune signals by implanting k-mer motifs. immuneML can also implant full sequences. Programs like OLGA [14] and IGoR [26] can be used to simulate immune receptors that fit the distribution of how they are generated through V(D)J-recombination. These can be used to form synthetic AIRRs that follow a realistic immune receptor distribution. The synthetic AIRRs will be like naive AIRRs that has not gone through any immune events.

1.5 Generation probability of immune receptor sequences

Though V(D)J-recombination is a stochastic process, it has been observed that due to biases in the process, that some clonotypes are created more often than others [9]. These biases are believed to occur due to convergent recombination, meaning that the same nucleotide sequence can be produced by multiple recombination events [9]. Multiple events leading to the same outcome means a higher probability that certain clonotypes are generated through V(D)J-recombination. The existence of large AIRR datasets generated from high-throughput sequencing has allowed the development of algorithms [14, 26, 27] that can infer the generation probabilities of clonotypes (hereafter referred to as P_{gen} or generation probability). V(D)J-recombination involves multiple events that all have a set of possible outcomes. These events mostly consist of the selection of V-, D- and J-gene templates, the stochastic number of deletions at each end of the gene segments, and the random inserted sequences in the junctions of the gene segments [14]. The same nucleotide sequence can be produced by more than one recombination event. The P_{gen} of a nucleotide sequence is therefore defined as the sum of all possible recombination events that produce that specific sequence. Since multiple codons (set of three nucleotides) can encode the same amino acid, the P_{gen} of an amino acid sequence is the sum of all the generation probabilities of the nucleotide sequences that translate into that amino acid sequence [14]. Due to the diversity of immune receptors, the P_{gen} of a typical clonotype is generally quite low and can range from about 10^{-60} to 10^{-5} [14]. It is also possible for sequences to have a predicted P_{gen} equal to 0, meaning that the immune receptor sequence is deemed to be impossible to generate through V(D)J-recombination (the reasons for sequences with P_{gen} equal to 0 appearing in actual experimental data are further discussed in section 5.5).

Comparing repertoires has shown that there are many shared TCR β clonotypes between repertoires. Pogorelyy et al. (2018) [27] describes multiple reasons for this. The two most important are: (1) There are many shared clonotypes due to some clonotypes having a much higher chance of generation through V(D)J-recombination (convergent recombination). (2) There are many shared clonotypes due to receptors encountering the same antigen, thus experiencing clonal expansion that spreads the same (or a similar) clonotype (convergent selection). The paper reasons that convergently selected clonotypes should generally have a lower generation probability than convergently recombined clonotypes. This reasoning is the basis for vdjRec[27], a method which attempts to identify likely candidate sequences that have many shared clonotypes due to convergent selection, thus finding possible sequences containing the immune signal. The program identifies these sequences by looking at the correlation between each sequence's generation probability and the number of repertoires the clonotype appears in.

Chapter 2

Problem statement and objectives

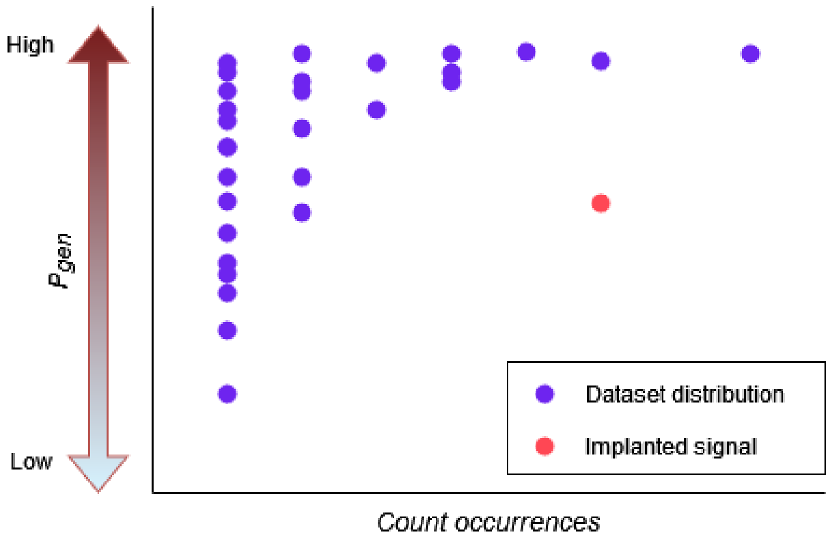


Figure 2.1: An implanted sequence could be easily spotted if it does not fall within the distribution of the rest of the dataset. Example of common distribution of P_{gen} (generation probability of a sequence) and number of count occurrences (total number of duplicate for a certain sequence in the dataset) per sequence in an AIRR dataset, and what an outlier implanted sequence might look like. Abundant sequences in the dataset will almost always have a high P_{gen} . Here, the implanted signal sequence is an outlier because its count occurrence is too high compared to its P_{gen} .

Simulating immune signals by implanting full sequences without a very high generation probability will often produce outliers that can be detected and exploited by the strategy used in programs like vdjRec. See Figure 2.1 for a visualization of what such an outlier might look like. Notice that even though there are sequences that appear more often than the implanted signal sequence, its low generation probability still leaves it very noticeable. For ML models to pick up on the pattern from the signal it must be implanted multiple times, which even with a very low implanting rate, will in all probability make the seed sequences from the signal appear noticeably more often than sequences with similar generation probability if the seed sequences do not have very high generation probability.

Benchmarking is the process of comparing tools and identifying the best-performing machine learning methods. If such simulated data is used to benchmark methods that use the same strategy as vdjRec, or a similar strategy, for example, based on sequence abundance and generation probability, this could give incorrect assessment of such methods. A machine learning model exploiting the P_{gen} outlier-problem could outperform other models when benchmarking with the simulated data, but could have much worse performance when classifying data from experimental samples. This is because simulated data with P_{gen} outlier-problem is a way in which the data is not like natural data. Natural data has clonotypes that are convergently selected, which can be called P_{gen} outliers, but these outliers are much less noticeable and exploitable than the simulated outliers can be. This is the reason vdjRec can only gather potential candidates that might be outliers due to clonal expansion. The outliers will still somewhat follow the P_{gen} to count occurrence distribution that the rest of the dataset follows. In natural data, there can be many more sequences that are convergently selected for other antigen that surround the specific antigen binding signal we are after.

This thesis presents two different implementations for full sequence signal implanting strategies that attempt to simulate immune signals that do not produce P_{gen} outliers in a way that programs like vdjRec can exploit. Both of the novel signal implanting strategies are implemented within the immuneML framework. immuneML already has an implementation for full sequence signal implanting, but this existing implementation does not attempt to solve this P_{gen} outlier problem. The new methods will extend this full sequence implanting. immuneML also has immune signal simulation based on k-mer implanting. These can both be used to test ML-models under different assumptions of how the immune signal will appear.

This thesis aims to: (1) illustrate how naive full sequence implanting strategies can create P_{gen} outliers and how this problem could be "exploited" by repertoire based ML models, (2) visualize the distribution of P_{gen} to count occurrences in different datasets, with and without implanted full sequence immune signals, (3) present two implementations of signal implanting strategies that will attempt to solve this P_{gen} outlier problem, (4) show that implanted sequences in the new signal implanting strategies can not be "found" by vdjRec.

Chapter 3

Methods

3.1 Tools

3.1.1 immuneML

immuneML (GitHub: https://github.com/uio-bmi/immuneML) is an extensible, open-source software ecosystem that implements each step of the AIRR machine learning process. It is accessible as a downloadable command line-tool, a Python package and as an internet application through a Galaxy web interface, and has an extensive documentation and tutorials both for users and developers [18]. Both signal implanting strategies and most of the tools implemented for this thesis were implemented withing or using the immuneML framework. The immuneML code library, as well as its additions presented in this thesis, are written in Python.

YAML-specification

When running immuneML as a command line-tool, the user specifies the wanted instructions and definitions using a YAML-file (examples can be found in Appendix A and B). YAML is a human-readable data-serialization language. The immuneML YAML-file is divided into two sections: definitions, where the user specifies analysis components, such as what dataset to use, signals to implant and what reports to use, and instructions, where the user specifies what kind of workflow to perform with the given analysis components. The types of instructions most relevant to this thesis are Simulation (implanting synthetic signals), ExploratoryAnalysis (analyzing datasets using encodings and reports) and TrainMLModel (training and estimating performance).

3.1.2 OLGA

OLGA (Optimized Likelihood estimate of immunoGlobulin Amino-acid sequences) [14] (GitHub: https://github.com/statbiophys/OLGA) uses dynamic programming to efficiently calculate the probability of generating a given clonotype. It was built to be a faster alternative to IGoR (Inference and Generation Of Repertoires) [26]. IGoR can only work with nucleotide sequences, while OLGA can work with both nucleotide and amino acid sequences. OLGA has default generative models of V(D)J-recombination for human T-cell alpha and beta, human B-cell heavy, and mouse T-cell beta, but users can also set their own custom VJ and V(D)J models. These models can also be used to generate immune receptor sequences. The program can be accessed through simple command line console scripts provided by OLGA or be imported as Python modules that can be incorporated in the user's own pipeline. All P_{gen} in this thesis are computed using OLGA. OLGA can compute the P_{gen} of both nucleotide and amino acid sequences, and can compute P_{gen} based on specific V- and J-genes.

Dynamic programming is a programming practice that involves splitting complex main problems up into multiple smaller sub-problems that are solved only once, but the answers are can be used multiple times. As mentioned in Section 1.5, the P_{gen} of a sequence is defined as all the sum of all the generative events that that can produce that sequence. These generative events consist of gene template selection (the random selection of V-, D- and J-genes) and the random insertions and deletions that occur in the VD and DJ junctions [14]. OLGA splits these into five segments (V, insertion segment, D, insertion segment, and J), and sums over the probabilities of all possible nucleotide sequences that are consistent with the given amino acid sequence, as well as the locations over the boundaries between each segment and the V-, D- and J-gene choices. These combinations of events are all collected in a matrix where the accumulative generation probability of an amino acid sequence is computed.

3.1.3 vdjRec

vdjRec[27] (GitHub: https://github.com/pogorely/vdjRec/ (this thesis uses version 0.1)) is a program that can find candidates for P_{gen} outliers using the P_{gen} outlier-exploit. The method was created to identify clonotypes that are likely to have high count occurrence due to selection from their response to a shared antigen, and not due to being likely to generate from V(D)J-recombination. vdjRec relies on the hypothesis that convergently selected clonotypes will generally have a lower P_{gen} than convergently recombined clonotypes. The method computes the likelihood of a sequence being overrepresented in a dataset compared to its P_{gen} , correcting for different repertoire sizes.

vdjRec uses a differently formatted files than the AIRR files used in immuneML. These files are exported when running the sequence generation probability distribution-data report.

3.1.4 Python packages

The implementations presented in this thesis are extensions of the immuneML framework. These were implemented on a branch (GitHub: https://github.com/uio-bmi/immuneML/tree/pgen_simulation) that was branched from the main immuneML branch with version 2.2.2. OLGA with version 1.2.4 was used in this thesis. All plots were created using plotly, and the parallelization of the P_{gen} computation was done using the multiprocessing package.

3.2 Datasets

3.2.1 Previously published experimental data

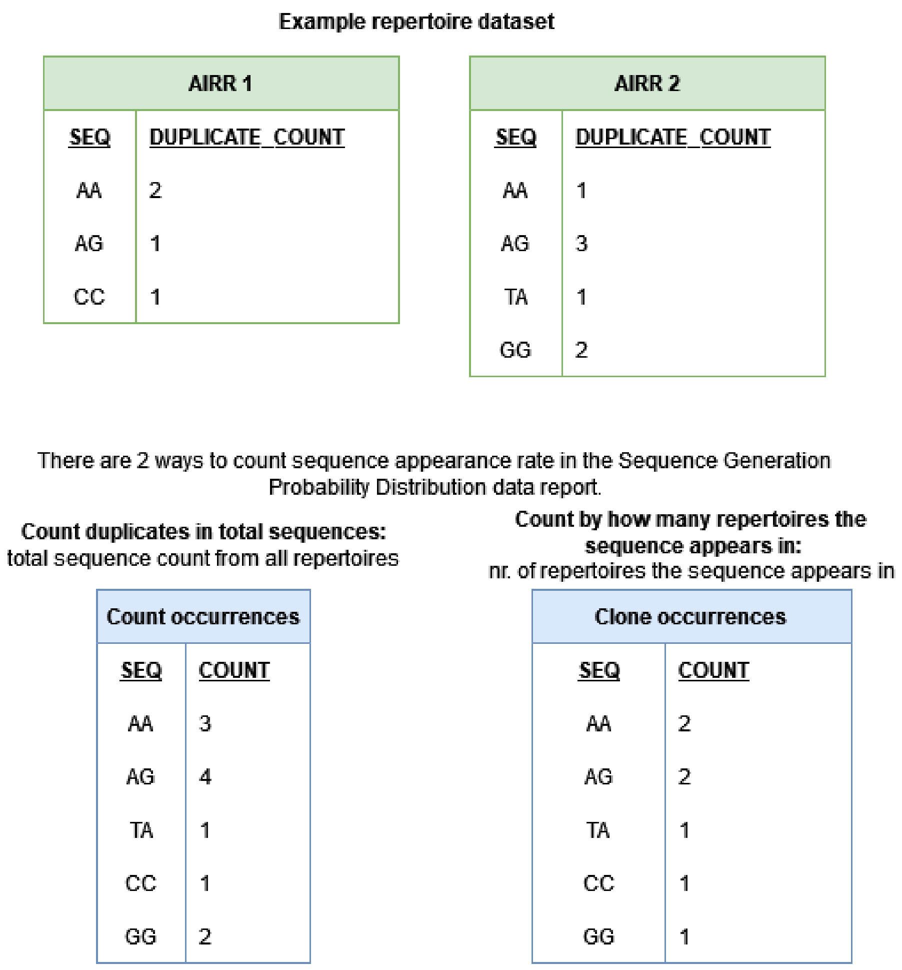
The main dataset used in this thesis is an experimental dataset from Emerson et al.[28]. Here, experimental data means actual biological data that have been derived from blood samples. The dataset contains 666 TCR beta sequence repertoires from humans with known cytomegalovirus serostatus. It was pre-processed by dropping all rows with missing values for the amino acid sequence and VJ-genes, as these are needed for computation of generation probability. Finally, 100 repertoires were then selected randomly and subsampled down to 100 000 sequences each to form the dataset used in this thesis. This leaves us with a total of 10 000 000 sequences between all 100 repertoires used. It would not be feasible to use the whole dataset because the computation of generation probability is quite resource-intensive and would take a very long time. This dataset will hereafter be referred to as the Emerson dataset or the experimental dataset.

3.2.2 Synthetic data

OLGA was used to generate a dataset to observe the behavior of a naive repertoire dataset, meaning a dataset that has not gone through any clonal expansion and is thus directly as if generated from V(D)J-recombination. This synthetic dataset (also referred to as OLGA dataset or simulated dataset) consists of 10 000 000 (100 AIRRs with 100 000 sequences each) human TCR beta sequences (the humanTRB generative model).

3.3 Visualization of distribution between generation probability and sequence occurrence rate

To observe how the distribution between generation probability and clonotype count occurrences, an immuneML data report was created. Data reports are a type of report that can be run within the exploratory analysis environment of immuneML, and shows some statistic or feature within a given dataset. This data report, called sequence generation probability distribution, takes an AIRR dataset, counts each clonotype, computes their P_{gen} , and visualizes the P_{gen} distribution for each clonotype count (or clone) occurrence. The non-implanted dataset sequences are displayed with violin plots and a line plot showing the lowest P_{gen} for each count occurrence column. This visualizes the space that an implanted signal sequence can be in without becoming an outlier. Implanted signal sequences are displayed as dots, so one can more easily see which sequences are implanted and if they are outliers.


Example repertoire dataset

AIRR 1

SEQ	DUPLICATE_COUNT
AA	2
AG	1
CC	1

AIRR 2

SEQ	DUPLICATE_COUNT
AA	1
AG	3
TA	1
GG	2

There are 2 ways to count sequence appearance rate in the Sequence Generation Probability Distribution data report.

Count duplicates in total sequences: total sequence count from all repertoires

Count occurrences

SEQ	COUNT
AA	3
AG	4
TA	1
CC	1
GG	2

Count by how many repertoires the sequence appears in: nr. of repertoires the sequence appears in

Clone occurrences

SEQ	COUNT
AA	2
AG	2
TA	1
CC	1
GG	1

Figure 3.1: There are 2 ways to count sequence occurrences using sequence generation probability distribution.

The computation of generation probability with OLGA can be very resource-intensive and time-consuming for larger datasets. The process was therefore parallelized using Python's multiprocessing-package. This was done by simply splitting the dataset evenly by a user specified number of processes, and applying the computation method in parallel to each part before concatenating the dataset parts after completion. Computing the P_{gen} of 100 000 sequences with 4 processes (on a computer with 4 cores), this parallelization achieved a speedup close to 4 (specifically for the P_{gen} computation-section).

The time complexity of the P_{gen} computation (as well as most other methods in this class) scales linearly. This means that it will in theory take about 10 times longer when using a dataset that is 10 times larger.

There are three input parameter unique to the Sequence Generation Probability data report: (1) *count_by_repertoire*: the user specifies whether to count sequences by count occurrences or clone occurrences (see Figure 3.1). (2) *mark_implanted_labels*: whether to mark the implanted signal sequence in a different color. (3) *default_sequence_label*: the name of the non-signal sequences in the plot legend (only if *mark_implanted_labels* is enabled).

