gyōza: a Snakemake workflow for modular analysis of deep-mutational scanning data

Authors: Romain Durand[1,2,3,4*], Alicia Pageau[1,2,3,4] and Christian R. Landry[1,2,3,4].

Affiliations:

[1]Département de Biochimie, de Microbiologie et de Bio-informatique, Faculté des Sciences et de Génie, Université Laval; Québec, G1V 0A6, Canada.

[2]Institut de Biologie Intégrative et des Systèmes (IBIS), Université Laval; Québec, G1V 0A6, Canada.

[3]PROTEO, Le regroupement québécois de recherche sur la fonction, l'ingénierie et les applications des protéines, Université Laval; Québec, G1V 0A6, Canada.

[4]Centre de Recherche sur les Données Massives (CRDM), Université Laval; Québec, G1V 0A6, Canada.

*Corresponding author. Email: romain.durand.1@ulaval.ca

Short title: gyōza, a Snakemake DMS analysis workflow

Keywords: gyōza, Snakemake, workflow, DMS, deep-mutational scanning, mutagenesis, sequencing, selection, mutational effects

Abstract:

Deep-mutational scanning (DMS) is a powerful technique that allows screening large libraries of mutants at high throughput. It has been used in many applications, including to estimate the fitness impact of all single mutants of entire proteins, to catalog drug resistance mutations and even to predict protein structure. Here, we present gyōza, a Snakemake-based workflow to analyze DMS data. gyōza requires little programming knowledge and comes with comprehensive documentation to help the user go from raw sequencing data to functional impact scores. Complete with quality control and an automatically generated HTML report, this new pipeline should facilitate the analysis of any DMS experiment. gyōza is freely available on GitHub (https://github.com/durr1602/gyoza).

1. Introduction

Deep-mutational scanning (DMS) is a molecular biology technique that evaluates the functional impact of a large number of mutations introduced at a particular locus (Fowler and Fields 2014). Owing to its broad range of applications and its relatively low cost, DMS has gained in popularity (Wei and Li 2023). Once the library of mutants is constructed, it is expressed in cells and subjected to screening (for example by applying selective pressure). Comparing allele frequencies at any post-screening time point to the initial time point allows the estimation of mutational effects. Several tools have been developed to perform this task. Enrich2, one of the most popular DMS analysis softwares, was developed early as a statistical framework to perform error estimation (Rubin *et al.* 2017). It has only very recently been updated to be able to run on modern versions of Python. DiMSum is a more recent software, written in R, which was also developed as a statistical framework to help users identify bottlenecks and other sources of error in their DMS dataset (Faure *et al.* 2020). Although DiMSum cannot handle multiple loci or multiple post-screening time points, the software is well maintained and the report it generates is informative. Other recently published softwares include Rosace (Rao *et al.* 2024), a Bayesian framework to analyze growth-based DMS data and popDMS (Hong *et al.* 2024), a DMS analysis method based on population genetics theory.

Despite the diversity of existing softwares to analyze DMS data, there are still experimental designs (and platform restrictions in certain cases) that preclude their use. For example, the Rosace environment can make it difficult to install and use on some systems, while popDMS uses a Gaussian prior distribution which is not suited for experiments involving strong selective pressure. This is typically the case for experiments involving proteins whose mutants confer antimicrobial resistance (Després *et al.* 2024; Chen *et al.* 2024; Bédard *et al.* 2024). In general, developing a DMS analysis software that can suit anyone's needs comes with a lot of challenges. Different experimental designs will lead to different sources of error, which means any statistical analysis is often project-dependent. We believe that an interesting compromise is a pipeline with a simple core that produces estimates of mutational effects ready for custom statistical analyses. Additionally, it should be usable by anyone with limited bioinformatic knowledge, yet modifiable by a more experienced user. To that aim, we designed gyōza, a pipeline that serves two purposes: analyze DMS data and generate interest in bioinformatics. To make sure that gyōza is accessible, we implemented several quality checks to help the user prepare everything needed for a workflow run. This includes detailed warnings in case of formatting issues or if any file is missing. gyōza is a Snakemake workflow, which means it comes with all advantages related to Snakemake, such as standardized code organization, increased reproducibility and portability. Other Snakemake-related aspects have previously been detailed elsewhere (Mölder *et al.* 2021; Corut and Wallace 2023).

To date, DMS analysis tools align sequencing reads to a reference, typically the wildtype nucleotide sequence of the mutated locus. This enables the identification of all possible

mutations in sequenced variants, including mutations that resulted from errors during the amplification of libraries or during sequencing. The current version of gyōza (at the time of writing) does not require alignment to a reference: we reasoned that in most cases, the experiment involves single mutants, which means it is computationally feasible to generate all expected mutants in silico and compare it to the sequencing dataset. Referring to the HTML report will inform the user on how the sequencing data were converted, with plots and detailed captions for each step, including an embedded interactive MultiQC report to explore the sequencing quality of all samples. Using this report, the user will know quickly if most of their sequencing reads were assigned to an expected sequence, or if they were lost at any of the early steps such as trimming or merging. Ultimately, read counts are converted into functional impact scores, reflecting mutational effects. Interestingly, the calculated score can accommodate different experimental designs, including growth assays, in which case the calculated scores are selection coefficients. gyōza can be used on a personal computer with limited computing resources or on typical high-performance computing (HPC) clusters. Most importantly, with comprehensive documentation and transparent source code, we hope to convince experimentalists to analyze their own DMS data.

