SHAIPER - AI for Surrogate Modelling Documentation

Table of Contents

1. SHAIPER - AI for Surrogate Modelling

1.1 About

Welcome to the `SHAIPER - AI for Surrogate Modelling` community!

`SHAIPER` is currently being developed at the IT4Innovations National Supercomputing Center. Its purpose is to offer a comprehensive toolbox for experimentation with and the utilization of various models for surrogate modelling. The goal is to implement a state of the art

software with respect to parallel scalability and efficiency while letting the researchers focus on more interesting aspects of their work, i.e. experiment with novel architectures, optimization methods etc.

`SHAIPER` is designed to offer comprehensive toolbox for:

- Efficient massivelly parallel dataset manipulation, analysis and information exports to standalone files.
- Modular AI models assembly.
- Hyperparameter space exploration for finding optimal model structures and learning behaviour.
- Model training monitoring and reporting.
- Feature optimizations based on gradient/genetic and mixed algorithms using the trained models.

Current Capabilities

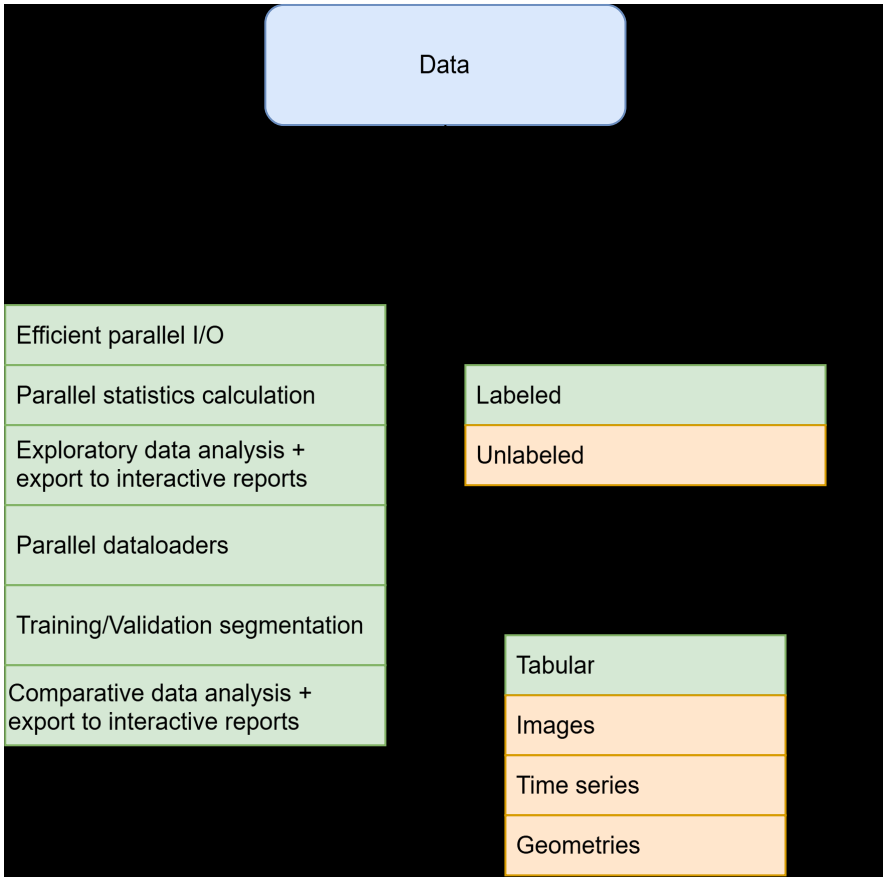
In the current state, the software is usable for the following tasks:

- Tabular Database creation, augmentation, transformation, utilization in training and statistics calculation + report exporting.
- Modular model assembly and utilization in training and parameter optimization
- Arbitrary Optimizer and Scheduler utilization
- Basic Loss function utilization
- Training progress reporting
- Parameter search via first order gradient methods

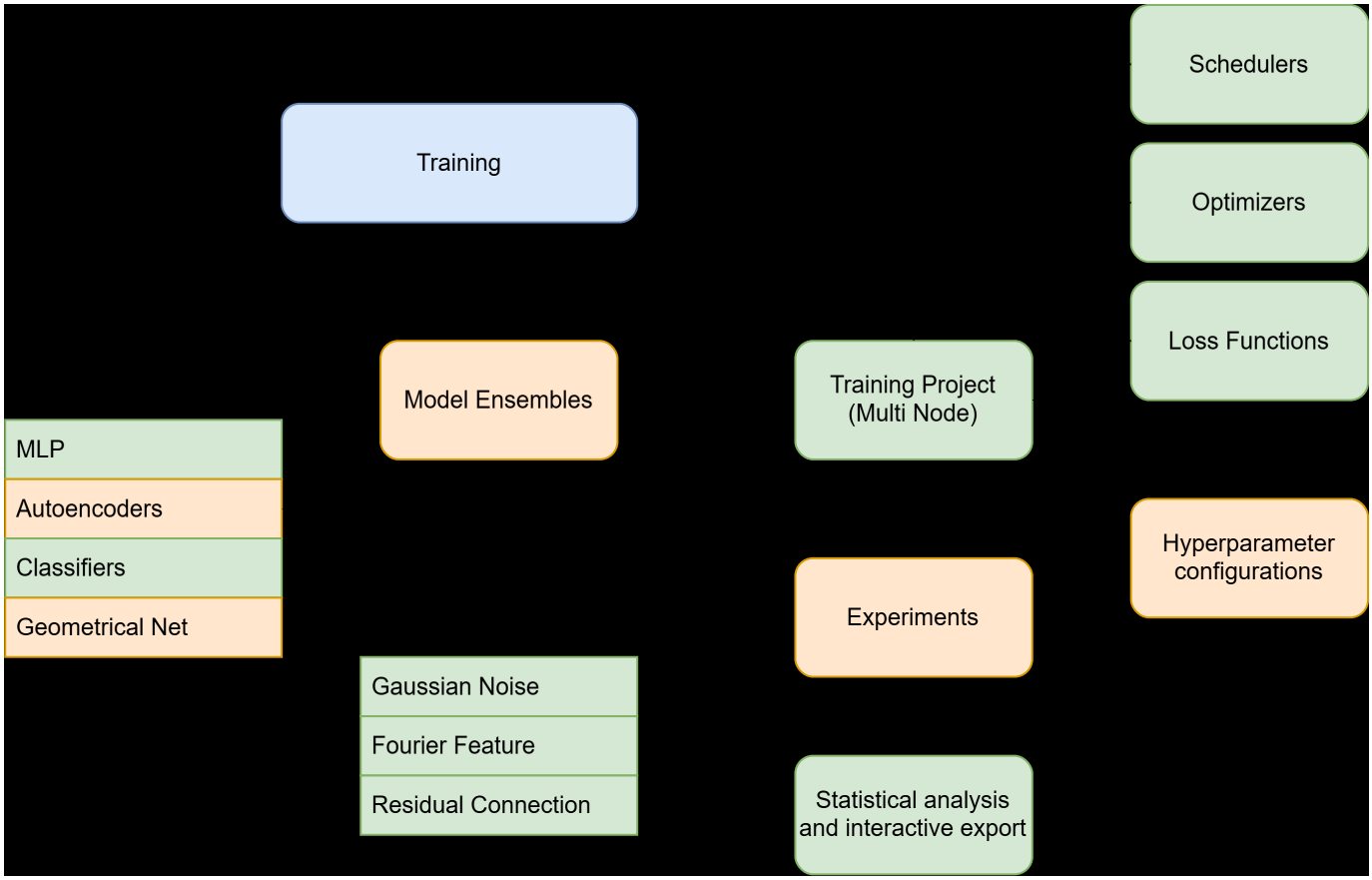
1.2 Current status of the development with respect to ecosystem around the source codes