Because the generation probability computation can be very time-consuming, every analysis in this thesis that requires P_{gen} computation of the whole dataset is done using sequence generation probability distribution. This means that, if a user needs a dataset with P_{gen} computed for more than one use, they only need to compute the P_{gen} once. The process for this data report is done in a series of steps, where the dataset is formatted as a pandas dataframe that is passed through a series of methods. This process is as follows: (1) load the dataset, which involves marking which (if any) sequences belong to which signal. (2) Count the sequences, either by count occurrences or clone occurrences. (3) Parallelized computation of the P_{gen} of the set of all clonotypes. (4) Plot the P_{gen} distribution. (5) Generate the occurrence limit P_{gen} range (for mutated sequence implanting). (6) Create tables used by vdjRec, which are formatted differently than the AIRRs used by immuneML. There are separate tables from samples (repertoires) and sequences. (7) Export plot, dataset with P_{gen} , occurrence limit P_{gen} range and vdjRec tables.

3.4 Integration with the immuneML platform

3.4.1 Signal implanter

The class signal implanter in immuneML handles going through AIRRs and "activating" each signal, so they are implanted into the correct AIRRs. Only the "diseased" AIRRs are handled, and immuneML exports an AIRR immediately after implanting the signal. This implementation complicated communicating information between different repertoires while implanting, which required extending the signal implanter-class with different solutions (further explained in Sections 3.5.2 and 3.5.3).

3.4.2 Signal implanting strategy

Signal implanting strategy is an abstract class that the signal implanting strategies classes described in this thesis will implement the repertoire-based implanting features of. All signals in immuneML must have a signal implanting strategy to control how the signal gets implanted.

Signal implanting strategy is given a dataset implanting rate and a repertoire implanting rate. These are both values between 0 and 1. The dataset implanting rate is how many repertoires to implant the sequence into. For example, for a dataset with 10 repertoires, a signal with dataset implanting rate 0.4 will be implanted in 4 repertoires. The repertoire implanting rate specifies how many times to implant the signal in a repertoire. For example, for a repertoire with 100 000 sequences, a signal with repertoire implanting rate 0.001 will be implanted 100 times in the repertoire. A signal can also be a set of multiple sequences. The implanting rates do not apply to each individual sequence, but the set as a whole. So for a repertoire with 100 000 sequences, a signal with multiple sequences and repertoire implanting rate 0.001, the set of sequences will be implanted 100 times in total in the repertoire.

3.4.3 General implementation design choices

The user can control whether the implanted sequences overwrite some sequences in the repertoires or to append the new sequences. Overwriting means the repertoire size stays the same. The repertoires that are being overwritten will be shuffled before implanting. Overwriting the last sequences without shuffling could be a problem if, for example, the sequences in the repertoires are sorted or there is already an implanted signal that is only at the end of the repertoires. In this thesis, all implanted sequences are appended to repertoires, to avoid distorting the distribution of the dataset.

The new classes' key features and methods are all unit tested. This allows for a simple, quick way to test that these methods behave as intended using small datasets with controlled components. Unit testing is also very useful for the future maintainability of these classes. Elaborate more on this

In immuneML, the setting for which part of the sequence to import (region type) is set to IMGT_CDR3 by default. This removes the first and last amino acids when importing sequences, as these are very conserved positions (often with the amino acids "C" for the first and "F" for the last position). OLGA needs these conserved positions to compute P_{gen} , so the new implanting strategies and the new data report needs them as well. Therefore, the region type IMGT_JUNCTION, which preserves the full sequence, must be used for these new implementations.

3.5 Designing novel signal implanting strategies

Two different ideas for how to solve the P_{gen} outlier-problem were implemented. See Figure 3.2 to see how these new classes fit into the existing simulation class structure of immuneML.

3.5.1 Naive full sequence implanting strategy

In this thesis, naive signal implanting means implanting full sequences without trying to account for the signal sequences becoming outliers. Figure 3.3 shows how naive full sequence implanting works. The signal is simply implanted according to the specified implanting rate in the "sick" repertoires.

3.5.2 Mutated sequence implanting

The idea behind mutated sequence implanting (Figure 3.2) is to split the seed sequences into similar sequences and implant them an appropriate amount of times for their P_{gen}

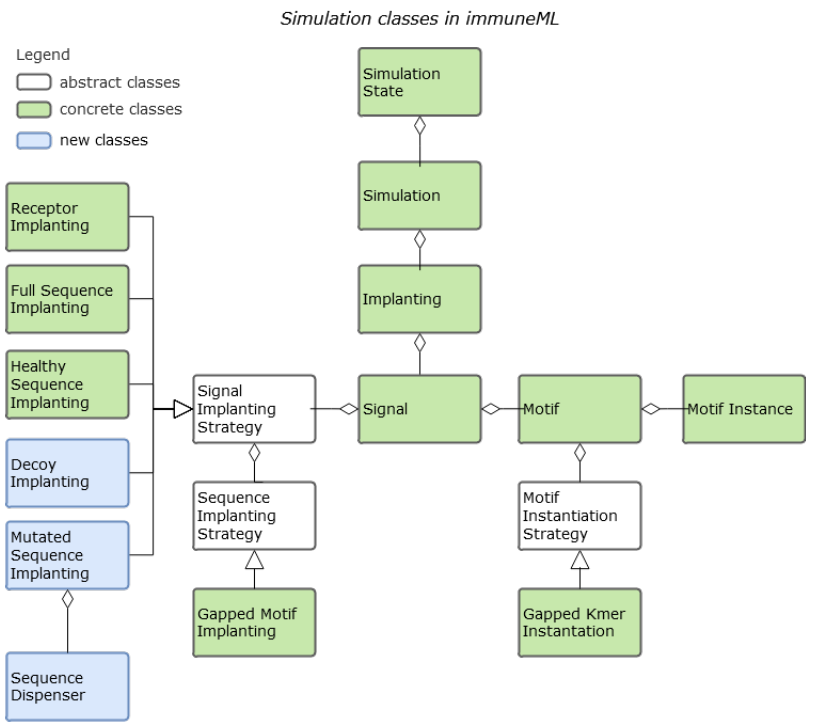


Figure 3.2: The mutated sequence implanting and decoy implanting classes implement the abstract signal implanting strategy-class. UML diagram showing the relations of the old and new simulation classes in immuneML. Mutated sequence implanting has a sequence dispenser for generating mutated sequences.

, thus preserving the signal while also making it adhere to the dataset's distribution. Here, seed sequences are the sequences given by the user that will be used as bases for mutation. These similar sequences are produced by mutating the original seed sequence(s), i.e., replacing one or more amino acids in the seed with different amino acids. For each time a signal sequence should be implanted, a mutated sequence is generated. The mutated sequences keep the properties such as V- and J-genes from the seed clonotype. The generation probability of the mutated sequence is computed, and the sequence gets assigned a limit to how many times it can get implanted in the dataset.

Since the main focus with this idea is to not let each mutated sequence be implanted too many times, so that it becomes an outlier, it was important to find a way to control how many times each generated mutated sequence has been implanted. This required communicating between different repertoires while implanting the signal, which as previously mentioned was made complicated by how the signal implanter-class handles the AIRRs. To solve this, a class called sequence dispenser was implemented (Figure 3.2). The sequence dispenser is shared among all repertoires, and given a set of sequence seeds, generates a random mutated sequence to implant and assigns a limit to how often it can be implanted according to its P_{gen}.

When mutated sequence implanting needs a sequence to implant, a seed sequence is selected and mutated according to the user's preference. The program then checks if this

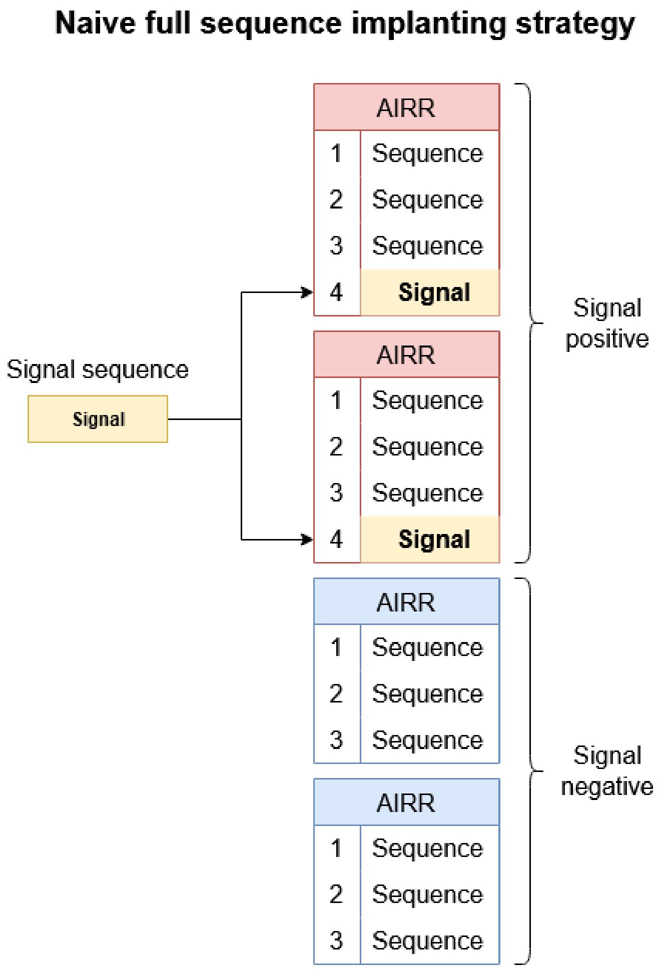


Figure 3.3: The naive signal implanting strategy simply implants one or more full sequences from the given immune signal into AIRRs, that are then marked as being positive for that immune signal.

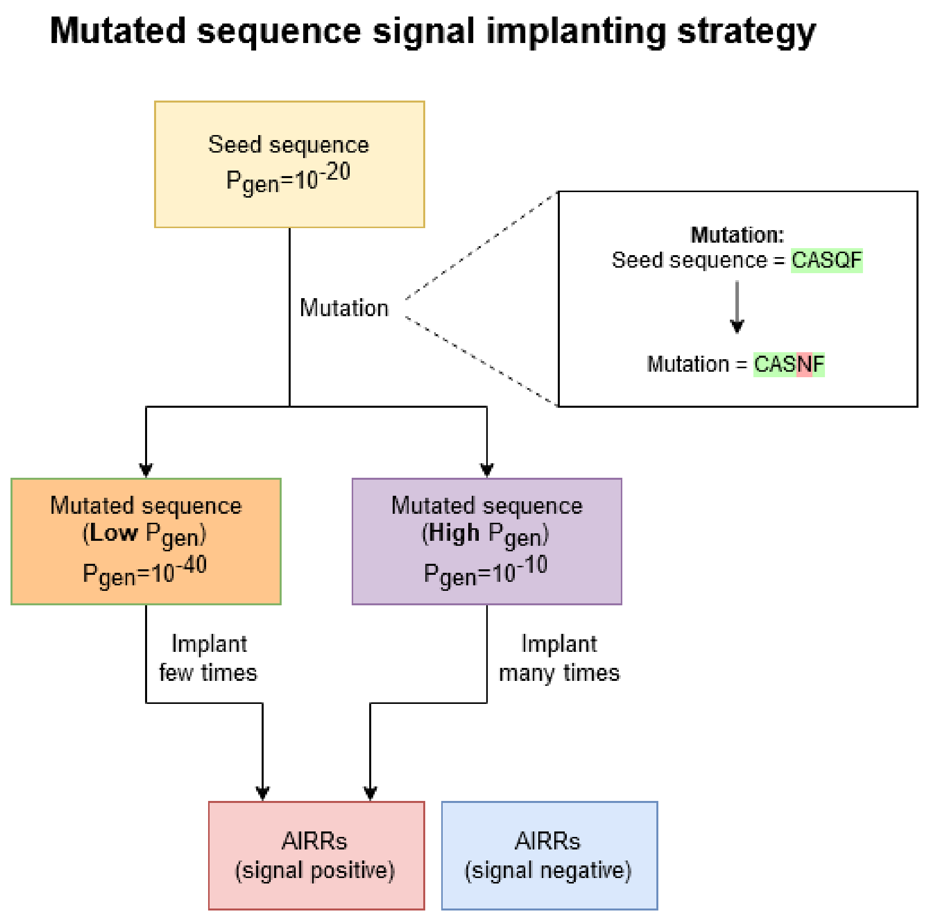


Figure 3.4: Mutated sequence signal implanting strategy implants mutated versions of the full sequences from the given immune signal. To fit the P_{gen} distribution in the dataset, the mutated sequences with low P_{gen} are mutated only a few times, while high P_{gen} mutations can be implanted many times (but might also only be implanted a few times).

mutated sequence is legal, meaning that it has not been implanted too many times or that the P_{gen} is 0 (meaning it would be impossible to generate through VDJ-recombination). If the mutated sequence is not legal, the class generates another one using the same seed sequence. If this loop does not find a legal mutated sequence to implant after a set number of attempts (which would ideally cover all possible mutations, but for performance reasons, the current implementation uses 200 max mutation attempts), the seed sequence is removed from the selection pool and tries the same procedure with the next seed sequence. If no seed sequences remain, the simulation is terminated and tells the user that no mutations were found for the given signal seed sequences.

Different datasets may have different P_{gen} and count occurrence-distributions. To control that the mutated sequences are implanted according to each dataset's distribution, the user can input the maximum count occurrence for P_{gen} ranges in the dataset. This occurrence limit P_{gen} range can be computed by running the sequence generation probability distribution-report. When a new mutated sequence is generated, this range decides the limit for how many times the mutated sequence can be implanted based on its P_{gen} . The occurrence limit P_{gen} range could for example look like:

```
occurrence_limit_pgen_range:
    1e-10: 2
    1e-8: 3
    1e-7: 4
    1e-6: 6
```

Here, mutated sequences with P_{gen} less than 1e-10 can only be implanted once. If the mutated sequence's P_{gen} is greater than 1e-10, then the max limit belonging to the P_{gen} value closest to the mutated sequence's P_{gen} is used. The count occurrence limit here is then drawn randomly from any integer between 2 and the max limit, inclusive. Using the occurrence limit P_{gen} range example above, sequences with P_{gen} closest to 1e-7 (such as e.g., 9e-8), will be assigned the limit 4. This actual max occurrence limit for this sequence will therefore be 2, 3 or 4. It is necessary to draw a random max limit instead of always using the exact max limit in case multiple mutated sequences draw the same very specific high limit. For example, if the max count occurrence limit for P_{gen} = 1e-6 is 50 instead of 6, there could be multiple mutated sequences that could be implanted exactly 50 times. This could create an artifact that ML models could possibly exploit instead of learning the intended signal pattern.

To make sure the duplicate count of a sequence is realistic compared to the rest of the duplicate counts (the number of duplicates for a clonotype within a repertoire) in the repertoire, rather than choosing an already existing duplicate count from the repertoire directly, the duplicate count of each mutated sequence is randomly chosen from a Gaussian distribution. The mean of the distribution is a randomly chosen duplicate count from the repertoire, and the standard deviation is half of the mean. This is to avoid multiple sequences having the exact same, very unusual, duplicate count. The

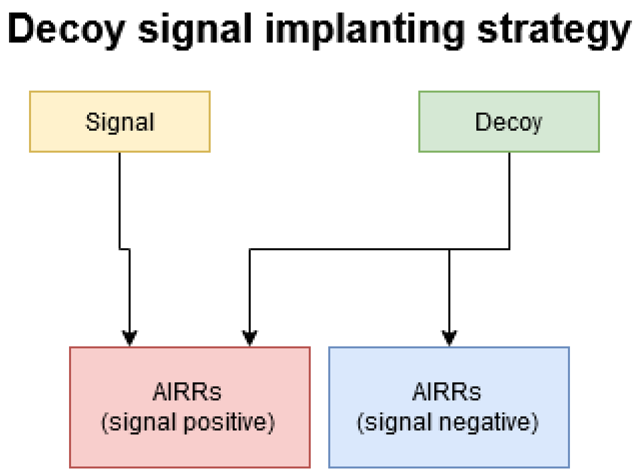


Figure 3.5: To hide the P_{gen} outliers from the signal, randomly generated decoy sequences are implanted in all AIRRs. If both signal positive ("sick") and signal negative ("healthy") repertoires contain P_{gen} outliers, then the P_{gen} outliers cannot be used as a way to classify repertoires.

duplicate count also takes into account the occurrence limit and does not implant more than is allowed.

The user can specify the mutation hamming distance, meaning how many positions to change the amino acid in the seed sequence when generating mutations. Hamming distance is the number of differences between two strings of the same length. The positions to mutate are picked randomly and could therefore be the same, meaning the real hamming distance between the seed and the mutated sequences has a chance to be lower in practice than the given mutation hamming distance. The amino acids to swap the positions to are also picked randomly from the amino acid alphabet, meaning they could be the same as the one already in the position.

Due to the V and J genes being more conserved than the D gene, mutating the middle of a sequence generally give much higher P_{gen} than mutating the start or end of a sequence (see Section 4.2). For this reason, the user decides what positions in the seed sequence can be mutated and how high the possibility of mutating different positions are. For example, if mutation position possibilities given is {4: 0.3, 5: 0.4, 6: 0.3}, then the only positions with index 4-6 can be mutated and there is a slightly higher chance of mutating position 5.

3.5.3 Decoy implanting

Decoy implanting (Figure 3.2) implants outliers in all repertoires, effectively masking the outliers produced by the signal. If both diseased and healthy repertoires have P_{gen} outliers, the ML model cannot exploit the outlier artifact to predict disease status.

The signal implanter-class only handles the AIRRs that will be introduced to the immune signal. Decoy sequences are implanted in both "sick" and "healthy" AIRRs.

This required altering the signal implanter-class, so it could handle all repertoires.

The user is given the option of how many decoys to generate, as well as the dataset implanting rate and repertoire implanting rate for each decoy. The same implanting rates are used for all decoys. These implanting rate options are the same as what the user must input when implanting the signal, so the decoy implanting rates default to being the same as the signal's implanting rates. To avoid all the decoys being implanted exactly the same amount, the program includes some randomization in the process. The duplicate count within each repertoire is chosen randomly from a Gaussian distribution, where the mean is equal to repertoire implanting rate times the repertoire size. The dataset implanting rate is used as the chance that a decoy will be implanted in the current repertoire.

The decoy sequences are generated using OLGA's generative model that uses the default model (organism, receptor type and chain type) that is inferred from the dataset.