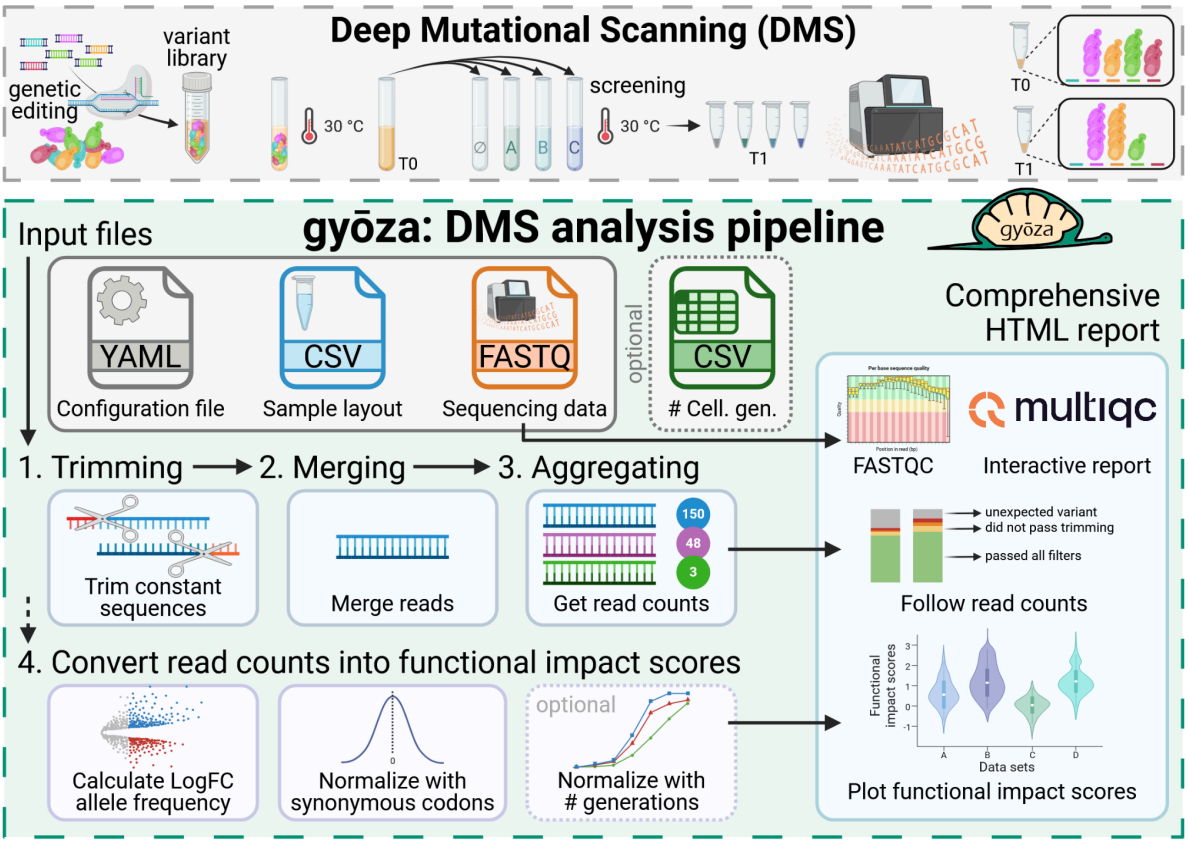


Figure 1. Overview of the typical experimental DMS and gyōza workflows. Deep mutational scanning can be done many ways. A schematic example is represented in the first section in gray. First, variant libraries are generated for every locus of interest, for example, by genetic editing. Then, libraries are cultured to express the variants. Non-deleterious variants are sequenced from samples collected at this initial time point (T0), as well as after screening, usually in a control condition and one or several other conditions (e.g. A, B, C). Here,

screening is done through a growth assay in selective conditions, but we could imagine an experiment where variants are selected based on cell morphology, fluorescence, etc. Ultimately, the loci of interest are sequenced at high throughput. The second section in green depicts the gyōza workflow. Variant read counts are obtained through three steps: adapter trimming using Cutadapt, read merging using PANDAseq and aggregating reads for unique sequences using VSEARCH. Read counts are converted into functional impact scores using several normalization steps, taking into account sample depth, allele frequency at the initial time point, synonymous codons as a reference point and optionally the number of cellular generations (if based on a growth assay). A single HTML report is generated upon completion of the workflow, featuring plots with detailed captions, organized in sections, to estimate the overall quality of the dataset as well as get a preliminary overview of the results. Figure created with BioRender.com. The top panel is a modified version of Fig 1B from (Durand *et al.* 2024).
Alt text. Schematic representation of a deep-mutational scanning experiment and the workflow to analyze the corresponding sequencing data.

2. Methods

Setup

gyōza can be installed on Linux and Mac systems or WSL2 for Windows users, by cloning the GitHub repository (see directory structure in Figure S1):

```
git clone https://github.com/durr1602/gyoza.git
```

Snakemake works best with the Conda package manager (Anaconda 2016) (more details in the “Conda” section in the supplemental material), which means its installation is a prerequisite. On the GitHub repository, we redirect the user to instructions on how to install Miniforge3, which will set up a Python 3 minimal environment and allow the use of Conda. The first necessary step is to create and activate a Conda environment in which Snakemake will be installed, using the provided requirements file:

```
conda env create --name=gyoza --file=env.yml
conda activate gyoza
```

The repository contains a toy dataset, which means the installation can be quickly tested by running the same command as for an actual run (see “Local execution of the workflow” section). For a standard use, the toy dataset can be replaced with real data by adapting the configuration and providing the necessary input files.

Configuration

The configuration file is a YAML file, which can be edited just as any text file, and is also validated against a provided YAML schema. This is particularly useful to ensure that no entry is missing and that all have the proper format. Entries include:

- file paths (see “Input files” section),

- the list of samples to be processed (or “all” to process all samples)
- the project-specific combination of attributes that feature as columns in the sample layout,
- the type of degenerate codons used to generate the single mutant libraries (currently supported are either “NNN” or “NNK”),
- three parameters used for filtering and plotting purposes (expected read count per sample and per variant, T0 read count threshold to label variants),
- a boolean switch to activate/deactivate quality check of all raw sequencing reads using FASTQC and MultiQC,
- a boolean switch to activate/deactivate normalization with the number of cellular generations,
- a boolean switch to activate/deactivate the automatic generation of the HTML report

Input files

The most important input files correspond to demultiplexed paired-end sequencing data (FASTQ) and the sample layout. Sequencing data are expected to have been obtained with an Illumina or Aviti sequencer, should be paired-end and can optionally be compressed (.fastq.gz). The sample layout is a CSV file containing nine mandatory columns:

- Sample_name: unique identifier for each sample,
- R1 and R2: name of the demultiplexed FASTQ files containing the forward and reverse reads, respectively,
- N_forward and N_reverse: 5’-3’ constant sequences on either side of the mutated locus,
- Mutated_seq: unique identifier for each mutated locus,
- Pos_start: starting position in the protein sequence for the mutated locus,
- Replicate: e.g. 1, 2 (more than 2 is handled but may lead to overplotting),
- Timepoint: T0 is mandatory, then one or more time points such as T1, T2, etc

The “Mutated_seq” column is a convenient specificity of gyōza, compared to existing DMS analysis pipelines. It was introduced so that the user can analyze DMS data from different loci in a single workflow run. Implementation of this functionality involves the generation of all single mutants of the DNA sequence for every possible value of “Mutated_seq”. DNA sequences are provided by the user in a mandatory CSV file containing exactly two columns:

- Mutated_seq: identifiers matching the ones in the sample layout,
- WT_seq: corresponding wild-type DNA sequence, assuming the first three bases constitute the first mutated codon

This is particularly useful for experiments in which a gene has been divided into several overlapping fragments. In such an experimental design, one would perform the DMS experiment on each fragment separately, then normalize the scores using the mutations in the overlaps. Because we cannot be sure the project-specific experimental design requires this additional normalization step, it is not included in the default workflow. Instead, we offer the possibility of labeling samples with any combination of attributes (e.g. attributes related to

the fragment, the genetic background, the screening condition, etc). These attributes are optional but can be entered as additional column names in the sample layout (and if so, should also be listed in the configuration file). Another useful application would be to normalize data obtained from different sequencing runs (assuming some samples were sequenced in at least two runs).

If the user indicates in the main configuration file that they wish to normalize with the number of cellular generations, a template CSV file is automatically generated with the following columns:

- Replicate (same as in the sample layout),
- Timepoint (here, only post-screening time points),
- one column per sample attribute,
- Nb_gen (initialized with "1" in the template)

When the workflow is run for the very first time (including if it is a dry run), an exception is raised and a message displays on the terminal to notify the user that the template file has been generated and needs to be edited. The user will then edit the Nb_gen column with the number of cellular generations between the indicated time point on the same row and the previous time point, e.g. at "T1", write "4.26" in the Nb_gen column to indicate that 4.26 doublings were measured between T0 and T1 for this condition. An example of the procedure applied to the toy dataset is provided in Figure S2.

All input files (except the sequencing data) are validated against dedicated YAML schemas to make sure that all mandatory columns are present and that the values in each column respect the expected format. If any formatting issue arises, detailed warnings should guide the user for troubleshooting.

In summary, for every new project, the user should edit the configuration file (YAML) and provide the following input files:

- the FASTQ files of the paired-end sequencing reads,
- the sample layout (CSV),
- the wild-type sequences (CSV),
- the number of generations between time points (CSV, based on generated template),
- (optional) the codon table if different from the provided one (*Saccharomyces cerevisiae* TAXID 559292)

Local execution of the workflow

Before we describe how to execute the workflow, let us introduce a few important concepts related to the use of Snakemake. A Snakemake workflow connects different modules, known as "rules", which process input files and optionally generate output files. Rules are defined in Snakemake files (.smk) which are located in the "rules" directory and are imported in the "Snakefile". The Snakefile additionally defines the main rule ("rule all") which specifies the ultimate target file(s). Snakemake essentially works backwards from the ultimate target file(s) to build the workflow, connecting modules so that the necessary input files for each step are

generated by an upstream module. Rule definitions include input, output, a main statement related to the actions and optional arguments such as parameters, resources, log files, etc. In summary, whenever an execution is triggered from the root directory (in our case, “gyoza”), Snakemake searches for the Snakefile and builds the workflow, starting with the main rule.

