panCRISPR Toolbox — a deep learning approach to improve CRISPR/Cas experiments

TUM Data Innovation Lab (TUM-DI-LAB)

Munich Data Science Institute (MDSI)

Technical University of Munich

&

Helmholtz AI Munich

Authors	Daria Yasafova, Dennis Gankin, Yevhenii Sharapov, Chelsea Bright, Firas Driss, Francesco Campi
Mentor(s)	Dr. Lisa Barros de Andrade e Sousa, Dr. Erinc Merdivan (Helmholtz AI Munich)
Project lead	Dr. Ricardo Acevedo Cabra (MDSI)
Supervisor	Prof. Dr. Massimo Fornasier (Board of Directors of MDSI)

Feb 2022

Abstract

The CRISPR (clustered regularly interspaced short palindromic repeats)/Cas system is a novel and powerful genome-editing technology. The two main components of the CRISPR/Cas system is a user-defined guide RNA (gRNA) and a CRISPR-associated (Cas) nuclease. The gRNA is a 20 base pair sequence specific (complementary) to the region of interest in a genome that guides the Cas nuclease which in turns makes a cut, knocking out the targeted gene. Thus, the user-defined gRNA must be designed specifically to allow for it to locate and bind effectively to the target location. To aid in the design of gRNAs that efficiently bind to the region of interest and simultaneously minimize the number of off-target bindings, we have built a user-friendly, integrative gRNA design toolbox which uses state-of-the-art machine and deep learning models, as well as bioinformatics techniques, to optimize gRNA design for use in CRISPR/Cas experiments.

Contents

1 Introduction

CRISPR is a technology that has been successfully harnessed to edit the genomes of prokaryotes, humans, and many other organisms. The tool targets a specifc location of the DNA with the goal to alter that specific piece of DNA and modify gene function (such as switching genes on and off). CRISPR is widely used for scientific research as it is a cheaper, faster, and easier alternative to previous gene editing techniques (i.e. TALENs and ZFN). The tool has many potential applications including correcting genetic defects, treating and preventing the spread of diseases and improving crops [21, 28].

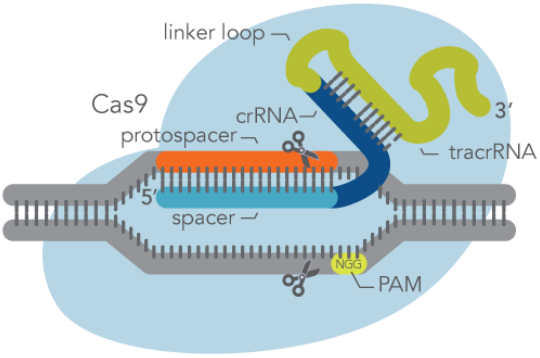

Figure 1: Visualization of the CRISPR/Cas editing system. The gRNA guides the Cas protein to the target DNA segment and binds to the target region, allowing the Cas protein to cut the DNA and let the natural DNA repair processes take over.

1.1 Biological background

CRISPR gene editing is based on a simplified version of the CRISPR-Cas9 antiviral defense system found in bacteria [1, 24]. The tool is made up of two main components, the Cas protein and the guide RNA (gRNA). The gRNA is a single stranded sequence consisting of a tracrRNA and crRNA. Both RNA and DNA are made up of building blocks called bases/nucleotides. Unlike RNA, DNA is double-stranded, with each complementary base from the opposite strand bound together to form a base pair. A base called adenine (A) pairs to the base thymine (T), and the base guanine (G) pairs with cytosine (C). RNA is made up of the same bases except for thymine, which is replaced with uracil (U), which also pairs with adenine. The Cas protein is an endonuclease protein that catalyzes the cleavage of the DNA. There are many types of Cas proteins found in bacteria, that help to defend against viruses. The Cas9 protein is the most widely used in gene editing experiments, as it can be easily programmed to locate and bind to a target region of DNA by supplying it with a specific gRNA. In particular, the crRNA, making up part of the gRNA, contains a 20-nucleotide sequence complementary to the target segment of the DNA, called the single guide RNA (sgRNA), which guides the Cas protein to the region of interest. At the target site, the Cas protein unravels the DNA helix, the weak bonds between the base pairs of the double stranded DNA break and the 20-nucleotide RNA sequence binds to a single strand of the target DNA by the formation of base pairs between complementary bases. This enables the Cas protein to make a double-stranded

break in the DNA. The targeted change is introduced by the cell's natural DNA repair mechanism when it repairs the cut. To prevent the Cas protein from cutting arbitrarily in the DNA, there is a short DNA sequence called a PAM that is located adjacent to the target DNA sequence on the opposite strand to which the sgRNA binds. If the Cas protein does not identify the PAM next to the target DNA sequence, it will not perform a cut. An example of a possible PAM sequence is NGG, where N can represent any nucleotide [10]. Thus, the guide RNA needs to be designed specifically for each target DNA segment, but this is a very delicate procedure, since the performances can vary widely for different sgRNAs.

1.2 Problem definition

To allow for the Cas protein to successfully bind and cleave the target DNA sequence, the user-defined RNA sequence must be designed to maximize efficiency and specificity. Efficiency measures how well the guide RNA binds to the target DNA segment, while specificity measures how often the guide RNA binds to regions outside of the target location (known as off-target regions).

1.3 State of the art CRISPR tools

Several state-of-the-art machine and deep learning methods have been developed to successfully predict on-target guide RNA activity and measure off-target effects. In [27], the authors developed a convolutional neural network, CRISPRon, which exhibits significantly higher prediction scores on unseen guide RNAs compared to existing methods. For example, CRISPRon outperforms another well-performing model, DeepHF, a Bidirectional LSTM (P-value < 0.001 of the two-sided Steiger's test) [23]. Transfer learning has also been used to try improve prediction of on-target guide RNA activity. For example, in [4], the authors developed a model named DeepCRISPR by using unsupervised learning to train a deep convolutionary denoising neural network-based autoencoder to automatically learn the underlying feature representation of the guide RNAs. The autoencoder was pretrained on a large-scale unlabelled guide RNA data source. The encoder was then fine-tuned on labelled guide RNA data and its output was used as input for a CNN.

Furthermore, numerous online guide RNA design tools have been developed to predict on-target guide RNA activity and measure off-target effects. For example, the authors of [27] developed separate on- and off-target online prediction tools. The authors of [4] unified their on- and off-target prediction tools into one framework to optimize guide RNA design. Although these existing tools implement state-of-the-art models that provide accurate gRNA efficiency and specificity predictions, they have certain drawbacks. For example, none of the tools excel when using heterogeneous data, as most models used for predictions have been trained on cell-specific or species-specific data and the tools cannot be customizable to different experiment settings. For example, DeepCRISPR focuses on conventional NGG-based sgRNA design for SpCas9 in humans. CRISPRon has been extended to cater for cross-species experiments; however use cases are still limited to experiments using NGG PAMs and SpCas9 proteins.

The CRISPor guide RNA design tool does well in combining functonalities of multiple existing tools and has been considered to be the best tool for designing gRNAs, as it caters to 417 genomes and 19 PAM types, meeting the needs of most users [14]. However, similar to most other gRNA design tools, CRISPRor lacks an integrative gRNA ranking system that takes into account specificity and efficiency scores and gRNA competition for the target gene location (that is, gRNAs that target the same location are undesirable as they will compete for the same location during CRISPR experiments). In fact, to the best of our knowledge, we have not come across a design tool that implements such an integrative guide selection procedure.

Another notable drawback of existing state-of-the-art design tools such as CRISPRon and DeepCRISPR is the inability to reproduce experiments using the existing software on GitHub. For example, neither of the software tools provide both raw and cleaned versions of the data sets they used and their associated splits to allow for reproducibility of experiments. In addition, neither of the software tools provide extensive guidelines for outside contribution and do not prioritize ease of use. For example, both repositories provide very limited documentation on their source code and on how their software tools should be used.

1.4 Technical prerequisites

We used GitHub for collaboration, code-synchronization, and potential code publication. To prevent code conflicts, we improved each module in separate feature branches and combined them towards the end of the project. Pre-commit hooks, i.e. autoflake, short, black[1], automatically advanced the code quality, cleanliness, and readability. In addition, we use extensive logging in the code, to make it easier to follow and debug the pipeline. Besides, we focus on a modular implementation into classes and simple class interfaces, so that the framework can be easily adjusted, improved, and extended.

Moreover, we use Hydra[2], a Python parameter configuration framework for simple configuration and easy experimentation. The framework allows extensive configuration and thus offers a lot of flexibility to the user. Each module is configurable by changing parameters in the corresponding configuration file. Hydra does not only simplify the use, but also tracks parameters for each experiment, supporting reproducibility. We also incorporated the functionalities of the Hydra Slurm plugin, which provides a user-friendly way of running scripts on remote computer clusters.

Since our project will be open-sourced, enabling outside contribution, we have also developed extensive documentation on all python scripts as well as a descriptive step-by-step guide to allow easy usage of the software. All experiment datasets and their associated splits are available for experiment reproduction. The packages needed to execute the pipeline are installed in a conda[3] environment, which can be easily cloned, to reproduce the same results. All in all, the design choices allow easy usage and understanding of the

[1] isort, autoflake, black
[2] `https://hydra.cc/`
[3] `https://docs.conda.io/en/latest/`

framework, fast configuration and experimentation, and simple extensibility. This is a very good foundation for experimental use and improvement.

1.5 Goals of the project

With the use of advanced machine learning and deep learning methods, the goal of our project was to create a toolbox that designs highly efficient and specific guide RNAs that can be successfully used for CRISPR gene editing experiments. Thus, our project was divided into two main components: on-target and off-target. For the on-target and off-target components, the goal was to utilize state-of-the-art machine and deep learning methods to accurately predict guide RNA efficiency and measure off-target effects, respectively. Figure 2 provides an overview of the toolbox developed in this project.