Chapter 4

Results

4.1 Distribution of generation probabilities and count occurrences in AIRR datasets

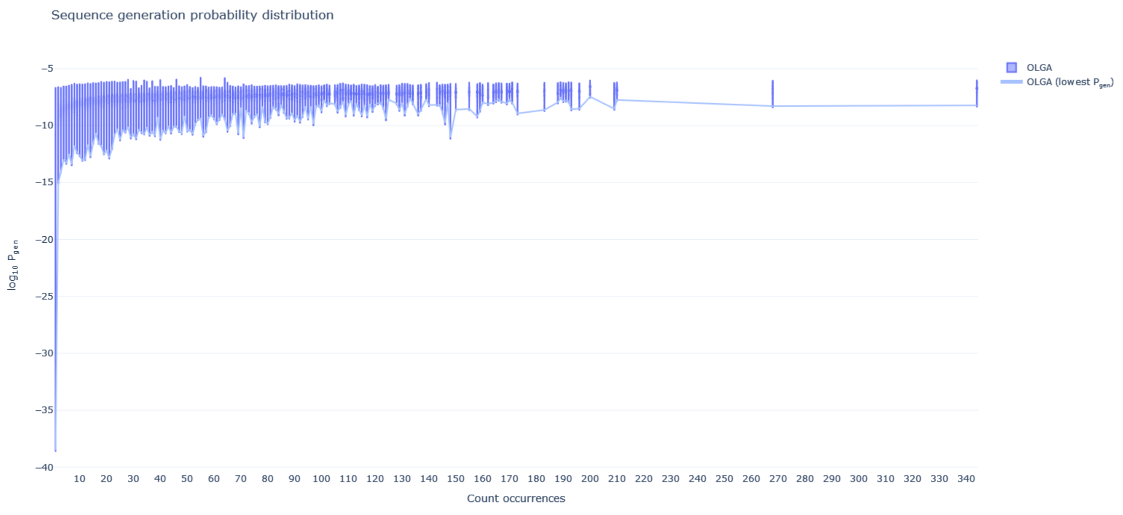
A series of experiments were made to observe how generation probability compares to how often sequences occur in an AIRR dataset. The data report Sequence Generation Probability Distribution (described in Section 3.3) was used to visualize the distribution in synthetic and experimental data.

4.1.1 Simulated data

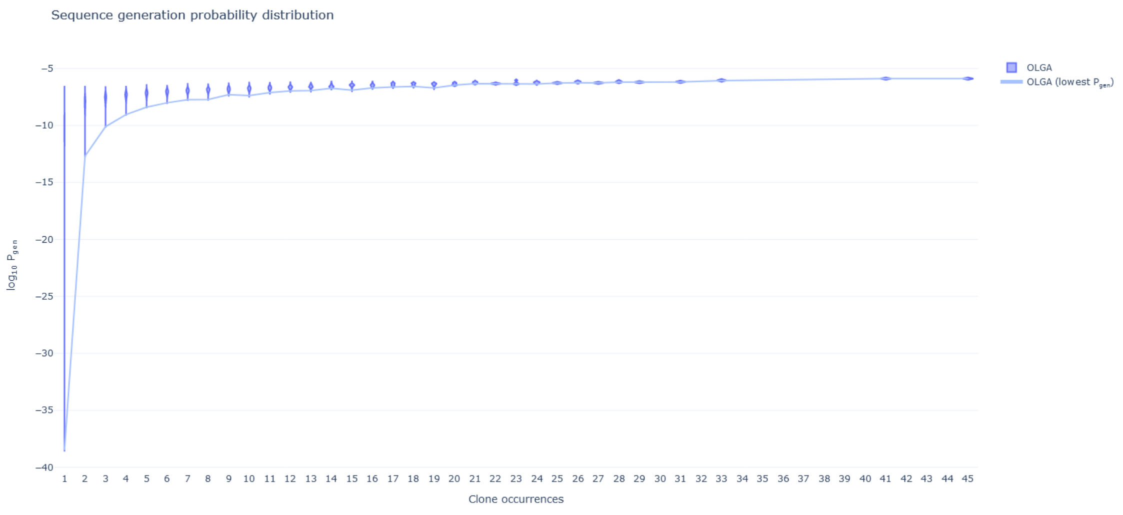
In Figure 4.1 we see the distribution of how often each sequence appears in simulated data compared to how high each sequence's P_{gen} is. This dataset is generated based on V(D)J-recombination statistics, and is therefore expected to be like an AIRR dataset that has never encountered an antigen, thus never experiencing clonal expansion. The figure shows a clear trend in the distribution, where sequences that appear more often in the dataset generally have a very high P_{gen} . This effect is stronger, meaning that the lowest P_{gen} is higher, for sequences with higher count occurrences. The majority of the sequences only appear once in the dataset. About 90 percent of the simulated sequences only appear once or twice in the whole dataset.

Counting clonotypes based on their count occurrences (Figure 4.1a) and their clone occurrences (Figure 4.1b) seem to both follow a similar heavy-tailed distribution. However, counting by clone occurrences seem to lead to a smoother and less noisy distribution.

P_{gen} distribution of simulated data



(a) Count sequences by count occurrences (sum of a sequence's count from all repertoires)



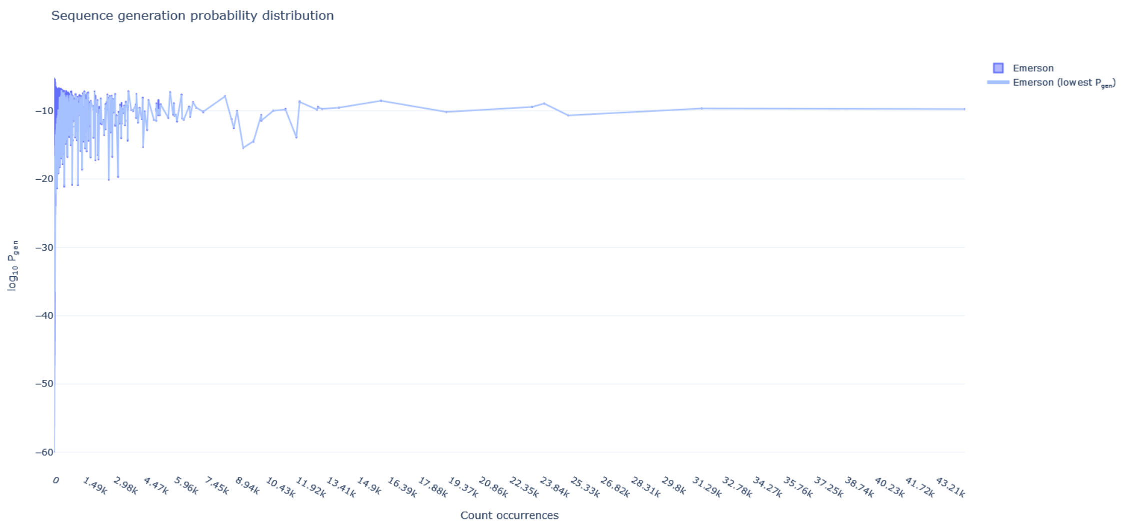
(b) Count by clone occurrences (number of repertoires where the sequence appears)

Figure 4.1: Synthetic data P_{gen} distribution. 10 000 000 sequences generated with OLGA, showing how each sequence's generation probability relates to how often the sequence occurs in the dataset.

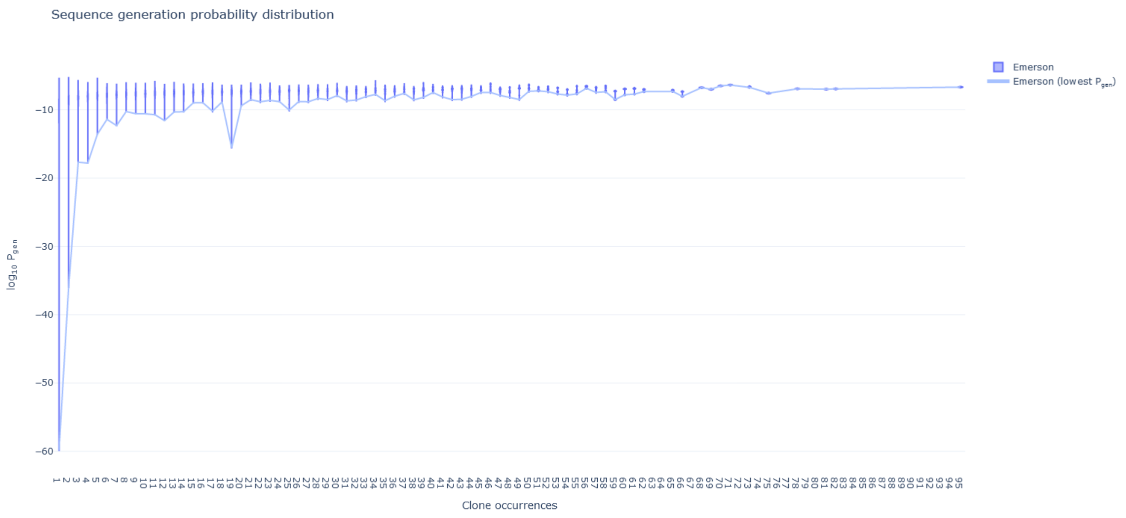
4.1.2 Experimental data

Figure 4.2 shows the distribution of how often each sequence appears in experimental data and how high each sequence's P_{gen} is. Similarly to the simulated data, the plots seem to follow a heavy-tailed distribution, i.e., most of the sequences appear once or twice and sequences that appear more often generally having a high P_{gen} . However, the distribution is generally less smooth and have more outliers than the simulated data P_{gen} distribution. About 60% of the clonotypes in the experimental data only appear once or twice in the entire dataset. Counting by count occurrences (Figure 4.2a) shows some very highly abundant clonotypes, where the most common clonotype appears about 43 000 times in the dataset. Counting by clone occurrences shows that some sequences appear in 95 of the 100 AIRRs. About 75 000 (0.75%) of the experimental data have P_{gen} equal to 0. See Section 5.5 for discussion around this.

P_{gen} distribution of experimental data



(a) Count sequences by count occurrences (sum of a sequence's count from all repertoires)

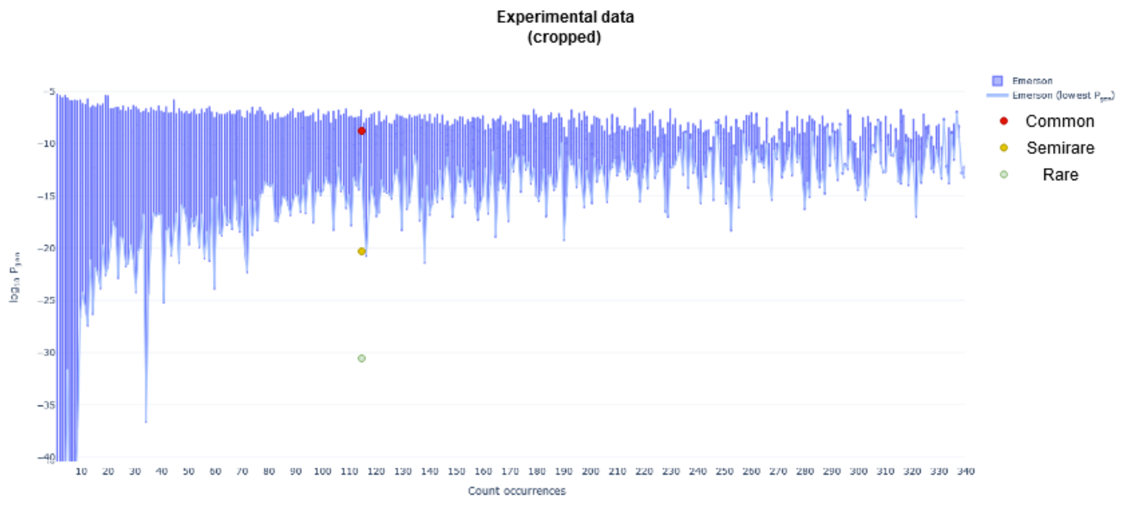


(b) Count by clone occurrences (number of repertoires where the sequence appears)

Figure 4.2: Experimental data P_{gen} distribution. 10 000 000 sequences from Emerson et al.[28], showing how each sequence's generation probability relates to how often the sequence occurs in the dataset.

4.1.3 Different datasets with rare and common implanted seeds

Figure 4.3 shows what implanted full sequences with different P_{gen} might look like in comparison to experimental and synthetic data. There are different P_{gen} distributions for different datasets. An implanted sequence may be an outlier in one dataset, but might fall within the commonly observed P_{gen} -distribution of another dataset. This means that the specific parameters when implanting in a dataset must be tailored to that dataset.



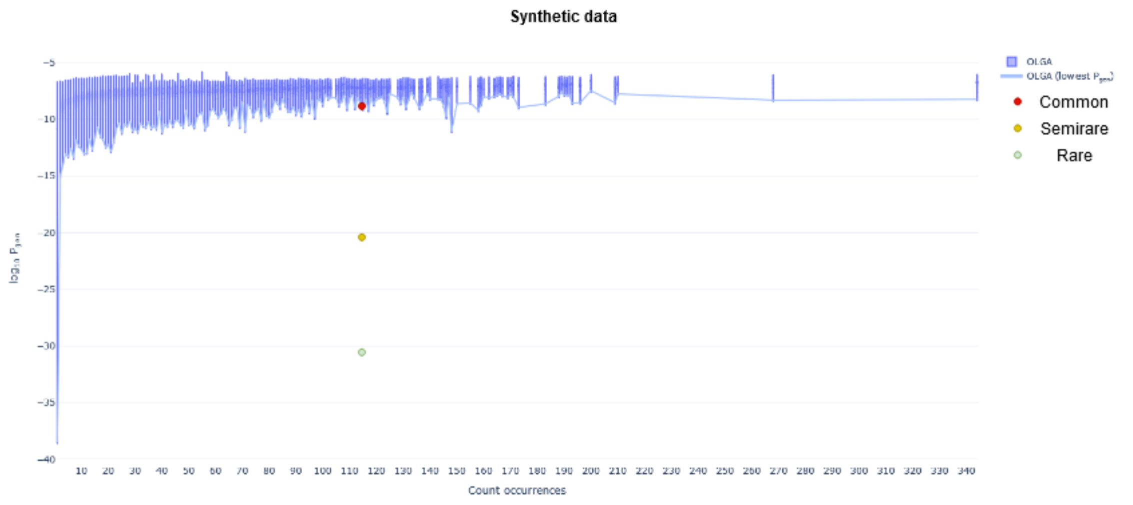


Figure 4.3: Illustration of rare, semirare and common sequences implanted in experimental (from Figure 4.2) and synthetic (from Figure 4.1) data. Experimental data has been cropped on both axis to have the same ranges as the synthetic data. The common sequence fits the distribution of both datasets. Semirare is an outlier in synthetic data, but not in experimental. Rare is an outlier in both datasets.

4.2 The effect of mutation on generation probability

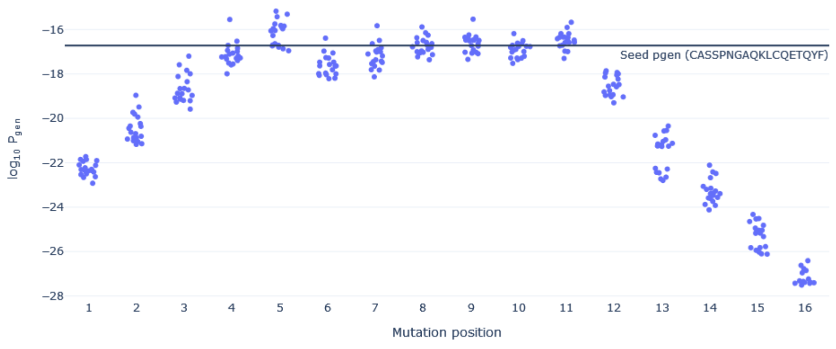
One way to implant signals without the implanted full sequences appearing more often than their generation probabilities allow compared to the rest of the dataset is to mutate the given seed, compute the generation probabilities of these new sequences, then implant some of them in the dataset with an implant rate that matches the probability (more details on mutated sequence implanting in Section 3.5.2). Here, mutation will be done by changing one or more positions in the seed sequence into randomly selected amino acids from the amino acid alphabet. To see how the generation probabilities are affected by mutating a sequence, a few different seed sequences were mutated to explore how their generation probability distribution were affected.

4.2.1 One mutated position

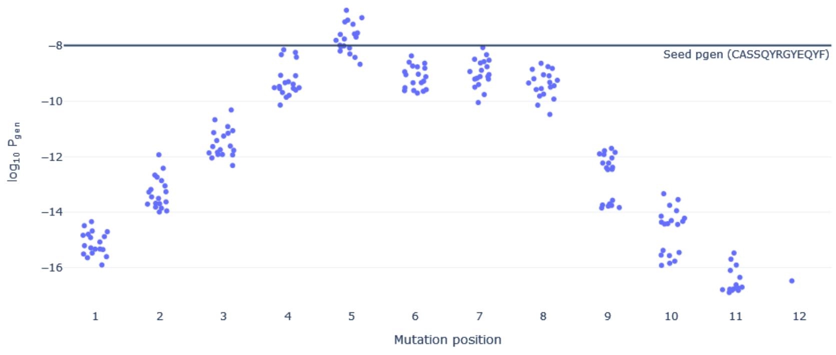
First the sequences were mutated once, meaning that only one position in the seed sequence were swapped with a different amino acid, i.e., the hamming distance between the original seed and the mutated sequences is equal to 1.

Mutating different positions in CDR3s (Figure 4.4) shows that mutated sequences with mutations in the middle generally have a higher P_{gen} than mutations in the start or end. There are also two high-P_{gen} peaks in the figures. The mutations with P_{gen} equal to 0 are not shown in the figure, as the axis is logarithmic. Mutations in the very conservative first and last position in the CDR3s almost always result in P_{gen} equal to 0.

