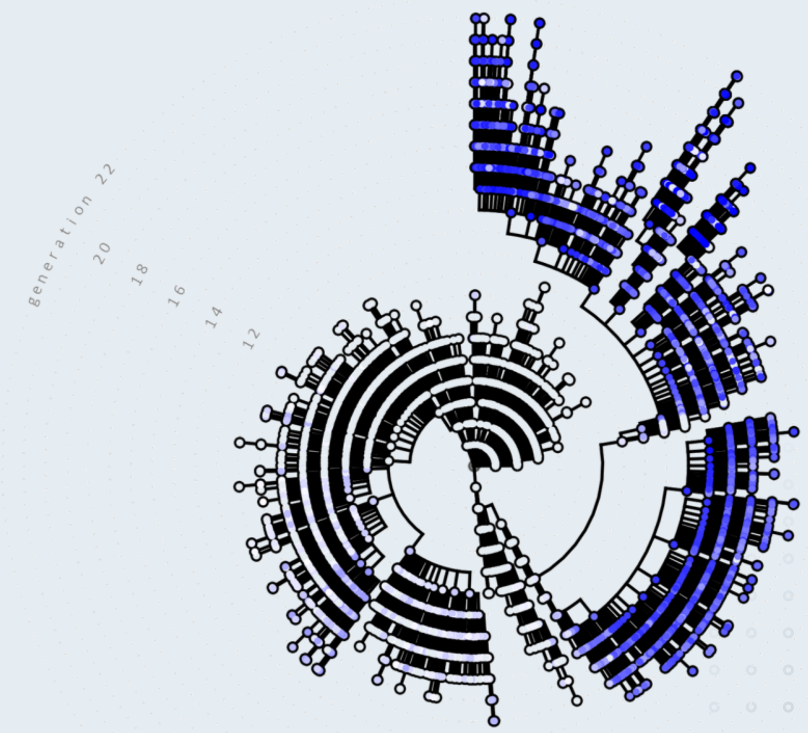
AI.zymes

A modular platform for evolutionary enzyme design



AI.zymes is in active development!

We are happy to assist you with running AI.zymes.
For questions, reach out to:
Adrian.Bunzel@mpi-marburg.mpg.de

Merlicek LP, Neumann J, Lear A, Degiorgi V, Waal MMdW, Cotet T-S, Mulholland AJ, Bunzel HA,
Angewandte Chemie International Edition 2025, https://doi.org/10.1002/anie.202507031 *(accepted)*.

Contents

1 Installation

We recommend installing AI.zymes with pip. For code development, AI.zymes can also be cloned from the GitHub repository. To install AI.zymes with pip, run:

```
pip install aizymes
```

AI.zymes works by wrapping around various bioengineering tools that need to be installed seperately. A list of all established tools in AI.zymes is given in the section **Available Tools**. For some of these tools, their paths need to be defined to connect with them. To that end, AI.zymes will create a **.config** file when you run AIzymes for first time.

Note

Several tools, such as Rosetta, need to be installed seperately. See **Available Tools** for guidelines on how to find the installation instructions.

Warning

AI.zymes currently only works with SLURM. If you want to use a different submission system, please contact Adrian.Bunzel@mpi-marburg.mpg.de.

1.1 Installation via GitHub

To install AI.zymes via GitHub, clone the repository, create the AI.zymes python environment, and add the repository to your pythonpath. To clone AI.zyme, run:

```
git clone https://github.com/bunzela/AIzymes.git
```

You can either create your own AI.zymes environment, or install all required packages in your existing environment.

```
cd AIzymes
# To build new environemnt
conda env create -f environment.yml --name AIzymes
# Alternative to install packages in curent environemnt:
# conda env update -f environment.yml --prune
```

Add the path to AI.zymes to your .bashrc. Replace **PATH_TO_AIZYMES** with the corect path to AI.zymes.

```
echo 'export PYTHONPATH="$PYTHONPATH:PATH_TO_AIZYMES/src/aizymes"' >> ~/.bashrc
source ~/.bashrc
```

2 Quick Start

Tip

Get started with AI.zymes using the notebook **example_notebooks/Example_KSI.ipynb**.

The notebook **Example_KSI.ipynb** can be used to improve the promoscious Kemnp eliminase activity of KSI. Here, each step of this notebook is explained briefly. For detailed instructions on how to use AI.zymes, check **5. Full Manual**.

2.1 Create an Instance of the AI.zymes class

To start AI.zymes, you first need to create an Instance of the AI.zymes class and set **FOLDER_HOME**. All data generated during the AI.zymes run are storred in **FOLDER_HOME**. Asides from the design structures, **FOLDER_HOME** contains information on the settings chosen during design (**variables.json**) and a database containing the sequences and scores of all designs (**all_scores.csv**).

```
# Import AI.zymes
from aizymes import *

# Create instance of AIzymes_MAIN
AIzymes = AIzymes_MAIN(FOLDER_HOME = 'Example_KSI')
```

2.2 Set up AI.zymes for design

Befor starting design, the basic folder structure and input files need to be set up with **AIzymes.setup**. The input settings used here are discussed in detail in **5. Full Manual**.

Important

All inputfiles provided need to be in the folder **Input** in the same directory as where you running the code. Note that structures can also be loaded from a different folder using **FOLDER_PAR_STRUC**.

Note

If the config file specified with **SYSTEM** does not exist, **AIzymes.setup** will prompt for inputs to generate a new config file. The config file contains system-specific information neccesary to run AI.zymes.