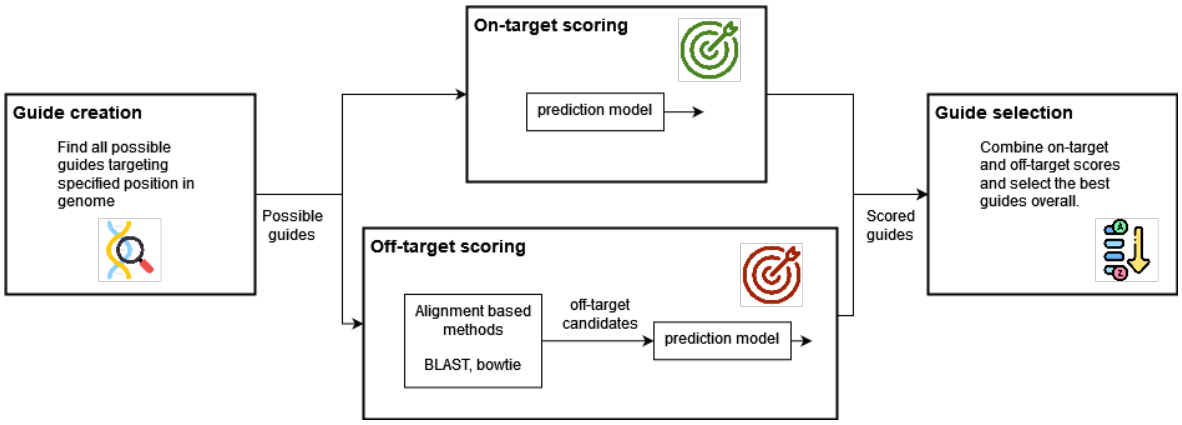

Figure 2: Pipeline steps for the guide design framework

2 panCRISPR Toolbox

A framework supporting biologists in picking well-suited guides for their experiments needs to combine multiple functionalities into one computational pipeline. This pipeline includes the initial search for possible guides targeting a specified gene in a user-defined genome, scoring these guides' effectiveness and specificity, and selecting the best ones for the user. Figure 2 depicts how the developed framework links these three steps together, offering a complete guide design pipeline.

2.1 Guide Creation Module

Guide creation comprises three phases, specifically, data retrieval, data preprocessing and data creation. The first stage involves setting up a persistent connection to the FTP server of ENSEMBL, that is a main source of human genome data in the pipeline, those are:

1. Homo_sapiens.GRCh38.dna_sm.toplevel.fa - contains the top level human genome.
2. homo_sapiens.gtf - human annotation file, which includes information about the position of the specific genes in the human genome.

Upon retrieval of the aforementioned files, annotation file is filtered based on the user input genes. The resulting truncated annotation is then applied to extracting the subsequences of the genome, which have the length of 20 bp, which are also followed by the specified PAM sequence. Strands of the negative sign are taken into account b considering the reverse compliment of those. The melting temperature and GC content - important biological features are calculated as well as positional features - position of the guide and the position of the exon. The resulting sequences are saved into the .fa files. The pipeline utilizes state-of-the-art python packages to concurrently process the data in a parallel manner as well as it allows simple usage by means of the agile configuration hierarchy.

The guide creation simply computes all potential guides without focusing on guide effectiveness yet. Therefore, the guide scoring step tries to evaluate the potential guides to be able to find the most desirable ones afterward. A desirable guide should be efficient and accurate. That means, first, it should bind to the target site with a high probability (on-target efficiency). Second, it should bind to as few non-targeted sites as possible (off-target activity). These two effects are evaluated and predicted in separate modules and combined afterwards.

2.2 On-target Module

The main goal of the on-target module is to estimate the knock-out efficiency of our guides. Alternatively, such an assessment when performed experimentally requires a lot of resources and uses complex experiments which tend to be expensive. However, exclusive experimental assessment tends to be unrealistic considering the growing number and the different area of applications of the CRISPR toolkit. Moreover, little is known on what makes guides more efficient than other due to the lack of data available for knockout experiments, and the lack of understanding of the whole human genome.

As an alternative, our on-target activity prediction relies on learning algorithms using the limited amount of open-source knock-out data. These models should be able to understand and to depict hidden pattern and properties of more efficient guides in terms of on-target activity.

2.2.1 Data

Data acquisition

To train the on-target models we have used the data published in the papers of Wang [25], Hart [8] and Doench [5],which are genetics screens performed using CRISPR-Cas9 on different human cell lines. In particular we have used the data of the following cell lines:

Publication	cell lines	size of sgRNA library
Wang	hl60, kbm7	$\sim$ 70 000 gRNA
Hart	HCT116, GBM, HeLa, RPE1	$\sim$ 90 000 gRNA
Doench	A735	$\sim$110 000 gRNA

The datasets contain the guide RNA sequences, the gene they belong to and the relative read counts taken at the beginning and during the experiment, where the read counts represent the abundance of a specific gRNA. Efficient sequences show a noticeable decrease in the read counts during experimentation. Specifically, the datasets contain the following information about the read counts.

- Wang: read counts taken at the beginning and at the end of the experiment in 1 replica.
- Hart: read counts taken at different time stamps (e.g. after 8, 12, 15, 18 days) and the experiments were conducted in different replicas (2 or 3).
- Doench: the data comes directly in log fold change (LFC) format, which is explained in more detail below in ([1]).

Data processing

Combining datastes of different cell lines for training should provide better generalization performances, as this prevents models dfrom learning the *efficiency prediction rules* of one specific cell line. For this reason, we wanted to make the data from the three experiments compatible, hence, we applied similar processing pipelines.

Firstly, we filtered the sequences with low initial read counts (< 200) as in [27], to ensure that enough gRNAs was present in the experiment. Since the dataset provided by Doench contained only the LFC, here we discarded the sequences having a high efficiency score (> 4), as suggested by the author. In fact, from the definition of LFC mentioned in ([1]), it is easy to see that low initial read counts can generate high values of LFC.
Then, the read counts were normalized with the procedure used by Doench, that is:

$$\text{read count}_{norm} = \frac{\text{read count} \times 10^6}{sum},$$

where *sum* is the sum of all the read counts in the same condition (e.g. initial or final) as the read count considered.

The dataset of Hart contained more detailed information about the experiment, so we applied additional filtering strategies in order to obtain cleaner data. We chose as final time stamp the one which showed better correlation between the different replicas and we used as final read count the the average of the different replicas. To compare the correlation of the different time stamps, we used both a scatter plot to have a visual representation and Pearson correlation coefficient to have a comparable value. Generally, the best time stamp was between 8 and 12 days.

Moreover, we considered as outliers and deleted the sequences whose difference between the replicas was higher than a certain threshold. The threshold was chosen specifically for each cell line such that it didn't delete too many sequences, since small fluctuations are admissible.

To translate the read counts into a predictable value, we used as efficiency metric the $\log_2$ fold change (LFC) of the read counts, which is a common metric in the literature [5, 4, 7] and is defined as:

$$\text{eff score} = \log_2\left(\frac{\text{read count}_{final}}{\text{read count}_{initial}}\right). \tag{1}$$

Then we applied the following processing steps to the efficiency scores to make them comparable across different genes and different cell lines (fig. 3).

1. Genes having less than 3 sequences were deleted in order not to affect the final evaluation of the models. The evaluation metrics given in Subsubsection 2.2.3 check the similarity of the ranks of the sequences w.r.t the true efficiencies of a gene and w.r.t. the predicted efficiencies, and with just one or two sequences also bad models could receive good scores.

2. If the scores of the sequences of the same gene showed an extremely low variance, than we deleted those sequences. The ranking based metrics are not suited to test the performances of the models in these situations, since the sequences are all almost equally good and the models wouldn't learn from these sequences.

3. Different genes could have biased efficiency scores for several biological reasons (e.g. how accessible is the DNA region due to DNA compaction). To reduce the bias and to make the efficiency scores comparable across different genes we applied to them a gene-wise median normalization.

   $$\text{eff score}_{norm} = \text{eff score} - median_{gene},$$

   where $median_{gene}$ is the median of the LFC of sequences belonging to that gene.

4. Finally, as in [4], we applied a descending rank based normalization to the efficiency scores of each cell to set that high efficiency scores mean high efficiency and to shrink the distribution of the scores to $[0, 1]$, allowing us to combine datasets of different experiments. Moreover, the condition eff score > 0, necessary to apply the NDCG metric, is fulfilled.

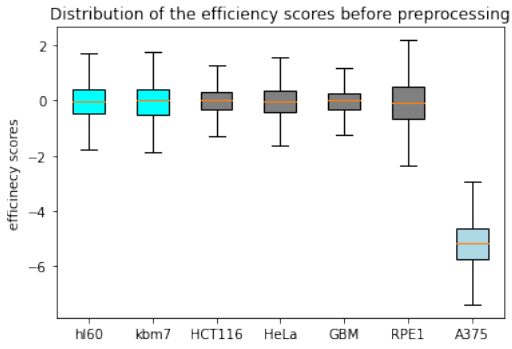

Figure 3: The distribution of the efficiency score before preprocessing: the ranges of the scores change between different cell lines and for this reason the scores are not comparable. Therefore, the processing pipeline is fundamental to combine the datasets.

2.2.2 Models

The models used are regression models, which aim is to predict the on-target efficiency score of a given sequence. Since in the literature there is not yet a clear winner between shallow models ad deep models for this specific problem [15], we trained and compared both the architectures.

Shallow Models

The shallow model we selected is a tree-based Gradient Boosted Regressor (GBR). The features used by the models to make the predictions are directly computed from the sequences in the following ways.

- Positional independent features: number of occurrences of n adjacent nucleotides (e.g. AC).
- Positional dependent features: binary features indicating whether n adjacet nucleotides appear at a specific position.
- n-Gapped features: the number of times that 2 given nucleotides appear at a certain distance (e.g. A _ _ _ _ C). These first three kind of features were used in [15].
- Important biological features ($*$), namely GC content and gRNA melting temperature, were computed using Biopython[4] and used in both shallow and deep models. These biological parameters are considered key features for predicting on-target gRNA efficiency [23, 27].