Development process
Continuous Integration
Code quality assessment and feedback
Unit tests
Documentation

1.3 Data management Design Intention and Development Status



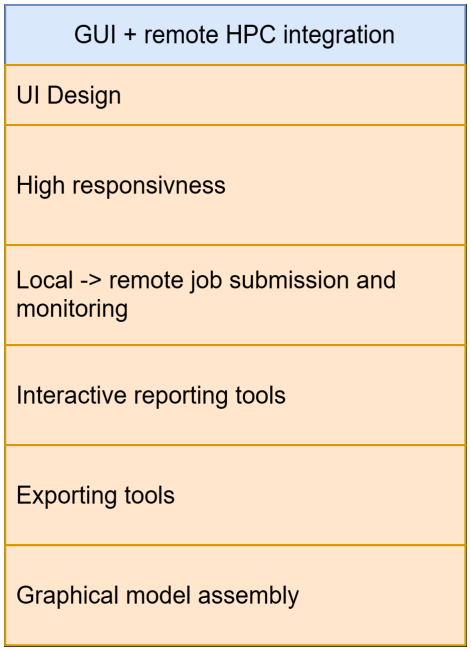
1.4 Training Workflow and Development Status



1.5 Utilization Development Status

Inference + Optimization
Automatic parametrization of the input space
Optimization of the input space with respect to the output metric
Avoiding invalid parameter spaces
Gradient optimization methods
Optimization methods based on Genetic Algorithms

1.6 Design Pillars of required UIX



2. Installation

For the installation steps, refer to the Installation Tutorial.

3. How to run the software

To learn about how to run & utilize the software, refer to the Guide for Users.

4. Want to participate in development?

If you are a contributing developer, please refer to the Note for Developers.

5. Downloading the repository

In order to download the git repository, follow these steps:

- If not already installed, install the Git versioning tool via `apt install -y git-all`.
- Choose an appropriate directory for the repository and navigate to it via the `cd` command.
- Download the git repository to your local storage via `git clone https://code.it4i.cz/kra568/ai-surrogate-modelling.git`.
- Navigate to the newly created directory via `cd ai-surrogate-modelling`.
- Setup your environment by calling the installation script via `sudo ./install/install.sh`
- To update the Python packages required by the software in the virtual Python environment, call `./install/install_python.sh` (this script is also called from the command above)

Now you have access to the source codes and can either start developing or install the package via pip.

6. Wheel Installation

You have downloaded the git repository, but do not wish to work on development. For this reason, we include a simple installation script.

Navigate to the project directory and run the command `sudo bash install/install_wheel.sh`. This installs a Python virtual environment `python_venv/.ai-for-surrogate-modelling` in your home directory, constructs the installation files and installs this software to the newly created virtual environment via pip.

Once installed, you can start experimenting with the software.

7. Uninstall the Wheel

To uninstall this software from the virtual environment, run the following series of commands from the terminal:

```
. $ai_for_surrogate_modelling_path/bin/activate
pip uninstall -y SHAIPER-AI-for-Surrogate-Modelling
```

8. Overview

This software has a single entry point, which parses the provided configuration file and performs the operations specified therein.

If you are using the installed package vie pip, the following commands will initiate the program.

```
. $ai_for_surrogate_modelling_path/bin/activate
mpirun -n N ai4sm --config path/to/main/config.yml
```

or alternatively, when you are working directly in the code-base (e.g. developing the software):

```
. $ai_for_surrogate_modelling_path/bin/activate
mpirun -n N python -m AI4SurrogateModelling.entry_point --config path/to/main/config.yml
```

where `N` denotes the desired number of mpi processes and `path/to/main/config.yml` denotes the main configuration file.

9. Main Configuration

The software uses a system of configuration files controlling the flow of the program. The aim is to lessen the hardships imposed by efficient parallel implementations, sofisticated reporting, database management, consistent training procedures and parallel I/O. Below we describe the basic structure of the configuration files. See Examples for more concrete tutorials.

9.1 Main Configuration File

So, you have run the software with the attribute `--config path/to/main/config.yml`, which points to the main configuration file. The main configuration YAML file consists of three components.

`main`

The top-level behaviour of the software is described inside the main section.

```
main:
    logging: info/debug                    # specifies the loggin level
    logging_directory: path/to/logs/storage # where should the logs be stored
    parallelization: distributed/isolated   # should all processes train a single model, or should each process do its own thing?
    seed: int                              # sets all random seeds to this value, for reproducibility purposes
```

`aliases`

Writing configuration files may contain very long and/or repeating values. To ease the writing of the configurations, the user can specify so-called aliases in the form of `alias_key: alias_value`. This tells the configuration parser, that all values inside the configuration files in the form `$alias_key$` should be replaced by the corresponding value `alias_value`.

Aliases can be recursively defined, i.e. one alias can use other alias in its value. Aliases cannot use aliases in their keys. The software checks if the aliases are defined correctly, i.e. that all aliases are replaced by an alias value and that nested aliases do not form a cyclic alias definition.

```
aliases:
    alias_key_0: alias_value_0
    alias_key_1: alias_value_1
    alias_key_2: alias_value_2-$alias_key_1$
```

`operations`

The last component of the main configuration file contains the operations to be performed by the program. Each operation has its unique numerical suffix which specifies the order of operations. The configuration parser checks the correctness of the operational IDs.

```
operations:
    operation_0: path/to/config_0.yml
    operation_25: path/to/config_25.yml
    operation_15: path/to/config_15.yml
    operation_10: path/to/config_10.yml
```

As you can see, the numerical suffices need not to form a sequence and do not need to be ordered in the configuration file. The existence of the files the operation points to is verified before any computation takes place.

10. Operational Configuration

The main configuration file describes the top-level configuration, which contains operations linking to the operational configuration files.

Each operational configuration file contains two basic sections: `objects` and `actions`.

`objects`

The object section contains the specification of objects used during the program for objects/actions specified in this operational file and all successive operational files.

```
objects:
    object_UID_0:
        constructor: path.to.class.definition_0
        parameters:
            parameter_name_0: parameter_value_0
            parameter_name_1: parameter_value_1
                        ...

    object_UID_1:
        constructor: path.to.class.definition_1
        parameters:
            parameter_name_0: parameter_value_0
            parameter_name_1: parameter_value_1
                        ...
```

This allows the user to specify any objects contained in the software. In the snippet above, the user specified two objects, each with a unique ID (the configuration parser checks the uniqueness requirement). Each objects needs to point to a method or a class in the code-base which returns the instance of the object initialized with the supplied parameters.

So, the user can use any objects, but needs to know how to construct them. The objects can be referenced by their unique IDs in the configuration files. For example, an action can be done using an object, or an object can use another object in its constructor, etc.