Warning

AI.zymes currently only works with SLURM. If you want to use a different submission system, please contact Adrian.Bunzel@mpi-marburg.mpg.de.

```
AIzymes.setup(
# General Design Settings
    WT                 = "1ohp",
    LIGAND             = "5TS",
    DESIGN             =
        "11,14,15,18,38,54,55,58,63,65,80,82,84,97,99,101,112,114,116",
    PARENT_DES_MED     = ['RosettaDesign',
                           'ESMfold','MDMin','RosettaRelax','ElectricFields'],
    DESIGN_METHODS     = [[0.5,'SolubleMPNN',\
                                'ESMfold','MDMin','RosettaRelax','ElectricFields
                                    '],\
                           [0.5,'RosettaDesign',\
                                'ESMfold','MDMin','RosettaRelax','ElectricFields
                                    ']],
# General Scoring Settings
    SELECTED_SCORES    = ["total","catalytic","interface","efield", "identical"],
# General Job Settings
    MAX_JOBS           = 72,
    MAX_GPUS           = 4,
    MEMORY             = 450,
    N_PARENT_JOBS      = 144,
    MAX_DESIGNS        = 5000,
    KBT_BOLTZMANN      = [1.0, 0.5, 0.02],
    SUBMIT_PREFIX      = "KSI_TEST",
    SYSTEM             = "AIzymes.config",
# RosettaDesign Settings
    CST_NAME           = "5TS_enzdes_planar_tAB100",
    CST_DIST_CUTOFF    = 40.,
# FieldTools settings
    FIELD_TARGET       = ":5TS@C9 :5TS@H04",
 )
```

2.3 Submit controller to run AI.zymes

Once the setup is completet you need to start the controller. **AIzymes.submit_controller()** will launch am HPC job that runs the controller.

```
AIzymes.submit_controller()
```

Tip

You can check the progress of AI.zymes and trouble-shoot errors with the controller.log. Alternatively, you can also check the output and errors of individual designs.

```
!tail -n 50 {FOLDER_HOME}/controller.log
!ls {FOLDER_HOME}/designs/0/scripts
```

2.4 Analysis tools to check AI.zymes

AI.zymes comes with several tools to analyze the design runs. Use **AIzymes.plot** to create a range of plots, showing how the scores evolved over time. Alternatively, you can use **AIzymes.print_statistics()** to print out key information from the **all_scores.csv** file.

Note

NORM is used to ensure comarability between design runs by normalizing all results to the same values.

```
AIzymes.plot(NORM = {
                'total_score': [310, 380],
                'catalytic_score': [-30, -4],
                'interface_score': [20, 30],
                'final_score': [-30, 0],
                'efield_score': [-40, 30],
                'identical_score': [0, 1.05],
             })
```

Tip

AIzymes.print_statistics() can be very helpfull to trouble-shoot!

```
AIzymes.print_statistics()
```

2.5 Extract best structures after AI.zymes run

Finally, to extract the hits from your AI.zymes run, execute **AIzymes.best_structures()**

Note

Structures will be saved with their final score as prefix, helping to identify the best designs.

```
AIzymes.best_structures()
```

3 Available Tools

AI.zymes works by wrapping around a range of bioengineering tools. While some of these tools can be readily installed using *pip* or *conda*, others need to be installed as stand-alone programs. Furthermore, some programs need to be registered in the **config** file genereatd during the first run of AI.zymes

Rosetta	Protein design and relaxation
LigandMPNN	Protein design with ligands
SolubleMPNN	Protein design without ligands
AlphaFold3	Structure prediciton with ligands
ESMFold	Structure prediciton without ligands
AmberTools	Structure minimization
FieldTools	Electric field calculation
BioDC	Redox Potential calculation

3.1 Protein Design Tolls

3.1.1 Rosetta

Rosetta is a comprehensive suite of software for physics-based macromolecular modeling and design. In AI.zymes, it is used for (1) structure relaxation and repacking, and (2) enzyme design.
Available at: https://rosettacommons.org/software/download/

3.1.2 LigandMPNN

LigandMPNN is a deep-learning design tool that focuses on optimizing protein–ligand interactions. Within AI.zymes, it can be integrated to tailor the active-site environment by adjusting residues that directly interact with the chemical transition state or substrate.
Available at: https://github.com/dauparas/LigandMPNN

3.1.3 SolubleMPNN and ProteinMPNN

SolubleMPNN is a deep-learning design tool used to generate protein sequences that fold into a given structural scaffold, often enhancing the stability of the input structure in the process. In AI.zymes, ProteinMPNN is used to remodel the protein scaffold, focusing on regions away from the active site to improve stability without compromising enzyme activity.
Available at: https://github.com/dauparas/ProteinMPNN

3.2 Structure Prediction Tools

3.2.1 AlphaFold3

AlphaFold3 is a deep learning-based structure prediction tool that can include a range of small-molecule ligands during prediction. In AI.zymes, AlphaFold3 is used to predict the three-dimensional structure of designed enzymes, especially those bound to ligands such as heme.
Available at: https://github.com/google-deepmind/alphafold3

3.2.2 ESMFold

ESMFold leverages large protein language models for structure prediction. It is usually faster than AlphaFold3 but cannot include small-molecule ligands. In AI.zymes, ESMFold is used in combination with RosettaRelax and MDMin to generate ligand-bound structural models. This combination offers a fast route to assess design variants for proper folding and active-site organization without the computational overhead of traditional methods.
Available at: https://github.com/facebookresearch/esm
Installed using: pip install transformers fair-esm

3.2.3 AmberTools

AmberTools is a collection of programs for molecular mechanics and dynamics, which help parameterize molecules and perform energy calculations. In the context of AI.zymes, AmberTools is used to minimize enzyme–ligand complexes (MDMin).
Available at: https://ambermd.org/AmberTools.php
Installed using: conda install conda-forge::ambertools

3.3 Scoring Tools

3.3.1 FieldTools

FieldTools is a computational tool for calculating electric fields in and around enzyme active sites. Since electrostatic interactions can greatly influence enzyme catalysis, AI.zymes incorporates FieldTools to quantitatively assess the catalytic electric field along a target bond. This evaluation is integrated into the multi-objective scoring system, enabling selection of variants that not only stabilize the protein and its active site but also optimize the electric field for enhanced catalysis.
Available at: https://github.com/bunzela/FieldTools

Note

FieldTools has been hard-coded into AI.zymes and does not need to be installed seperately.

3.3.2 BioDC

Work in progress...