These features have different natures (binary, counting, tempretaure) and belong to different domains. Therefore, it is necessary to normalize the features to ensure that the distribution of each feature in the training set has mean 0 and variance 1. This step was implemented with the StandardScaler class of the library scikit-learn[5], which computes the normalization parameters from the training, saves them, and uses them in the evaluation process to normalize the features of the input sequence. Training the StandardScaler only on the training data does not leak any information from the training set.

Deep Models

We implemented two deep learning models, a baseline fully connected neural network and a convolutional neural network with the same structure as CRISPRon, an existing top-performing model in the literature [27]. The baseline neural network (baseline_nn) has two hidden layers and each linear layer (except the last layer) is followed by a Leaky ReLu activation function and then batch normalization. Adding leaky ReLu prevents the "dying ReLu problem". Batch normalization aids in generalization of the model and, just like Leaky ReLu, has been shown to accelerate training.

The convolutional neural network (CRISPRon) builds upon baseline_nn by adding convolutional layers, whose output is flattened, concatenated with the biological features($*$),

[4] https://biopython.org/
[5] https://scikit-learn.org/stable/

and fed into fully connected layers. The detailed achitecture can be seen in Fig. 4

Both these neural networks take a one-hot-encoded sequence as input and output an efficiency score for each sequence as the final output. The one-hot-encoding (Fig. 4) is a 1-dimensional image with 4 channels, one for each nucleotide, and each pixel represents which nucleotide has the sequence in that position (i.e. the channel corresponding to the nucleotide is 1, the others are 0).

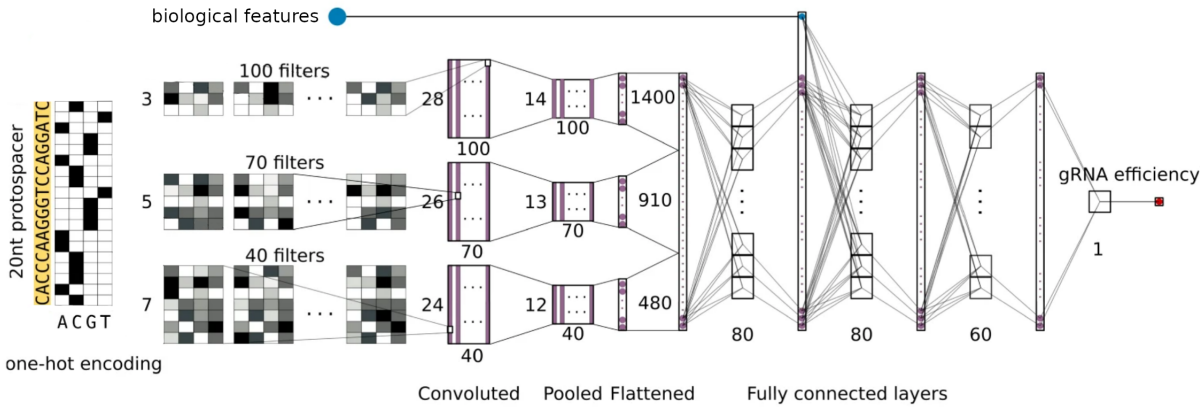

Figure 4: CRISPRon architecture with one-hot-encoded sequences reproduced from Xi Xiang et. al. [27]

Training strategy

To train the models, we split the preprocessed data into train, validation, and test sets with a splitting ratio of approximately 60/20/20. The splits have been computed w.r.t. the genes, that is the three splits contain sequences belonging to three disjoint sets of genes. This ensures more reliable evaluation results, as the sequences in the test and the validation sets belong to genes completely unseen by the models.

The hyperparameters of the models were finely tuned using the evaluation metrics results on the validation set. For the deep models, we monitored the model performance during the training using the MSE calculated on the validation set (validation loss). To prevent over-fitting on the training data, we implemented early stopping with a patience set to 20 such that when the validation loss does not decrease after 20 epochs, the training will end.

2.2.3 Evaluation metrics

By definition, the on-target activity prediction task should highlight which among the guides is more efficient. A suitable metric, should hence, attribute high scores for models where the predicted efficiencies of a set of guides follow the same order as the actual efficiencies of these guides. Such metrics are referred to as rank-based correlation and are widely used in the literature such as in [5, 8, 25].

The most commonly adopted correlation measure in the literature is called Spearman-Rank Correlation Coefficient, r_s, [18]. It provides a measure for the statistical dependence between the ranking of two variables. While Spearman's correlation assesses the monotonic relationship between two variables, it uses Pearson's correlation to assess the linear relationship between the respective rankings of the two variables. It can be computed as the following

$$r_s = \frac{R(X)R(Y)}{\sigma_{R(X)}\sigma_{R(Y)}}$$

where X,Y represents respectively the ground truth and the predicted efficiency scores. R(.) denotes the rank function.

Another less commonly used metric is called Normalized Discounted Cumulative Gain, NDCG [22]. It measures the ranking quality and is often used in information retrieval. DCG can be used under the following assumption: "highly relevant documents are more useful when appearing earlier in a search engine result list". It uses a rank logarithmic discount factor to penalize relevant documents that appear at the end. The formula is as follows:

$$NDCG_p = \frac{DCG_p}{IDCG_p}$$

, where $DCG_p = \sum_{i=1}^{n} \frac{eff_i}{log_2(i+1)}$ and $IDCG_p = \sum_{i=1}^{|REL_p|} \frac{eff_i}{log_2(i+1)}$

, with REL_p representing the set of the most efficient guides up to position p.

NDCG compared to Spearman correlation, can be parameterized with an integer p to only focus on the p most efficient guides. Selecting a model with a relatively high $NDCG_p$ means that the model is likely to perform better on the top p efficient guides. On the other hand, Spearman Correlation considers the whole set of guides and penalizes rank mismatches equally among all positions. In guide selection, our aim is to reduce the number of guide candidates to a smaller, more efficient set. NDCG seems to offer an advantage over Spearman Correlation, and hence it has been used in the evaluation of our models.

Considering that our test sets from different cell lines contain guides targeting different genes, we decided to run the evaluation gene-wise. That is, we calculate the correlation scores with respect to each gene, and then we average the gene-wise score to get the final model evaluation. Moreover, while focusing on the NDCG correlation, we set $p = 1$ to put more emphasis on the top ranked gRNA.

2.2.4 Results

The models implemented were tested in two different experimental set-ups to thoroughly investigate their performances in possible use cases.

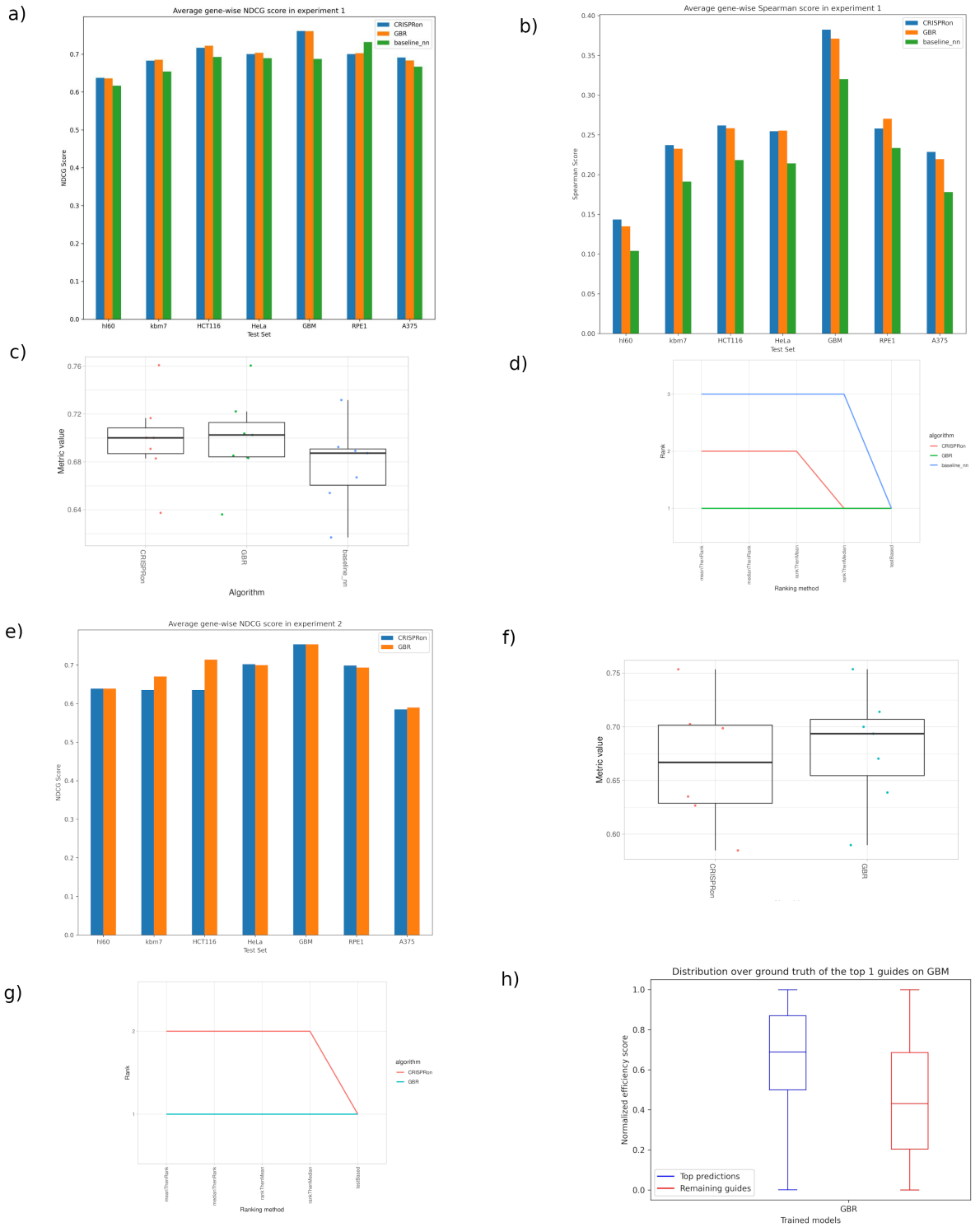

Figure 5: Results showing a comparison between the implemented models in Experiments 1 and 2. (a) Comparison of the average gene-wise NDCG scores and (b) Spearman correlations of the implemented models trained under the setting of Experiment 1 and tested on each cell line. (c) Box-plot (from ChallengeR) of descriptive statistics for NDCG scores over all test sets of each model under the setting of Experiment 1. (d) Comparisons of outcomes of five different ranking strategies (from ChallengeR) for Experiment 1. (e) Comparison of the average gene-wise NDCG scores of the implemented models trained under the setting of Experiment 2 and tested on each cell line. (f),(g) same as (c),(d) for Experiment 2. (h) Boxplots comparing the distribution of the ground truth efficiency scores of the top ranked guides with the remaining guides for the GBM cell line.