Object UIDs, paths to constructors, parameter names and values can use aliases specified in the main configuration file.

`actions`

Constructing objects and not using them is not very productive. Thus, the final building block of our configuration system is the system of `actions`. An action can reference an instance of an object, a static function or a global function.

In the snippet below, you can see `action_0` referencing an instance of the object specified above with UID `object_UID_0` and calling its member function `class_function_name` with the supplied parameters (if any).

The action `action_42` references either a static function or a global function with the supplied parameters.

The system performs the actions in order with respect the the integer suffix. A check is performed whether the referenced functions are available in the code-base and whether the objects have been specified in the configuration files (i.e. the program knows how to construct them).

```
actions:
  action_0:
    object_uid: object_UID_0
    fname: class_function_name
    parameters:
        parameter_name_0: parameter_value_0
        parameter_name_1: parameter_value_1
                    ...

  action_42:
    fname: path.to.function.call
    parameters:
        parameter_name_0: parameter_value_0
        parameter_name_1: parameter_value_1
                    ...
```

11. Example Tabular (Random)

Here you can find an overview of a complete example in which you learn:

```
- How to define random Tabular datasets with various data subgroups
- How to process the dataset and export its various statistics
- How to define a basic loss function
- How to define optimizers
- How to define schedulers
- How to define and train a model
- How to use a trained model in search of optimal parameter configurations
```

The configuration scripts for this example can located at `conf/examples/random`.

To run this example, use the commands depicted in the Overview.

11.1 The Main configuration file

(`conf/examples/random/run_config.yml`) This file defines a lot of aliases and the complete workflow in terms of the operational configuration files. The system of aliases was chosen to simplify the definition of the model size, size of the dataset and datatype of the datasets and models being used.

```yaml
main:
  logging: info # possible values: info/debug
  logging_directory: logs/examples/random # default value: logs
  parallelization: distributed # possible values: distributed/isolated
  seed: 42

aliases:
  project_path: AI4SurrogateModelling.src
  dtype: float32

  input_key_known: inputs_group_known
  input_key_unknown: inputs_group_unknown
  output_key: outputs_group

  input_dim: 7
  output_dim: 4
  nrows: 10000

  ConfigDirectory: conf/examples/random

  DatabaseTabular: $project_path$.database.labeled.tabular.database_tabular.DatabaseTabular
  Importer: $project_path$.importer.tabular.database_importer_tabular_random.ImporterTabularRandom

  dim_layer_1: 256
  dim_layer_2: 128
  dim_layer_3: 64
  dropout_layer_1: 0.25
  dropout_layer_2: 0.25
  dropout_layer_3: 0.25

  Example_ID: random_example
  DatabaseDirectory: DATABASE_EXAMPLES/$Example_ID$
  results_dir: results/random_example

operations:
  operation_0: $ConfigDirectory$/data_create.yml
  operation_1: $ConfigDirectory$/data_export.yml
  operation_10: $ConfigDirectory$/define_models.yml
  operation_20: $ConfigDirectory$/define_optimizers.yml
  operation_30: $ConfigDirectory$/define_schedulers.yml
  operation_40: $ConfigDirectory$/define_losses.yml
  operation_51: $ConfigDirectory$/model_refine.yml
  operation_60: $ConfigDirectory$/find_parameters.yml
```

11.2 Operation - Create Database

(`conf/examples/random/data_create.yml`) The first operation performed is to define the dataset. For this we need to define two objects, the database itself and at least one importer. In our case, we use a tabular database and a random tabular importer (i.e. it fills the database with random data)

The database object has UID `database` and the importer object has UID `random_importer` for future reference in other operational configuration files. The random importer is tasked to define tabular data with 11 columns, 7 of wchich are inputs and 4 of which are the

outputs. The 7 inputs are further divided into two groups. Group `$input_key_known$` represents those inputs, for which we prescribe known values in the utilization step. Group `$input_key_unknown$` represents a subset of inputs the values of which we are tasked to find during the utilization step.

After the object definition, several actions are defined.

- `action_0` clears the whole database by calling a member function `clear_database()` of the `TabularDatabase` class.
- `action 1` uses another member function of the TabularDatabase `add_data()`, which uses the supplied importers to fill the database.
- `action_2` takes the data in the database and transforms each column in each data group to fit into the specified range.
- `action_4` removes all columns which do not contain enough information, e.g. columns which are highly correlated to ther columns, or columns which do not contain enough variance in the data.
- `action_5` defines a split of the database into three groups, `train` with 80% of rows, `test` with 20% of rows and `validation` with 0% of rows.

In this example, the database is stored in the directory `DATABASE_EXAMPLES/random_example` is created.

```
objects:
  database:
    constructor: $DatabaseTabular$
    parameters:
      path_tgt: $DatabaseDirectory$

  random_importer:
    constructor: $Importer$
    parameters:
      nentries: $nrows$
      data_column_groups:
        $input_key_known$: [0, 1]
        $input_key_unknown$: [2, 3, 5, 7, 9]
        $output_key$: [4, 6, 8, 10]

actions:
  action_0:
    object_uid: database
    fname: clear_database

  action_1:
    object_uid: database
    fname: add_data
    parameters:
      importers: [
        random_importer,
      ]

  action_2:
    object_uid: database
    fname: transform_range
    parameters:
      keys: [$input_key_known$, $input_key_unknown$, $output_key$]
      target_range: (-1, 1)

  action_4:
    object_uid: database
    fname: cleanup_columns
    parameters:
      keys_to_consider: [$input_key_known$, $input_key_unknown$, $output_key$]
      correlation_eps: 1e-6
      variance_eps: 1e-6

  action_5:
    object_uid: database
    fname: train_test_split
    parameters:
      train_ratio: 0.8
      test_ratio: 0.2
      validation_ratio: 0.0
```

11.3 Operation - Database Export

Each database can be analyzed and results presented. One of the methods is to generate a standalone HTML file containing predefined information.

This operation file defines one object `database`, located at the path identical to the `database` in the previous configuration file.

There is only one action to be performed, to call the method `export_to_html` located deep in the source code hierarchy, which takes the supplied tabular database and exports various statistics to the specified file.