4 Introduction

4.1 Coding philosophy

AI.zymes is a modular program that seamlessly combines computational methods for design, structure prediction, and machine learning in a coherent enzyme design workflow (Fig. 1). To that end, AI.zymes employs a controller function that orchestrates the design process. The controller collects information from the designs and stores them in a shared database, selects which variants to submit for design, and decides what type of design or structure prediction algorithms to run with the selected variant. The controller is designed to manage a user-defined number of parallel design jobs, automatically starting a new job as soon as a previous one is completed.

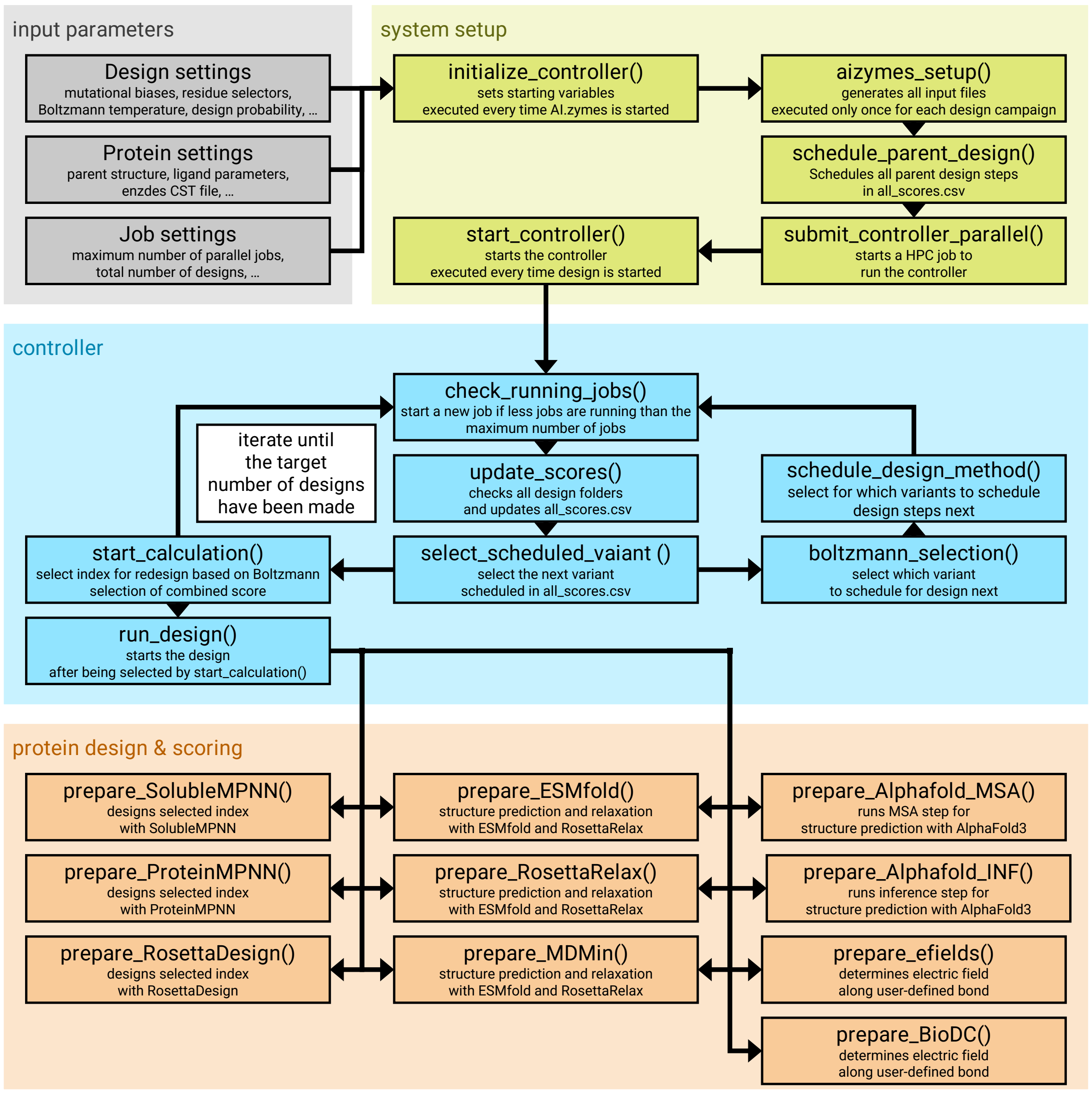

input parameters
Design settings
mutational biases, residue selectors, Boltzmann temperature, design probability, ...
Protein settings
parent structure, ligand parameters, enzdes CST file, ...
Job settings
maximum number of parallel jobs, total number of designs, ...
system setup
initialize_controller()
sets starting variables
executed every time AI.zymes is started
aizymes_setup()
generates all input files
executed only once for each design campaign
schedule_parent_design()
Schedules all parent design steps in all_scores.csv
start_controller()
starts the controller
executed every time design is started
submit_controller_parallel()
starts a HPC job to run the controller
controller
check_running_jobs()
start a new job if less jobs are running than the maximum number of jobs
iterate until the target number of designs have been made
update_scores()
checks all design folders and updates all_scores.csv
schedule_design_method()
select for which variants to schedule design steps next
start_calculation()
select index for redesign based on Boltzmann selection of combined score
select_scheduled_vaiant ()
select the next variant scheduled in all_scores.csv
boltzmann_selection()
select which variant to schedule for design next
run_design()
starts the design after being selected by start_calculation()
protein design & scoring
prepare_SolubleMPNN()
designs selected index with SolubleMPNN
prepare_ESMfold()
structure prediction and relaxation with ESMfold and RosettaRelax
prepare_Alphafold_MSA()
runs MSA step for structure prediction with AlphaFold3
prepare_ProteinMPNN()
designs selected index with ProteinMPNN
prepare_RosettaRelax()
structure prediction and relaxation with ESMfold and RosettaRelax
prepare_Alphafold_INF()
runs inference step for structure prediction with AlphaFold3
prepare_RosettaDesign()
designs selected index with RosettaDesign
prepare_MDMin()
structure prediction and relaxation with ESMfold and RosettaRelax
prepare_efields()
determines electric field along user-defined bond
prepare_BioDC()
determines electric field along user-defined bond

Fig. 1 | General Flow Chart of AI.zymes. Based on a set of input variables (grey), AI.zymes will be set up and started (yellow). The main program of AI.zymes is the controller (blue), which controls the overall workflow, decides what action to take (salmon, red), and writes information into the databases (green).