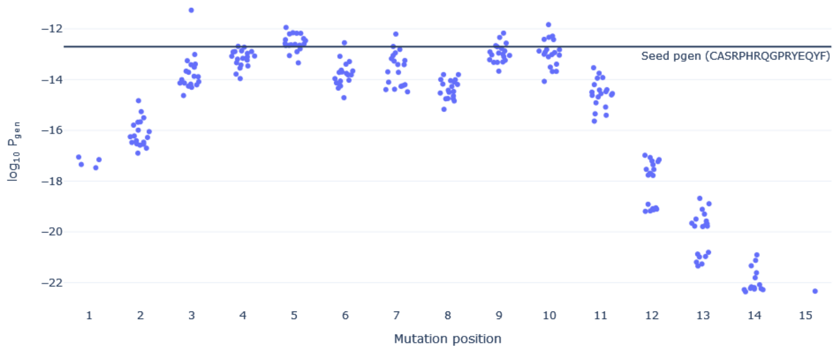
While the figure shows that the P_{gen} distribution of mutations are similar for different seed sequences, the lengths of the different segments (start, peak, middle, peak, end) differ a bit. This could perhaps be affected by the length of the seed sequence. The P_{gen} ranges are quite different and might be affected by the P_{gen} of the original sequence.



(a) CASSPNGAQKLCQETQYF



(b) CASSQYRGYEQYF

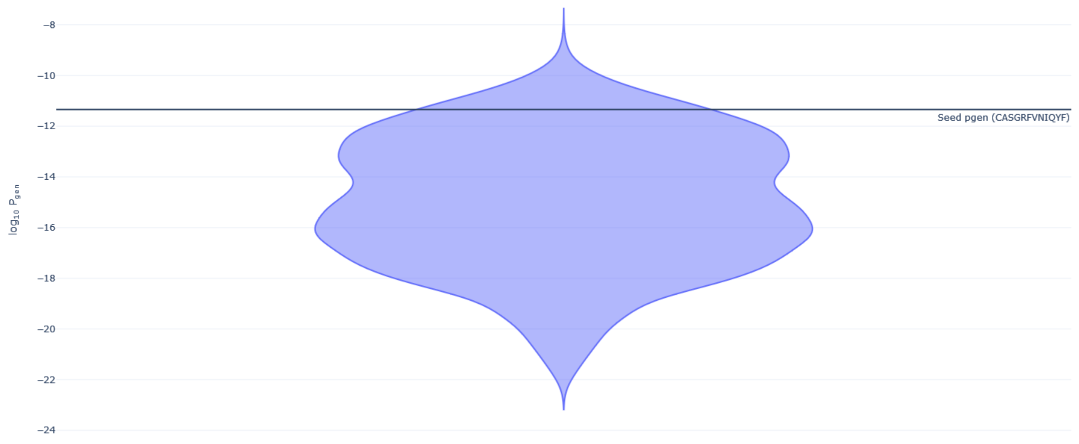


(c) CASRPHRQGPRYEQYF

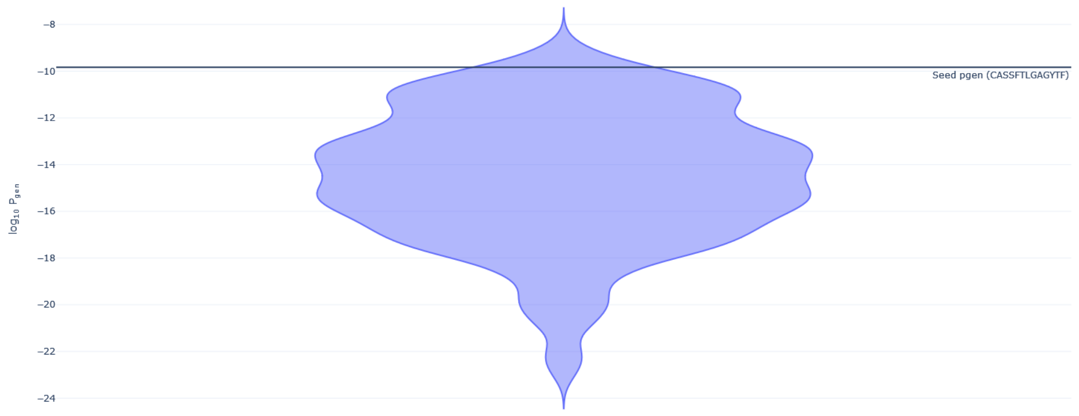
Figure 4.4: The effect of mutating different positions in three sequences generated randomly with OLGA. Each position in the sequences were swapped with each amino acid in the amino acid alphabet, meaning that these are all possible mutations with hamming distance 1 from the seed sequence. Mutated sequences with P_{gen} equal to 0 are not shown.

4.2.2 Two mutated positions

Mutating 2 positions in a sequence at a time (Figure 4.5) shows that the majority of the mutated sequences generally have a lower P_{gen} than the original seed, and only about 0.02-0.03% of the mutations produce a higher P_{gen} than the original sequence. A large portion of the mutations have P_{gen} equal to 0.



(a) Sequence: CASGRFVNIQYF.
45% of mutations have P_{gen} equal to 0.
0.03% of mutations P_{gen} are higher than the seed sequence's P_{gen} .



(b) Sequence: CASSFTLGAGYTF.
32% of mutations have P_{gen} equal to 0.
0.02% of mutations P_{gen} are higher than the seed sequence's P_{gen} .

Figure 4.5: Mutating amino acids in 2 positions at a time in a sequence produces a wide range of generation probabilities, but the majority of the generation probabilities are lower than the original sequence and many are 0. The horizontal line shows the generation probability for the original sequence.

4.3 Spotting P_{gen} outliers implanted in datasets using the new signal implanting strategies

A series of tests were conducted to see if the new signal implanting strategies solve the P_{gen} outlier problem. Signal sequences were implanted into the Emerson dataset (see Section 3.2.1) using decoy implanting and mutated sequence implanting. In addition to this, the same sequences were implanted using a naive full sequence signal implanting strategy as a control test to test whether outliers can be recovered using P_{gen} and count (or clone) occurrence. Here, naive signal implanting means implanting full sequences without trying to account for the signal sequences becoming outliers.

The implanted signal was the same for all implanting strategies and consisted of 5 clonotypes (see Table 4.1). These clonotypes were generated using OLGA, and were selected to have a wide range of P_{gen} .

The datasets with implanted signals were analyzed using vdjRec to test whether the implanted signal sequences can be retrieved using only their P_{gen} and count occurrence. These tests use the pval_post metric from vdjRec to judge which sequences are likely to be P_{gen} outliers or not. pval_post is the P-value that the clone occurrence pattern of a sequence is explained by convergent recombination alone. This means that sequences that are abundant due to clonal expansion or due to being implanted many times in the dataset might have a very low P-value. Sequences with high P-value are sequences where the clone occurrences match the computed P_{gen} , and therefore have an appropriate abundance for being produced solely through V(D)J-recombination.

There were sequences in the dataset with P_{gen} 0. Before passing the data to vdjRec, the sequences with P_{gen} 0 were removed from the dataset. Even though these sequences are outliers, they are not relevant for the P_{gen} distributions used here. Sequences with P_{gen} 0 is discussed further in Section 5.5.

AIR amino acid sequence	V-gene	J-gene
CASRSGNEKLFF	TRBV7-8*02	TRBJ1-4*01
CASSWIEPHIKGEGQTYEQYF	TRBV6-3*01	TRBJ2-7*01
CASSYLVAENSGANVLTF	TRBV6-5	TRBJ2-6
CASSVMCQDRVSSYEQYF	TRBV6-4	TRBJ2-7
CATGGFFSYEQYF	TRBV7-7*03	TRBJ2-7*01

Table 4.1: Clonotypes that were implanted in the datasets, that together form the simulated immune signal.

Parameter	Value
Number of AIRRs	100
AIRs per repertoire	100 000
Dataset implanting rate	0.5 (50%) (50 repertoires)
Repertoire implanting rate	0.001 (0.1%) (100 implanted sequences per repertoire)

Table 4.2: Parameters and dataset properties that were used in all simulations.

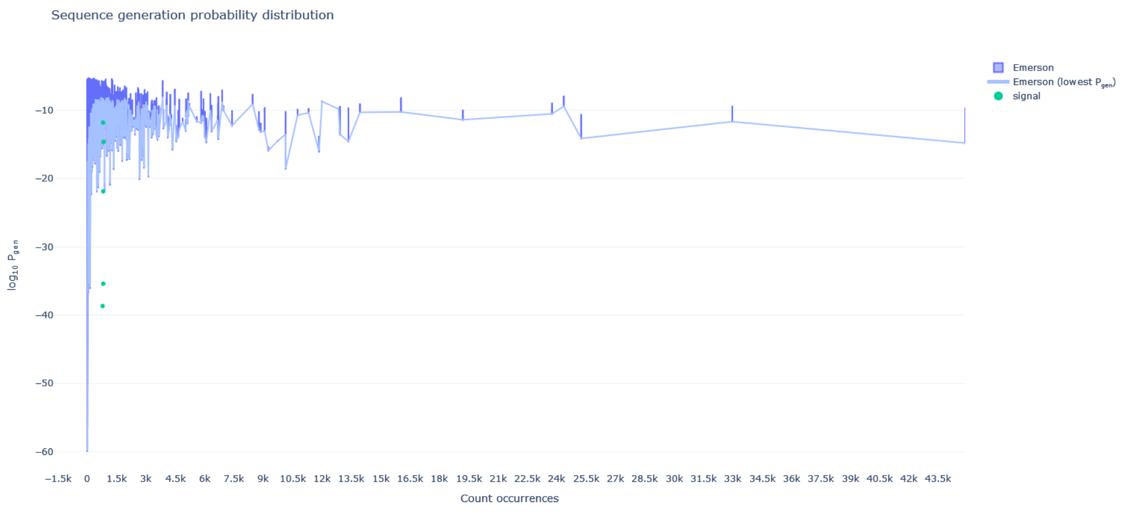
4.3.1 Naive full sequence implanting

Implanting immune signals naively is done here by using the decoy implanting strategy, but with 0 decoys. The full yaml-specification can be found in Appendix A.3.

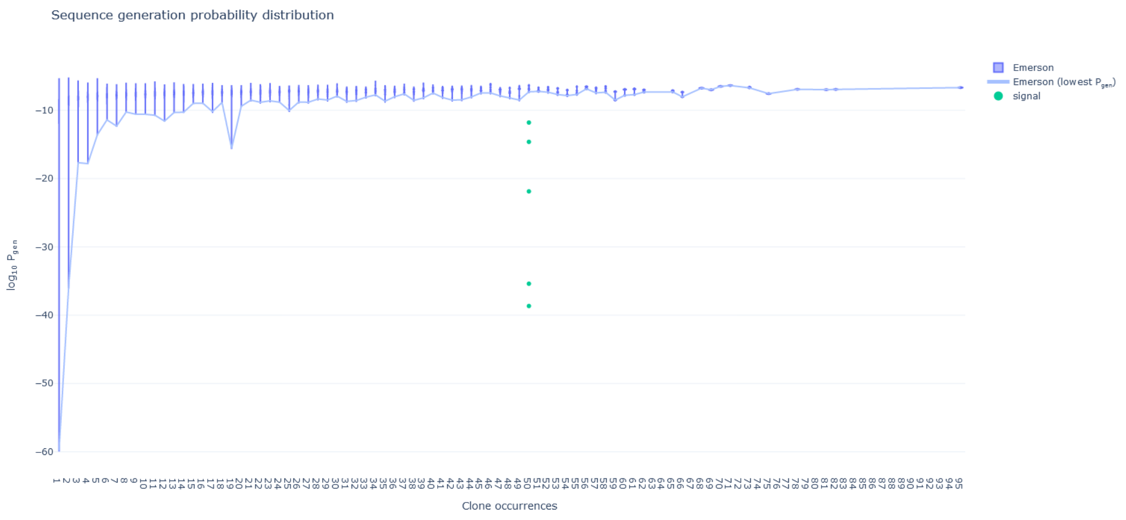
As shown in Figures 4.6, it is possible to spot the full sequences that are P_{gen} outliers in a distribution showing the full experimental dataset. It is easier to spot the implanted sequences when counting by clone occurrence (Figure 4.6b) instead of counting by count occurrence (Figure 4.6a).

Using the vdjRec method to find these outliers shows that the full sequences from the signal has a low P-value compared to the rest of the dataset (Figure 4.7), however the outliers are not as distinct as in Figure 4.6. The experimental dataset mostly has sequences with a high P-value, but there are some low P-value sequences. There are some non-signal sequences from the dataset that have similar or lower P-value than the signal sequences.

Naive full sequence implanting: P_{gen} distribution



(a) Count by count occurrences



(b) Count by clone occurrences

Figure 4.6: Full sequence signal implanted with a naive implanting strategy in experimental data. Showing the P_{gen} distribution compared to each sequence's count occurrence makes it possible to spot sequences with low P_{gen} that are implanted many times.

4.3.2 Decoy implanting

The decoy implanting generated 1000 decoys to hide the P_{gen} outliers from the signal. The decoy implanting rates were the same as each sequence in the signal. As the immune signal was formed by five sequences, the repertoire implanting rate had to be divided by five. See the full yaml-specification in Appendix A.2.

Both Figure 4.8 and 4.9 shows that the P_{gen} outliers are a little hidden by the decoy sequences. The reasons as to why these are not hidden more are discussed in Section

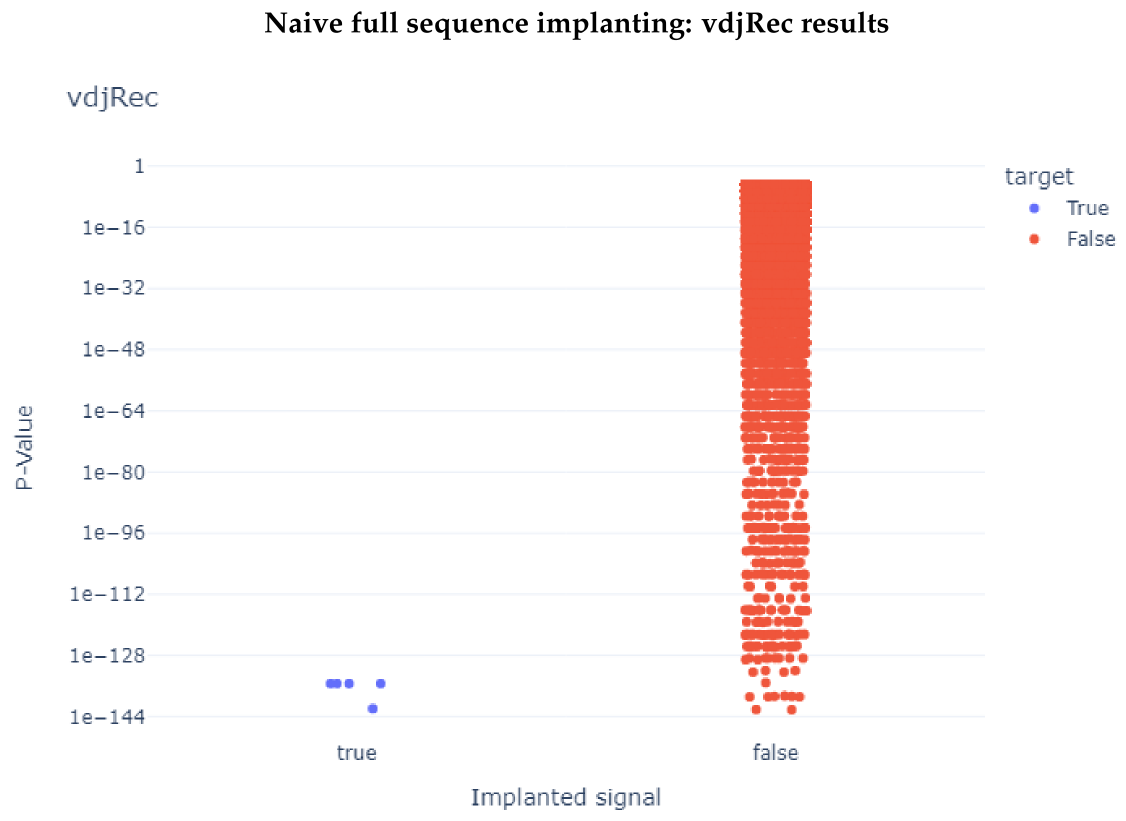
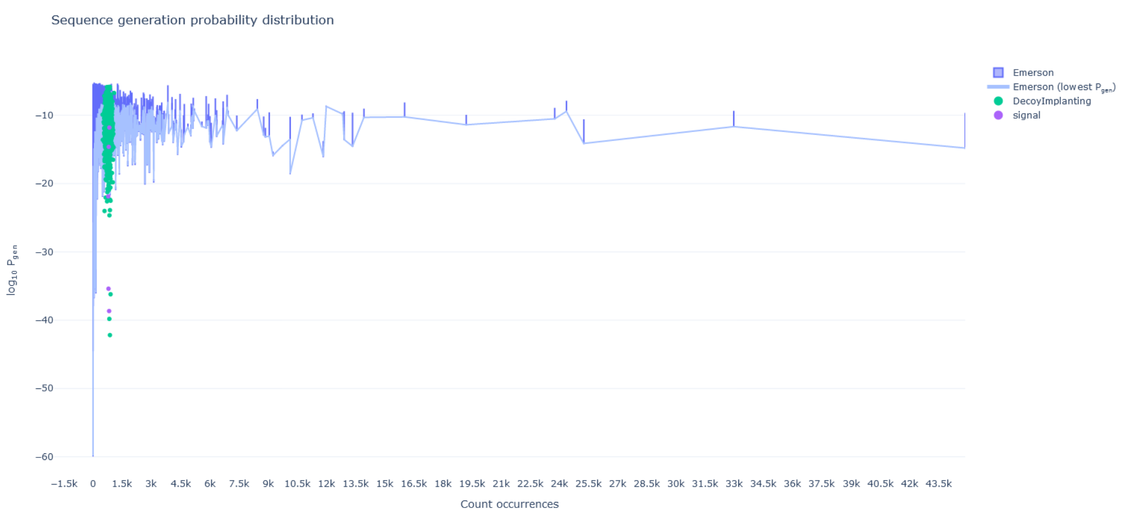


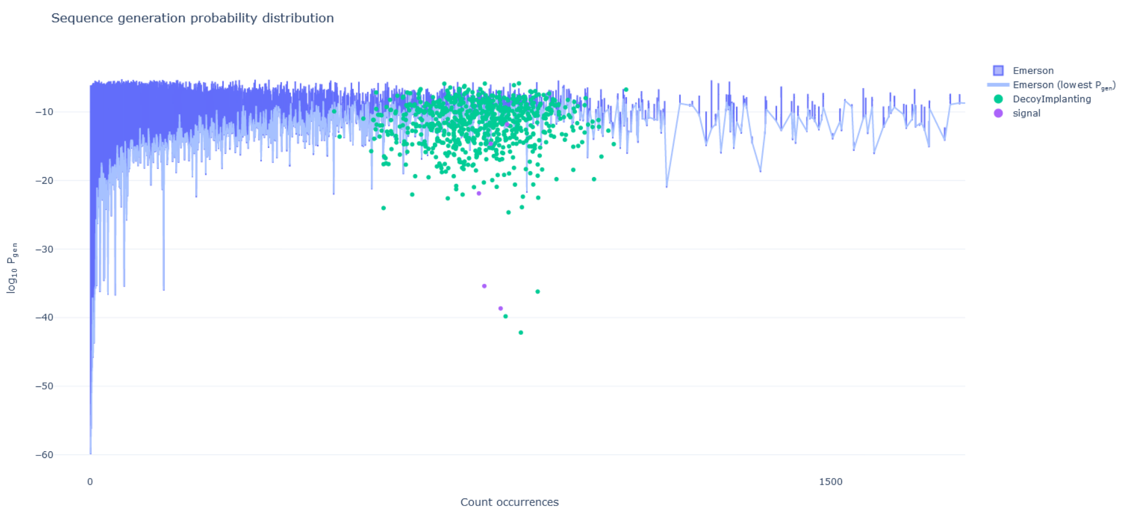
Figure 4.7: Naively implanted signal has low likelihood of occurring through V(D)J-recombination compared to the rest of the experimental dataset. The y-axis is the P-value that the clone occurrence pattern of a sequence is explained by convergent recombination alone, and not due to clonal expansion or high implanting rates.