Experiment 1. In this experiment we combined the training sets of all the cell lines in one data set and used it for training. Then we tested on the test set of each cell line (fig. 5a, b). To rank our models trained on all cell lines based on their performance on each test set, consisting of a single cell line, w.r.t. the NDCG score we utilize the challengeR visualization toolkit [26] (fig. 5c, d). For more details about the ranking of the plots as well as a detailed explanation of all graphs obtained using the ChallengeR toolbox see the Appendix. Fig 5d shows the ranking of the model across different ranking strategies and the GBR and the CRISPROn architectures are ranked as the best models, but the test cannot determine a clear winner. However, under the first 3 conditions GBR is ranked as first.

Experiment 2. In this experiment, we combined all cell lines except one into a single data set on which we trained our 2 top performing models. The left-out cell line was used to test the transferability of the models to unseen cell lines (fig. 5e). AS above challengeR visualization toolkit [26] (fig. 5f, g) has been used to rank the two models and GBR is slightly better and for this reason we select it for the final pipeline.
Moreover, we used the Wilcoxon test to see if combining different cell lines gives better transferability than using only one cell line for training for the GBR model . We trained the model on a single cell line and tested on all the others. Our null hypothesis was that the NDCG scores of the models trained on a single cell line are higher or equal to to the scores obtained by the models trained using the method in this experiment. The p-values obtained are:

Table 1: p-values for the test: combining cell lines translates to better transferability

model	HCT116	HeLa	GBM	RPE1	hl60	kbm7	A375
GBR	0.03125	0.03125	0.078125	0.078125	0.03125	0.078125	0.78125

Finally, we wanted to visualize whether the models actually are able to detect which are the best guides. Using the model from Experiment 1, for each gene we selected the guide having the highest predicted efficiency score. Then we compared the distributions of the ground truth scores of the best selected guides and of the discarded guides (fig. 5h). We would expect that the distribution of the selected guides is shifted upwards, because this would imply that the guides the models predict as efficient are actually more efficient than the others.
Moreover, we used the Wilcoxon test to verify if the shift between the two distributions is statistically significant. In particular, for each discarded guide we subtracted from its ground truth efficiency score the efficiency score of the best guide for that gene:

$$\text{eff score}_{guide} - \text{eff score}_{best\ guide}$$

.
We tested with the null hypothesis that the median is greater or equal to 0 (i.e. the predicted guides are worse or equivalent to the discarded ones). The p-values obtained are:

Table 2: p-values for the test: selected guides are better than discarded guides

model	hl60	kbm7	HCT116	HeLa	GBM	RPE1	A375
CRISPROn	9.32E-86	5.59E-191	4.42E-125	4.52E-159	2.23E-218	1.65E-221	0.0
GBR	1.74E-89	1.44E-197	1.68E-140	4.71E-173	3.53E-218	4.68E-228	0.0

2.3 Off-target Module

Not only is the efficiency important in guide design but also its specificity, i.e. whether a guide has off-targets, as off-target cleavage by CRISPR/Cas9 can cause undesired and even harmful side-effects on non-coding regions of the genome. Due to their mismatch tolerance, gRNAs target sequences that are similar to the target site but not identical. These sequences are called off-target sequences (OTSs) or off-targets.
However, experimental data shows that guides don't bind to every possible off-target sequence, only to some of them. To this end, we train models on data produced in silico with alignment tools and labeled using in vitro verified data to classify off-targets as true or false, depending on whether they were found in the experimental data or not, respectively. Trained models are then used for off-target prediction in the main framework.

2.3.1 Data

The off-target module is centered around the experimental data [29], that contains off-targets verified in vitro with the Circle-seq method [20]. It has 3095 records of off-target data with the following information: sgID (guide name), gRNA, OTS, Chr (chromosome, in which OTS was found), Strand (forward or reverse strand of DNA), Mismatch (number of mismatches), Start and End (positions of OTS in the genome). It contains 11 unique guide sequences, and the number of mismatches varies from 1 to 7. The genome used for the paper is the human genome (hg19). The guide RNA sequences are 23 nucleotides long, 20 of which are the gRNA itself, and 3 are the NGG PAM sequence. Strand column refers to the two DNA strands, the reading direction is 5' to 3' and hence, there exist a forward (in 5'-3' direction) and a reverse (in 3'-5' direction) strand.
For our experiments we use the same guides and the same genome as the paper. We align the guides against the human genome using alignment tools (bowtie and BLAST).

2.3.2 Featurization

This section encompasses results from bowtie and BLAST, feature computation and encoding.

Bowtie

Bowtie [12] is a fast end-to-end alignment tool that runs on the command line. It is geared towards aligning short sequences (also called reads) to large genomes. Bowtie performs an end-to-end read alignment, which means that it searches for alignments involving all of the read characters. Bowtie allows alignments to have at most 3 mismatches.

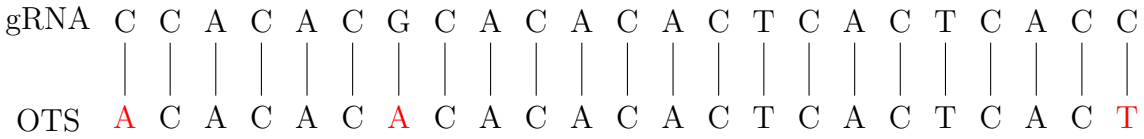

Figure 6: gRNA and OTS with 3 mismatches found by bowtie

Bowtie configuration

We wrapped bowtie into python so we can call it within our pipeline. All arguments for the bowtie command are passed through Hydra from a config file and then the command is generated and passed to the bowtie wrapper. The additional parameters with which we run bowtie are "-a" which instructs bowtie to report all alignments for each guide, and "-v 3" for bowtie to find all alignments with up to 3 mismatches.

Bowtie output

Standard bowtie output contains the following columns: sgID (guide name), Strand, Chr (chromosome), Start (start position), gRNA (guide sequence), Mismatch (its content is explained below). sgID contains the guide names we pass through the file with guides. Strand can be forward and reverse. gRNA is strand-dependent: for a forward strand it is reported in the 5'-3' direction, for a reverse strand, the sequence is reported 3'-5' and all nucleotides are replaced by their complements. Mismatch column contains a comma-separated list of mismatch descriptors, and a single descriptor has the format $offset : OTS - base > gRNA - base$, where offset is strand-specific (has the direction 5'-3' for a forward strand, 3'-5' for a reverse).

Bowtie OTS

In order to train a model to classify off-targets we need to feed it both a gRNA and an OTS, because only by having both can a model see the differences between the two. Both bowtie and blastn don't have OTS in their standard output. For bowtie we can construct OTS using gRNA, Strand and Mismatch columns. Using the contents of the Mismatch column (described in 2.3.2 Bowtie output) we can replace gRNA-base of gRNA with OTS-base of OTS in the offsets they mismatch, taking into consideration that for forward strands the positions/offsets are indexed in forward direction, and for reverse strands in reverse direction, which will yield OTS.

Bowtie results

In order to compare results of bowtie with the experimental data we merge the databases with pandas[6] on the four matching columns: sgID (guide name), Chromosome, Strand, Start. During the analysis of the bowtie output, we noticed the number of true off-targets was incomplete. Further investigations suggested the cause was due to the 1. Incorrect processing of the input sequencing file containing gRNAs by bowtie, and 2.

[6] https://pandas.pydata.org/

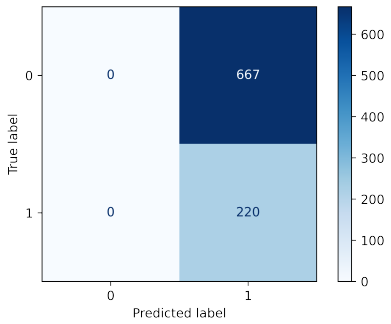

Figure 7: Confusion matrix of bowtie predictions for off-targets with <4 mismatches

Wrong information about the number of mismatches in the experimental data provided. Fixing the issues gave us 220 true off-targets and drastically reduced overall number of alignments (from 16190 to 887). To conclude, bowtie contributed the most data for off-target models with 887 records, 220 of which were true off-targets.