4.2 Basic concepts

4.2.1 Evolutionary ”rounds” in AI.zymes

The evolutionary algorithm in AI.zymes is not based on performing classical evolutionary rounds comprising mutagenesis and screening. Instead, the algorithm aims to maximize performance by constantly running a set number of design jobs set by **max_jobs**. Whenever the number of running jobs falls below **max_jobs**, AI.zymes spawns another design or structure prediction job. To choose the parent variant for design, AI.zymes performs multi-objective Boltzmann selection. Importantly, all designs are treated equally during Boltzmann selection, which selects from the growing pool of designs irrespective of which round the designs stemmed. Thus, design is not limited by a maximum number of rounds but a maximum number of designs (**max_designs**).

4.2.2 Scoring metrics

AI.zymes employs various scoring metrics to assess various properties relevant to enzyme activity. The **total_score** corresponds to the total energy of the system. This score reflects the **total_score** of a structure in Rosetta and is referred to as ”stability score” in the main text. The **interface_score** corresponds to the binding energy of the ligand to the protein. This score was calculated using Rosetta’s InterfaceScoreCalculator mover and is referred to as the ”interface score” in AI.zymes. The **catalytic_score** corresponds to the score of the catalytic interaction. It is calculated from the ideal interaction geometry described in a Rosetta_Match constraint file format. Finally, the **efield_score** describes the electric fields along the scissile C-H bond. Fields were calculated using FieldTools, and this score is referred to as the ”electric field” in the main text.
Note that each variant in AI.zymes can have two sets of scores. All variants have scores from their initial design run. Variants selected for redesign also undergo structure prediction and relaxation, generating an updated set of scores. If available, Boltzmann selection is based on these updated scores; otherwise, the original design scores are used.
Furthermore, to run evolution in a forward-thinking manner, where the scores of the direct descendants of a variant can be included during screening, a metric called ”potential” was introduced. The potential is the average of the score of the current variant and the scores of all its direct descendants.

4.2.3 System startup

To start AI.zymes, the system must be set up using **setup_aizymes()** to create the overall file structure and stores all design-relevant parameters in the variables.json file. Among others, these include the name of the protein, ligand, and which residues can be designed. Furthermore, general settings can be set, such as how many jobs may run in parallel and how many designs will be made in total. Following the initial setup, **startup_controller()** reads these parameters from the variables.json file and initializes all other required variables. Thus, **setup_aizymes()** is only rone once, whereas **startup_controller()** is executed every time the code is restarted.

4.2.4 Controller

The controller is the central program of AI.zymes. It constantly cycles between the following steps:
1) **check_running_jobs()** determines how many jobs are currently running on the system. If **MAX_JOBS** are running, the controller sleeps for 20 s and rechecks the number of running jobs, or else the controller initiates a new design cycle.
2) **update_scores()** iterates through all designs and updates the **all_scores.csv** database. **update_scores()** also unblocks all indices for which the structure prediction runs are completed.
3) **check_parent_done()** checks if the initial non-evolutionary design for the parent variant has concluded. If fewer designs than N_PARENT_JOBS have been made, the controller skips the Boltzmann selection and instead runs **start_parent_design()**, starting a **run_RosettaDesign()** job on the parent structure.
4) **boltzmann_selection()** initiates the Boltzmann selection to identify the next scaffold for design.
5) **start_calculation()** finally starts the design based on the selected input structure. **start_calculation()** checks if there is a structure of the selected design that went through **ESMfold_RosettaRelax()**. If not, the controller will start the structure prediction and block the selected index. If there is a relaxed structure, the controller will generate a new index into which the design will be stored. This involves creating a folder for the new design and appending the **all_scores.csv** file with the selected index. To allow for a blend of design methods, the selected design will use **ProteinMPNN** instead of **RosettaDesign** with a user-defined probability **ProteinMPNN_PROB**.

4.2.5 Boltzmann selection

In contrast to all other scores, the **catalytic_score** has a clear minimum, with a score of zero reflecting a perfect agreement of the design with the target catalytic geometry. Thus, **catalytic_score** was not included during Boltzmann selection but used as a binary cutoff to exclude variants before selection. To that end, the mean plus one standard deviation of the **catalytic_score** of all designs is calculated, and all variants with a **catalytic_score** below that cutoff are removed. Additionally, structures that are currently undergoing structure prediction are excluded from Boltzmann selection to prevent redundant structure prediction on the same structure.
Boltzmann selection is based on the design potentials and not on their scores. Boltzmann selection is performed on the combined_potential, which is calculated from the average of the z-score standardized **total_potential**, **interface_potential**, and **efield_potential**. Note that during normalization, potentials for which lower values are better are inverted (**total_potential** and **interface_potential**). Thus, higher combined_potentials correspond to better variants. Boltzmann selection is performed at a user-defined temperature kbt_boltzmann. To increase selection stringency during design, kbt_boltzmann decreases with each new design from an initial **KBT_BOLTZMANN** value with a **KBT_BOLTZMANN_DECAY** rate in a single exponential decay. Boltzmann selection

4.2.6 Database

The all_scores.csv database is the primary file containing all information from an AI.zymes run. Among other details, the database includes data on the specific settings for making each design and its resulting scores. Additionally, **all_scores.csv** keeps track of which structures are currently undergoing structure prediction and are therefore excluded from Boltzmann selection. This exclusion prevents redundant structure prediction from being performed multiple times on the same structure.