5.1.2. Most of these decoy sequences have a high P_{gen} , but some have low enough P_{gen} to somewhat hide the signal sequences with the lowest P_{gen} . Figure 4.10 shows that, while most of the decoy sequences have a high P-value, some decoy sequences have similar P-value to the signal sequences.

Decoy implanting: P_{gen} distribution



(a) Count by count occurrences



(b) Cropped version of (a). X-axis range: 0-1800.

Figure 4.8: Full sequence signal implanted alongside 1000 randomly generated decoy sequences in experimental data. Showing the P_{gen} distribution compared to each sequence's count occurrence makes it possible to spot sequences with low P_{gen} that are implanted many times. The decoy sequences are implanted in both sick and healthy repertoires to attempt to "hide" the outliers in the signal sequences that only appear in sick repertoires.

4.3.3 Mutated sequence implanting

The mutated sequence implanting used a mutation hamming distance of 2. This means that up to 2 positions in the seed sequences were mutated for each mutated sequence. This mutation hamming distance was chosen to be low enough to preserve as much of the signal seeds as possible, while being high enough to be able to produce enough

Decoy implanting: P_{gen} distribution

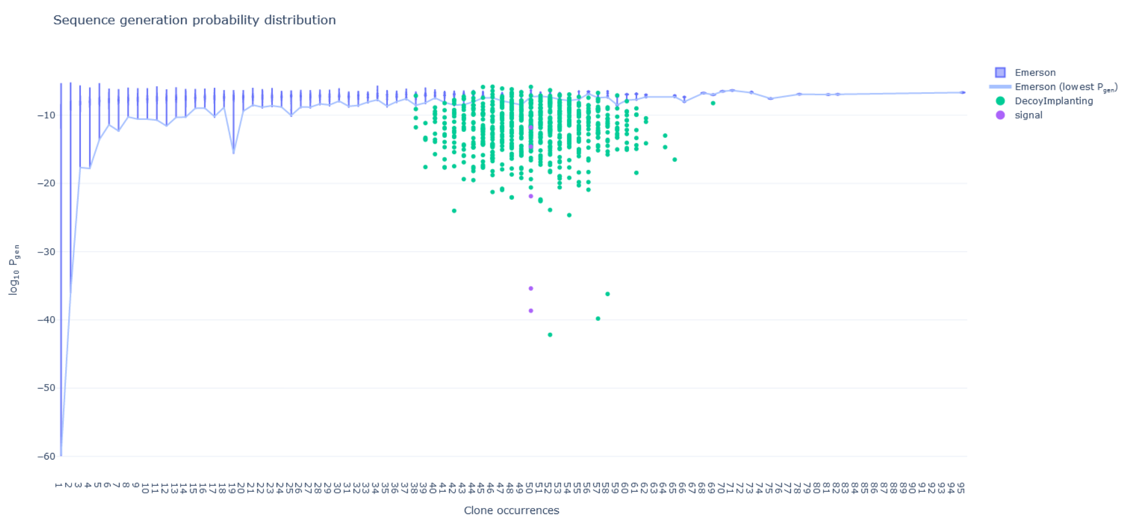


Figure 4.9: Count by clone occurrences: Full sequence signal implanted alongside 1000 randomly generated decoy sequences in experimental data.

varied mutations. The method used a wide mutated position occurrence limit, that gave a slightly higher chance of mutations in the middle of the sequence. The full yaml-specification can be found in Appendix A.1.

Both Figure 4.11 and 4.12 show that the mutated sequences mostly match the distributions of the rest of the dataset. Most of the mutated sequences have a low count (and clone) occurrence. There are some sequences that might appear to not follow the distribution perfectly, and have a count occurrence that is higher than most other sequences with similar P_{gen} . Figure 4.13 shows that the P-value of the mutated sequences are varied. The P-values are mostly high, but some are lower and similar to the naively implanted signal's P-value.

Decoy implanting: vdjRec results

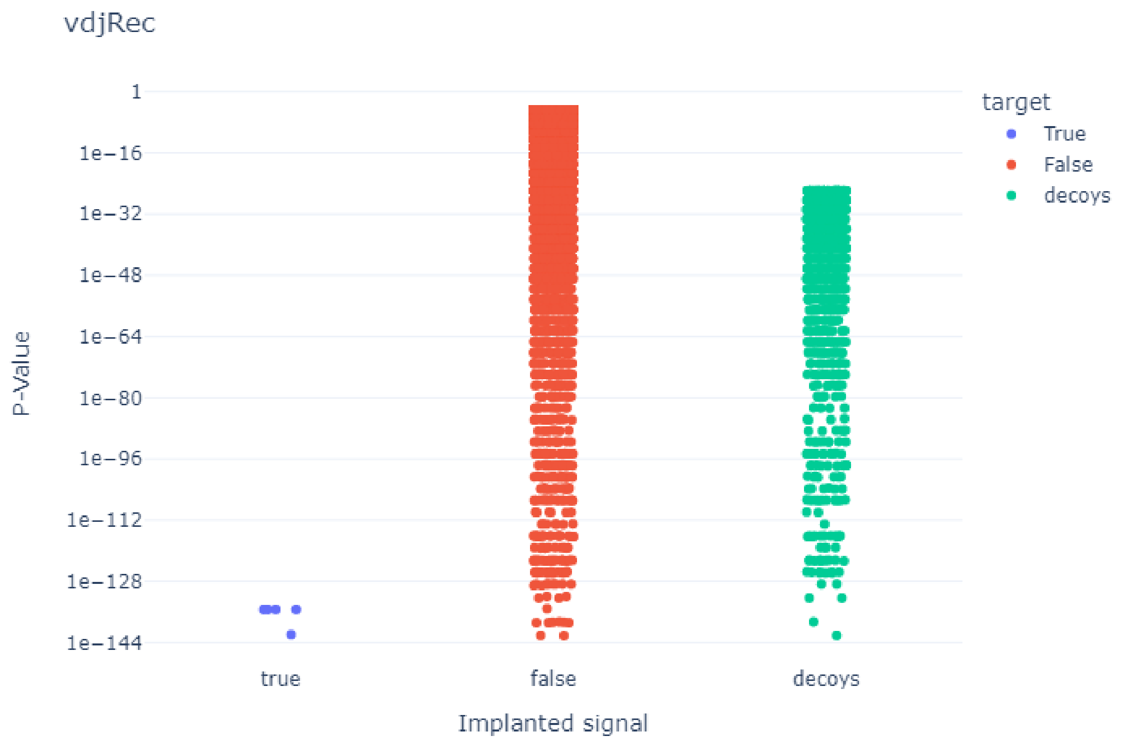
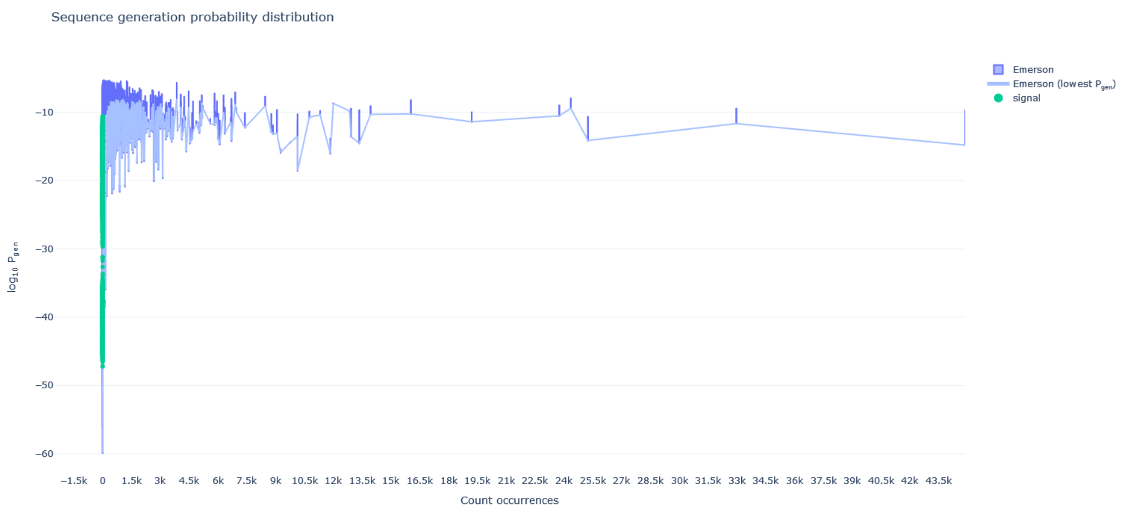
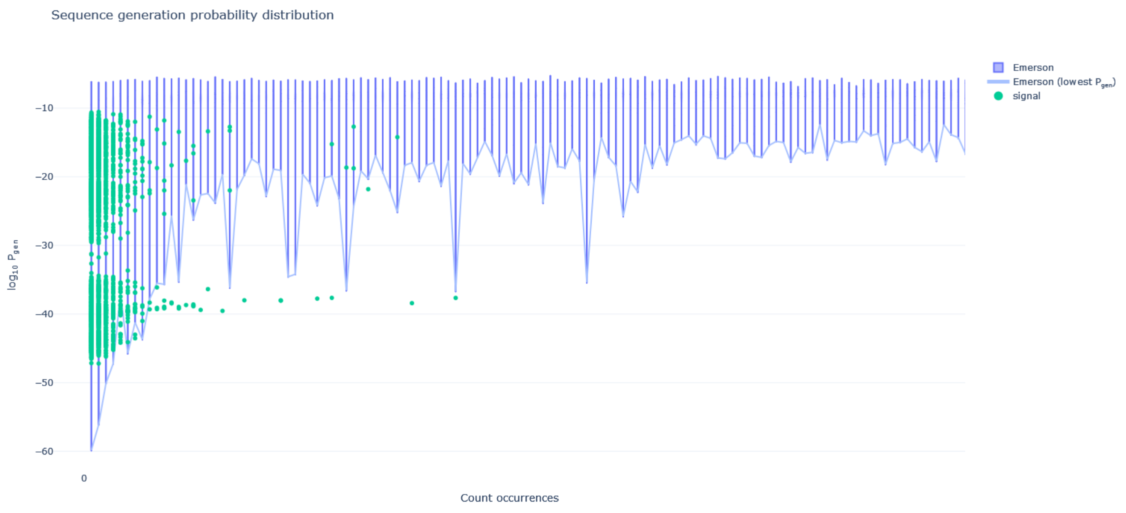


Figure 4.10: Implanted signal has a low likelihood of occurring through V(D)J-recombination compared to the rest of the experimental dataset. The implanted decoy sequences have a large P-value range, where some decoys have similar P-value to the signal sequences. The y-axis is the P-value that the clone occurrence pattern of a sequence is explained by convergent recombination alone, and not due to clonal expansion or high implanting rates.

Mutated sequence implanting: P_{gen} distribution



(a) Count by count occurrences



(b) Cropped version of (a). X-axis range: 0-117.

Figure 4.11: Mutated sequences from a signal seed implanted in experimental data. Showing the P_{gen} distribution compared to each sequence's count occurrence makes it possible to spot sequences with low P_{gen} that are implanted many times. The mutated sequences are implanted to follow the distribution of the experimental data.

Mutated sequence implanting: P_{gen} distribution

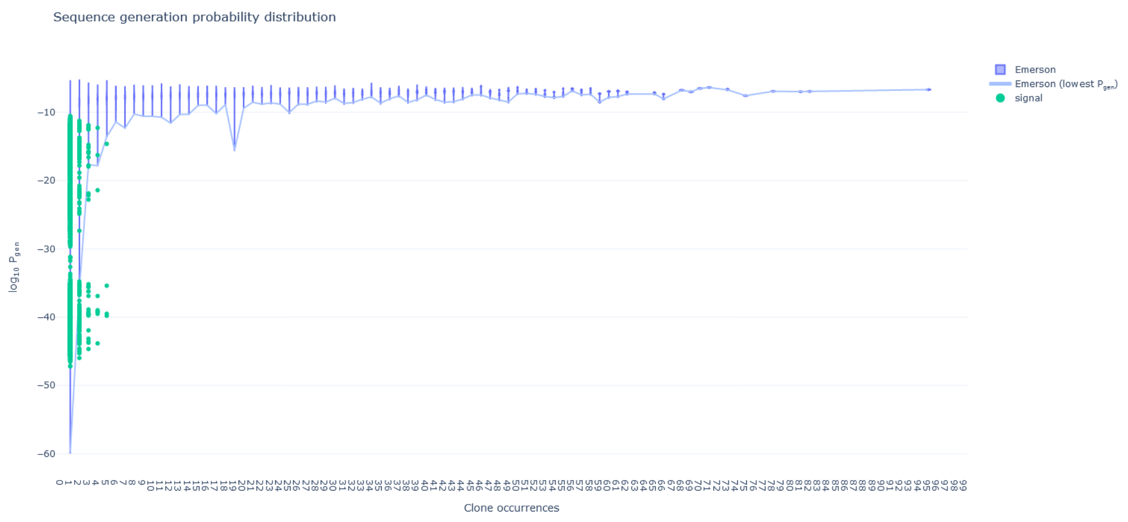


Figure 4.12: Mutations of the seed sequences from the immune signal have similar likelihood of occurring naturally as the rest of the dataset. P-value is the likelihood of each sequence occurring from V(D)J-recombination.

Mutated sequence implanting: vdjRec results

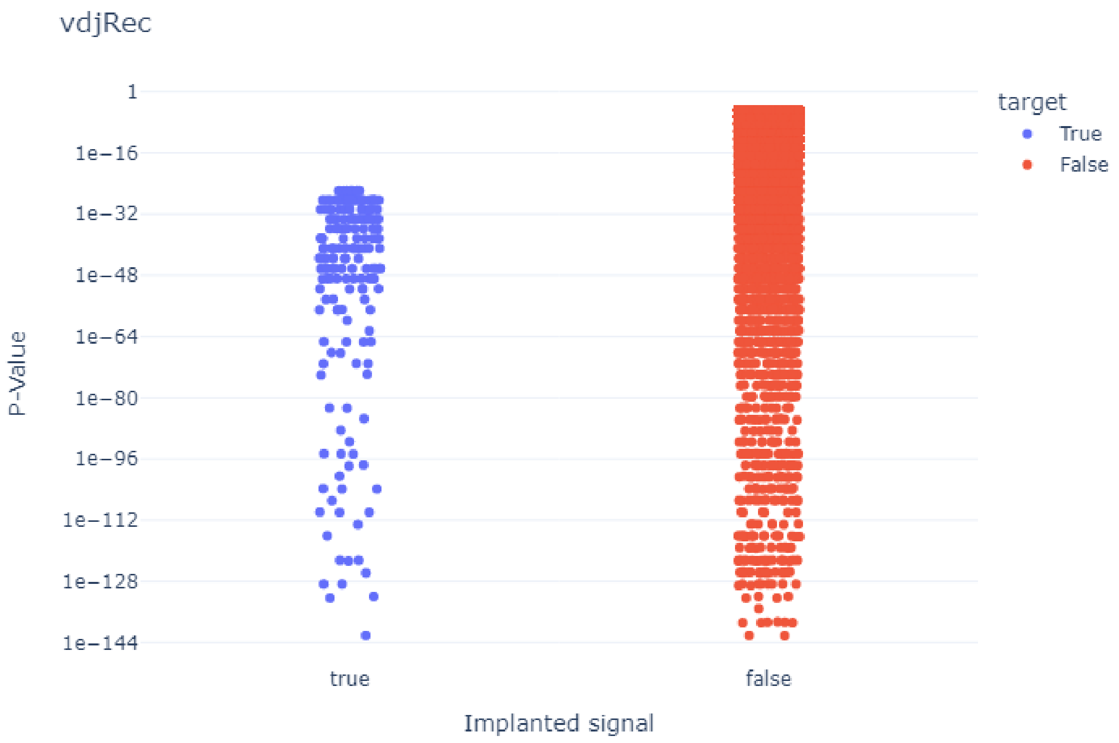


Figure 4.13: Mutations of the seed sequences from the immune signal have similar likelihood of occurring naturally as the rest of the dataset. The y-axis is the P-value that the clone occurrence pattern of a sequence is explained by convergent recombination alone, and not due to clonal expansion or high implanting rates.