In order to quickly check the integrity of the workflow (for example to ensure the file paths have been properly entered in the configuration file), it is strongly recommended to perform a “dry run” by typing the following command line from the “gyoza” directory:

```
snakemake --use-conda --dry-run
```

For a run on the toy dataset (assuming the user has not modified any file after cloning the repository), simply remove the “dry-run” flag to execute the workflow:

```
snakemake --use-conda
```

The same command line should be used for an actual run (assuming the user has prepared all files as described above). Upon first use of gyōza, dedicated Conda environments will be installed and activated on demand to make sure the jobs of each rule are run in environments with the appropriate packages (more details in the “Conda” section in the supplemental material).

Snakemake cannot know the expected content of output files and will therefore not execute the workflow if the target files, identified by their path and name, already exist. On the other hand, some intermediary files are conserved by default, which means that Snakemake will always run whatever is “missing” for the workflow to complete. For example, if the workflow is accidentally aborted halfway through, rerunning the same command line (with the same ultimate target as final output) will automatically detect that some files already exist and move on to the ones that are either incomplete or that have not been generated yet.

By default, all available cores will be provided. The user can scale down by adding the “cores” flag and specifying the number of CPUs to use in parallel, as such:

```
snakemake --use-conda --cores 10
```

Note that if you are on a shared node, or if the dataset is rather large, it is strongly recommended to submit the jobs to the SLURM scheduler (see “Running the workflow on HPC clusters” section).

Workflow steps

Quality control

First, raw FASTQ files are analyzed using FastQC (Andrews 2010) to evaluate the overall quality of sequencing data. A single interactive report is generated using MultiQC (Ewels *et al.* 2016) (and later on embedded in the gyōza report).

Getting read counts

There are three main steps necessary to go from demultiplexed paired-end sequencing data to read counts: adapter trimming, read merging and counting the number of reads for each unique sequence. gyōza uses three packages to do that: Cutadapt (Martin 2011), PANDAseq (Masella *et al.* 2012) and VSEARCH (Rognes *et al.* 2016).

First, forward and reverse reads are trimmed with Cutadapt using the specified constant sequences on either side of the mutated locus as 5' adapters. By default, indels are forbidden and the maximum error rate is 0.15 (which means three substitutions in a 20-bp sequence would still match the adapter. The untrimmed sequences are discarded, but their number is retrieved by a downstream script (see "Gather read statistics" section).

Trimmed forward and reverse reads are merged with PANDAseq. Among other technical details (which are transparent in the source code), we specify that the maximum overlap between forward and reverse reads from a pair be 625 bp. This is in line with PANDAseq recommendations in case of highly overlapping sequences, which is typically the case for Aviti sequencing data. Finally, merged reads are collapsed into 100% identity clusters with VSEARCH (referred to as "dereplication" in the VSEARCH manual, with the function --fastx-uniques). In this final step, we discard all singletons (sequences only sequenced once), since they might result from sequencing errors. The output is formatted as a single FASTA file for each sample, containing entries for every non-singleton unique sequence, as well as the corresponding number of reads.

Gather read statistics

A custom Python script is used to parse all log files from the three main steps performed to get usable read counts: trimming, merging and aggregating. The script outputs a single dataframe for all samples, in which read counts are followed throughout these processing steps. To facilitate interpretation, this dataframe is processed in the next step (see "Comparing read counts with expectations" section) and plotted as a stacked barplot, which is ultimately featured in the automatically generated HTML report (see "Exploring the results" section).

Using the Jupyter platform to visualize the transformation of raw data into functional scores

Once the read counts are obtained, we need to answer two main questions: does the sequencing data match expectations? and what do they mean in terms of mutational effects? For both, we believe the involved steps are easier to visualize if the code is written in Jupyter notebooks (Kluyver *et al.* 2016). A Jupyter notebook allows to organize the code, for example using section headings (written in the Markdown language) and to include plots. Template notebooks in gyōza include code written in Python for consistency. Jupyter integration in a Snakemake workflow implies that template notebooks (the ones available on the GitHub repository) are parameterized by Snakemake in order to execute them. The snakemake object imported in the notebook gives access to all workflow objects such as input, output, config entries, etc. One can retrieve the executed notebook to visualize the output, or even to export it as an HTML file. If an error is raised during a step involving a Jupyter notebook, we recommend referring to the executed notebook (in the logs/notebooks directory) to get more details on the error. If the user wants to re-run the notebook or alter the template notebook (to modify, remove or add steps, to edit plots, etc.), interactive editing is available using a dedicated snakemake command line (see "Alter existing notebooks" section).

Comparing read counts with expectations

In order to compare the observed sequences to the expected sequences, we go through three steps: 1) generate and annotate all expected single mutants of the starting sequences (mutated loci), 2) convert all FASTA files containing read counts into a single dataframe of observed sequences, 3) compare sets using the merge function from the Python package pandas. The first step is performed by a dedicated Jupyter notebook, while the second and third steps are performed by a separate notebook. This ensures that the workflow is separated into modules, which can easily be altered or replaced. For example, if one were to analyze a DMS experiment in which the libraries include a defined list of mutants, they would need to replace the notebook to generate single mutants by a script that creates a similarly-formatted dataframe with the defined mutant list (see "Edit the core workflow" section in the supplemental material).

The standard practice for analyzing DMS data is often different. First, reads would be aligned to a reference, which would allow to spot any number of mutations of any type. In most cases, experiments are designed to mutate a single codon at a time. If a second mutation is introduced (e.g. due to off-target editing, error during amplification or during sequencing), gyōza will label the sequences and their associated read count as "Unexpected". Ultimately, the tool generates a stacked barplot which shows the number of reads that were discarded at each step (trimming, merging, aggregating, unexpected) and the number of usable reads. The rationale is that the user can refer to this plot to quickly determine the quality of the dataset and resort to "manual" troubleshooting if need be. Some issues could be easier to troubleshoot than others: for example, if many reads are discarded at the trimming stage, the user might identify an error in the specified adapters. On the other hand, if a large proportion

of reads are attributed to unexpected sequences, the user might need to either inspect the content of the sequences (e.g. directly on the Jupyter notebook), or use a different tool to retrieve more detailed information.

Processing read counts

A dedicated Jupyter notebook is used to process read counts, with the ultimate goal to convert them into functional impact scores. First, variants are labeled with a confidence score based on the number of reads at T0: 1, high confidence = above specified threshold in all replicates; 2, medium confidence = above threshold in at least one replicate; 3, low confidence = equal or below threshold in all replicates. The general rule to set a relevant threshold is to find the balance between:

- setting the threshold too low, which results in the inclusion of variants likely resulting from sequencing errors,
- setting the threshold too high, which diminishes the size of the dataset and may mask the full fitness range

In the very last steps of the workflow, gyōza saves a first dataframe with all functional scores and a second one including only average scores (and error) from high confidence variants across replicates.

The following steps are performed to convert read counts into functional impact scores:

- add 1 to all read counts,
- calculate read frequencies (normalize with sample depth),
- calculate log2 fold-change (frequency at a later time point replicate divided by frequency in the matching replicate at T0),
- (optional) normalize with the number of cellular generations between T0 and a later time point), in which case the ultimate score will correspond to a selection coefficient,
- subtract the median of normalized log2 fold-change values obtained for variants encoding the wild-type amino acid sequence, except for the wild-type nucleotide sequence

We purposefully exclude the wild-type nucleotide sequence from the last normalization step because it is often over-represented in libraries generated using oligonucleotide pools. However, the user can modify the notebook to include it if need be. Finally, we aggregate over synonymous codons at each position to get a single functional impact score per amino acid sequence. At this stage, the table ("all_scores.csv") is saved in the results/df directory. In addition, we also calculate the median and error across replicates for high-confidence variants, then save the corresponding table ("avg_scores.csv"). In this last table, for each output time point, three columns "fitness", "lower_err" and "upper_err" correspond to the median, lower and upper error (from 2.5th to 97.5th percentile).