4.3 Available Modules in AI.zymes

Currently, AI.zymes has established **run_RosettaDesign()** and **run_ProteinMPNN()** for design. For structure prediction and scoring, **run_ESMfold_RosettaRelax()** and **calc_efields_score()** have been established. Importantly, AI.zymes is built highly modularly, facilitating the future addition of other protein engineering packages to augment the computational evolution algorithm.

4.3.1 Design with RosettaDesign

run_RosettaDesign() setups and submits a RosettaDesign run for the selected index based on RosettaScripts. AI.zymes dynamically generates the input .xml files that control RosettaScripts. An example .xml file can be found in 6.4 Example .xml file for RosettaDesign. Briefly, a geometry bias from the RosettaMatch constraint file is introduced with the AddOrRemoveMatchCsts (6.2 Rosetta Match constraint file), and the protein is repacked and minimized using the EnzRepackMinimize mover. Subsequently, the protein is designed using FastDesign for 3 repeats while applying a bias to the input sequence using the FavorSequenceProfile mover with a weight of **CST_WEIGHT**. Designable active-site residues are defined with DESIGN, and all mutations but cysteine are permitted to avoid the introduction of disulfide bonds, which can affect reproducibility if redox states are not tightly controlled. For the catalytic residue, only glutamate and aspartate are permitted to maintain the catalytic mechanisms relying on a carboxylate base for protein abstraction. Catalytic residues were defined via a Rosetta Match Constraint File. After design, the protein is relaxed for 1 repeat with FastRelax mover without the geometry bias from the RosettaMatch constraint file. The final scores, including the **interface_score** calculated with InterfaceScoreCalculator, are given for the relaxed structure.

4.3.2 Design using ProteinMPNN

run_ProteinMPNN() setups and submits a ProteinMPNN run for the selected index using its sequence as input. A **bias_by_res.json** file is generated that applies a bias to the input sequence. The input structure was parsed using the ProteinMPNN helper scripts (**parse_multiple_chains.py**, **assign_fixed_chains.py**, **make_fixed_positions_dict.py**) to define the target chain and specify the fixed positions. Residues in DESIGN are excluded from design with ProteinMPNN and are only designed with Rosetta. ProteinMPNN is executed with the specified sampling temperature ProteinMPNN_T and the generated bias file, producing a set of 100 candidate sequences. The highest-scoring sequence in terms of **global_score** is used for the subsequent modeling steps. Because ProteinMPNN does not provide a structure but only a sequence, **run_ProteinMPNN()** always spawns a **run_ESMfold_RosettaRelax()** to generate a structure and scores for Boltzmann selection.

4.3.3 Structure prediction with ESMfold, RosettaRelax and MDMin

run_ESMfold_RosettaRelax() using both the sequence and structure as input to predict the protein structure with ESMFold. Because ESMFold cannot predict the structure of the substrate-enzyme complex, the coordinates of the ligand are transferred from the parent structure file into the predicted structure after the alignment of the two structures. Afterward, sidechains are stripped from the ESMfold model, and the resulting backbone-ligand complex is repacked and relaxed using Rosetta with the FastRelax mover. For designs that do not provide a structure (e.g., ProteinMPNN), **run_ESMfold_RosettaRelax()** uses the structure of the parent variant as input.

4.3.4 Electric field calculations

calc_efields_score() is used to determine active-site electric fields of the input structures. Electric field calculation is performed with FieldTools (https://github.com/bunzela/FieldTools). Fields are calculated using point charges from the ff19SB AMBER forcefield. All stated fields correspond to the effective field along the scissile C-H bond. FieldTools relies on Coulomb's law using the point charges from the system's topology file and the coordinates from the input structure or trajectory to calculate the electric field along a target bond. FieldTools calculates the electric field vectors E using the Coulomb constant and the vector from the center of the target bond to the charges in the system. Subsequently, the effective field E_eff projected along the target bond is calculated from the scalar product of the directional unity vector along that bond and the total field vector E.

5 Full Manual

5.1 Input preperation

Input file in same directory as Jupyter notebook.

5.1.1 Input files for Rosettta

Add description how to make .params and .cst files

Does input structure need REMARK match?

5.1.2 Input files Amber (MDMin, FieldTools)

Add description how to make .frcmod and .prepi files

5.2 Create an Instance of the AI.zymes class

Add description here to what this cell is doing

```
FOLDER_HOME = '/raven/ptmp/bunzela/AIzymes/TEST'
import sys, os
from AIzymes_015 import *
AIzymes = AIzymes_MAIN(FOLDER_HOME = FOLDER_HOME)
```

5.3 Set up AI.zymes for design

Add description here to what this cell is doing AIzymes.setu

AIzymes.setup() options

WT	Name of the protein scaffold. Must fit to the input pdb structure in **Input**. if **FOLDER_PAR_STRUC** is set (see below), a **WT** structure is still required to create a reference sequence.
LIGAND	Ligand name. Must fit all ligand input files in the folder **Input**.
DESIGN	Comma-seperated string of residues that will be targeted with **RosettaDesign**. Usually, these are active-site or ligand-binding residues. Residue numbering starts with 1.
PARENT_DES_MED	List of Methods to be used for the re-design of the parent input structures. Default: ['RosettaDesign','ElectricFields']
DESIGN_METHODS	List of Lists defining the design methods after parent design is completed. In addition to the design methods, each list must start with a float. The flot indictes the probability of the specific design methdos to be used.

	Default:
	[[0.7,'RosettaDesign','ElectricFields'],\
	~~[0.3,'ProteinMPNN','ESMfold','RosettaRelax','ElectricFields']]
EXPLORE.	Set to True to activate EXPLORE modus.
	During EXPLORE, design will be performed faster
	Only use this EXPLORE for testing and not for actually design!
	Default: None
RESTRICT_RESIDUES .	
FOLDER_PARENT	
	Default: 'parent'
FOLDER_PAR_STRUC .	

5.3.1 General settings

5.3.2 General design settings

5.3.3 RosettaDesign settings

5.3.4 RosettaDRelax settings

5.3.5 ProteinMPNN settings

5.3.6 AlphaFold3 settings

5.3.7 ESMFold settings

5.3.8 MDMin settings

MDMin relies on Amber to minimize the input structure. MD minimization is requested by adding **MDMin** to **DESIGN_METHODS**.