BLAST

BLAST (Basic Local Alignment Search Tool) is a widely used program to efficiently align a sequence against a database and find meaningful matches that share some similarity. Alignments can be performed for DNA or protein sequences, but in this case, DNA sequences are compared (BLASTn). As visualized in figure 8, instead of computing costly pairwise alignments with each sequence in the database, BLAST first finds high scoring words in the database, extends those, and filters significant matches with a high score and low e-value. For efficiency, only word pairs that are found in close proximity are extended (two-hit method), as they are more likely to yield a high scoring extension. Lastly, gaps are also introduced by the gapped BLAST algorithm. [3, 2]

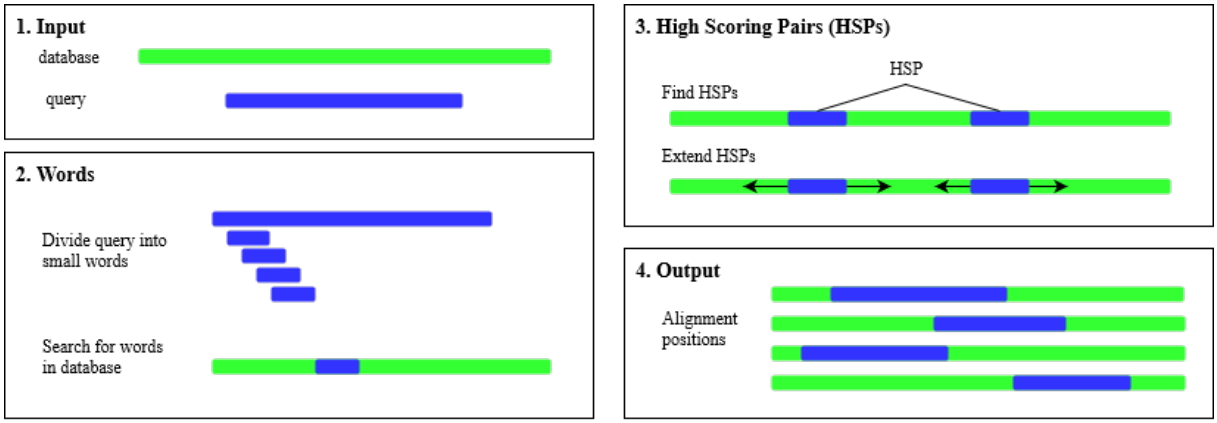

Figure 8: BLAST sequence alignment algorithm visualization

BLAST configuration

Even though it is widely used in bioinformatics, BLAST comes with a few caveats.

Firstly, BLAST aims to find functional similarity between sequences, for example to detect gene families. Therefore, an alignment with a high similarity score represents sequences that are likely functionally or evolutionary connected. Similarly, the e-value computed for alignments by BLAST denotes the probability to find another alignment with a greater similarity score in the searched database. [3, 2] According to current literature, off-target effects do not depend on evolutionary or functional relationships, though, but on the number and positions of mismatches as well as some molecular features (i.e. melting temperature) [5, 16]. Consequently, BLAST by itself will not be a good indicator for off-target activity and needs to be complemented with other ways of prediction.
Secondly, BLAST is a complex program with many parameters that adjust the search to different use-cases. In addition, the alignment is time consuming. So, a good set of parameters should be restrictive enough to reduce the runtime and prevent an explosion of alignments, while at the same time allow to detect as many experimentally confirmed off-targets as possible. A grid search determined the parameter set to be: `word_size=7` and `evalue=100`. Nevertheless, the framework allows the user to adjust any BLAST parameter manually in the configuration file if needed.

BLAST parsing

BLAST returns the alignments and corresponding features, such as the alignment position, its score and e-value, in a tab separated file. The format and returned features differ from the output generated by bowtie. So, to combine results from both programs, the BLAST data needs to be parsed and processed accordingly. First, BLAST does not necessarily align the whole sequence, but it can align parts of it. Thus, the parsing step recomputes the reported number of mismatches to incorporate the whole guide.
Second BLAST only reports sequence positions instead of returning the aligned sequences themselves. Therefore, we developed a way to read out the sequences from the genome's fasta file, similar to section 2.1. This approach is much faster than using the built in `blastdbcmd` tool. The sequences are retrieved from the genome fasta file in three phases:
1. Load annotation file: The annotation file provides indexing for the genome fasta file for quick access with information about where certain nucleotide ranges are saved in the fasta file.
2. Map ranges to OTS: Next, to know which ranges to access, they need to be mapped to sequences lying in these ranges. Using pandas[7], the sequence dataframe can be merged with the range dataframe to achieve the mapping. But as the files are large we merge the dataframe in chunks and decrease the allocated memory.
3. Load data from fasta file: Lastly, we load the identified ranges from the fasta file and extract the OTS at the correct positions. Overall, this procedure decreases the runtime compared to `blastdbcmd` by 50 to 100 times depending on the number of OTS, and allows a feasible OTS retrieval.

BLAST results

Comparing BLAST alignments with experimentally confirmed off-targets from [29], proves the shortcomings of BLAST. The confusion matrix in figure 9 shows that BLAST finds

[7] `https://pandas.pydata.org/`

both a lot of false off-targets, and also only a small percentage of the experimentally confirmed off-targets.
At the same time, the feature correlation matrix in Figure 10 indicates the correlation between some BLAST features and the alignment being an experimental off-target. So, the off-target effect seems to have no obvious relation with the features reported by BLAST.

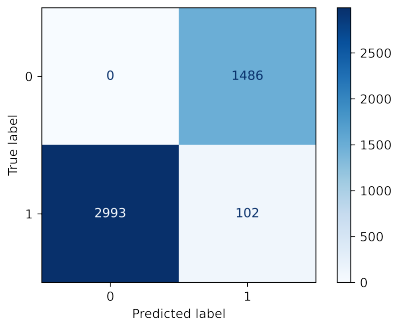

Figure 9: Confusion matrix for BLAST off-target prediction on experimental data from [29].

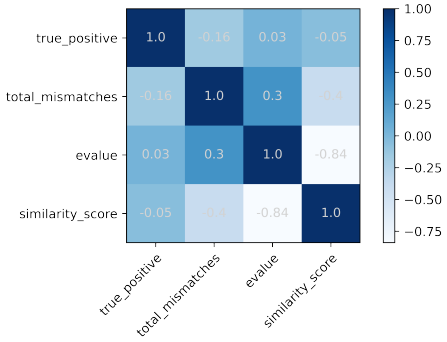

Figure 10: Feature correlation matrix for BLAST features, using standard correlation coefficient.

As a result, BLAST on its own, cannot be used to predict off-target effects reliably. But BLAST offers one advantage compared to bowtie: it reports alignments with more than three mismatches. Thus even, with its shortcomings, it provides important additional data for data preprocessing.

Features

As part of the feature engineering process, gRNA and OTS are selected to be used later for training models, as they carry the most relevant information. Additionally, GC-content and melting temperature are computed for both the gRNA and OTS, as important biological features that were shown to impact off-target activity of guides. GC-content is the percentage of G and C nucleotides in a sequence. Melting temperature refers to the temperature at which a DNA double helix dissociates into single strands. There are several formulas for computing melting temperature for a sequence, we use the one based on nearest neighbor thermodynamics. Both the biological features are computed using Bio.SeqUtils[8] package of Biopython.

gRNA-OTS pair encoding

The gRNA and OTS are encoded as a pair using a 6-bit encoding scheme, that leverages directional encoding [13]. Each sequence pair is considered as a fixed length vector with the four-bit channel (A, G, C, T) and two-bit direction channel. The four-bit channel preserves the nucleotides of the on-target site and the off-target site while the direction channel

[8] https://biopython.org/docs/1.75/api/Bio.SeqUtils.html

is designed to identify the mismatch directions; for example, "00110-10" represents the mismatch "G → C" ("G" is the on-target site and "C" is the off-target site) while "00110-01" represents the mismatch "C → G". The proposed encoding schema significantly reduces the coding length from eight bits to six bits, which can be advantageous given the amount of the off-target data we have.

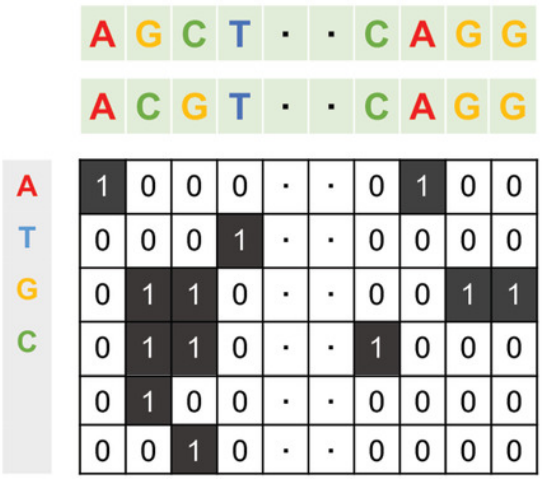

Figure 11: 6-bit encoding scheme

2.3.3 Models

Both shallow and deep models were trained for the binary classification task of whether an off-target found by the alignment tools is true or not.

Shallow Models

Several shallow models were trained, including a support-vector machine (SVM), a decision tree, a multilayer perceptron (MLP) with a few hidden layers, and a random forest. All shallow models utilized the 6-bit encoding scheme described in 2.3.2. The features with GC-content and melting temperature were normilized.

Deep Models

A deep convolutional neural network was trained to hierarchically learn the weights of the positional features and combine them with the output of the fully connected network, that processes the non-positional features. The features are first encoded, by means of one-hot-encoding of the sequences of the base pairs (single nucleotides, pairs, triplets and so on). For the sake of dimensionality reduction only singletons were taken into consideration. The positional features are then processed by the deep convolutional neural network, combined with dropout and parametric relu. A fully connected neural network was trained in parallel to combine its output with the output of the convolutional neural network and eventually produce the logits using the final fully connected layer on the end.