In this example, a file at `results/random_example/reports/database.html` is created.

```
objects:
  database:
    constructor: $DatabaseTabular$
    parameters:
      path_tgt: $DatabaseDirectory$

actions:
  action_20:
    fname: $project_path$.export.reports.tabular.ReporterTabular.export_to_html
    parameters:
      database: database
      tgt_fn: $results_dir$/reports/database.html
```

11.4 Operation - Model Definition

This operational file showcases the possibilities of creating nested object dependencies and arbitrary dependent structures. This file defines several pytorch models, all of them combining into a single `final_model` to be trained and used utilization.

```
objects:

  block_model_MLP_01:
    constructor: $project_path$.model.models.other.mlp.MultiLayerPerceptron
    parameters:
      dtype: $dtype$
      layer_vertices: [$dim_layer_1$, $dim_layer_1$]
      layer_activations: [['layer_norm_$dim_layer_1$', 'dropout_$dropout_layer_1$'], ['gelu']]
      layer_biases: [False, True]
      name: "The first hidden MLP block"
      input_keys: [$input_key_known$]
      output_keys: [$input_key_known$]

  block_model_MLP_02:
    constructor: $project_path$.model.models.other.mlp.MultiLayerPerceptron
    parameters:
      dtype: $dtype$
      layer_vertices: [$dim_layer_2$, $dim_layer_2$]
      layer_activations: [['layer_norm_$dim_layer_2$', 'dropout_$dropout_layer_2$'], ['gelu']]
      layer_biases: [False, True]
      name: "The second hidden MLP block"
      input_keys: [$input_key_known$]
      output_keys: [$input_key_known$]

  block_model_MLP_03:
    constructor: $project_path$.model.models.other.mlp.MultiLayerPerceptron
    parameters:
      dtype: $dtype$
      layer_vertices: [$dim_layer_3$, $dim_layer_3$]
      layer_activations: [['layer_norm_$dim_layer_3$', 'dropout_$dropout_layer_3$'], ['gelu']]
      layer_biases: [False, True]
      name: "The third hidden MLP block"
      input_keys: [$input_key_known$]
      output_keys: [$input_key_known$]

  block_model_MLP_input:
    constructor: $project_path$.model.models.other.mlp.MultiLayerPerceptron
    parameters:
      dtype: $dtype$
      layer_vertices: [$input_dim$, $dim_layer_1$]
      layer_activations: [[None], ['gelu']]
      layer_biases: [False, True]
      name: "An input MLP block"
      input_keys: [$input_key_known$, $input_key_unknown$]
      output_keys: [$input_key_known$]

  transfer_layer_00:
    constructor: $project_path$.model.models.other.mlp.MultiLayerPerceptron
    parameters:
      dtype: $dtype$
      layer_vertices: [$dim_layer_1$, $dim_layer_2$]
      layer_activations: [[None], ['gelu']]
      layer_biases: [False, True]
      name: "A transfer MLP block"
      input_keys: [$input_key_known$]
      output_keys: [$input_key_known$]

  transfer_layer_01:
    constructor: $project_path$.model.models.other.mlp.MultiLayerPerceptron
    parameters:
      dtype: $dtype$
      layer_vertices: [$dim_layer_2$, $dim_layer_3$]
      layer_activations: [[None], ['gelu']]
      layer_biases: [False, True]
      name: "A transfer MLP block"
      input_keys: [$input_key_known$]
      output_keys: [$input_key_known$]
```

```
  transfer_layer_output:
    constructor: $project_path$.model.models.other.mlp.MultiLayerPerceptron
    parameters:
      dtype: $dtype$
      layer_vertices: [$dim_layer_3$, $output_dim$]
      layer_activations: [[None], ['tanh']]
      layer_biases: [False, True]
      name: "A final output MLP block"
      input_keys: [$input_key_known$]
      output_keys: [$output_key$]

  residual_layer_00:
    constructor: $project_path$.model.models.layers.residual.ResidualLayer
    parameters:
      dtype: $dtype$
      model: block_model_MLP_01

  residual_layer_01:
    constructor: $project_path$.model.models.layers.residual.ResidualLayer
    parameters:
      dtype: $dtype$
      model: block_model_MLP_02

  residual_layer_02:
    constructor: $project_path$.model.models.layers.residual.ResidualLayer
    parameters:
      dtype: $dtype$
      model: block_model_MLP_03

  final_model:
    constructor: $project_path$.model.models.other.sequential.Sequential
    parameters:
      layers: [
        block_model_MLP_input,
        residual_layer_00,
        transfer_layer_00,
        residual_layer_01,
        transfer_layer_01,
        residual_layer_02,
        transfer_layer_output
      ]
      name: "A composite model for the vodik voltage prediction task"
```

11.5 Operation - Optimizer Definition

This file contains a definition of three built-in optimizers with some of its most relevant parameters exposed. Users can reference any built in optimizers or custom-built ones.

```
objects:
  optimizer_adam:
    constructor: torch.optim.Adam
    parameters:
      lr: 1e-3
      betas: (0.8, 0.99)
      eps: 1e-06
      weight_decay: 0
      amsgrad: False

  optimizer_adamW:
    constructor: torch.optim.AdamW
    parameters:
      lr: 1e-3
      weight_decay: 1e-5

  optimizer_SGD:
    constructor: torch.optim.SGD
    parameters:
      lr: 1e-3
      momentum: 0
      dampening: 0
      weight_decay: 0
      nesterov: False
```

11.6 Operation - Scheduler Definition

This file contains a definition of one learning rate scheduler, namely the `LambdaLR` scheduler. User can reference any built in and applicable callable object to be used as the lambda function.

We offer one lambda function example located at `AI4SurrogateModelling.src.training.schedulers.lambdas.cyclical_lr.CyclicalLR` which serves as a cyclical learning rate manipulator.

```
objects:
  lambda_scheduler_function_0:
    constructor: $project_path$.training.schedulers.lambdas.cyclical_lr.CyclicalLR
    parameters:
      log_factor_low: -1
      log_factor_high: 1
      periods: [500, 1000, 2000]

  scheduler_LAMBDA_0:
    constructor: torch.optim.lr_scheduler.LambdaLR
    parameters:
      lr_lambda: lambda_scheduler_function_0
```

11.7 Operation - Loss Definition

This file contains an example of a very basic loss object used in the training procedure. The loss object has UID `loss_0` and the object constructor is located at `AI4SurrogateModelling.src.training.loss.loss_base.LossBase`. The constructor parameters allow the user to define losses based on the

- weights & biases of the model
- difference in data calculated by the model specified by a nested dictionary. In this example, the loss is calculated between `real['outputs_group']` and `predicted['outputs_group']`
- partial derivatives of data calculated by the model with respect to other data, together with the prescription on the intervals in which the partial derivatives should lie.

The loss function keeps track of all individual losses with coefficients > 0 and provides a single criterion value to be used by a hyperparameter optimization procedure.

```
objects:
  loss_0:
    constructor: $project_path$.training.loss.loss_base.LossBase
    parameters:
      model_parameters:
        mae: 0
        mse: 0
        rmae: 0
        rmse: 0

      outputs:
        # a key contained in the data provided by the model
        $output_key$:
          # a key contained in the data provided by the dataloader
          $output_key$:
            mae: 0
            mse: 1
            rmae: 0
            rmse: 0

      # derivatives:
      #   # a key contained in the data provided by the model
      #   $output_key$:
      #     # a key contained in the data provided by the dataloader
      #     $input_key_known$:
      #       # partial derivative of output with index 0 w.r.t. input with index 0
      #       'out[0]/in[0]':
      #         constraints: ['value >= 0', 'value <= 1'] # we want to enforce two constraints on the partial derivative
      #         mae: 0 # MAE loss component coefficient
      #         mse: 0 # MSE loss component coefficient
```

11.8 Operation - Refine Model

This file initializes the database to be used during training and performs a single action located at `AI4SurrogateModelling.src.training.trainers.feed_forward.refine`. This action initializes a model (if it does not exist) and trains it for the specified number of epochs using the supplied scheduler, optimizer and loss function.