5.3.9 Electric Field settings

Selection for electric fields using **efield** in **SELECTED_SCORES**. Electric field calculation is requested by adding **ElectricFields** to **DESIGN_METHODS**.

AIzymes.setup() options WEIGHT_EFIELD . . . . . . Adjust the weight of the **efield_score**

FIELDS_EXCL_CAT	Excluding the field effect of catalytic residues, recommended False
FIELD_TARGET	Define the bond along which the electric field is calculated.
	For Kemp this was: **:5TS@C9 :5TS@H04**.
	Atoms along which the field is calculated are defined using Amber selection masks.
	":" indicates the residue name or number, wheras **"@"** describes the atom name.

Note

In its current version, AI.zymes aims to design the highest possible poisitive electric field. To design negative electic fields, swap the two atoms defined in the **FIELD_TARGET**.

5.3.10 BioDC settings

work in progress...

5.4 Submit controller to run AI.zymes

Add description here to what this cell is doing

```
AIzymes.submit_controller()
```

5.5 Analysis tools to check AI.zymes

Add description here to what this cell is doing

```
AIzymes.plot(...)
AIzymes.print_statistics()
```

5.6 Extract best structures after AI.zymes run

Add description here to what this cell is doing

```
AIzymes.best_structures()
```

6 Code

6.1 design_AlphaFold3.py

Prepares commands to run protein structure prediction using AlphaFold3.

Functions:

seq_to_json()	Converts a sequence file into a JSON input for AlphaFold3.
prepare_AlphaFold3_MSA()	Prepares the AlphaFold3 MSA stage.
prepare_AlphaFold3_INF()	Prepares the AlphaFold3 inference stage.

6.1.1 seq_to_json()

Converts a sequence file (with a .seq extension) into a JSON input file for AlphaFold3.

Parameters:

seq_input (str)	Input sequence file location
working_dir (str)	working dir for AlphaFold3

6.1.2 prepare_AlphaFold3_MSA()

Performs MSA for structure prediction with AlphaFold3.

Parameters:

index (str)	The index of the protein variant to be predicted.
cmd (str)	Growing list of commands to be exected by run_design using submit_job.

Returns:

cmd (str)	Command to be exected by run_design using submit_job.

6.1.3 prepare_AlphaFold3_INF()

Performs MSA for structure prediction witt AlphaFold3.

Parameters:

index (str)	The index of the protein variant to be predicted.
cmd (str)	Growing list of commands to be exected by run_design using submit_job.

Returns:

cmd (str)	Command to be exected by run_design using submit_job.

6.2 design_ESMfold.py

Design ESMfold Module

Manages the structure prediction of protein sequences using ESMfold within the AIzymes project.

Functions:

- prepare_ESMfold	Prepares commands for ESMfold job submission.

6.2.1 prepare_ESMfold()

Predicts structure of sequence in {index} using ESMfold. Uses ligand coordinates from previous RosettaDesign.

Parameters:

index (str)	The index of the protein variant to be predicted.
cmd (str)	Growing list of commands to be exected by run_design using submit_job.

Returns:

cmd (str)	Command to be exected by run_design using submit_job.

6.3 design_match.py

Design Match Module

Provides functionalities for matching protein designs with specific constraints and requirements in the AIzymes workflow.

Functions:

- prepare_LigandMPNN	Executes the LigandMPNN pipeline for protein-ligand structure adaptation.

Modules Required:

6.3.1 run_RosettaMatch()

6.3.2 run_Matcher()

6.3.3 run_Matcher_apo()

6.3.4 separate_matches()

6.4 design_MDMin.py

Prepares commands to run MD minimization using Amber to refine protein structures.

Functions:

- prepare_MDMin()	Sets up commands for sidechain building (Rosetta), generating Amber input files (tleap), and MD minimization (sander)

6.4.1 prepare_MDMin()

Minimises protein structure in {index} using Amber.

Parameters:

- index (str)	The index of the protein variant to be relaxed.
- cmd (str)	collection of commands to be run, this script wil append its commands to cmd

6.5 design_MPNN.py

Manages all MPNN methods for protein design

Functions:

prepare_ProteinMPNN()	Prepares commands for ProteinMPNN job submission.
prepare_SolubleMPNN()	Prepares commands for SolubleMPNN job submission.
prepare_LigandMPNN()	Prepares commands for LigandMPNN job submission.
make_bias_dict()	Creates and writes a bias dictionary for MPNN based on an input PDB.
ProteinMPNN_check()	Checks that ProteinMPNN is installed.
LigandMPNN_check()	Checks that LigandMPNN is installed.

Modules Required:

6.5.1 prepare_LigandMPNN()

Uses LigandMPNN to redesign the input structure with index.

Parameters:

index (str)	The index of the designed variant.
cmd (str)	Growing list of commands to be exected by run_design using submit_job. Optional Parameters:
gpu_id (int)	ID of the GPU to be used. Returns:
cmd (str)	Command to be exected by run_design using submit_job

6.5.2 prepare_ProteinMPNN()

Uses ProteinMPNN to redesign the input structure with index.

Args: index (str): The index of the designed variant. cmd (str): Growing list of commands to be exected by run_design using submit_job.

Returns:

cmd (str)	Command to be exected by run_design using submit_job

6.5.3 prepare_SolubleMPNN()

Uses SolubleMPNN to redesign the input structure with index.

Args: index (str): The index of the designed variant. cmd (str): Growing list of commands to be exected by run_design using submit_job.

Returns:

cmd (str)	Command to be exected by run_design using submit_job

6.5.4 make_bias_dict()

Creates a bias dictionary for the input PDB and writes it to a JSON file.

Args: PDB_input (str): Path to the input PDB file. folder_mpnn (str): Folder where the bias JSON will be saved.

6.5.5 ProteinMPNN_check()

Checks if ProteinMPNN is installed

6.5.6 LigandMPNN_check()

Checks if LigandMPNN is installed

6.6 design_RosettaDesign.py

Prepares commands to run RosettaDesign methods for protein design.