Chapter 5

Discussion

5.1 Spotting P_{gen} outliers implanted in datasets using different signal implanting strategies

This thesis proposes two methods which aimed to implant full sequence immune signals such that methods can not exploit unrealistic P_{gen} outlier sequences. These new implanting strategies, and a naive full sequence implanting strategy, were tested by visually observing how implanting sequences in different ways relate to the rest of the P_{gen} distribution in the dataset, as well as applying the vdjRec-method on the ground truth-data.

5.1.1 Naive full sequence implanting

By visualizing the P_{gen} distribution of the dataset along with the implanted signal sequences, we can clearly see that the naively implanted signal produce outliers (Figure 4.6). However, the number of outliers differed based on the method of counting the sequences in the dataset. This is discussed further in Section 5.2.

The vdjRec method did successfully identify the naively implanted sequences as P_{gen} outlier candidates (Figure 4.7). The P-value of all the signal sequences was very low. This means that the program deems it likely that the signal sequences occur too often in the dataset to simply be produced by V(D)J-recombination. There are some sequences in the dataset with similar P-value. However, these low P-value sequences are very few, only about 10 have similar P-value to the signal. If one were to select all sequences under a certain P-value threshold to be significant outliers, one could easily select a threshold (e.g., 1e-130) where a large portion of the candidate sequences would be the signal. As the dataset used is very large (10 000 000 sequences), narrowing the candidates down to such a small number should be seen as successfully "finding" the outliers. The non-implanted sequences with very low P-value could be due to the experimental dataset having some naturally occurring "outliers", likely due to clonal expansion or sequencing errors.

5.1.2 Decoy implanting

Results

The signal was somewhat hidden by decoy sequences that are implanted in both sick and healthy repertoires (Figure 4.8 and 4.9). The signal sequences with higher P_{gen} were hidden well by a large number of decoy sequences, but signal sequences with low P_{gen} only had three decoy sequences with similar P_{gen} . Further testing would be required to say conclusively whether this fully "hides" the P_{gen} outliers. Using vdjRec shows that some decoy sequences have P-value similar to the signal sequences (Figure 4.10). If one were to select all candidates with P-value lower than, e.g., 1e-130, one would select the entire signal, along with 2-4 decoy sequences (and some non-implanted sequences). Ideally, there would a larger number of decoy sequences with lower P-value, but this example still illustrates the effectiveness of the decoy implanting strategy.

Strengths and weaknesses

Decoy implanting does not always preserve the P_{gen} to count occurrence distribution of the dataset, but, in contrast to mutated sequence implanting, the decoy implanting strategy implants and preserves the full signal instead of mutating it.

One weakness of the decoy implanting-method is that the randomly generated decoy sequences often have a high P_{gen} , as these are per definition the most likely to be generated. The decoys with high P_{gen} will often not be outliers, and will therefore not mask the outlier sequences from the signal. A solution to this problem could simply be to generate and implant a higher number of decoys, as there is a higher chance of generating low-P_{gen} sequences and hiding the P_{gen} outliers from the signal. However, with more and more decoy sequences, the dataset distribution can become more and more distorted. In the example of decoy implanting provided in this thesis, 1000 decoy sequences are generated and implanted about 500 times each in the dataset. This is 500 000 sequences, or 5 percent of the dataset, that are added to a dataset of 10 000 000 sequences. This could potentially fundamentally change the P_{gen} distribution of the dataset.

In the future, this method could be extended in order to only select decoy sequences with similar P_{gen} as the signal sequences, thus always masking the signal without having to generate and implant too many decoy sequences. A downside to this solution could be that computing P_{gen} for each generated decoy could easily become computationally heavy and time-consuming, as it could take many attempts to generate decoy sequences with low enough P_{gen} . Alongside this extension, the method could let the user specify their own decoy sequences, which would give more control to the user.

5.1.3 Mutated sequence implanting

Results

The mutated signal sequences that were implanted appropriately according to their P_{gen} followed the P_{gen} distribution of the experimental dataset (Figure 4.11 and 4.12). Most of the mutated sequences followed the P_{gen} distribution very well, however there were some sequences that may appear to be P_{gen} outliers. These are the more abundant sequences around $\log_{10}$ P_{gen} -40 (most noticeable in Figure 4.11b). These sequences appear to be P_{gen} outliers (and might still be deemed outliers by programs such as vdjRec), and are caused by P_{gen} outlier "peaks" in the experimental data. The occurrence limit P_{gen} range (explained further in Section 3.5.2) controls how many times each sequence can be implanted based on their P_{gen} . This occurrence limit P_{gen} range is strictly increasing, meaning that it will follow these outlier peaks and deem them as within the acceptable distribution of the dataset. However, the "outliers" produced by this are quite few, not very extreme, and as they are part of the dataset, it can be argued that they do follow the distribution of the experimental data. In the future, the computation of this occurrence limit P_{gen} range could be altered in a way that attempts to not include the P_{gen} outlier peaks so that the mutated sequences fit the general distribution better.

The mutated signal sequences had a broad P-value range, and most of the mutated sequences had a mid to high P-value compared to the rest of the dataset. If one were to select all P_{gen} outlier candidates by selecting all sequences with P-value under a certain threshold, one would not so easily capture a significant amount of signal sequences. If one did capture many signal sequences, one would also capture many times more non-implanted sequences.

The P_{gen} of the mutated sequences seem to group into two major clusters. These groups appear to match the P_{gen} of the seed sequences. This could show that mutating mostly towards the middle of the sequences produces mutations with similar P_{gen} to the seeds.

Strengths and weaknesses

It can be said that mutated sequence implanting is the more sophisticated of the two new signal implanting strategies, as it preserves the P_{gen} to count occurrence distribution of the implanted dataset while preserving most of the full sequence seed. However, the method can be critiqued for not truly implanting the full seed sequences, and might not fully work very well for machine learning models that rely on full sequences instead of sequence similarity or subsequences. For instance, the binary classifier proposed by Emerson et al. [28] relies on the presence of identical sequences across disease-labelled repertoires.

5.2 Counting methods: count occurrences and clone occurrences

Experimental data (Figure 4.2a) have some very high count occurrence sequences (highest sequence count occurrence is about 43 000). These sequences were selected randomly from the very large Emerson dataset, along with their duplicate counts. As these repertoires are very large, these abundant clonotypes could be due to very high duplicate counts in just a few repertoires. About half of these Emerson AIRRs have positive CMV status[28], so the high-count clonotypes could possibly be part of this CMV immune signal. However, the Emerson data is bulk-sequenced, which can heavily distort the sequence count distribution. The extremely high-count occurrence clonotypes could also be a product of the added level of random noise produced by this distortion.

The P_{gen} distribution was generally smoother when counting by clone occurrences for both experimental (Figure 4.2a) and synthetic data (Figure 4.1a). Counting by clone occurrences always lead to smaller x-axis, with the maximum x-value being totals number of AIRRs in the dataset (here, 100). This smaller and more limited axis can give a better (and often more readable) view of the general behavior of the P_{gen} distribution.

Counting by clone occurrences (Figure 4.6) was more effective than counting by count occurrences (Figure 4.6b) as a counting method for spotting naively implanted P_{gen} outliers using the dataset and implanting parameters used in this thesis. The signal was implanted a few times in half the repertoires, meaning the signal had quite a high clone occurrence, but a low count occurrence. The signal sequences were selected to have a wide range of P_{gen} , as it was necessary to implant clonotypes with P_{gen} low enough to become P_{gen} outliers within the experimental dataset used. However, these sequences were not all outliers for both counting methods. Only two of the signal sequences were noticeable outliers with the count occurrence-method, but all the sequences were clear P_{gen} outliers for the clone occurrence-method.

5.3 Effect of mutation on generation probability

Mutating different positions in CDR3s (Figure 4.4) shows that mutated sequences with mutations in the middle generally have a higher P_{gen} than mutations in the start or end. This could be explained by the fact that the V- (start) and J-genes (end) are more conserved than the D gene (middle). Mutations in these more conserved regions could perhaps produce less common sequences, i.e., sequences with lower P_{gen} . There are also two high-P_{gen} peaks in the figures. These peaks could be due to the added diversity in the junctions from random insertions and deletions during V(D)J-recombination. This added junctional variability could mean that mutating these positions are more tolerated.

5.4 Performance and implementation

The implementation currently has its own branch, but there are plans to merge it with the main immuneML branch in the future. However, to do this, the code likely needs to go through multiple rounds of feedback to make it fit into the framework in a way that allows future development and code maintenance to be easily done by others. The new programs need very thorough documentation and tutorials (example tutorial that could be used for the immuneML site can be found in section B).

P_{gen} computation

The implementation is designed to be quite efficient. The main time-consuming action is to compute the P_{gen} of all the sequences in the datasets. For the large datasets used in this thesis (10 000 000 sequences each), the P_{gen} computation was done on a high-performance computing server. Even when using 8 processes with 12 GB memory each, computing the P_{gen} of the about 5 500 000 unique sequences from the experimental data still took more than 7 hours. Therefore, this thesis could not use the entire Emerson dataset, which is much bigger with its 89 840 865 unique TCR β sequences [28].

A number of choices were made to ensure that P_{gen} computation of many sequences were not made unnecessarily. All files that require the P_{gen} of all the sequences in the dataset are exported from the sequence generation probability distribution-data report. For instance, one of these exported files is the occurrence limit P_{gen} range. To implant mutated sequences so that they follow the P_{gen} distribution of the dataset, we need the lowest P_{gen} for each value of count/clone occurrence. Computing the P_{gen} of the whole dataset every time we simulate with mutated sequence implanting would take a ridiculous amount of time, so we limit the number of times we need to do this computation.

Sequence generation probability distribution

Sequence generation probability distribution visualizes the P_{gen} to count (or clone) occurrences for each sequence in an AIRR dataset. Showing each clonotype as a point in a scatter plot often gives the clearest view of the behavior of the dataset's P_{gen} distribution. However, showing every single sequence as a dot is not scalable, and easily becomes very difficult for the computer to display. Therefore, a violin plot is used for the non-implanted sequences. This is easier for the computer to handle. For large datasets, these will appear more like lines, however, they still clearly show the distribution of the P_{gen} and sequence occurrences in the dataset.

The plot features a line displaying the lowest P_{gen} for each value of x. This is an attempt to find a border that delineates the general distribution of the dataset. This can be used to have a clear way to distinguish between outliers and that follow the dataset distribution. In the future, this line could be smoothed in some way so it better reflects the general distribution of the dataset. This "smoothed distribution line" could also be used to compute the occurrence limit P_{gen} range.

5.5 Generation probabilities in the datasets

5.5.1 P_{gen} distribution of experimental and synthetic data

AIRRs generated using OLGA (Figure 4.1a) can be expected to have a similar P_{gen} distribution to AIRRs that have never gone through any clonal expansion. The experimental dataset (Figure 4.2a) has many more sequences that appear more often than the synthetic dataset. There are also sequences appearing often in the dataset that has a lower P_{gen} than what we see in the simulated data. This could possibly be explained by clonal expansion, and fits the reasoning from Pogorelyy et al. [27]: that the convergently selected, i.e., sequences that appear more often due to clonal expansion, have a generally lower P_{gen} than the convergently recombined, i.e., sequences that appear more often due to biases during V(D)J-recombination. However, the dataset generated using OLGA is not experimental data, and we should be careful when comparing the two. The synthetic dataset is generated through a very generalized generative model based on V(D)J-recombination. The sequence counts in the Emerson dataset is heavily distorted due to sequencing. This distortion will appear only in the experimental data.

5.5.2 Sequences deemed impossible to generate

The sequence generation probability distribution-data report (described in Section 3.3) does not visualize sequences with P_{gen} equal to 0. These sequences are deemed impossible to generate through VDJ-recombination by the P_{gen} computation program. These sequences would always be outliers no matter the count occurrence, as they can never occur naturally. The variety of P_{gen} means the P_{gen} axis must be log-scale, which does not include 0.

The experimental Emerson dataset used in this thesis has about 75 000 sequences with P_{gen} equal to 0 (according to OLGA's computations). This is about 0.75 percent of dataset sequences. The AIRR dataset generated using OLGA does not have any sequences with P_{gen} equal to 0, as the model for P_{gen} computation and generation are the same. There are a few factors as to why these "impossible" receptor sequences appear in experimental human blood data. Individuals have different biological models that determine the mechanisms of V(D)J-recombination, meaning that P_{gen} for the same AIR sequence can be different across individuals [29]. OLGA is trying to generalize these repertoire generation models into a single model, which leads to P_{gen} computation likely never being exactly correct for any specific individual. This is a major limitation of OLGA (as well as methods that also generalize models similarly), and thus also a limitation of the new methods presented in this thesis. The databases for V, D and J genes are very incomplete. As OLGA relies on incomplete gene reference databases, the program might not be aware of the genes used to create some of the different receptors. Genetics especially differ with race, and the genetics of people with European ancestry has been much better mapped than other people of other ancestries [30]. Ethnic minorities might therefore have less accurate P_{gen} computation. Errors when sequencing the dataset could also be a factor for these sequences with P_{gen} 0, for example, if there is a sequencing error in a conserved region in a receptor sequence.

5.5.3 Range of generation probabilities in the datasets

The P_{gen} ranges for synthetic data and experimental data are quite different. The experimental Emerson data (Figure 4.2) has sequences with ($\log_{10}$) P_{gen} down to about -60, while the lowest P_{gen} value for synthetic data from OLGA (Figure 4.1a) doesn't even go down to -40. The OLGA model is trying to estimate the P_{gen} using an incomplete database of V, D and J genes. Sequences generated by OLGA might therefore be a little biased towards having a higher P_{gen} , because the same model that created the sequences is rating how likely it is to create this sequence. If there is any difference between the statistics of the V, D, and J genes used by OLGA and the gene statistics a given dataset is based on, then these sequences would generally be less likely to generated according to the OLGA model. OLGA can generate all sequences it deems to have P_{gen} higher than 0. The full P_{gen} range of OLGA sequences is therefore much larger than what is found in the synthetic dataset used here. The more sequences are generated, the higher the chance of finding sequences with very low P_{gen} . This "full P_{gen} range" would only be found if one generated an infinitely large dataset.

The highest P_{gen} in the different datasets are roughly the same. There is likely a natural maximum value of P_{gen} , at least according to the OLGA model. Since both datasets have such a large size, and since the sequences with high P_{gen} are by definition the most common sequences, it is likely that they both have sequences with P_{gen} that match or approach this maximum P_{gen} value.

Chapter 6

Conclusion

This thesis presents how P_{gen} outliers can be created when simulating immune signals by implanting full sequences. These outliers were shown to be noticeable using a visualization of the P_{gen} and count occurrences of each sequence in a dataset. The two new implementations for signal implanting strategies did not produce P_{gen} outliers that could be noticed using the same visualization method. vdjRec is a method for finding the candidates that are least likely to be formed solely through V(D)J-recombination. This method was used to successfully detect candidates for P_{gen} outliers produced by a naive signal implanting strategy. vdjRec could not reliably be used to find the signal sequences implanted using the new signal implanting strategies.

Both the new signal implanting strategies successfully showed that they can be used to simulate full sequence immune signals, however, they have different strengths and weaknesses. Mutated sequence implanting makes the signal sequences follow the P_{gen} distribution of the rest of the dataset, but has to alter the signal sequences slightly in order to do so. Decoy implanting preserves the full sequences, but may alter the P_{gen} distribution of the dataset by adding many decoy sequences that do not follow the dataset distribution. This thesis suggests that the new methods can be used complementary, filling different roles depending on what ML model is used.

6.1 Future work

6.1.1 P_{gen} computation in large datasets

The Emerson dataset [28] is still one of the bigger AIRR datasets in the field, but as our sequencing technologies continue to improve, these large datasets are likely going to be more prevalent. The P_{gen} computation used in this thesis is still too slow to handle these very large datasets (at least without pouring a lot of resources into the analysis). Future versions of the program could perhaps only compute a subset of these sequences. The P_{gen} to count occurrence distribution of datasets follow a heavy-tailed distribution, meaning there are much more sequences with low count occurrence. For this reason we could, for example, only compute the P_{gen} of 1 percent of all sequences that occur in 1 repertoire, 5 percent of sequences that occur in 2 repertoires, etc., and compute the P_{gen} for all sequences that occur in more than 10 repertoires.

The uses for the generation probability of immune receptors are still quite niche within the AIRR analysis field. If this interest expanded, we could see more large studies on, e.g., how P_{gen} distribution behaves in large datasets. If these studies published the large datasets with fully computed P_{gen} for all sequences, we could import them and use them for our own analyses, without having to go through computing the P_{gen} .

6.1.2 Future work for full sequence signal implanting strategies

In the future, the new signal implanting strategies should be tested using data from other chains and AIR types. The new methods have only been tested thoroughly with human TCR β (for both experimental and simulated data). Based on how repertoires are formed through V(D)J-recombination and later through clonal expansion, one can assume that the P_{gen} to count occurrence distribution follows a similar heavy-tailed distribution as TCR β repertoires. The method behavior is dynamic for heavy-tailed distributions, so these methods will very likely work just as well for other organisms, AIR cell types and chains.

Additionally, the new methods should be tested thoroughly with ML. The current implementations for the new signal implanting strategies mostly preserve the signal, as well as the sequences in the dataset, so the methods will very likely work for some ML scenarios. Datasets with signals with different implant strategies should be tested with different ML models, so we can compare what models work best for what strategies. For example, as mutated sequence implanting mutates the full signal sequences, it may not work very well for models that rely on identical shared sequences for classification. However, ML models that would learn to detect shared features between clusters of similar sequences could perform well on this kind of data.