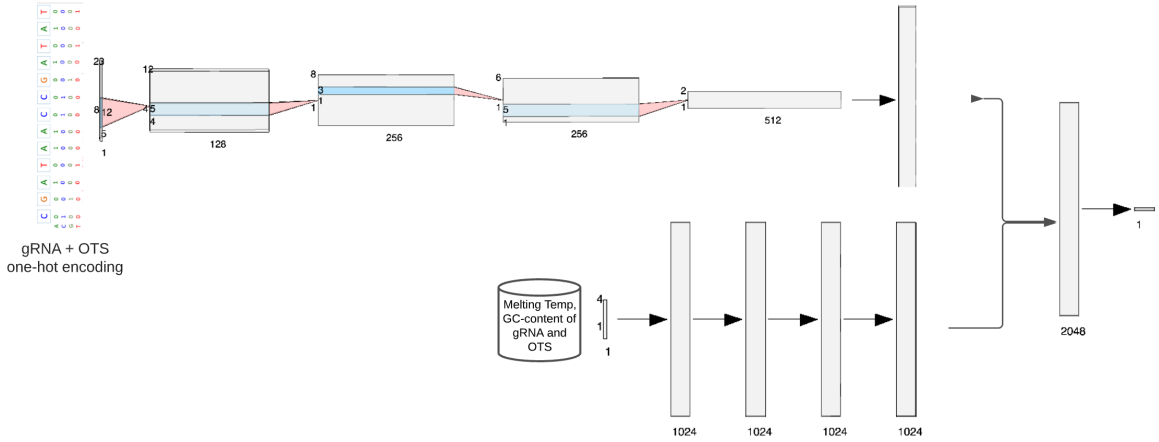

Figure 12: Convolutional neural network

Training strategy

The models were trained using the split over the training samples of 80/20 and the held-out sample for testing. A standard cross validation was used to split the data into n shards and validate on the cut-out shard. The models were then being compared based on the mean metric over the splits. For training the deep neural network, a weighted categorical cross entropy was used - specifically weights of [1., 2.] for non-binding and binding. Due to the high risk of the over-fitting certain approaches were used. To prevent over-fitting the following advanced approaches and techniques were utilized:

1. stochastic weight averaging - a technique to store the last models during the epoch and consider the exponential moving average of those;
2. learning rate reduction on plateau;
3. early stopping based on the validation metric;

Combining Models

During the training many models were taken into the consideration. After conducting hyperparameter optimization and choosing the best candidates for the deep and shallow models we arrived at the point where the models were close in their metrics. The final score of the models was about 0.73 F1 score. To reduce the variance of the models the weighted average over the predicted probabilities was employed, which allowed contemplating the models as an ensemble to be used during the final scoring phase.

2.3.4 Evaluation metrics

To meticulously estimate the probabilities of the OTS, is is essential to choose the preference of minimization between the false positives and false negatives to fall onto the second one. Since any unintended cleavage site may cause severe consequences for the experiment it is crucial to evaluate how well the model predicts those. One shouldn't also

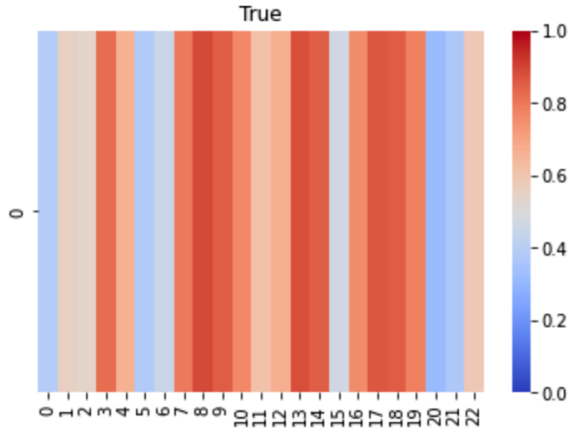

Figure 13: Occlusion of the guide, including the PAM sequence

forget about the false positives, since they can tarnish the experimental potential of the guide sequence by compromising it with the over-estimated number of the possible sites. This brings us to idea, that multiple classification metrics are to be applied. In this work we opted above all for F1-score, recall and precision. We were also monitoring the general confusion matrix.
To evaluate how well did the models actually learn the underlying specifics of the off-target effects, we tried to plot heatmaps of the occlusion of the nucleotides. This means, some of the nucleotides of the true OTS were hidden from the model and then it was offered to classify it. The higher the value of the nucleotide is, the more relevant was the occlusion of the gene to the final decision.
Figure 7 demonstrates, that indeed, removing the PAM sequence, dramatically reduces the probability of the OTS. At the same time some of the border nucleotides also play a decisive role as opposed to those in proximity of the middle point.

2.4 Guide selection Module

Following the guide creation, on- and off-target activity prediction, the guide selection step aims to filter out guides according to the trade-of higher on-target activity versus lower off-target activity. However, some guides might be competing over the same target sequence position of the genome. The latter might lead to experimental issues, hence we need to make sure the selected guides do not compete on the the binding sites. However, we also found out experiments where competing guides are allowed, especially the type of experiments where they optimize for activation. Below, we introduce our guides selection method for each use-case.

2.4.1 Guide selection without positional encoding

In the case where we allow competing guides, the selection problem becomes linear, and it suffices to pick the guides that shows the highest on-target activity and the lowest off-target activity. Considering two real numbers $0 < a, b < 1$ representing respectively on-target and off-target factors, we formulate the problem for n candidate guides as follows:

$$x^* = \arg \max_{x=[x_k]_{k=1..n}} x^T(as_{on} - bs_{off}) \quad \text{subject to } \Sigma_i x_i = p \tag{2}$$

Where x is a binary vector and each element x_i encodes whether i^{th} guide is picked, s_{on} and s_{off} denote respectively the normalized on-target and off-target scores, and p is the number of selected guides. a, b are initially set to 0.5 each, and can be fine-tuned when running the pipeline. On an implementation level, we calculate the overall score for each candidate guide as the linear formula $as_{on} - bs_{off}$. The p best scoring guides are then selected.

2.4.2 Guide selection with positional encoding

Adding the positional encoding in the objective function to avoid competing guides, increases its complexity to be quadratic. Before we dive into the new selection problem, we first present the conflict matrix $M = [M_{i,j}]_{1 \leq i,j \leq n}$ for n candidate guides.

$$M_{i,j} = \begin{cases} 1 & \text{if } i^{th} \text{ and } j^{th} \text{ guides are conflicting} \\ 0 & \text{else} \end{cases}$$

M is symmetric by definition and $M_{i,i} = 1$ for every i in $1...n$. In addition to the on-target and off-target factors a, b we also introduce a conflict penalty factor λ. The matrix M is created using the binding position information that was generated along with the guide creation step.
The main objective function is then defined as the following

$$x^* = \arg \max_{x=[x_k]_{k=1..n}} x^T(as_{on} - bs_{off}) - \lambda x^T M x \quad \text{subject to } \Sigma_i x_i = p \tag{3}$$

where again x is a binary vector and each element x_i encodes whether i^{th} guide is selected, s_{on} and s_{off} denote respectively the normalized on-target and off-target scores, and p is the number of selected guides.
The term $x^T M x$ represents the sum of the pairwise conflicts between guides. Clearly higher values of λ lead to less conflicting set of guides. However, λ should be carefully tuned so that the term $\lambda x^T M x$ will not be dominant in the objective function. The default value is set 0.5 that is equal to default values a, b.
In the following we present our attempts to solve the optimization problem.

Polynomial Constrained Boolean Optimization:
In order to solve the constrained optimization problem we used the python library qubovert[9]. The library provides a solver for Polynomial Constrained Boolean Optimization which can be used to solve the optimization problem. However, the solver considers the problem as

[9] https://github.com/jtiosue/qubovert

combinatorial and brute force through all combinations, hence its complexity is exponential. We also noticed that from around 30 guides the solver starts to take considerable amount of time, and by increasing it further the solver did not terminate running on an average computer.

Greedy Approach:
Alternatively, and since the last approach suffers from exponential running time, we decided to implement a greedy approach. Our solution considers the best scoring guide at every position and discard conflicts, and is detailed in the Appendix, Algorithm 1.
The function **DiscardConflicts**, shown in 1, ensures that in every iteration all the guides conflicting with the best selected guide are discarded from the pool. Although this approach doesn't surely converge to the global maximum, it runs much faster, hence can be useful in some cases.

3 Discussion

Overall, the on-target models showed promising results in performance and transferability. The differences in performance, shown in (fig 5a, b) in Subsubsection 2.2.4, are an indicator of the issues of the Spearman correlation related to our purpose, highlighted in (2.2.3). For this reason, we have considered only the NDCG score for the second experiment (Fig d).

The ranking results of the challengeR visualization toolkit [26] (fig. 5c, d, f, g) show that the GBR model is the most promising one, however CrisprOn has very similar performances and there is no statistical evidence in choosing the GBR. However, GBR, as a shallow model, is more suitable for an explainablility analysis, which is left for future investigations.

In Table 1, the p-values show that for 6 out of 7 cell lines the null hypotheses (the models trained on a single cell line have better transferability), can be rejected with a confidence level of 0.08. Given this, a model trained in Experiment 1 is likely to be the best when used with unseen cell lines, since it uses all the cell lines available in our data set. Therefore, it has been used to train the model that provides the efficiency predictions in the final framework.
The p-values in Table 2 are extremely low. This shows that the shift of the two distributions in fig. 5h is statistically significant and that, on our data, the gene-wise selection of the guides based on our model's predictions chooses guides that are at least above the average. This result is very encouraging because it gives a guarantee about the reliability of the predictions made by our models. Moreover, the performances obtained in Experiment 1 and Experiment 2 are, apart from A375, extremely close to each other. Hence our models show a surprisingly good transferability to unseen cell lines.

The ranking of the off-target models was based on the comparison on the test sample, allocated beforehand. To score the models, the results were compared to those of the alignment based approaches to certify that the machine learning methods do indeed re-

veal concealed patterns.

In the period many models had been considered. Among those to be mentioned were support vector machine, utilizing different positional encodings and biological features. Those models performed significantly worse than those of the models offered in the work. We conducted the tests to verify, the models had outperformed the non-machine learning methods. This allows us to conclude, that the offered approaches could be used as an alternative to the commonly used alignment methods. To check the models generalization capacity, several approaches were used. For example guides occlusion during the training and verifying on the unseen guides.

From a technical perspective, only a few requirements limit the framework's usage. The limiting factors are mainly runtime and storage. There must be at least enough space for the genome and BLAST and bowtie databases on the disc. This can be a lot for personal computers, becasue e.g. computation on the human genome requires more than 100GB disc space. At the same time, especially the BLAST runtime increases quickly with the amount of guides. While only a few guides take seconds to align, hundreds of guides can lead to a BLAST runtime of 30 minutes or more, decreasing the framework's practicability.