Exploring the results

Throughout the workflow, plots are generated and stored as SVG files in the "graphs" directory, as well as in any format specified by the user in the main configuration file. In order to facilitate interpretation of the analysis, a comprehensive HTML report is generated at

the root of the “results” directory. This report will be generated regardless of the state of completion of the workflow, which means only completed steps will be included. It contains a main view, which opens on a diagram of the workflow, and a side panel on the left with a search bar and a menu organized in sections (Fig. 2A). Overall, the user has access to the following:

- hyperlinks to the main tools (Cutadapt, PANDAseq, VSEARCH),
- source code for each rule with a list of input files, output files and packages with specified versions,
- runtime statistics,
- four main sections for the results:
    - Quality control: embeds the interactive MultiQC report on all FASTQ files,
    - Read filtering: stacked barplot of filtered reads and distributions of read counts for unexpected variants in each sample,
    - Read processing: example of distribution of raw read count and frequency per variant, example of overlap across time points and replicates, distribution of allele frequencies at each time point,
    - Functional impact: distribution of functional impact scores, correlation between replicates, correlation between time points, functional impact over time

Each plot comes with a detailed caption to help interpret the data. Examples are shown in Figure 2. The file tree obtained upon full completion of the workflow is provided in Figure S1.

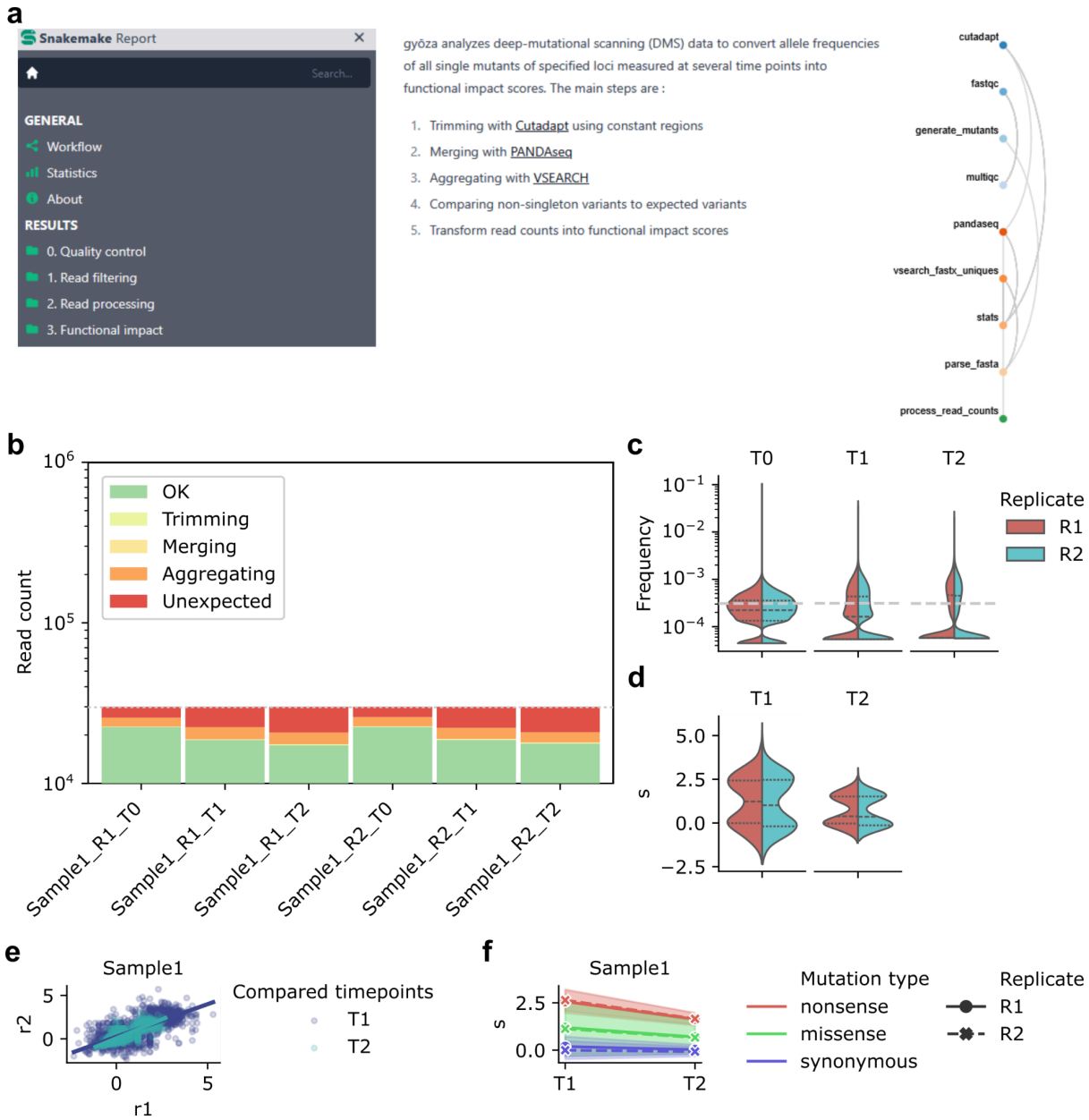


Figure 2. Examples of output generated by gyōza. a) Main view of the HTML report. Navigating the menu on the left will lead to several plots with detailed captions, including the following examples. b) Filtered reads. For each sample, read depths are plotted as stacked bars. The color key indicates the number of reads that passed all filters ("OK"), or the number of reads lost at the indicated step. c) Distribution of allele frequencies (read count per variant normalized by sample depth) at each time point. d) Distribution of functional impact scores at each post-screening time point. e) Correlation between replicates. f) Functional impact over time. Bands indicate the standard deviation for each mutation type. Plots have been slightly altered in Inkscape compared to the ones generated when running gyōza on the toy dataset, to maximize space efficiency. In general, the format of generated plots might vary depending on the number of samples and their properties (names, number of replicates, time points, etc). **Alt text.** Examples illustrating the report generated by the workflow, here with plots corresponding to the provided toy dataset.

Running the workflow on HPC clusters

If the user needs to scale up the workflow, gyōza supports the automatic submission of jobs to the SLURM scheduler. The user should first edit the "technical" configuration file in the "profile" directory, then run the following command line to launch the workflow:

```
snakemake --profile profile
```

The technical configuration file is again a YAML file that can easily be modified by the user and constitutes a replacement for the traditional bash files that are commonly used to submit jobs to SLURM. A user familiar with this procedure will find standard arguments in the sbatch statement. Make sure you keep only what is necessary and edit the arguments in the "default-resources" section. As their name indicates, these are only the default resources, in case some of them are not specified in the rule definitions. However, in order to prevent over-allocation, gyōza defines rule-specific resources for almost every rule, which are dynamically requested based on the total input size and number of attempts. Priority will always be given to command line arguments, then to rule definitions, then to this technical configuration file. Other convenient options (activated by default) include a naming scheme for every job that should make it easier for the user to find the proper log files if any error should happen. Log files related to SLURM jobs are located in the "SLURM_out" directory and sorted by rule. They should contain the same output as what is usually displayed on the terminal and therefore may inform on errors related to Snakemake or SLURM. Errors related to the tool or the script itself would typically show up in the sample log files, which are located upstream in the "logs" directory (and are also sorted by rule).

By default, an email is sent to the specified address every time a job fails (can be changed by modifying the "mail-type" argument in the technical configuration file). Conveniently, errors related to a lack of resources (job either reaches the time limit or is killed by an out-of-memory event) will be labeled accordingly in the subject of the email.

3. Results and discussion

gyōza comes with a toy dataset so that the user can test the installation. The same dataset was analyzed with DiMSum (Faure *et al.* 2020), although we selected only the first two time points (T0, T1), since DiMSum does not handle multiple post-screening time points. Overall, the setup is quite similar, except the preparation of input files for gyōza is fairly intuitive, with validations and warnings providing helpful indications to set up the workflow. For a local execution (i.e. a small dataset), runtime is comparable once Conda environments for gyōza have been installed. However, scaling up with gyōza should prove fast both to set up and run. Internal tests with a 145 GB unpublished dataset (many conditions, including several time points, replicates and mutated loci) ran under 5 hours with a relatively small amount of computing resources (40 CPUs per task maximum at any time and up to 5 GB of memory).