The training state and progress are stored inside the directory `results/random_example/training`. The standalone HTML report visualizing the training progress and best model properties is generated after the training finishes and is located in `results/random_example/reports/training.html`

```
objects:
  database:
    constructor: $DatabaseTabular$
    parameters:
```

```
      path_tgt: $DatabaseDirectory$

actions:
  action_0:
    # training function to be used
    fname: $project_path$.training.trainers.feed_forward.refine

    parameters:
      checkpoint_dir: $results_dir$/training
      number_of_epochs: 10
      batch_size: 1024
      dtype: $dtype$

      data_uid: database
      model_uid: final_model
      optimizer_uid: optimizer_adam
      scheduler_uid: scheduler_LAMBDA_0
      reset: True
      report_dir: $results_dir$/reports

      loss_uid: loss_0
```

11.9 Operation - Model Utilization

Finally, we take the best model obtained via the refinement process with UID `loaded_model`, the underlying database with UID `database`, a gradient optimizer, scheduler and loss and perform a single action: gradient based parameter optimization located at `AI4SurrogateModelling.src.utilization.parameter_estimation.gradient_sampler.GradientParameterSampler.estimate_parameters_tabular`.

The optimization runs for the specified number of epochs and attempts to find the specified number of parameter configurations. The Parameters are kept inside the known value boundaries (inferred from the supplied database). The search space is restricted by a known set of input/output values. In this example, the inputs from the group `inputs_group_known` are to be found by the optimization, the inputs from group `inputs_group_unknown` and outputs `outputs_group` need to be specified by the user.

The optimization attempts to find a `P` such that for all prescribed input values `X` and prescribed output values `Y`, the equality `model([P, X]) = Y` holds.

```
objects:
  loaded_model:
    constructor: $project_path$.model.model_base.ModelBase.load_model
    parameters:
      fn: '$results_dir$/training/model_best.bin'

  database:
    constructor: $DatabaseTabular$
    parameters:
      path_tgt: $DatabaseDirectory$

  optimizer_adam_inverse:
    constructor: torch.optim.Adam
    parameters:
      lr: 1e-3
      betas: (0.8, 0.99)
      eps: 1e-12
      weight_decay: 0
      amsgrad: False

  lambda_scheduler_function_inverse:
    constructor: $project_path$.training.schedulers.lambdas.cyclical_lr.CyclicalLR
    parameters:
      log_factor_low: -1
      log_factor_high: 1
      periods: [500, 1000, 2000]

  scheduler_LAMBDA_inverse:
    constructor: torch.optim.lr_scheduler.LambdaLR
    parameters:
      lr_lambda: lambda_scheduler_function_inverse

  loss_inverse:
    constructor: $project_path$.training.loss.loss_base.LossBase
    parameters:
      outputs:
        $output_key$:
          $output_key$:
            mae: 0
            mse: 1
            rmae: 0
            rmse: 0

actions:
  action_0:
    fname: $project_path$.utilization.parameter_estimation.gradient_sampler.GradientParameterSampler.estimate_parameters_tabular

    parameters:
```

```
        ncandidates: 50
        nepochs: 100
        dtype: $dtype$

        model_uid: loaded_model
        optimizer_uid: optimizer_adam_inverse
        scheduler_uid: scheduler_LAMBDA_inverse
        loss_uid: loss_inverse

        parameter_key: $input_key_unknown$ # assume the data is split so that one group contains the values to be optimized
        database_uid: database

        # some values may be known
        known_values:
            Column_000: [0.025, 0.025, 0.015]
            Column_001: [0.005, 0.015, 0.026]
            Column_004: [0.045, 0.039, 0.029]
            Column_006: [0.045, 0.039, 0.029]
            Column_008: [0.045, 0.039, 0.029]
            Column_010: [0.045, 0.039, 0.029]
        report_dir: '$results_dir$/reports'
```

12. Notes for Developers

12.1 Setup your local machine

We assume you have a local copy of the git repository obtained as described in the Installation guide

12.2 Git Guidelines

We will aim to follow the approach to git development described nicely in this blog post.

- Intialize the Git flow wrapper via the `git flow init` command.
- use `master` branch for production release.
- use `development` as a future release branch.
- use `feature/` as Feature branches prefix.
- use `release/` as Release branches prefix.
- use `hotfix/` as Hotfix branches prefix.
- use `support/` as Support branches prefix.
- use `bugfix/` as Bugfix branches prefix.
- use nothing for version prefix.

Afterwards, try to follow the suggestions below, in case of any issues or suggestions, contact Michal Kravčenko.

General Structure of the repo

- The production code is in the branch `master`, this branch contains the latest version of whatever code is available to the users.
- The development is contained in the branch `development`.

Feature Development

Feature branches are designed for development of new functionalities.

- When the developer wants to work on a feature titled `feature_x`, they should perform the following commands:
- `git flow feature start feature_x` (initializes the appropriate local branch and switches the developer to it).
- Then continue developing the feature, add changes via the `git add ...` command, commit the changes via the `git commit -m 'commit message'` command. The commit message should contain a brief description of the changes being commited and, if available, link to an issue in the repository via the `#N` command (where `N` is the number of the issue).
- Call `git flow feature publish` to push your feature to the remote repository.
- Once the feature is ready to be pushed to the development branch, call `git flow feature finish`. This switches the developer to the `development` branch and updates its content with that of the branch `feature/feature_x`. The branch `feature/feature_x` is then deleted.
- Call `git push` to push the updated development branch to the remote repository.
- When the developer wants to work on another feature `feature_y` developed by someone else, call the `git flow feature pull origin feature_y` command.

Hotfix Development

Hotfixes are urgent bug fixes, the bugs should be pushed to the production branch and development branches as soon as possible.

- When working on a hotfix with version `VERSION`, they should perform the following commands:
- `git flow hotfix start VERSION` (initializes the appropriate local branch asn switches the developer to it).
- Then continue developing the hotfix, add changes via the `git add ...` command, commit the changes via the `git commit -m 'commit message'` command.
- Once the hotfix is ready to be pushed to the development branch, call `git flow hotfix finish VERSION`. This switches the developer to the `development` branch and updates its content with that of the branch `hotfix/VERSION`. The hotfix is also merged to the `master` branch, which is also tagged with the version change caused by the hotfix. The branch `hotfix/VERSION` is then deleted.

Making a new Release

Release branch is designed for preparation of a new production release. Hotfixes are urgent bug fixes, the bugs should be pushed to the production branch and development branches as soon as possible.

- To create a new release branch for version `VERSION`, call the `git flow release start RELEASE` command.
- Allow other developers to push changes to the release branch via the `git flow release publish RELEASE` command.
- To finish the relase of a new version (big deal!), call the `git flow release finish RELEASE` command.
- Push the appropriate versioning via the `git push origin --tags` command.

12.3 Code Quality

Developers can call `./install/code_check.sh` script to automatically improve and analyze the quality of the produced code.

13. Module `entry_point`

13.1 **mod** AI4SurrogateModelling.entry_point

A main entry point of the program. Parses the configuration file and serves as a hub for all functionalities.

func main

```
main()
```

func main_debug

```
main_debug(config)
```

func main_function

```
main_function(config: dict)
```

Takes the provided configuration dictionary and performs the specified operations in order.