Mutated sequence implanting randomly chooses which positions to mutate and which amino acids to mutate to. The method allows the user to control which positions in the seed sequences should be able to have mutations, and what the probability of mutating each position is. The method could be extended further to allow the user to control the probability of mutating to certain amino acids. Amino acids have different biochemical properties, so this could perhaps affect how mutation behaves.

The new signal implanting strategies have individual strengths and weaknesses, and will likely work differently for different ML models. Other ideas for signal implanting strategies that seek out to solve the P_{gen} outlier problem could improve or complement the implementations presented in this thesis. This thesis provides the groundwork for the development of such new methods.

Bibliography

[1] Jonathan Sprent. 'Immunological memory'. en. In: *Current Opinion in Immunology* 9.3 (June 1997), pp. 371–379. ISSN: 0952-7915. DOI: 10.1016/S0952-7915(97)80084-2. URL: https://www.sciencedirect.com/science/article/pii/S0952791597800842 (visited on 12/04/2023).

[2] Maryam B. Yassai, Yuri N. Naumov, Elena N. Naumova and Jack Gorski. 'A clonotype nomenclature for T cell receptors'. In: *Immunogenetics* 61.7 (July 2009), pp. 493–502. ISSN: 0093-7711. DOI: 10.1007/s00251-009-0383-x. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2706371/ (visited on 12/04/2023).

[3] Hongmei Liu et al. 'The methods and advances of adaptive immune receptors repertoire sequencing'. In: *Theranostics* 11.18 (Aug. 2021), pp. 8945–8963. ISSN: 1838-7640. DOI: 10.7150/thno.61390. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8419057/ (visited on 12/04/2023).

[4] Edus H. Warren, Frederick A. Matsen IV and Jeffrey Chou. 'High-throughput sequencing of B- and T-lymphocyte antigen receptors in hematology'. In: *Blood* 122.1 (July 2013), pp. 19–22. ISSN: 0006-4971. DOI: 10.1182/blood-2013-03-453142. URL: https://doi.org/10.1182/blood-2013-03-453142 (visited on 12/04/2023).

[5] *The innate and adaptive immune systems*. en. Publication Title: InformedHealth.org [Internet]. Institute for Quality and Efficiency in Health Care (IQWiG), July 2020. URL: https://www.ncbi.nlm.nih.gov/books/NBK279396/ (visited on 17/04/2023).

[6] Lonneke Scheffer. 'Simulation and analysis of immune receptor repertoire frequency distributions'. eng. Accepted: 2019-08-15T23:45:44Z. MA thesis. 2019. URL: https://www.duo.uio.no/handle/10852/69136 (visited on 04/05/2023).

[7] Oliver Backhaus. 'Generation of Antibody Diversity'. en. In: *Antibody Engineering*. IntechOpen, Feb. 2018. ISBN: 978-953-51-3826-6. DOI: 10.5772/intechopen.72818. URL: https://www.intechopen.com/chapters/58467 (visited on 05/05/2023).

[8] Jorg J. A. Calis and Brad R. Rosenberg. 'Characterizing immune repertoires by high throughput sequencing: strategies and applications'. English. In: *Trends in Immunology* 35.12 (Dec. 2014). Publisher: Elsevier, pp. 581–590. ISSN: 1471-4906, 1471-4981. DOI: 10.1016/j.it.2014.09.004. URL: https://www.cell.com/trends/immunology/abstract/S1471-4906(14)00155-0 (visited on 05/05/2023).

[9] Daniel J. Laydon, Charles R. M. Bangham and Becca Asquith. 'Estimating T-cell repertoire diversity: limitations of classical estimators and a new approach'. In: *Philosophical Transactions of the Royal Society B: Biological Sciences* 370.1675 (Aug. 2015), p. 20140291. ISSN: 0962-8436. DOI: 10.1098/rstb.2014.0291. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4528489/ (visited on 12/04/2023).

[10] Kenneth B. Hoehn, Anna Fowler, Gerton Lunter and Oliver G. Pybus. 'The Diversity and Molecular Evolution of B-Cell Receptors during Infection'. In: *Molecular Biology and Evolution* 33.5 (May 2016), pp. 1147–1157. ISSN: 0737-4038. DOI: 10.1093/molbev/msw015. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4839220/ (visited on 25/04/2023).

[11] Jean-Pierre Cabaniols, Nicolas Fazilleau, Armanda Casrouge, Philippe Kourilsky and Jean M. Kanellopoulos. 'Most α/β T Cell Receptor Diversity Is Due to Terminal Deoxynucleotidyl Transferase'. In: *Journal of Experimental Medicine* 194.9 (Nov. 2001), pp. 1385–1390. ISSN: 0022-1007. DOI: 10.1084/jem.194.9.1385. URL: https://doi.org/10.1084/jem.194.9.1385 (visited on 14/04/2023).

[12] David G. Schatz and Yanhong Ji. 'Recombination centres and the orchestration of V(D)J recombination'. en. In: *Nature Reviews Immunology* 11.4 (Apr. 2011). Number: 4 Publisher: Nature Publishing Group, pp. 251–263. ISSN: 1474-1741. DOI: 10.1038/nri2941. URL: https://www.nature.com/articles/nri2941 (visited on 12/04/2023).

[13] Jr Charles A Janeway, Paul Travers, Mark Walport and Mark J. Shlomchik. 'The generation of diversity in immunoglobulins'. en. In: *Immunobiology: The Immune System in Health and Disease. 5th edition* (2001). Publisher: Garland Science. URL: https://www.ncbi.nlm.nih.gov/books/NBK27140/ (visited on 12/04/2023).

[14] Zachary Sethna, Yuval Elhanati, Curtis G. Callan, Aleksandra M. Walczak and Thierry Mora. 'OLGA: fast computation of generation probabilities of B- and T-cell receptor amino acid sequences and motifs'. eng. In: *Bioinformatics (Oxford, England)* 35.17 (Sept. 2019), pp. 2974–2981. ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btz035.

[15] John L. Xu and Mark M. Davis. 'Diversity in the CDR3 Region of VH Is Sufficient for Most Antibody Specificities'. English. In: *Immunity* 13.1 (July 2000). Publisher: Elsevier, pp. 37–45. ISSN: 1074-7613. DOI: 10.1016/S1074-7613(00)00006-6. URL: https://www.cell.com/immunity/abstract/S1074-7613(00)00006-6 (visited on 13/04/2023).

[16] Victor Greiff, Gur Yaari and Lindsay G. Cowell. 'Mining adaptive immune receptor repertoires for biological and clinical information using machine learning'. en. In: *Current Opinion in Systems Biology*. Systems immunology & host-pathogen interaction (2020) 24 (Dec. 2020), pp. 109–119. ISSN: 2452-3100. DOI: 10.1016/j.coisb.2020.10.010. URL: https://www.sciencedirect.com/science/article/pii/S2452310020300524 (visited on 11/04/2023).

[17] Chakravarthi Kanduri et al. 'Profiling the baseline performance and limits of machine learning models for adaptive immune receptor repertoire classification'. In: *GigaScience* 11 (Jan. 2022), giac046. ISSN: 2047-217X. DOI: 10.1093/gigascience/giac046. URL: https://doi.org/10.1093/gigascience/giac046 (visited on 11/04/2023).

[18] Milena Pavlović et al. 'The immuneML ecosystem for machine learning analysis of adaptive immune receptor repertoires'. en. In: *Nature Machine Intelligence* 3.11 (Nov. 2021). Number: 11 Publisher: Nature Publishing Group, pp. 936–944. ISSN: 2522-5839. DOI: 10.1038/s42256-021-00413-z. URL: https://www.nature.com/articles/s42256-021-00413-z (visited on 19/04/2022).

[19] Xiaoqing Yu, Farnoosh Abbas-Aghababazadeh, Y. Ann Chen and Brooke L. Fridley. 'Statistical and Bioinformatics Analysis of Data from Bulk and Single-Cell RNA Sequencing Experiments'. en. In: *Translational Bioinformatics for Therapeutic Development*. Ed. by Joseph Markowitz. Methods in Molecular Biology. New York, NY: Springer US, 2021, pp. 143–175. ISBN: 978-1-07-160849-4. DOI: 10.1007/978-1-0716-0849-4_9. URL: https://doi.org/10.1007/978-1-0716-0849-4_9 (visited on 04/05/2023).

[20] Michael J. T. Stubbington, Orit Rozenblatt-Rosen, Aviv Regev and Sarah A. Teichmann. 'Single-cell transcriptomics to explore the immune system in health and disease'. In: *Science* 358.6359 (Oct. 2017). Publisher: American Association for the Advancement of Science, pp. 58–63. DOI: 10.1126/science.aan6828. URL: https://www.science.org/doi/10.1126/science.aan6828 (visited on 04/05/2023).

[21] Enkelejda Miho, Alexander Yermanos, Cédric R. Weber, Christoph T. Berger, Sai T. Reddy and Victor Greiff. 'Computational Strategies for Dissecting the High-Dimensional Complexity of Adaptive Immune Repertoires'. In: *Frontiers in Immunology* 9 (2018). ISSN: 1664-3224. URL: https://www.frontiersin.org/articles/10.3389/fimmu.2018.00224 (visited on 04/05/2023).

[22] Geir Kjetil Sandve and Victor Greiff. 'Access to ground truth at unconstrained size makes simulated data as indispensable as experimental data for bioinformatics methods development and benchmarking'. In: *Bioinformatics* (Sept. 2022), btac612. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btac612. URL: https://doi.org/10.1093/bioinformatics/btac612 (visited on 12/09/2022).

[23] Milena Pavlović et al. *Improving generalization of machine learning-identified biomarkers with causal modeling: an investigation into immune receptor diagnostics*. arXiv:2204.09291 [cs, q-bio]. Apr. 2023. DOI: 10.48550/arXiv.2204.09291. URL: http://arxiv.org/abs/2204.09291 (visited on 20/04/2023).

[24] Melanie R. Shapiro et al. *Human immune phenotyping reveals accelerated aging in type 1 diabetes*. en. Pages: 2023.02.24.529902 Section: New Results. Feb. 2023. DOI: 10.1101/2023.02.24.529902. URL: https://www.biorxiv.org/content/10.1101/2023.02.24.529902v1 (visited on 03/05/2023).

[25] Cédric R Weber et al. 'immuneSIM: tunable multi-feature simulation of B- and T-cell receptor repertoires for immunoinformatics benchmarking'. In: *Bioinformatics* 36.11 (June 2020), pp. 3594–3596. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btaa158. URL: https://doi.org/10.1093/bioinformatics/btaa158 (visited on 13/05/2023).

[26] Quentin Marcou, Thierry Mora and Aleksandra M. Walczak. 'High-throughput immune repertoire analysis with IGoR'. en. In: *Nature Communications* 9.1 (Feb. 2018). Number: 1 Publisher: Nature Publishing Group, p. 561. ISSN: 2041-1723. DOI: 10.1038/s41467-018-02832-w. URL: https://www.nature.com/articles/s41467-018-02832-w (visited on 22/04/2022).

[27] Mikhail V Pogorelyy et al. 'Method for identification of condition-associated public antigen receptor sequences'. In: *eLife* 7 (Mar. 2018). Ed. by Arup K Chakraborty. Publisher: eLife Sciences Publications, Ltd, e33050. ISSN: 2050-084X. DOI: 10.7554/eLife.33050. URL: https://doi.org/10.7554/eLife.33050 (visited on 31/01/2023).

[28] Ryan O. Emerson et al. 'Immunosequencing identifies signatures of cytomegalovirus exposure history and HLA-mediated effects on the T cell repertoire'. en. In: *Nature Genetics* 49.5 (May 2017). Number: 5 Publisher: Nature Publishing Group, pp. 659–665. ISSN: 1546-1718. DOI: 10.1038/ng.3822. URL: https://www.nature.com/articles/ng.3822 (visited on 18/01/2023).

[29] Andrei Slabodkin et al. 'Individualized VDJ recombination predisposes the available Ig sequence space'. en. In: *Genome Research* 31.12 (Jan. 2021). Company: Cold Spring Harbor Laboratory Press Distributor: Cold Spring Harbor Laboratory Press Institution: Cold Spring Harbor Laboratory Press Label: Cold Spring Harbor Laboratory Press Publisher: Cold Spring Harbor Lab, pp. 2209–2224. ISSN: 1088-9051, 1549-5469. DOI: 10.1101/gr.275373.121. URL: https://genome.cshlp.org/content/31/12/2209 (visited on 22/04/2022).

[30] Kerui Peng et al. 'Diversity in immunogenomics: the value and the challenge'. In: *Nature methods* 18.6 (June 2021), pp. 588–591. ISSN: 1548-7091. DOI: 10.1038/s41592-021-01169-5. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8842483/ (visited on 18/04/2023).

Appendix A

Documentation

A.1 Mutated Sequence Implanting

Example of YAML specification for mutated sequence implanting:

```yaml
definitions:
  datasets:
    my_dataset: # user-defined dataset name
      format: ImmuneML
      params:
        region_type: IMGT_JUNCTION
        is_repertoire: true # we are importing a repertoire dataset
        path: path/to/dataset
        metadata_file: path/to/metadata
  motifs:
    seed1:
      seed: CASRSGNEKLFF
      v_call: TRBV7-8*02
      j_call: TRBJ1-4*01
      mutation_position_possibilities:
        2: 0.1
        3: 0.1
        4: 0.1
        5: 0.2
        6: 0.2
        7: 0.1
        8: 0.1
        9: 0.1
      instantiation: GappedKmer
    seed2:
      seed: CASSWIEPHIKGEGQTYEQYF
      v_call: TRBV6-3*01
      j_call: TRBJ2-7*01
      mutation_position_possibilities:
        2: 0.05
        3: 0.05
        4: 0.05
        5: 0.05
        6: 0.05
        7: 0.05
        8: 0.1
        9: 0.1
        10: 0.1
        11: 0.1
        12: 0.05
        13: 0.05
        14: 0.05
        15: 0.05
        16: 0.05
        17: 0.05
      instantiation: GappedKmer
    seed3:
      seed: CASSYLVAENSGANVLTF
      v_call: TRBV6-5
      j_call: TRBJ2-6
      mutation_position_possibilities:
        2: 0.05
        3: 0.05
        4: 0.05
        5: 0.05
        6: 0.1
        7: 0.1
        8: 0.1
        9: 0.1
        10: 0.1
        11: 0.1
        12: 0.05
        13: 0.05
        14: 0.05
        15: 0.05
      instantiation: GappedKmer
    seed4:
      seed: CASSVMCQDRVSSYEQYF
      v_call: TRBV6-4
      j_call: TRBJ2-7
      mutation_position_possibilities:
        2: 0.05
        3: 0.05
        4: 0.05
        5: 0.05
        6: 0.1
        7: 0.1
        8: 0.1
        9: 0.1
        10: 0.1
        11: 0.1
        12: 0.05
        13: 0.05
        14: 0.05
        15: 0.05
      instantiation: GappedKmer
    seed5:
      seed: CATGGFFSYEQYF
      v_call: TRBV7-7*03
      j_call: TRBJ2-7*01
      mutation_position_possibilities:
        2: 0.1
        3: 0.1
        4: 0.1
        5: 0.1
        6: 0.2
        7: 0.1
        8: 0.1
        9: 0.1
        10: 0.1
      instantiation: GappedKmer

  signals:
    signal:
      motifs:
        - seed1
        - seed2
        - seed3
        - seed4
        - seed5
      implanting:
        MutatedSequence:
          mutation_hamming_distance: 2
          occurrence_limit_pgen_range: # from
              SequenceGenerationProbabilityDistribution
            7.220680062850662e-57: 2
            7.794254937992779e-51: 3
            6.107826743718888e-48: 4
            1.821959985799395e-46: 6
            2.0234293791767077e-44: 8
            1.4945643833176732e-38: 9
            2.6581458481634067e-37: 36
            6.937145322949189e-26: 43
            1.4747398205160718e-24: 63
            5.988660467194418e-23: 76
            5.059241689500405e-22: 146
            9.634161404115809e-22: 489
            1.4251105809584813e-21: 1161
            8.749768826698754e-21: 2670
            2.218250077134713e-20: 3124
            3.727010758499254e-16: 9242
            3.5027165748446177e-15: 9750
```

```
            1.313278270448748e-14: 11844
            2.0882156617027488e-11: 25145
            1.7465323761588067e-10: 44559
          overwrite_sequences: False
      sequence_position_weights:
        0: 1

  simulations:
    my_simulation:
      my_implanting:
        signals:
          - signal
        dataset_implanting_rate: 0.5
        repertoire_implanting_rate: 0.001

instructions:
  my_simulation_instruction:
    type: Simulation
    dataset: my_dataset
    simulation: my_simulation
    export_formats: [AIRR, ImmuneML] # export the simulated dataset to
        these formats
```

A.2 Decoy Implanting

Example of YAML specification for decoy implanting:

```
definitions:
  datasets:
    my_dataset: # user-defined dataset name
      format: ImmuneML
      params:
        region_type: IMGT_JUNCTION
        is_repertoire: true # we are importing a repertoire dataset
        path: path/to/dataset
        metadata_file: path/to/metadata
  motifs:
    seed1:
      seed: CASRSGNEKLFF
      v_call: TRBV7-8*02
      j_call: TRBJ1-4*01
      instantiation: GappedKmer
    seed2:
      seed: CASSWIEPHIKGEGQTYEQYF
      v_call: TRBV6-3*01
      j_call: TRBJ2-7*01
      instantiation: GappedKmer
    seed3:
      seed: CASSYLVAENSGANVLTF
      v_call: TRBV6-5
      j_call: TRBJ2-6
      instantiation: GappedKmer
    seed4:
      seed: CASSVMCQDRVSSYEQYF
      v_call: TRBV6-4
      j_call: TRBJ2-7
      instantiation: GappedKmer
    seed5:
      seed: CATGGFFSYEQYF
      v_call: TRBV7-7*03
      j_call: TRBJ2-7*01
      instantiation: GappedKmer

  signals:
    signal:
      motifs:
        - seed1
        - seed2
        - seed3
        - seed4
        - seed5
      implanting:
        Decoy:
          nr_of_decoys: 100
          overwrite_sequences: False
          dataset_implanting_rate_per_decoy: 0.5
          repertoire_implanting_rate_per_decoy: 0.0002 # signal implanting
              rate / 5 (because 5 seed sequences)
      sequence_position_weights:
        0: 1

  simulations:
    my_simulation:
      my_implanting:
        signals:
          - signal
        dataset_implanting_rate: 0.5
        repertoire_implanting_rate: 0.001

instructions:
  my_simulation_instruction:
    type: Simulation
    dataset: my_dataset
    simulation: my_simulation
    export_formats: [AIRR, ImmuneML] # export the simulated dataset to
        these formats
```