Ultimately, most sequences (99.5 %) were picked up by both tools. Only 18 out of 3,867 sequences were discarded as singletons by one tool but not the other and vice versa (all with a

read count of 2 instead of 1). Despite the differences in calculating the functional impact scores (DiMSum implements an error model), the values calculated by both tools correlate significantly (Fig 3). Unfortunately, attempts at testing the dumpling pipeline (which uses Rosace and Enrich2) were unsuccessful because of platform issues, which are being addressed at the time of writing.

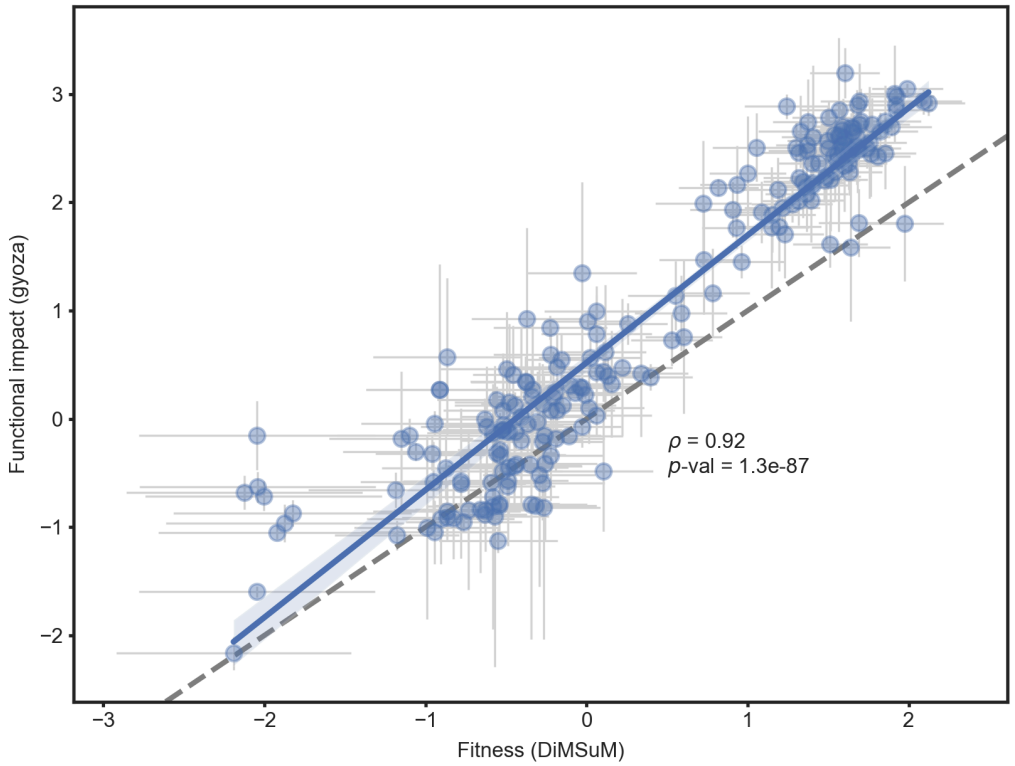


Figure 3. Correlation of functional impact scores calculated by gyōza and DiMSum v1.3.2. Each data point corresponds to an amino acid sequence from the gyōza toy dataset. Functional scores (with 95 percentile interval as asymmetric error bars) calculated by gyōza are indicated on the y axis. "Fitness" and "sigma" (error) values calculated by DiMSum are shown on the x axis. Unique variants which were flagged as singletons (at T0 or T1) by either gyōza (n=1) or DiMSum (n=17) were excluded. For both tools, scores were initially calculated for 3,849 unique nucleotide variants with a single mutated codon relative to the wildtype sequence. Average fitness values were calculated only from high confidence variants (i.e. variants from the toy dataset with a read count above 10 in all T0 replicates), resulting in 217 unique amino acid sequences. Correlation was estimated by calculating the Spearman's rank correlation coefficient (ρ) and the corresponding *p*-value. Deviation from the diagonal can be explained by the use of different methods in normalizing the scores.
Alt text. Graph comparing mutational effects, as estimated by the workflow and a similar existing tool.

In summary, gyōza provides a portable, intuitive and transparent way to quickly analyze DMS data.

4. Data availability

gyōza is freely available (under MIT license) from GitHub (https://github.com/durr1602/gyoza). A separate repository contains everything that relates to tests reported here (https://github.com/Landrylab/Durand_et_al_2025_gyoza).

5. Acknowledgments

We thank Philippe C. Després for the initial optimization of PANDAseq and VSEARCH arguments as well as critical reading of the manuscript, Stéphane Larose for technical assistance with HPC and Victor Haffreingue for precious advice on the use of version control.

6. Funding

Canadian Institutes of Health Research (CIHR) (Foundation grant 387697, CRL)
Genome Canada and Genome Quebec (grant 6569, CRL)
Canada Research Chair in Cellular Systems and Synthetic Biology (CRL)
Natural Sciences and Engineering Research Council of Canada (NSERC) postdoctoral fellowship (RD)

7. References

Anaconda, 2016 *Anaconda Software Distribution*.

Andrews, S., 2010 FastQC: A Quality Control tool for High Throughput Sequence Data.

Bédard, C., I. Gagnon-Arsenault, J. Boisvert, S. Plante, A. K. Dubé *et al.*, 2024 Most azole resistance mutations in the Candida albicans drug target confer cross-resistance without intrinsic fitness cost. Nat. Microbiol. 9: 3025–3040.

Chen, J. Z., M. Bisardi, D. Lee, S. Cotogno, F. Zamponi *et al.*, 2024 Understanding epistatic networks in the B1 β-lactamases through coevolutionary statistical modeling and deep mutational scanning. Nat. Commun. 15: 8441.

Corut, A. K., and J. G. Wallace, 2023 kGWASflow: a modular, flexible, and reproducible Snakemake workflow for k-mers-based GWAS. G3 (Bethesda) 14: jkad246.

Després, P. C., A. K. Dubé, M.-È. Picard, J. Grenier, R. Shi *et al.*, 2024 Compensatory mutations potentiate constructive neutral evolution by gene duplication. Science 385: 770–775.

Durand, R., A. G. Torbey, M. Giguere, A. Pageau, A. K. Dubé *et al.*, 2024 Mutational landscape and molecular bases of echinocandin resistance. bioRxiv 2024.07.21.604487.

Ewels, P., M. Magnusson, S. Lundin, and M. Käller, 2016 MultiQC: summarize analysis results for multiple tools and samples in a single report. Bioinformatics 32: 3047–3048.

Faure, A. J., J. M. Schmiedel, P. Baeza-Centurion, and B. Lehner, 2020 DiMSum: an error

model and pipeline for analyzing deep mutational scanning data and diagnosing common experimental pathologies. Genome Biol. 21: 207.

Fowler, D. M., and S. Fields, 2014 Deep mutational scanning: a new style of protein science. Nat. Methods 11: 801–807.

Hong, Z., K. S. Shimagaki, and J. P. Barton, 2024 popDMS infers mutation effects from deep mutational scanning data. Bioinformatics 40: btae499.

Kluyver, T., B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier *et al.*, 2016 Jupyter Notebooks -- a publishing format for reproducible computational workflows, pp. 87–90 in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, edited by F. Loizides and B. Schmidt. IOS Press.

Martin, M., 2011 Cutadapt removes adapter sequences from high-throughput sequencing reads. EMBnet.journal 17: 10–12.

Masella, A. P., A. K. Bartram, J. M. Truszkowski, D. G. Brown, and J. D. Neufeld, 2012 PANDAseq: paired-end assembler for illumina sequences. BMC Bioinformatics 13: 31.