Functions:

prepare_RosettaDesign	Prepares RosettaDesign commands for job submission.

6.6.1 prepare_RosettaDesign()

Designs protein structure in {index} using RosettaDesign.

Args: index (str): Index assigned to the resulting design.

Returns:

cmd (str)	Command to be exected by run_design using submit_job.

6.7 design_RosettaRelax.py

Prepares commands to run RosettaRelax methods to refine protein structures.

Functions:

prepare_RosettaRelax	Sets up commands for RosettaRelax job submission.

6.7.1 prepare_RosettaRelax()

Relaxes protein structure in {index} using RosettaRelax.

Parameters:

index (str)	The index of the protein variant to be relaxed.
cmd (str)	collection of commands to be run, this script wil append its commands to cmd

Optional Parameters:

PreMatchRelax (bool)	True if ESMfold to be run without ligand (prior to RosettaMatch).

6.8 helper.py

Contains utility functions and supporting routines used across multiple modules within the AIzymes project.

Functions:

normalize_scores()	Normalizes scores for each selected score type in a DataFrame.

one_to_three_letter_aa()	Converts a one-letter amino acid code to its corresponding three-letter code.
run_command()	Executes a shell command with optional output capturing and error handling.
load_main_variables()	Loads main configuration variables from a JSON file into the project.
save_main_variables()	Saves current project variables to a JSON file.
submit_job()	Submits a computational job using system-specific commands.
sequence_from_pdb()	Extracts the amino acid sequence from a PDB file.
generate_remark_from_all_scores_df()	Generates REMARK annotations from catalytic residue data.
save_cat_res_into_all_scores_df()	Extracts catalytic residue indices and names from a PDB file and saves them into the scores DataFrame.
reset_to_after_parent_design()	Resets folders and updates the scores DataFrame after completing parent designs.
reset_to_after_index()	Removes all design entries and corresponding folders beyond a specified index.
save_all_scores_df()	Atomically saves the scores DataFrame to a CSV file.
get_best_structures()	Identifies the best structure designs, generates plots, and archives selected structures.
remove_intersection_best_structures()	Removes overlapping structure files between designated best-structure folders.
trace_mutation_tree()	Traces the mutation pathway through design generations and plots score progression.
print_average_scores()	Prints the average scores of designs (if applicable).
wait_for_file()	Waits for a specified file to exist and reach a non-zero size.
hamming_distance()	Computes the Hamming distance between two sequences.
exponential_func()	Evaluates an exponential decay function.

Modules Required:

6.8.1 normalize_scores()

Normalizes scores for each selected score type from the provided DataFrame.

Parameters:

unblocked_all_scores_df (df)	DataFrame containing the unnormalized score columns.
norm_all (bool)	If True, use min-max normalization based on predefined ranges; if False, use Z-score normalization.
extension (str)	Suffix appended to the score keys (defaults to ”score”). Returns:
dict	A dictionary containing the normalized score arrays for each score type.

6.8.2 one_to_three_letter_aa()

Converts a one-letter amino acid code to its corresponding three-letter code in all uppercase.

Parameters:

one_letter_aa (str)	A one-letter amino acid code. Returns:
str	The corresponding three-letter amino acid code.

6.8.3 run_command()

Executes a command (typically for running Python scripts) with optional output capturing and error handling.

Parameters:

command (list of str)	The command to run, provided as a list of strings.
cwd (str, optional)	The working directory in which to run the command.
capture_output (bool, optional)	If True, captures standard output and error; otherwise, output is suppressed. Defaults to False. Returns:
str	The standard output of the command, if captured.

6.8.4 load_main_variables()

Loads main configuration variables from a JSON file and sets them as attributes of the provided object.

Parameters:

FOLDER_HOME (str)	The path to the main folder where the variables JSON file is located.

6.8.5 save_main_variables()

Saves main configuration variables from the object's attributes to a JSON file.

6.8.6 sequence_from_pdb()

Extracts the amino acid sequence from a PDB file.

Parameters:

pdb_in (str)	The base path (without extension) to the PDB file. Returns:
str	The amino acid sequence extracted from the PDB file.

6.8.7 generate_remark_from_all_scores_df()

Generates a remark string from catalytic residue data contained in the all_scores_df DataFrame for a given index.

Parameters:

index	The index (row key) in the DataFrame for which to generate the remark. Returns:
str	A multi-line string containing REMARK lines for each catalytic residue.

6.8.8 save_cat_res_into_all_scores_df()

Finds catalytic residue indices and names from a PDB file and saves them into the all_scores_df DataFrame at the specified index.

Parameters:

index	The DataFrame row index where catalytic residue information should be stored.
PDB_path (str)	The path (without extension) to the PDB file. Optional Paramters:

save_resn (bool, optional)	If True, saves the residue names; otherwise, only saves indices.

6.8.9 reset_to_after_parent_design()

6.8.10 reset_to_after_index()

This function resets the run back to a chosen index. It removes all later entries from the all_scores.csv and the home dir. index: The last index to keep, after which everything will be deleted.

6.8.11 save_all_scores_df()

6.8.12 save_resource_log_df()

6.8.13 get_best_structures()

6.8.14 remove_intersection_best_structures()

6.8.15 trace_mutation_tree()

6.8.16 wait_for_file()

Wait for a file to exist and have a non-zero size.

6.8.17 hamming_distance()

6.8.18 exponential_func()

6.8.19 create_new_index()

The create_new_index function is responsible for generating a new index. It adds a new row to all_scores_df and makes the folders for the index.

Parameters:

parent_index (str)	The index of the parent variant for designs. luca (str): design_method (str): next_steps (list):

6.8.20 count_mutations()

Counts the number of differing positions (mutations) between two sequences.

Parameters:

seq1 (str)	The first sequence.
seq2 (str)	The second sequence. Returns:
int	The number of mutations.