4 Conclusion

To conclude, in this project we developed a full-fledged, and flexibly configurable framework for CRISPR guide design. While there are multiple approaches in the literature on both off-target and on-target prediction, only a few known platforms combine this with a guide creation and guide selection step to enable a complete pipeline from target selection to the selection of the best guides. Thus, we implemented guide creation, guide scoring and guide selection modules and streamlined them into a single pipeline. The guide scoring module follows state-of-the-art approaches for on- and off-target prediction by training and comparing several shallow and deep models in both cases. As often done by similar programs [11], the guide selection module ranks guides based on their score. But on top, the module offers selection algorithms to pick best guides without target overlaps, too.

Besides its functionality, the framework stands out from other tools because of its architecture and coding practices, too. First of all, the project is open source and due to its modularity, configurability, and clean code, the framework can easily be improved and built upon by anyone. This does not only provide reproducibility, but also an excellent experimental playground for researchers in this field, to test new models quickly. With the inference pipeline completely connected, developers can adjust or exchange each module separately, taking advantage of the framework's great extensibility.

5 Outlook

Future work can focus on improving each separate module. First, by adjusting the guide creation step one can allow more specific target selection possibilities. For example, the module can offer targeting only certain PAMs, sub-regions of genes or sub-regions of exons or pre-filtering guides by optimal GC-content ranges. Second, testing more models and training on additional data from other species [6, 9] or from more human CRISPR screening databases [19, 17] can increase prediction model accuracy. Also, including other types of features, such as epigenetic marks or gene expression data, might enhance the model performance.

In addition, code improvements for robustness against exceptions and runtime reduction would lead to a smoother user experience. Thus, each module should tolerate wrong user input and missing files or parameters with more argument and sanity checks before execution. Adding a testing framework and automating unit and integration tests can support in eliminating errors and uncovering edge cases. Optimizing the BLAST execution is the most important task to reduce the overall runtime, because the alignment tool takes up most of the pipeline's execution time. To do so, BLAST could be run in parallel mode and with more RAM, or a larger wordsize and smaller evalue could be chosen. Besides, several pipeline steps, such as the off-target and on-target prediction are parallelizable.

While the current framework solely focuses on the guide design for CRISPR knockout experiments, extensions could offer guide designs for other CRISPR experiments, too. CRISPRi and CRISPRa experiments, for example, require different guide properties, which need be addressed by adjusted prediction and selection modules.
Lastly, apart from improving the framework's performance, adding visualization, a graphical user interface, and explainability tools will be crucial for the tool to become helpful to biologists in their daily work.

Bibliography

[1] Mazhar Adli. *The CRISPR tool kit for genome editing and beyond. Nat Commun 9: 1911.* 2018.

[2] Altschul et. al. "Basic local alignment search tool". In: *Journal of molecular biology* 215.3 (1990). ISSN: 0022-2836. DOI: 10.1016/S0022-2836(05)80360-2.

[3] Camacho et. al. "BLAST+: architecture and applications". In: *BMC bioinformatics* 10 (2009), p. 421. DOI: 10.1186/1471-2105-10-421.

[4] Guohui Chuai et al. "DeepCRISPR: optimized CRISPR guide RNA design by deep learning". In: *Genome biology* 19.1 (2018), pp. 1–18.

[5] John G Doench et al. "Optimized sgRNA design to maximize activity and minimize off-target effects of CRISPR-Cas9". In: *Nature biotechnology* 34.2 (2016), pp. 184–191.

[6] Wataru Fujii et al. "Efficient generation of large-scale genome-modified mice using gRNA and CAS9 endonuclease". In: *Nucleic Acids Research* 41.20 (Aug. 2013), e187–e187. ISSN: 0305-1048. DOI: 10.1093/nar/gkt772. eprint: https://academic.oup.com/nar/article-pdf/41/20/e187/25361999/gkt772.pdf. URL: https://doi.org/10.1093/nar/gkt772.

[7] Maximilian Haeussler et al. "Evaluation of off-target and on-target scoring algorithms and integration into the guide RNA selection tool CRISPOR". In: *Genome biology* 17.1 (2016), pp. 1–12.

[8] Traver Hart et al. "High-resolution CRISPR screens reveal fitness genes and genotype-specific cancer liabilities". In: *Cell* 163.6 (2015), pp. 1515–1526.

[9] Vivek Iyer et al. "Off-target mutations are rare in Cas9-modified mice". In: *Nature methods* 12.6 (June 2015), p. 479. ISSN: 1548-7091. DOI: 10.1038/nmeth.3408. URL: https://doi.org/10.1038/nmeth.3408.

[10] Fuguo Jiang and Jennifer A. Doudna. "CRISPRâ€"Cas9 Structures and Mechanisms". In: *Annual Review of Biophysics* 46.1 (2017). PMID: 28375731, pp. 505–529. DOI: 10.1146/annurev-biophys-062215-010822. eprint: https://doi.org/10.1146/annurev-biophys-062215-010822. URL: https://doi.org/10.1146/annurev-biophys-062215-010822.

[11] Kornel Labun et al. "CHOPCHOP v3: expanding the CRISPR web toolbox beyond genome editing". In: *Nucleic Acids Research* 47.W1 (May 2019), W171–W174. ISSN: 0305-1048. DOI: 10.1093/nar/gkz365. eprint: https://academic.oup.com/nar/article-pdf/47/W1/W171/28880274/gkz365.pdf. URL: https://doi.org/10.1093/nar/gkz365.

[12] Ben Langmead et al. "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome". In: *Genome Biology* 10.3 (Mar. 2009), R25. ISSN: 1474-760X. DOI: 10.1186/gb-2009-10-3-r25. URL: https://doi.org/10.1186/gb-2009-10-3-r25.

[13] Jiecong Lin et al. "CRISPR-Net: A Recurrent Convolutional Network Quantifies CRISPR Off-Target Activities with Mismatches and Indels". In: *Advanced science* 7.13 (2020).

[14] Guanqing Liu, Yong Zhang, and Tao Zhang. "Computational approaches for effective CRISPR guide RNA design and evaluation". In: *Computational and Structural Biotechnology Journal* 18 (2020), pp. 35–44. ISSN: 2001-0370. DOI: https://doi.org/10.1016/j.csbj.2019.11.006. URL: https://www.sciencedirect.com/science/article/pii/S2001037019303551.

[15] Ali Haisam Muhammad Rafid et al. "CRISPRpred (SEQ): a sequence-based method for sgRNA on target activity prediction using traditional machine learning". In: *BMC bioinformatics* 21.1 (2020), pp. 1–13.

[16] Muhammad Naeem et al. "Latest Developed Strategies to Minimize the Off-Target Effects in CRISPR-Cas-Mediated Genome Editing". In: *Cells* 9 (2020).

[17] Benedikt Rauscher et al. "GenomeCRISPR - a database for high-throughput CRISPR/-Cas9 screens". In: *Nucleic Acids Research* 45.D1 (Oct. 2016), pp. D679–D686. ISSN: 0305-1048. DOI: 10.1093/nar/gkw997. eprint: https://academic.oup.com/nar/article-pdf/45/D1/D679/8847389/gkw997.pdf. URL: https://doi.org/10.1093/nar/gkw997.

[18] C Spearman. *Tthe proof and measurement of association between two things, American J.* 1904.

[19] Florian StÃ¶rtz and Peter Minary. "crisprSQL: a novel database platform for CRISPR/-Cas off-target cleavage assays". In: *Nucleic Acids Research* 49.D1 (Oct. 2020), pp. D855–D861. ISSN: 0305-1048. DOI: 10.1093/nar/gkaa885. eprint: https://academic.oup.com/nar/article-pdf/49/D1/D855/35364360/gkaa885.pdf. URL: https://doi.org/10.1093/nar/gkaa885.

[20] Shengdar Q Tsai et al. "CIRCLE-seq: a highly sensitive in vitro screen for genome-wide CRISPR-Cas9 nuclease off-targets". en. In: *Nat Methods* 14.6 (May 2017), pp. 607–614.

[21] Fathema Uddin, Charles M. Rudin, and Triparna Sen. "CRISPR Gene Therapy: Applications, Limitations, and Implications for the Future". In: *Frontiers in Oncology* 10 (2020). ISSN: 2234-943X. DOI: 10.3389/fonc.2020.01387. URL: https://www.frontiersin.org/article/10.3389/fonc.2020.01387.

[22] Hamed Valizadegan et al. "Learning to Rank by Optimizing NDCG Measure". In: Jan. 2009, pp. 1883–1891.

[23] Daqi Wang, Chengdong Zhang, Bei Wang, et al. "Optimized CRISPR guide RNA design for two high-fidelity Cas9 variants by deep learning". In: *Nature communications* 10.4284 (2019).

[24] Haifeng Wang, Marie La Russa, and Lei S Qi. "CRISPR/Cas9 in genome editing and beyond". In: *Annual review of biochemistry* 85 (2016), pp. 227–264.

[25] Tim Wang et al. "Genetic screens in human cells using the CRISPR-Cas9 system". In: *Science* 343.6166 (2014), pp. 80–84.

[26] Manuel Wiesenfarth et al. "Methods and open-source toolkit for analyzing and visualizing challenge results". In: *Scientific reports* 11.1 (2021), pp. 1–15.

[27] Xi Xiang et al. "Enhancing CRISPR-Cas9 gRNA efficiency prediction by data integration and deep learning". In: *Nature communications* 12.3238 (2021).

[28] Yuanyuan Xu and Zhanjun Li. "CRISPR-Cas systems: Overview, innovations and applications in human disease research and gene therapy". In: *Computational and Structural Biotechnology Journal* 18 (2020), pp. 2401–2415. ISSN: 2001-0370. DOI: https://doi.org/10.1016/j.csbj.2020.08.031. URL: https://www.sciencedirect.com/science/article/pii/S2001037020303846.