A.3 Naive Implanting

Example of YAML specification for naive implanting (by using decoy implanting with 0 decoy sequences):

```
definitions:
  datasets:
    my_dataset: # user-defined dataset name
      format: ImmuneML
      params:
        region_type: IMGT_JUNCTION
        is_repertoire: true # we are importing a repertoire dataset
        path: path/to/dataset
        metadata_file: path/to/metadata
  motifs:
    seed1:
      seed: CASRSGNEKLFF
      v_call: TRBV7-8*02
      j_call: TRBJ1-4*01
      instantiation: GappedKmer
    seed2:
      seed: CASSWIEPHIKGEGQTYEQYF
      v_call: TRBV6-3*01
      j_call: TRBJ2-7*01
      instantiation: GappedKmer
    seed3:
      seed: CASSYLVAENSGANVLTF
      v_call: TRBV6-5
      j_call: TRBJ2-6
      instantiation: GappedKmer
    seed4:
      seed: CASSVMCQDRVSSYEQYF
      v_call: TRBV6-4
      j_call: TRBJ2-7
      instantiation: GappedKmer
    seed5:
      seed: CATGGFFSYEQYF
      v_call: TRBV7-7*03
      j_call: TRBJ2-7*01
      instantiation: GappedKmer
  signals:
    signal:
      motifs:
        - seed1
        - seed2
        - seed3
        - seed4
        - seed5
      implanting:
        Decoy:
          nr_of_decoys: 0
          overwrite_sequences: False
      sequence_position_weights:
        0: 1

  simulations:
    my_simulation:
      my_implanting:
        signals:
          - signal
        dataset_implanting_rate: 0.5
        repertoire_implanting_rate: 0.001

instructions:
  my_simulation_instruction:
    type: Simulation
    dataset: my_dataset
    simulation: my_simulation
    export_formats: [AIRR, ImmuneML] # export the simulated dataset to
       these formats
```

A.4 Sequence generation probability distribution

Example of YAML specification for sequence generation probability distribution:

```
definitions:
  datasets:
    my_dataset: # user-defined dataset name
      format: ImmuneML
      params:
        region_type: IMGT_JUNCTION
        is_repertoire: true # we are importing a repertoire dataset
        path: path/to/dataset
        metadata_file: path/to/metadata
  reports:
    my_data_report:
      SequenceGenerationProbabilityDistribution:
        default_sequence_label: Emerson

instructions:
  my_expl_analysis_instruction: # user-defined instruction name
    type: ExploratoryAnalysis # which instruction to execute
    analyses: # analyses to perform
      my_analysis: # user-defined name of the analysis
        dataset: my_dataset
        report: my_data_report
    number_of_processes: 8
```

Appendix B

Tutorial

The branch with the new implementations can be found here: https://github.com/uio-bmi/immuneML/tree/pgen_simulation.

B.1 How to set up immuneML with the branch used in this thesis

First, clone the immuneML pgen_simulation-branch:

```
    git clone -b pgen_simulation https://github.com/uio-bmi/immuneML.git
```

Then, from the project directory, install the requirements and environment:

```
    pip install -r requirements.txt
    pip install -e .
```

Run immuneML with a yaml-specification by running

```
    immune-ml <yaml-file.yaml> <output/path>
```

B.2 How to simulate full sequence disease-associated signals in AIRR datasets without creating exploitable outliers

The immuneML documentation has tutorials (https://docs.immuneml.uio.no/latest/tutorials.html) that explain how to use certain features. If the new classes presented in this thesis are merged into the main immuneML branch, there should be a tutorial explaining their new features. Here is an example of what such a tutorial might look like. All the files and datasets used in this tutorial are available on the pgen_simulation branch (https://github.com/uio-bmi/immuneML/tree/pgen_simulation/example_files_for_tutorial).

In immuneML, it is possible to implant immune signals in an AIRR dataset in order to simulate the effect of an immune event on the repertoires. A more detailed tutorial for implanting immune signals can be found in the immuneML documentation (https://docs.immuneml.uio.no/latest/tutorials/how_to_simulate_antigen_signals_in_airr_datasets.html).

B.2.1 Generating a synthetic AIRR dataset with OLGA

This can be done with any AIRR dataset, but here we will generate a synthetic dataset.

We start by generating a dataset using OLGA. Here we use a dataset with 100 repertoires consisting of 1000 sequences each. As computing the generation probability of datasets can be time-consuming, we start with a small dataset. Here is a bash script that can be used for this:

```
#!/bin/bash

echo "Generating repertoires..."

mkdir repertoires

echo filename,identifier,subject_id > metadata.csv

for i in {1..100}
do
   olga-generate_sequences --humanTRB -n 1000 -o repertoires/rep$i.tsv
   echo repertoires/rep$i.tsv,rep$i,rep$i >> metadata.csv
done

echo "Finished generating repertoires"
```

B.2.2 Creating outliers by implanting immune signals naively

Generation probability (P_{gen}) is the probability that an immune receptor sequence is created naturally, solely through V(D)J-recombination. Here, we will implant a sequence with low P_{gen} in the generated dataset. If implanted enough times, this will create a P_{gen} outlier.

```
definitions:
  datasets:
    my_dataset: # user-defined dataset name
      format: OLGA
      params:
        region_type: IMGT_JUNCTION # we use the full sequence
        is_repertoire: true # we are importing a repertoire dataset
        path: . # path to the repertoires-directory
        metadata_file: metadata.csv # path to metadata
  motifs :
    seq_with_low_pgen:
      seed : CAPTALDRATWKRSPLQEQYF
      v_call : TRBV7-2
      j_call : TRBJ2-7
      instantiation : GappedKmer
  signals:
    signal_with_pgen_outlier:
      motifs:
        - seq_with_low_pgen
      implanting:
        Decoy:
          nr_of_decoys: 0 # decoy implanting with 0 decoys is the same as
              naive implanting
          overwrite_sequences: False
      sequence_position_weights:
        0: 1 # unimportant when implanting full sequences

  simulations:
    my_simulation:
      my_implanting:
        signals:
          - signal_with_pgen_outlier
        dataset_implanting_rate: 0.05 # implanting in 5 percent of the
            repertoires
        repertoire_implanting_rate: 0.001

instructions:
  my_simulation_instruction:
    type: Simulation
    dataset: my_dataset
    simulation: my_simulation
    export_formats: [AIRR] # export the simulated dataset to these formats
```

B.2.3 Analyzing the P_{gen} distribution for the dataset

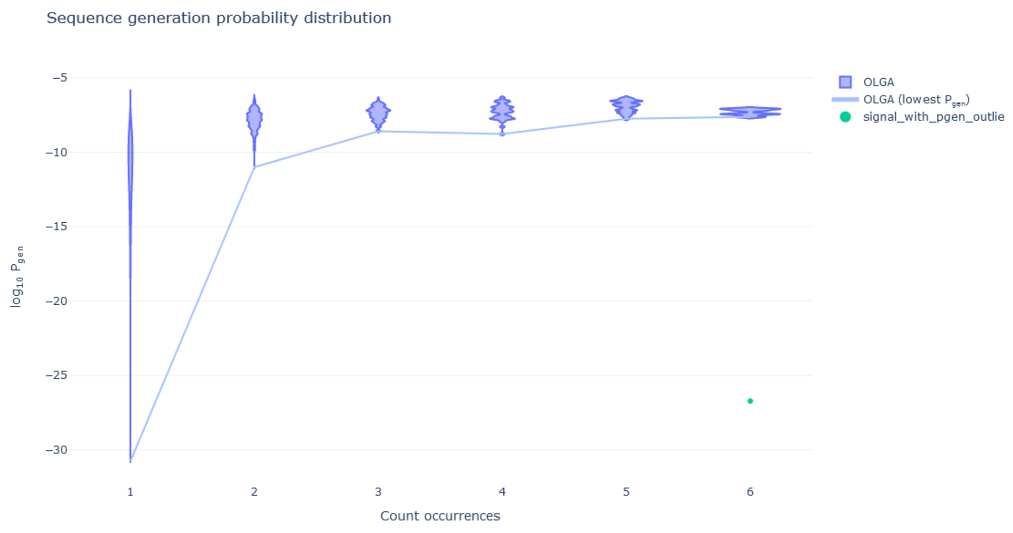
To find a P_{gen} outlier we need to compute the P_{gen} and count all the sequences in the whole dataset. This is done here using a data report called SequenceGenerationProbabilityDistribution. Computing the P_{gen} of a whole dataset can be quite time-consuming (for 100 000 sequences, about 5–15 minutes using 4 processes). It is recommended to use multiple processes, as this speeds up the process by a lot.

```
definitions:
  datasets:
    my_dataset: # user-defined dataset name
      format: AIRR
      params:
        region_type: IMGT_JUNCTION
        is_repertoire: true # we are importing a repertoire dataset
        path: path/to/the/repertoires/directory
        metadata_file: path/to/metadata/of/the/dataset
  reports:
    my_data_report:
      SequenceGenerationProbabilityDistribution:
        default_sequence_label: OLGA

instructions:
  my_expl_analysis_instruction: # user-defined instruction name
```

```
    type: ExploratoryAnalysis # which instruction to execute
    analyses: # analyses to perform
      my_analysis: # user-defined name of the analysis
        dataset: my_dataset
        report: my_data_report
    number_of_processes: 4 # number of processes to use
```

Applying this analysis to our dataset with naive implants, we get a plot similar to this:



The signal sequence is easily noticeable due to its high count occurrence (total count in all AIRRs in the dataset) compared to its low P$_{gen}$.

B.2.4 Implanting decoy sequences along the immune signal to mask the exploitable outliers

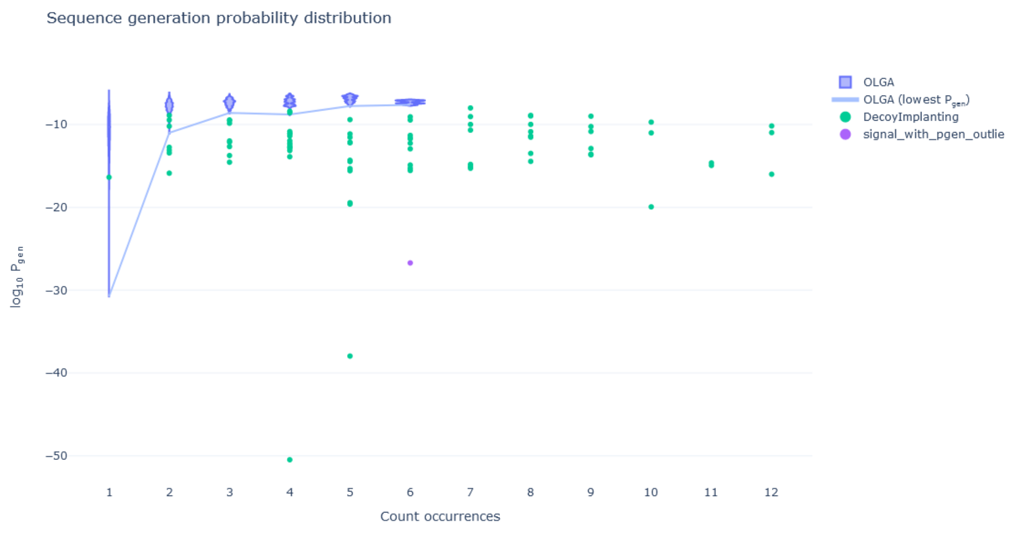
The signal is only implanted in diseased repertoires. Implanting decoy sequences in both diseased and healthy repertoires will mask the outliers from the signal.

```
definitions:
  datasets:
    my_dataset: # user-defined dataset name
      format: OLGA
      params:
        region_type: IMGT_JUNCTION # we use the full sequence
        is_repertoire: true # we are importing a repertoire dataset
        path: . # path to the repertoires-directory
        metadata_file: metadata.csv # path to metadata
  motifs :
    seq_with_low_pgen:
      seed : CAPTALDRATWKRSPLQEQYF
      v_call : TRBV7-2
      j_call : TRBJ2-7
      instantiation : GappedKmer
  signals:
    signal_with_pgen_outlier:
      motifs:
        - seq_with_low_pgen
      implanting:
        Decoy:
          nr_of_decoys: 100
          overwrite_sequences: False
          dataset_implanting_rate_per_decoy: 0.05
          repertoire_implanting_rate_per_decoy: 0.001
      sequence_position_weights:
        0: 1

  simulations:
    my_simulation:
      my_implanting:
        signals:
          - signal_with_pgen_outlier
        dataset_implanting_rate: 0.05 # implanting in 5 percent of the repertoires
        repertoire_implanting_rate: 0.001

instructions:
  my_simulation_instruction:
    type: Simulation
    dataset: my_dataset
    simulation: my_simulation
    export_formats: [AIRR] # export the simulated dataset to these formats
```

Applying the P_{gen} -analysis to the dataset with implanted signal and decoys, we get a plot similar to this:



Some of the decoys are outliers and mask the signal.

B.2.5 Implanting mutated versions of the immune signal sequences to avoid outliers

Mutating the signal sequence into multiple similar sequences and implanting these mutated sequence in a way that fits the P_{gen} to count occurrence distribution of the dataset will not produce any outliers. To do this, we need an occurrence limit P_{gen} range for our dataset, which controls how often each mutated sequence is allowed to be implanted according to their P_{gen} . This occurrence limit P_{gen} range is computed automatically when running SequenceGenerationProbabilityDistribution (if the other steps in the tutorial have been followed, this P_{gen} range is already computed for our dataset can be found in the exported files in the output directory or in the index.html).

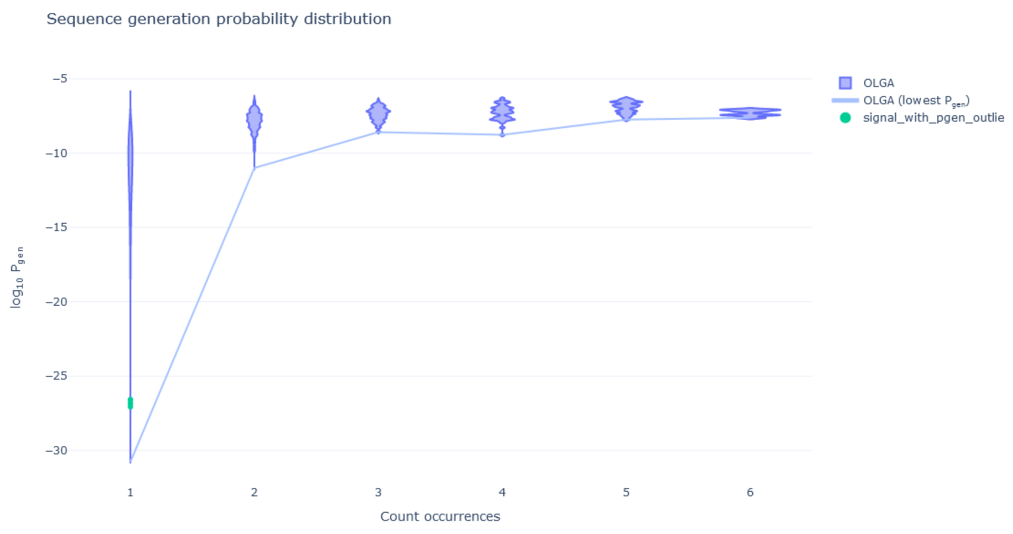
```
definitions:
  datasets:
    my_dataset: # user-defined dataset name
      format: OLGA
      params:
        region_type: IMGT_JUNCTION # we use the full sequence
        is_repertoire: true # we are importing a repertoire dataset
        path: . # path to the repertoires-directory
        metadata_file: metadata.csv # path to metadata
  motifs :
    seq_with_low_pgen:
      seed : CAPTALDRATWKRSPLQEQYF
      v_call : TRBV7-2
      j_call : TRBJ2-7
      mutation_position_possibilities: # possibilities for which positions
          in the sequence to mutate. Here, the mutations are even along the
           middle.
        4: 0.1
        5: 0.1
        6: 0.1
        7: 0.1
        8: 0.1
        9: 0.1
        10: 0.1
        11: 0.1
        12: 0.1
        13: 0.1
      instantiation : GappedKmer
  signals:
    signal_with_pgen_outlier:
      motifs:
        - seq_with_low_pgen
      implanting:
        MutatedSequence:
          mutation_hamming_distance: 2
          occurrence_limit_pgen_range: # this will differ for each
              generated dataset
```

```
            1.0167556408918248e-11: 2
            1.671167468445542e-09: 4
            1.7699642495296634e-08: 5
            2.3559356353530645e-08: 6
          overwrite_sequences: False
      sequence_position_weights:
        0: 1

  simulations:
    my_simulation:
      my_implanting:
        signals:
          - signal_with_pgen_outlier
        dataset_implanting_rate: 0.05 # implanting in 5 percent of the
            repertoires
        repertoire_implanting_rate: 0.001

instructions:
  my_simulation_instruction:
    type: Simulation
    dataset: my_dataset
    simulation: my_simulation
    export_formats: [AIRR] # export the simulated dataset to these formats
```

Applying the P_{gen} -analysis to the dataset with implanted mutated signal sequences, we get a plot similar to this:



All the mutated signal sequences are implanted only once each. This means that they now follow the distribution of the rest of the dataset.