6.8.21 print_statistics_df()

6.8.22 neg_norm_array()

6.8.23 get_mutations()

6.8.24 count_offspring()

6.8.25 plot_scores()

6.8.26 get_active_site_sequence()

6.9 main_design.py

Main Design Module. Coordinates various design steps, managing the workflow of Rosetta, ProteinMPNN, and other modules within the AIzymes project.

Functions:

run_design	Runs the selected design step based for the given index.

Modules Required:

6.9.1 run_design()

Start a job by creating and executing a submission script based on the current index and design_step.

Parameters:

index (int)	The index representing the current design.
design_step (str)	The design_step to be executed.

6.10 main_running.py

Contains the main control functions for running AIzymes, including job submission, score updating, and Boltzmann selection. The functions in this file are responsible for the high-level management of the design process, interacting with the AIzymes_MAIN class to initiate, control, and evaluate design variants.

Functions:

start_controller()	Runs the design loop until the maximum number of designs is reached.
check_running_jobs()	Returns the count of current and scheduled jobs.
update_potential()	Appends new scores to potential files and recalculates averages.
update_scores()	Updates variant scores from outputs and unblocks completed jobs.
boltzmann_selection()	Chooses a variant based on Boltzmann-weighted potentials.
check_parent_done()	Checks if a parent variant's design jobs are finished.
start_parent_design()	Initiates design processing for parent variants.
start_calculation()	Launches the next design step for a selected variant.
create_new_index()	Creates a new design index, updates scores, and logs the new entry.

Modules Required:

6.10.1 start_controller()

The start_controller function is called in the AIzyme_0X script's controller function. The controller function decides what action to take, assures that the maximum number of design jobs are run in parallel,

collects information from the designs and stores them in a shared database, selects the variants to submit for design, and decides the type of structure prediction to perform with the selected variant.
Each ofthese tasks is performed by the functions introduced before, and thereforer the start_controller function controls the flow of actions.

6.10.2 select_scheduled_variant()

6.10.3 check_running_jobs()

The check_running_job function returns the number of parallel jobs that are running by counting how many lines in the qstat output are present which correspond to the job prefix. This is true when working on the GRID, for the other system the same concept is being used but the terminology differs.

Returns:

int	Number of running jobs for the specific system.

6.10.4 update_potential()

Updates the potential file for a given score type at the specified variant index.
Creates or appends to a '¡score_type¿_potential.dat' file in 'FOLDER_HOME/¡index¿', calculating and updating potentials for the parent variant if necessary.

Parameters:

score_type (str)	Type of score to update (e.g., total, interface, catalytic, efield).
index (int)	Variant index to update potential data.

6.10.5 update_scores()

Updates the all_scores dataframe.
This function iterates over design variants, updating scores based on files generated by different processes. It also updates sequence information, tracks mutations, and saves the updated DataFrame.

6.10.6 update_resource_log()

6.10.7 boltzmann_selection()

Selects a design variant based on a Boltzmann-weighted probability distribution.
Filters variants based on certain conditions (e.g., scores, block status), then computes probabilities using Boltzmann factors with a temperature factor ('KBT_BOLTZMANN') to select a variant for further design steps.

Returns:

int	Index of the selected design variant.

6.10.8 schedule_design_method()

6.10.9 start_calculation()

6.11 main_scripts.py

6.11.1 make_main_scripts()

6.12 main_startup.py

6.12.1 submit_controller_parallel()

6.12.2 initialize_controller()

6.12.3 prepare_input_files()

6.12.4 initialize_variables()

6.12.5 initialize_logging()

6.12.6 check_settings()

6.12.7 aizymes_setup()

6.12.8 make_empty_all_scores_df()

Makes the starting all_scores_df dataframe that collects all information about an AI.zymes run

6.12.9 make_empty_resource_log_df()

Makes the starting resource_log dataframe to track jobs running over time

6.12.10 schedlue_parent_design()

Adds all parent design runs to the all_score_df dataframe

6.13 plotting.py

6.13.1 make_plots()

6.13.2 load_plot_scores_df()

6.13.3 plot_statistics()

6.13.4 plot_scores_hist()

6.13.5 plot_scores_v_index()

6.13.6 plot_scores_v_generation()

6.13.7 plot_mutations_v_generation()

6.13.8 print_vals()

6.13.9 combine_images()

6.13.10 plot_resource_log()

6.13.11 plot_score_v_index()

6.13.12 plot_score_hist()

6.13.13 plot_score_v_generation_violin()

6.14 plotting_tree.py

6.14.1 plot_tree()

6.14.2 calculate_subtree_sizes()

6.14.3 assign_polar_positions()

6.14.4 adjust_uniform_angles()

6.14.5 polar_to_cartesian()

6.15 scoring_BioDC.py

Calculates Redox potential using the BioDC software.

Functions:

- prepare_BioDC	Prepares commands for BioDC job submission.

6.15.1 prepare_BioDC()

Redox potential calculation for {index} with BioDC Currently hardcoded for m4D2

Parameters:

index (str)	The index of the protein variant to be analyzed.
cmd (str)	Growing list of commands to be exected by run_design using submit_job.

Returns:

cmd (str)	Command to be exected by run_design using submit_job.

6.15.2 get_redox_score()

6.15.3 BioDC_check()

Checks if BioDC is installed

6.16 scoring_efields.py

6.16.1 prepare_efields()

Calculate electric fields for structure in {index}.

Paramters: index (str): The index of the protein variant for which efields are to be calculated. cmd (str): Growing list of commands to be exected by run_design using submit_job.

Returns:

cmd (str)	Command to be exected by run_design using submit_job.

6.16.2 get_efields_score()

6.16.3 update_efieldsdf()

Adds a new row to "{FOLDER_HOME}/electric_fields.csv" containing the electric fields generated by FieldTools.py for all residues in the protein

6.17 setup_system.py

Contains system specifc information. At the Moment, this is all hard-coded. In the future, this will be part of the installation of AIzymes.

set_system() contains general variables. submit_head() constructs the submission header to submit jobs.

6.17.1 set_system()

6.17.2 submit_head()