Mölder, F., K. P. Jablonski, B. Letcher, M. B. Hall, C. H. Tomkins-Tinch *et al.*, 2021 Sustainable data analysis with Snakemake. F1000Res. 10: 33.

Rao, J., R. Xin, C. Macdonald, M. K. Howard, G. O. Estevam *et al.*, 2024 Rosace: a robust deep mutational scanning analysis framework employing position and mean-variance shrinkage. Genome Biol. 25: 138.

Rognes, T., T. Flouri, B. Nichols, C. Quince, and F. Mahé, 2016 VSEARCH: a versatile open source tool for metagenomics. PeerJ 4: e2584.

Rubin, A. F., H. Gelman, N. Lucas, S. M. Bajjalieh, A. T. Papenfuss *et al.*, 2017 A statistical framework for analyzing deep mutational scanning data. Genome Biol. 18: 150.

Wei, H., and X. Li, 2023 Deep mutational scanning: A versatile tool in systematically mapping genotypes to phenotypes. Front. Genet. 14: 1087267.

8. Supplemental material

```
├── LICENSE
├── README.md
├── config
│   ├── README.md
│   ├── config_file.yaml
│   ├── project_files
│   │   ├── ScerevisiaeTAXID559292_Cocoputs_codon_table.csv
│   │   ├── layout.csv
│   │   ├── nbgen.csv
│   │   └── wt_seq.csv
│   └── reads
│     └── *.fastq.gz
├── env.yml
├── fulldoc
│   └── README.md
├── logs
│   ├── 0_qc
│   │   ├── *_fastqc.log
│   │   └── multiqc.log
│   ├── 1_trim
│   │   ├── cutadapt-sample=*.err
│   │   └── cutadapt-sample=*.stats
│   ├── 2_merge
│   │   └── pandaseq-sample=*.stats
│   ├── 3_aggregate
│   │   └── vsearch-sample=*.stats
│   ├── 4_stats
│   │   └── stats.log
│   └── notebooks
│     ├── generate_mutants.ipynb
│     ├── parse_fasta.ipynb
│     └── process_read_counts.ipynb
├── profile
│   ├── config.v8+.yaml
│   └── status-sacct.sh
├── results
│   ├── 0_qc
│   │   ├── *_fastqc.html
│   │   ├── *_fastqc.zip
│   │   └── multiqc.html
│   ├── 3_aggregate
│   │   └── *_aggregated.fasta
│   │
│   │
```

```
│   ├── df
│   │   ├── all_scores.csv
│   │   ├── avg_scores.csv
│   │   ├── master_layout.csv.gz
│   │   └── readcounts.csv.gz
│   ├── done
│   ├── graphs
│   │   ├── hist_plot.svg
│   │   ├── rc_filter_plot.svg
│   │   ├── rc_var_plot.svg
│   │   ├── replicates_heatmap_plot.svg
│   │   ├── replicates_plot.svg
│   │   ├── s_through_time_plot.svg
│   │   ├── scoeff_violin_plot.svg
│   │   ├── timepoints_plot.svg
│   │   ├── unexp_rc_plot.svg
│   │   └── upset_plot.svg
│   ├── read_stats.csv
│   └── report.html
└── workflow
    ├── Snakefile
    ├── envs
    ├── notebooks
    │   ├── generate_mutants.py.ipynb
    │   ├── parse_fasta.py.ipynb
    │   └── process_read_counts.py.ipynb
    ├── report
    ├── rules
    │   ├── common.smk
    │   ├── cutadapt.smk
    │   ├── generate_mutants.smk
    │   ├── pandaseq.smk
    │   ├── parse_fasta.smk
    │   ├── process_read_counts.smk
    │   ├── qc.smk
    │   ├── stats.smk
    │   └── vsearch.smk
    ├── schemas
    └── scripts
        └── get_read_stats.py
```

Figure S1. File tree upon full completion of the workflow. Executed notebooks are stored in the logs/notebooks directory, whereas template (unparameterized) notebooks are located in the workflow/notebooks directory (with other scripts).

```
normalize:
      # Set to True if you want to normalize impact scores with the
number of cellular generations
      with_gen: True
```

```
(gyoza) <USER>@<MACHINE>:~/gyoza$ snakemake -n
Main config validated.
Sample layout validated.
/home/rodur28/git_repos/gyoza/workflow/rules/common.smk:30: UserWarning:
The column 'Pos_stop' is not listed in your sample attributes
  print('Sample attributes imported.')
Sample attributes imported.
WT imported.
Exception in file
/home/rodur28/git_repos/gyoza/workflow/rules/common.smk, line 51:
>>Please fill in the file config/project_files/nbgen.csv with the number
of cellular generations<<
>>(or deactivate this normalization in the main config file)<<
  File "/home/rodur28/git_repos/gyoza/workflow/rules/common.smk", line
51, in <module>
```

```
(gyoza) <USER>@<MACHINE>:~/gyoza$ cat config/project_files/nbgen.csv
Species,Mating_type,Fragment,Replicate,Timepoint,Nb_gen
CN,alp,F1,r1,T1,1
CN,alp,F1,r2,T1,1
CN,alp,F1,r1,T2,1
CN,alp,F1,r2,T2,1
```

Figure S2. Normalization with the number of cellular generations. When opting in for normalization with the number of cellular generations in the main configuration file (first code block), the workflow generates a template file and prompts the user to fill it with numbers for each indicated condition (second block). In the corresponding file (third code block), we can see each combination of sample attributes (as defined in the main configuration file), replicate and time point arranged on different rows. For each row, the number of cellular generations between the indicated time point (on the same row) and the previous one has been initialized to 1. From there, the user can either modify the file directly or use a custom script to generate one (making sure the format copies the template's).

Conda

Snakemake has been designed to work well with the Conda package manager. The main drawback is that it often comes with the installation of a large volume of files and may, in certain rare instances, interfere with already installed dependencies. It is one of the reasons why Conda is prohibited on some systems. Unfortunately, there is no easy fix for this. Until

Snakemake allows the specification of predefined virtual environments, we believe the current solution offers the most interesting tradeoff between robustness and flexibility. Briefly, rule definitions must include a “conda” statement which points to a YAML requirements file (either locally in the “envs” directory, or online if the rule uses a wrapper). Each file lists the necessary packages with versions to be installed in a virtual Conda environment to ensure that there are no software incompatibilities. The first time the workflow is run (or anytime a YAML requirements file is edited), Snakemake will download and install all necessary packages in a virtual Conda environment. This step might take time but is skipped if the environment already exists (e.g. if the workflow is run a second time). Two convenient arguments allow the user to respectively create all environments and list all existing environments:

```
snakemake --conda-create-envs-only
snakemake --list-conda-envs
```

During the execution of the workflow, rule-dedicated environments are automatically activated.

Edit the core workflow

If the user wants to adapt the workflow to suit their needs, they can do the following:

- alter the existing scripts or notebooks,
- alter or replace an existing module,
- add new modules.

Alter existing scripts

Editing a Python script only requires modification of the .py file in the “scripts” directory. As long as the files that are necessary in downstream steps are properly generated, modifications should not be too impactful. However, in the core workflow, the only Python script is a parsing script that retrieves read count statistics from the log files of tools like Cutadapt, PANDAseq and VSEARCH. This means the only case that would require modification by the user would be if log files are formatted differently in the future (in a way that prevents the current parsing script from functioning), or if the user wants to extract additional information from log files (or other files already generated at this stage). If the user does extract more information and wants to include it in the corresponding plot, modifications to the Jupyter notebook in charge of plotting (“parse_fasta.py.ipynb”) would be necessary.