Parameters:

Name	Type	Description	Default
`config`	`dict`	a configuration dictionary	*required*

func main_info

```
main_info(config)
```

14. Module `src.aux`

14.1 **mod** AI4SurrogateModelling.src.aux

Module containing uncategorized auxilliary functions

class ndarray_comparator

```
ndarray_comparator(value: ndarray, eps: float)
```

A BASIC COMPARATOR FOR 1D NUMPY.NDARRAY OBJECTS A AND B.

1) len(A) < len(B) => A < B 2) A[j] == B[j] and A[i] < B[i] => A < B for j < i

Flattens the input array and calculates relevant metrics for future comparison.

Parameters:

Name	Type	Description	Default
`value`	`ndarray`	input numpy.ndarray to be compared in the future	*required*
`eps`	`float`	tolerance parameter, two numbers a and b are considered the same when |a - b| <= eps	*required*

attr eps `instance-attribute`

```
eps = eps
```

attr n `instance-attribute`

```
n = len(value)
```

attr value `instance-attribute`

```
value = flatten()
```

meth __eq__

```
__eq__(other: ndarray_comparator) -> bool
```

Implements the equal (==) operator.

Parameters:

Name	Type	Description	Default
`other`	`ndarray_comparator`	Value to compare against	*required*

Returns:

Name	Type	Description
`bool`	`bool`	True when self == other, otherwise False

meth __ge__

```
__ge__(other: ndarray_comparator) -> bool
```

Implements the greater than or equal (>=) operator

Parameters:

Name	Type	Description	Default
`other`	`ndarray_comparator`	Value to compare against	*required*

Returns:

Name	Type	Description
`bool`	`bool`	True when self >= other, otherwise False

`meth __gt__`

```
__gt__(other: ndarray_comparator) -> bool
```

Implements the greater than (>) operator

Parameters:

Name	Type	Description	Default
`other`	`ndarray_comparator`	Value to compare against	*required*

Returns:

Name	Type	Description
`bool`	`bool`	True when self > other, otherwise False

`meth __le__`

```
__le__(other: ndarray_comparator) -> bool
```

Implements the lower than or equal (<=) operator

Parameters:

Name	Type	Description	Default
`other`	`ndarray_comparator`	Value to compare against	*required*

Returns:

Name	Type	Description
`bool`	`bool`	True when self <= other, otherwise False

`meth __lt__`

```
__lt__(other: ndarray_comparator) -> bool
```

Implements the lower than (<) operator

Parameters:

Name	Type	Description	Default
`other`	`ndarray_comparator`	Value to compare against	*required*

Returns:

Name	Type	Description
`bool`	`bool`	True when self < other, otherwise False

meth __str__

```
__str__() -> str
```

Returns the string representation of this array.

Returns:

Name	Type	Description
`str`	`str`	Returns the native string representation of the flattened numpy array

func f_not_implemented

```
f_not_implemented(func) -> Callable
```

A decorator to be used on functions which are not implemented yet. Useful when the codebase is large and you are unsure what functions are called and where. Aborts the program on all MPI processes and produces a call trace for debugging.

Parameters:

Name	Type	Description	Default
`func`	`Callable`	A function to be decorated	*required*

Returns:

Name	Type	Description
`Callable`	`Callable`	the wrapped function

func f_should_remove

```
f_should_remove(func: Callable) -> Callable
```

A decorator to be used on functions which are planned to be removed or made obsolete.

Parameters:

Name	Type	Description	Default
`func`	`Callable`	A function to be decorated.	*required*

Returns:

Name	Type	Description
`Callable`	`Callable`	the wrapped function

func get_index_eq_min

```
get_index_eq_min(
    u: ndarray_comparator, V: ndarray, V_ordering: list[int]
) -> int
```

Provided the comparator u and the associated array, find and retrieve the highest row index i such that V[i] > u.

Parameters:

Name	Type	Description	Default
`u`	`ndarray_comparator`	referential row to compare against	*required*
`V`	`ndarray`	row space in which to look for the rows	*required*
`V_ordering`	`list[int]`	An ordering of rows in V such that V[V_ordering[i]] <= V[V_ordering[i + 1]]	*required*

Returns:

Name	Type	Description
`int`	`int`	the highest row index i such that V[i] > u

`func` get_maximal_matching

```
get_maximal_matching(
    graph_edges: set[tuple[int, int]],
) -> set[tuple[int, int]]
```

A greedy algorithm finding a maximal matching from the provided edges.

Parameters:

Name	Type	Description	Default
`graph_edges`	`set[tuple[int, int]]`	Each entry is a tuple (u, v), meaning that vertex u is connected to vertex v	*required*

Returns:

Type	Description
`set[tuple[int, int]]`	set[tuple[int, int]]: Subset of the input graph_edges such that adding any edge from: graph_edges - output will connect to an edge already in the output.

`func` get_similar_row_pairs

```
get_similar_row_pairs(
    U: ndarray,
    U_ordering: list[int],
    U_indices: list[int],
    V: ndarray,
    V_ordering: list[int],
    V_indices: list[int],
    eps: float,
) -> set[tuple[int, int]]
```

Takes two arrays U with shape (nU, k) and V with shape (nV, k) and finds rows of U which are also in V given the provided tolerance eps > 0. Outputs the global row indices of corresponding pairs.

Parameters:

Name	Type	Description	Default
`U`	`ndarray`	First array containing the first set of rows.	*required*
`V`	`ndarray`	Second array containing the second set of rows.	*required*
`U_ordering`	`list[int]`	An ordering of rows in U such that U[U_ordering[i]] <= U[U_ordering[i + 1]]	*required*
`U_indices`	`list[int]`	The array U is possibly distributed among multiple MPI processes. This list keeps track of the global row indices for array U.	*required*
`V_ordering`	`list[int]`	An ordering of rows in V such that V[V_ordering[i]] <= V[V_ordering[i + 1]]	*required*
`V_indices`	`list[int]`	The array V is possibly distributed among multiple MPI processes. This list keeps track of the global row indices for array V.	*required*
`eps`	`float`	tolerance parameter to be used in the comparator	*required*

Returns:

Name	Type	Description
`set`	`set[tuple[int, int]]`	Outputs a set of corresponding pairs, i.e. each (u, v) in the output means that U[u] == V[v]. Outputs an empty set when no matches are found.

`func` get_simplex_distance

```
get_simplex_distance(
    *,
    q: ndarray,
    points: ndarray,
    simplices: list[tuple[int, ...]]
) -> float
```

Calculates the shortest distance from point q to any point lying in the provided simplex set.

Parameters:

Name	Type	Description	Default
`q`	`ndarray`	Coordinates of the point to calculate for.	*required*
`points`	`ndarray`	Coordinates of the simplex vertices	*required*
`simplices`	`list[tuple[int, ...]]`	Simplex adjacency	*required*

Returns:

Name	Type	Description
`float`	`float`	min ||q - x|| over x, where x is any point in the provided simplex set.

`func` lies_near_simplex_set

```
lies_near_simplex_set(
    q: ndarray,
    points: ndarray,
    simplices: list[tuple[int, int]],
```

```
    tol: float,
) -> bool
```

Determines whether the provided point q lies near any of the provided simplices.