[29] Jifang Yan et al. "Benchmarking and integrating genome-wide CRISPR off-target detection and prediction". In: *Nucleic Acids Research* 48.20 (Nov. 2020), pp. 11370–11379. ISSN: 0305-1048. DOI: 10.1093/nar/gkaa930. eprint: https://academic.oup.com/nar/article-pdf/48/20/11370/34368248/gkaa930.pdf. URL: https://doi.org/10.1093/nar/gkaa930.

Appendix

Algorithm 1: Greedy approach for guides selection

function GreedySelection
  $scores \leftarrow a * s_{on} - b * s_{off}$
  $sorted \leftarrow$ **sort**$(scores)$
  $p \leftarrow$ *number of selected guides*
  $selectedGuides \leftarrow []$
  for $0 \leq i \leq p$ **do**
    $guide \leftarrow sorted[-1].guide$
    $sorted \leftarrow$ **DiscardConflicts**$(guide, sorted)$
    $selectedGuides \leftarrow selectedGuides + guide$
  end
  return *selectedGuides*

ChallengeR Results

To rank our models trained on all cell lines based on their performance on each test set, consisting of a single cell line, we utilize the challengeR visualization toolkit [26]. Figures 14 and 15 give visualizations on the rankings of the models. The models are ordered based on a test-based ranking scheme. In particular, the ranking is made according to the number of significant one-sided test results for each model where the significance test used is Wilcoxon signed-rank test. In what follows, we give a brief description of each graph that the challengeR toolbox outputs. As this only serves as an explanation of the types of graphs, we have have only considered the setting of Experiment 1.

Figure 14 shows a box-plot representing descriptive statistics for the NDCG scores over all test sets (the multi-coloured points) of each model considered.

Figure 15 shows a podium plot for ranking the considered models based on their NDCG scores. Each model is colour coded and each coloured dot represents the NDCG score of a test set for the respective model. The lines connect the NDCG scores corresponding to the same test set but different models. The models are ranked from left to right on the x-axis. For each test set the models are ranked and each coloured dot is assigned to a podium based on the rank the model received for each test. The bars at the bottom represent the frequency of each coloured dot assigned to the respective podium.

Figure 16 shows the outcomes of five different ranking strategies. Methods meanThenRank and medianThenRank aggregate the metric values across test sets for each model using the mean and median, respectively. The models are then ranked based on the aggregation value. Methods rankThenMean and rankThenMedian first rank the test sets based on their metric score for each model and then aggregate the ranks (using mean and median, respectively) of test sets to obtain the final rank. The method testBased is the one-sided Wilcoxon test as described above. This allows us to evaluate the robustness of ranking across different ranking methods. Each coloured line represents a separate model.

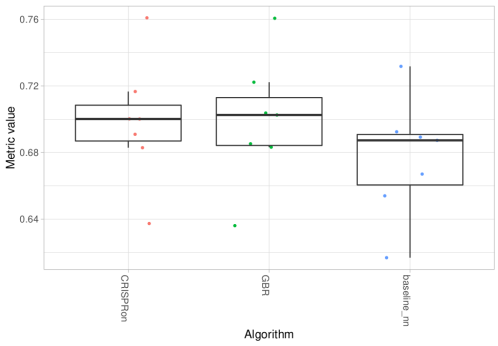

Figure 14: Box-plot of descriptive statistics for NDCG scores over all test sets of each model considered.

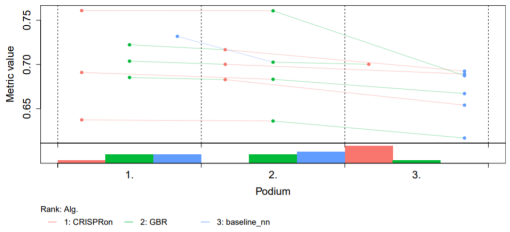

Figure 15: Podium plot for ranking the considered models based on their NDCG scores.

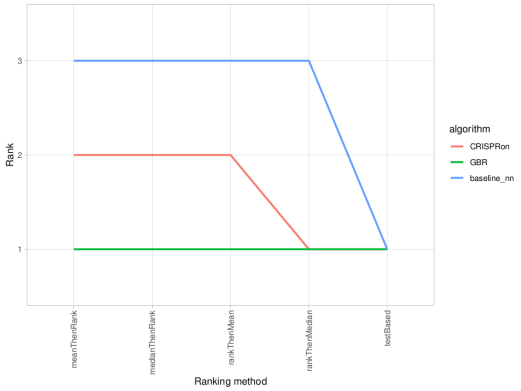

Figure 16: Comparisons of outcomes of five different ranking strategies.

For each ranking method listed on the x-axis, the height of line represents the model's corresponding rank. Horizontal lines indicate identical ranks for all methods.

Detailed timeline of the panCRISPR Toolbox project

Week(s)	Module	Detailed explanation
Week 1-4	Implementation	1. Researched implementation tools (pytorch, pyscaffold, Pytorch lightning, Sklearn, Hydra, Optuna) 2. Created project directory structure 3. Set up git repository and design git workflow (Structure consists of using Hydra config files to conveniently change parameters in the python scripts. This facilitates the model training step and makes results reproducible.) 4. Tested LRZ architecture with Hydra Submitted launcher plugin
	On- and off-target	1. Get the basics of molecular biology needed for the project 2. Read on CRISPR/Cas9 technology 3. Reviewed literature on on- and off-target prediction 4. Reviewed existing tools for on- and off-target prediction 5. Reviewed literature on datasets 6. Found data for on- and off-target pipeline

Week 5-9	On-target	1. Searched for state-of-the-art datasets of on-target efficiency scores 2. Conducted data preprocessing. 3. Challenges: (a) Many research papers using the same data did not give adequate details of their data preprocessing steps, preventing us from reproducing their experiments. (b) In each dataset there were differences in how on-target efficiencies were calculated. This required us to search the raw read counts and compute the efficiency scores manually for each dataset 4. Conducted featurization of sequence data for shallow models and one-hot-encoding of input for deep models 5. Split datasets into train, test, and validation with respect to the genes (to train and test on different sets of genes) and save to repository to allow for reproducibility of experiments. 6. Created data modules for shallow and deep models 7. Created model classes for shallow and deep models using Sklearn and Pytorch lightning 8. Created configuration files for each data module and model class to allow for convenient parameter tuning 9. Set up logging tools for training models. (We use Weights and Biases to track our training and validation losses). Again, we set up Hydra config files for logging and callbacks (such as early stopping) so that we do not have to hardcode in the Python scripts. 10. Added class and config file for evaluation of models. Research different evaluation metrics to use. 11. Tested evaluation pipeline and obtain first results on deep and shallow model performance.

Week 5-9	Off-target	1. Developed the roadmap of the project 2. Studied bowtie and blast, their differences, pros, cons and limitations. 3. Found sample guides from the literature search 4. Performed queries to bowtie and BLAST. 5. Found the best configuration for BLAST and bowtie 6. Created wrappers for bowtie and blast to run them within our pipeline 7. Created configuration files for BLAST and bowtie 8. Got output from alignment methods and parsed it 9. Compared results from bowtie with experimental data 10. Created guide generating pipeline 11. Set up the remote workspaces 12. Performed EDA 13. Created Querying tools for blastn to retrieve the ots from the genome files 14. Created FTP download interface

Week 10-14	On-target	1. Optimized data preprocessing: decrease preprocessing time and implement steps to obtain cleaner data to improve model performance 2. Improved model performance by using Optuna sweeps for hyperparameter tuning (Automatically sweeps through different choices of parameters specified in config file to find optimal hyperparameters) 3. Shallow models implemented so far: Baseline linear regressor, support vector regressor, gradient boosting machine 4. Deep models implemented so far: a. baseline neural network (2 fully connected hidden layers, ReLu activation function, and batch normalization for generalization) b. CrisprOn model from literature (CNN with ReLu activation function and dropout for generalization). 5. Added more evaluation plots for better visualization of results 6. Added additional guide RNA features from Biopython library (GC content/ melting temperature) to improve shallow and deep model performance. Examine the relevance of these features for both deep and shallow models. 7. Compared performance of shallow models using positional features of different orders. 8. Investigated model transferability between cell lines (train on one cell line and test on another)

Week 10-14	Off-target	1. Solved issues with bowtie, improving its results 2. Created OTS for BLAST and bowtie 3. Computed GC-content and melting temperature, adding them to the list of features 4. Split datasets into train and test and saved to repository for reproducibility of experiments. 5. Trained models on bowtie data 6. Created modularized classes of alignment methods, added them to the off-target module 7. Created Cross Validation pipeline 8. Created NGRAM and 6-bit encoders 9. Created data Preprocessing pipelines 10. Created deep convolutional models estimate the off-target score of the guide 11. Created shallow models: svm, random forest and others to estimate the off-target score of the guide 12. Created Scoring methods 13. Integrated all submodules into inference pipeline and creating modularized classes

Week 15-18	On-target	1. Compared different evaluation metrics (Spearman correlation and NDCG) 2. Considered feature selection algorithm for shallow models 3. Incorporated different ranking methods to rank models. Visualize ranking results using ChallengeR toolkit 4. Combined on-target and off-target pipelines to create a full working pipeline that takes a gene as input, generates a list of relevant guide RNAs and predicts efficiency and specificity scores for each guide using the best on-target and off-target models, respectively. 5. Implemented knapsack algorithm to rank the guide RNAs taking into account both efficiency and specificity scores as well as guide RNA overlap (we do not want the chosen top ranked guide RNAs to be competing for the same position on the DNA.) 6. Continued to improve performance of models: a. Combine datasets and train on multiple cell lines b. Augment training data to improve generalization
Week 15-18	Guide selection	1. Created guide selection module 2. Merged on- and off-target scores 3. Researched possible ranking algorithms with and without positional constraints 4. Implemented weighted sum, simple knapsack algorithm, greedy algorithm, quadratic optimization for guide selection