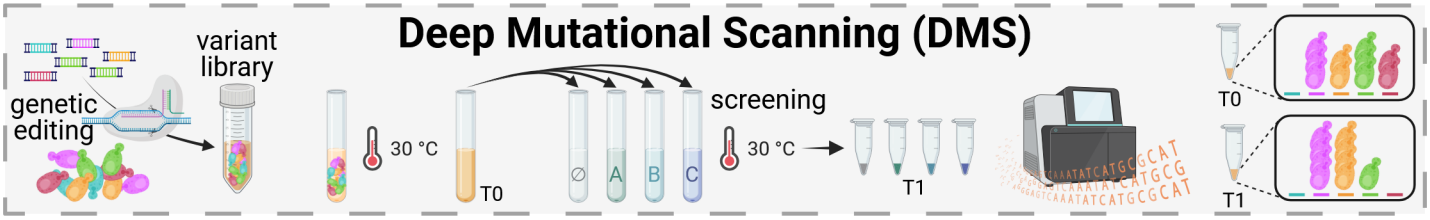
Alter existing notebooks

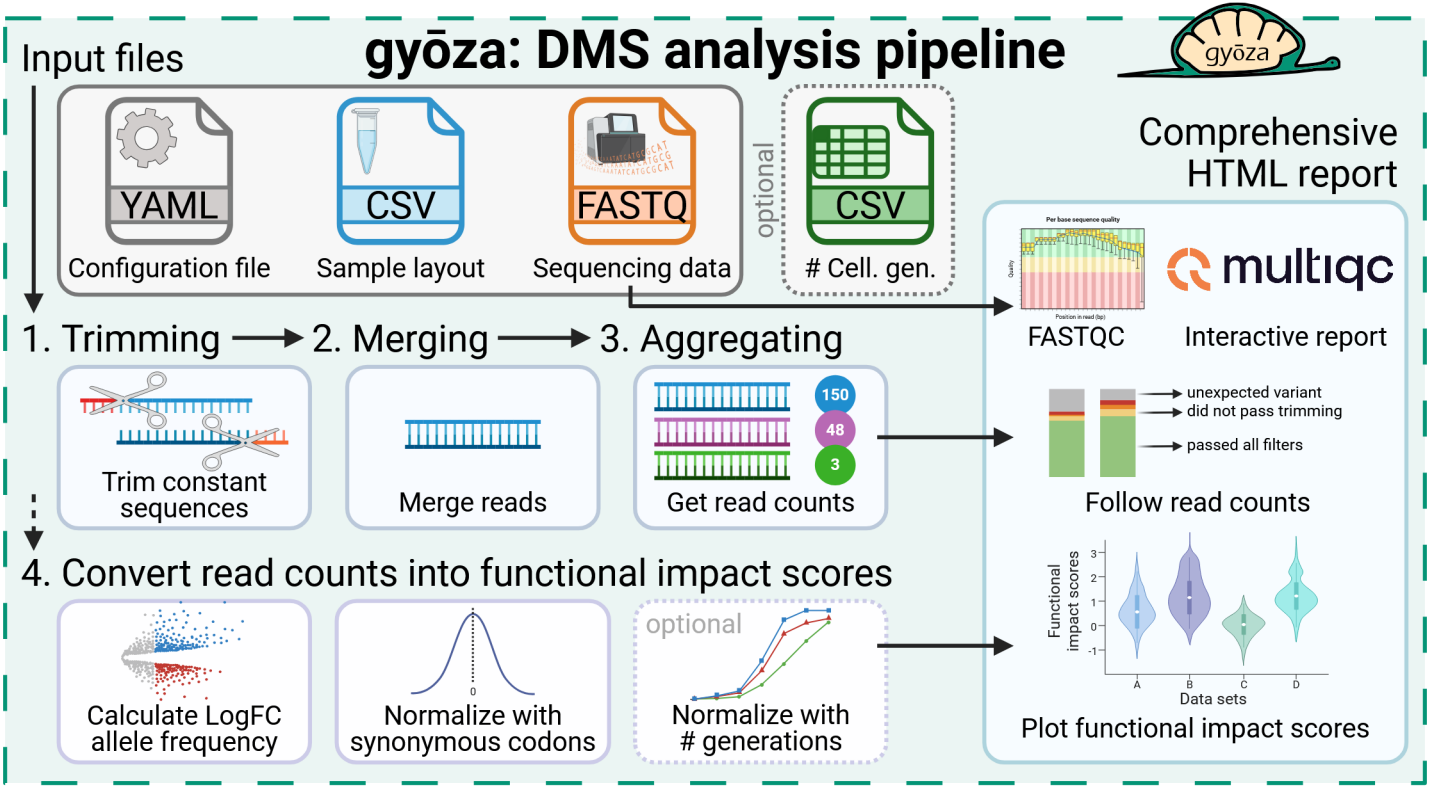
Editing a parameterized Jupyter notebook is more complicated than editing a script, since it requires knowledge of the kernel specifications, as well as knowledge of the values for the different parameters, including file paths. Thankfully, Snakemake allows interactive editing of any Jupyer notebook, using a single command line to open it. From a single target (a dataframe or a plot generated by the notebook), the workflow will determine which template notebook is responsible for generating the target and will automatically run the necessary

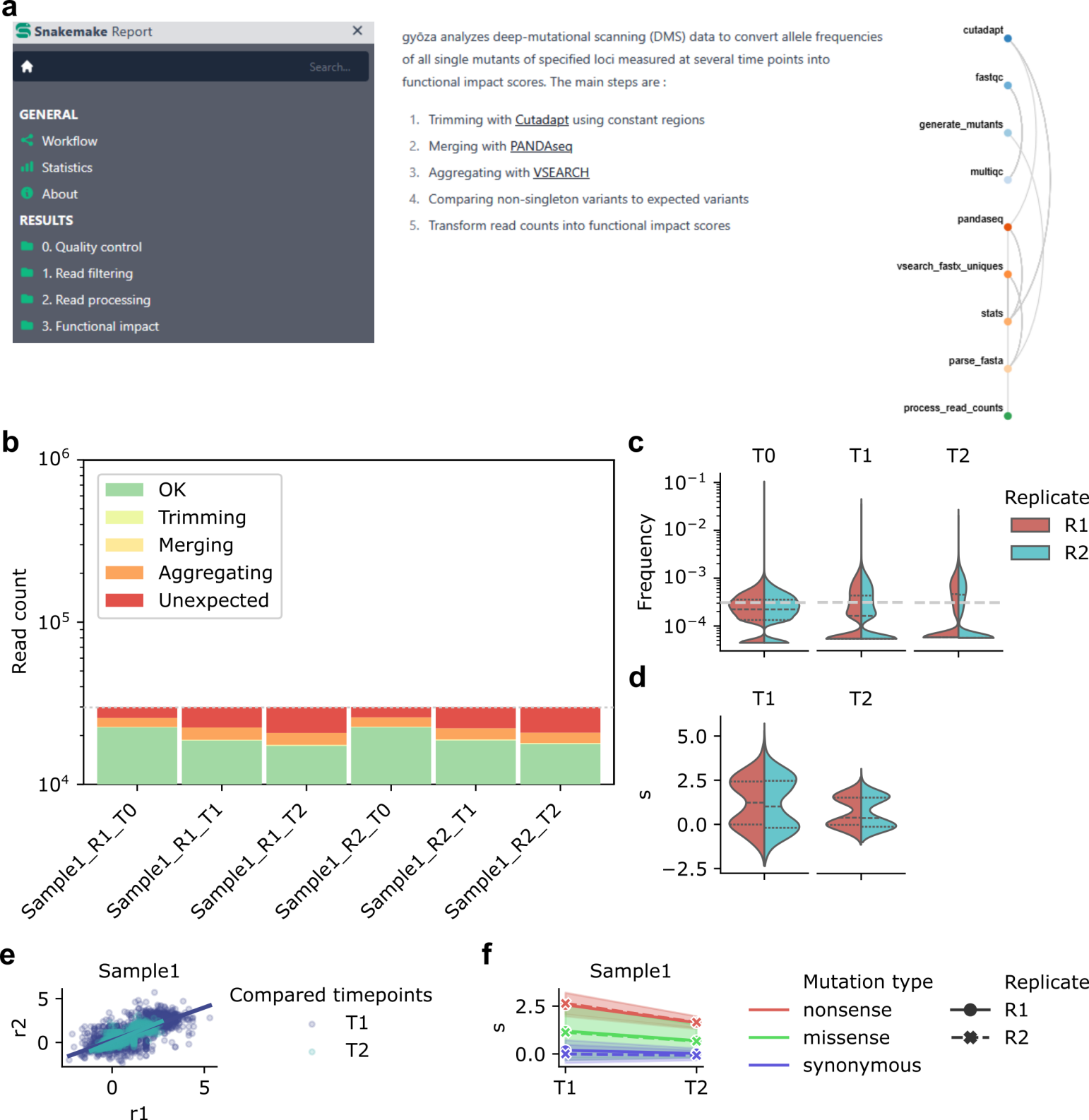
steps to get there. Once the workflow has confirmed that all necessary input files exist, a Jupyter server is opened so that the user can enter the parameterized notebook in a ready-for-use environment. More specifically, Snakemake activates the Conda environment (as specified in the rule definition) and exports the kernel specifications, meaning that all necessary tools will be installed. In order to open the notebook, the user needs to access the generated URL, which leads to the Jupyter hub dashboard. Because the server is opened in the local browser, users on login nodes of HPC clusters might need to open a SSH tunnel between their local machine and the node (instructions on the GitHub repository). Because the notebook is already parameterized at this point, there is no need to manually enter the file paths or the values for other parameters. The user can simply run one cell after the other until they get to the section where they want to introduce a modification, in which case they can also test the modifications right away. If the user wants to introduce a change that necessitates a different set of requirements, the requirements should be properly specified in the rule definition, ideally pointing to a brand new YAML file located in the "envs" directory. Of course, these modifications would need to be brought before opening the notebook so that the changes get reflected in the kernel specifications.