Parameters:

Name	Type	Description	Default
`q`	`ndarray`	The point for which to determine whether it lies near a set of simplices of not.	*required*
`points`	`ndarray`	Coordinates of the simplices to test against	*required*
`simplices`	`list[tuple[int, ...]]`	Vertex adjacency of the simplices.	*required*
`tol`	`float`	Measure of what 'near' and 'not near' mean.	*required*

Returns:

Name	Type	Description
`bool`	`bool`	True if q is at most tol distance (Euclidean) from any point in the provided simplices. False otherwise.

`func` permute_columns

```
permute_columns(
    data: list[ndarray],
    keys_current: list[str],
    keys_new: list[str],
) -> list[ndarray]
```

Given the list of numpy arrays data and its column labeling keys_new, find and apply a column permutation such that the data and its column labeling corresponds to the labeling keys_current.

Parameters:

Name	Type	Description	Default
`data`	`list[ndarray]`	List of arrays to permute	*required*
`keys_current`	`list[str]`	Target column ordering	*required*
`keys_new`	`list[str]`	Current column ordering	*required*

Returns:

Type	Description
`list[ndarray]`	list[np.ndarray]: a list of elemenets in data (in the same order) with permuted columns.

`func` transform_function

```
transform_function(func) -> Callable
```

A decorator to be used around functions producing data transformations. The decorator checks whether the input contains the required argument and whether the output contains the required arguments.

Checks for missing arguments. In case of a failure aborts the program on all MPI processes. Checks and informs the user if any error in input/output definition is present.

Parameters:

Name	Type	Description	Default
`func`	`Callable`	A function to be decorated	*required*

Returns:

Name	Type	Description
`Callable`	`Callable`	the wrapped function

15. Module `src.config`

15.1 `mod` AI4SurrogateModelling.src.config

Contains modules and functions related to parsing, processing and verification of user supplied configurations.

`class` Configuration

```
Configuration(config: Config)
```

Validates the user supplied configuration and performs basic pre-processing steps: - evaluates aliases such that no aliases contain variables - uses the evaluated aliases to replace all variables in the configuration tree.

Parameters:

Name	Type	Description	Default
`config`	`dict`	Contains the user supplied configuration.	*required*

`attr` alias_evaluation_map `instance-attribute`

```
alias_evaluation_map = _get_alias_evaluation_map(
    cfg["aliases"]
)
```

`attr` operations `instance-attribute`

```
operations = [{}] * len(cfg['operations'])
```

16. Module `src.history_progress`

16.1 **mod** AI4SurrogateModelling.src.history_progress

class HistoryProgress

```
HistoryProgress()
```

Keeps track of various metrics obtained during the training process.

attr measurements `instance-attribute`

```
measurements = {}
```

attr measurements_parsed `instance-attribute`

```
measurements_parsed = {}
```

meth add

```
add(key: str, value: Any, log: bool = False)
```

Adds a new measurement at the current internal index position.

Parameters:

Name	Type	Description	Default
`key`	`str`	Path to the nested dictionary.	*required*
`value`	`Any`	Value to be added	*required*

meth add_transpose_last

```
add_transpose_last(key: str, value: Any, log: bool = False)
```

Adds a new measurement at the current internal index position. Also exchanges the last entry in the key with the data key.

Parameters:

Name	Type	Description	Default
`key`	`str`	Path to the nested dictionary.	*required*
`value`	`Any`	Value to be added	*required*

meth get_last

```
get_last() -> dict
```

meth parse

```
parse()
```

Converts the raw lists of measurements as dictionaries into a nested dictionary of lists.

meth should_terminate

```
should_terminate(
    key: str, niterations: int = 10, treshold: float = 1e-06
) -> bool
```

17. Module `src.io`

17.1 `mod` AI4SurrogateModelling.src.io

Module conatining methods and classes utilized in efficient parallel access to a filesystem.

`class` ParallelFile

```
ParallelFile(fn: str, initial_file_size: int = 10 ** 6)
```

Wrapper for easy parallel file manipulation of an LMDB file.

Sets all internal state variables to default values. Initializes the database file. - filename to the database - initial size of the database file - load queue - save queue - remove queue - move queue

Parameters:

Name	Type	Description	Default
`fn`	`str`	Path to the directory storign the related database files.	*required*
`initial_file_size`	`int`	Initial size of the database file. Defaults to 10**6.	`10 ** 6`

`attr` fn `instance-attribute`

```
fn = fn
```

`attr` initial_size `instance-attribute`

```
initial_size = initial_file_size
```

`attr` queue_load `instance-attribute`

```
queue_load = {}
```

`attr` queue_move `instance-attribute`

```
queue_move = {}
```

`attr` queue_remove `instance-attribute`

```
queue_remove = {}
```

`attr` queue_save `instance-attribute`

```
queue_save = {}
```

`meth` append

```
append(*, data: dict[str], msg: str) -> None
```

Takes the supplied data and inserts them at the end of this database. Must be called by ALL mpi processes (even if some processes might have empty data).

Parameters:

Name	Type	Description	Default
`data`	`dict[str]`	A dictionary containing a prefix as a key and the associated array of objects. The controller will store the objects under the path starting with the prefix key.	*required*
`msg`	`str`	An informative message for the user's benefit.	*required*

`meth` get_default_metadata

```
get_default_metadata() -> dict
```

Constructs and returns the default metadata structure. Currently, it contains the number of stored files for each prefix in the database.

Returns:

Name	Type	Description
`dict`	`dict`	Metadata

`meth` load

```
load(
    *, keys: list[str], msg: str
) -> tuple[dict[str : (list[Any])], list[int]]
```

Loads all the files with the specified prefix. Each MPI process holds a portion of the files. Must be called by ALL MPI processes.

Parameters:

Name	Type	Description	Default
`keys`	`list[str]`	A list of prefixes for which to perform the loading operation	*required*
`msg`	`str`	An informative message for the user's benefit.	*required*

Returns:

Type	Description
`tuple[dict[str:(list[Any])], list[int]]`	tuple[dict[str: list[Any]], list[int]]: A list of loaded objects for each supplied prefix and the associated list of global file indices.

`meth` load_single_process

```
load_single_process(
    *, keys: list[str], msg: str, rank: int = 0
) -> tuple[dict[str : (list[Any])], list[int]]
```

Loads all the files with the specified prefix, can be called by any MPI processes.

Parameters:

Name	Type	Description	Default
`keys`	`list[str]`	A list of prefixes for which to perform the loading operation	*required*
`msg`	`str`	An informative message for the user's benefit.	*required*
`rank`	`int`	Rank of the process to load data for. Defaults to 0.	`0`

Returns:

Type	Description
`tuple[dict[str:(list[Any])], list[int]]`	tuple[dict[str: list[Any]], list[int]]: A list of loaded objects for each supplied prefix and the associated list of global file indices.

meth load_subset

```
load_subset(
    *, indices_global: list[int], keys: list[str], msg: str
) -> dict[str : (list[Any])]
```

For each of the supplied prefix, loads all files corresponding to the provided global indices.

Parameters:

Name	Type	Description	Default
`keys`	`list[str]`	A list of prefixes for which to perform the loading operation	*required*
`msg`	`str`	An informative message for the user's benefit.	*required*
`indices_global`	`list[int]`	A list of file indices to load.	*required*

Returns:

Type	Description
`dict[str:(list[Any])]`	dict[str: list[Any]]: A list of loaded objects for each supplied prefix.

meth remove

```
remove(
    *, keys: list[str], indices_global: list[int], msg: str
) -> None
```

Given the list of prefixes and global file indices for each prefix, removes the corresponding files. Automatically recalculates which files need to be moved so that the resulting files for each prefix use a continuous indexation. The number of file io operations is minimized.