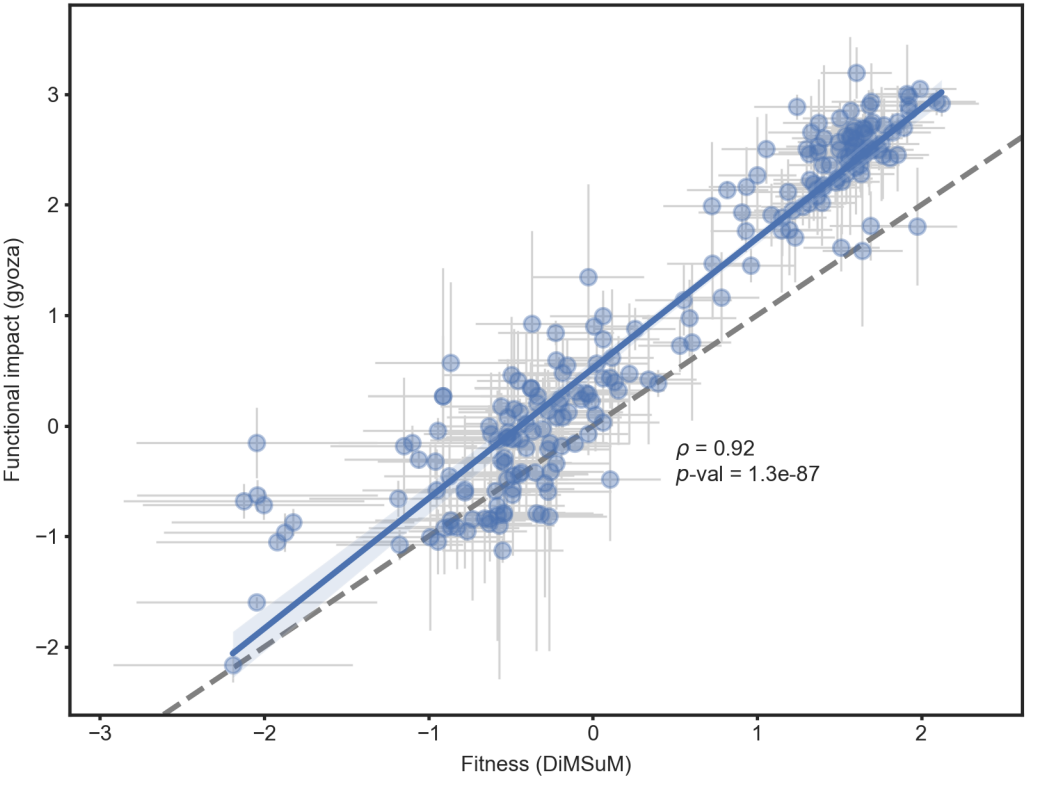
Alter an existing module or add a new one

Alteration of a module, or "rule" as it is referred to in Snakemake, necessitates editing of the rule definitions directly in the .smk files. Unless one wants to alter significantly the core workflow, it is recommended not to change the input or output statements. Instead, one might be interested in changing the actions performed by the rule, in which case they would alter the main statement in the rule definition. gyōza uses three types of statements: shell, script or notebook. The shell statement, written in bash, is often used to load an existing tool. In our case, PANDAseq is called from a shell statement. Cutadapt, VSEARCH, FASTQC and MultiQC are also called from shell statements, although gyōza uses wrappers for these tools. The script and notebook statements point to a custom script or a Jupyter notebook, respectively. Conveniently, either may be written in one of several programming languages (Python, R,...). Ultimately the way the code is written does not matter much, as long as the proper actions are performed, typically transforming input files into properly formatted output files, so that workflow compatibility is maintained. Adding a module that branches out of the core workflow, meaning that its output is not reused downstream, requires the corresponding files to be listed in the input statement of the main rule ("rule all"). It is highly recommended to specify rule-specific environments (either load a wrapper or add a new YAML file in the "envs" directory and point to the file in the "conda" statement of the rule).


Deep Mutational Scanning (DMS)
genetic editing
variant library
30 °C
T0
∅
A
B
C
screening
30 °C
T1
T0
T1


gyōza: DMS analysis pipeline
gyōza
Input files
YAML
Configuration file
CSV
Sample layout
FASTQ
Sequencing data
optional
CSV
Cell. gen.
Comprehensive HTML report
1. Trimming
2. Merging
3. Aggregating
Trim constant sequences
Merge reads
150
48
3
Get read counts
FASTQC
multiqc
Interactive report
unexpected variant
did not pass trimming
passed all filters
Follow read counts
4. Convert read counts into functional impact scores
Calculate LogFC allele frequency
Normalize with synonymous codons
optional
Normalize with # generations
Functional impact scores
Data sets
A
B
C
D
Plot functional impact scores


a
Snakemake Report
Search...
GENERAL
Workflow
Statistics
About
RESULTS
0. Quality control
1. Read filtering
2. Read processing
3. Functional impact
gyōza analyzes deep-mutational scanning (DMS) data to convert allele frequencies of all single mutants of specified loci measured at several time points into functional impact scores. The main steps are :
1. Trimming with Cutadapt using constant regions
2. Merging with PANDAseq
3. Aggregating with VSEARCH
4. Comparing non-singleton variants to expected variants
5. Transform read counts into functional impact scores
cutadapt
fastqc
generate_mutants
multiqc
pandaseq
vsearch_fastx_uniques
stats
parse_fasta
process_read_counts
b
Read count
OK
Trimming
Merging
Aggregating
Unexpected
Sample1_R1_T0
Sample1_R1_T1
Sample1_R1_T2
Sample1_R2_T0
Sample1_R2_T1
Sample1_R2_T2
c
T0
T1
T2
Frequency
Replicate
R1
R2
d
T1
T2
s
e
Sample1
r2
r1
Compared timepoints
T1
T2
f
Sample1
s
T1
T2
Mutation type
nonsense
missense
synonymous
Replicate
R1
R2


Functional impact (gyoza)
Fitness (DiMSuM)
ρ = 0.92
p-val = 1.3e-87
3
2
1
0
−1
−2
−3
−2
−1
0
1
2