Must be called by ALL mpi processes and assumes ONLY mpi process with rank 0 has the relevant data.

Parameters:

Name	Type	Description	Default
`keys`	`list[str]`	A list of prefixes for which to perform the removal operation	*required*
`indices_global`	`list[int]`	Indices of files to be removed. One list for all prefixes. All processes have the same list, no need for synchronization.	*required*
`msg`	`str`	An informative message for the user's benefit.	*required*

meth reset

```
reset() -> None
```

Sets the internal state to initial values. Empties all queues, sets its own metadata to default values and stores them in the database file.

meth update

```
update(
    *,
    indices_global: dict[list[int]],
    data: dict[str],
```

```
    msg: str
) -> None
```

Takes the supplied data and the corresponding global indices and updates the corresponding EXISTING content of this database. Must be called by ALL mpi processes (even if some processes might have empty data).Must be called by ALL mpi processes.

Parameters:

Name	Type	Description	Default
`indices_global`	`dict[list[int]]`	*description*	*required*
`data`	`dict[str]`	A dictionary containing a prefix as a key and the associated array of objects. The controller will store the objects under the path starting with the prefix key.	*required*
`msg`	`str`	An informative message for the user's benefit.	*required*

`func` get_first_active_element

```
get_first_active_element(
    arr: dict[str : (list[str])],
) -> dict[str:int]
```

Determines where to start the processing of the supplemented files. This is necessary, because the queue is processed in predetrmined chunks, the processing of which can fail and need to be restarted.

Parameters:

Name	Type	Description	Default
`arr`	`dict`	A dictionary of (key -> filenames) pairs.	*required*

Returns:

Name	Type	Description
`dict`	`dict[str:int]`	For each key in arr, contains the lowest index of the filename not yet processed

`func` lmdb_controller

```
lmdb_controller(func) -> Callable
```

A decorator function used to wrap parallel io methods. It splits the file operation into chunks of size 1000. When the database does not have enough storage, it increases it by a predefined factor and restarts the file processing from the last unfinished chunk.

e decorated function requires three arguments

lmdb_environment_fn (str): path to an LMDB file synchronous_access (bool): if true, processes do not access the files simultaneously, but go in a round-robin fashion. When one process finishes all its work, another one line continues with its work. synchronize (bool): if true, processes access the files simultaneously.

Parameters:

Name	Type	Description	Default
`func`	`Callable`	A function working with files in parallel.	*required*

Returns:

Name	Type	Description
`Callable`	`Callable`	the wrapped function

18. Module `src.logging`

18.1 `mod` AI4SurrogateModelling.src.logging

Module containing classes for logging functionality.

`class` Logger

A static class containing methods and decorators for comprehensive logging functionality.

`attr` level `class-attribute` `instance-attribute`

```
level = 0
```

`attr` logger_level `class-attribute` `instance-attribute`

```
logger_level = NONE
```

`attr` mpi_is_master `class-attribute` `instance-attribute`

```
mpi_is_master = True
```

`attr` mpi_rank `class-attribute` `instance-attribute`

```
mpi_rank = 0
```

`attr` mpi_size `class-attribute` `instance-attribute`

```
mpi_size = 1
```

`attr` target_dir `class-attribute` `instance-attribute`

```
target_dir = 'logs'
```

`meth` debug `staticmethod`

```
debug(msg: str = '') -> None
```

Outputs DEBUG level logging information to the MPI specific files.

Parameters:

Name	Type	Description	Default
`msg`	`str`	Message to be logged. Default ''.	`''`

`meth` get_result_log_string `staticmethod`

```
get_result_log_string(obj: Any) -> str
```

A recursive function representing any object as a string. Used for logging purpose where some object can be extremely large or contain many elements and implicit string conversion is infeasible.

Based on the type of the object, these conversion rules apply: str: returns the string as is, unless its longer than 30 characters, in which case it returns 'string, length[n]', where n = len(obj)

```
bytes: returns 'binary data, length[n]', where n = len(obj)

number: returns the string representation of the number.

iterable: other objects which can be iterated over, like lists,           dictionaries, tuples and sets, are parsed recursively.

numpy array: Numpy arrays result in 'numpy.ndarray(shape = (n)',           where n is the shape of obj.
```

Parameters:

Name	Type	Description	Default
`obj`	`Any`	Object to be represented as a string.	*required*

Returns:

Name	Type	Description
`str`	`str`	String representation of the object.

meth indent `staticmethod`

```
indent() -> str
```

Returns a whitespace string. The length depends on the depth of the logged function in the stack.

Returns:

Name	Type	Description
`str`	`str`	OF WHITE SPACES LINEARY PROPORTIONAL TO THE STACK DEPTH

meth info `staticmethod`

```
info(msg: str = '') -> None
```

Outputs INFO level logging information to the MPI specific files.

Parameters:

Name	Type	Description	Default
`msg`	`str`	Message to be logged. Default ''.	`''`

meth init `staticmethod`

```
init(
    logger_level: LoggerInfoLevel,
    mpi_rank: int,
    mpi_size: int,
    mpi_is_master: bool,
) -> None
```

Initializes the static logging environment. Each MPI process has its own logging file. Logging should not be conditioned based on the MPI process rank, this may result in program hanging.

Parameters:

Name	Type	Description	Default
`logger_level`	`LoggerInfoLevel`	The amount of logged information.	*required*
`mpi_rank`	`int`	The global MPI rank.	*required*
`mpi_size`	`int`	The total number of MPI processes	*required*
`mpi_is_master`	`bool`	Is this MPI process a master?	*required*

meth is_iterable `staticmethod`

```
is_iterable(obj: Any) -> bool
```

Determines if the object can be iterated over.

Parameters:

Name	Type	Description	Default
`obj`	`Any`	Object to be analyzed.	*required*

Returns:

Name	Type	Description
`bool`	`bool`	True if it can be iterated over, False otherwise.

meth logged `staticmethod`

```
logged(commentary='', show_debug=False) -> Any
```

Decorator to log function calls, arguments, return values, and optional commentary.

Parameters:

Name	Type	Description	Default
`commentary`	`str`	Supplementary debug information. Defaults to ''.	`''`
`show_debug`	`bool`	Optional flag. When the logging level includes DEBUG information, this flag can enable/disable it for select functions. Defaults to False.	`False`

Returns:

Name	Type	Description
`Any`	`Any`	decorator

meth warning `staticmethod`

```
warning(msg: str = '') -> None
```

Outputs WARNING level logging information to the MPI specific files.

Parameters:

Name	Type	Description	Default
`msg`	`str`	Message to be logged. Default ''.	`''`

`class` LoggerInfoLevel

Bases: `Enum`

Enum class determining the level of logging to be used in the program

attr DEBUG `class-attribute` `instance-attribute`

```
DEBUG = 2
```

attr INFO `class-attribute` `instance-attribute`

```
INFO = 1
```

attr NONE `class-attribute` `instance-attribute`

```
NONE = 0
```

19. Module `src.memory`

19.1 **mod** AI4SurrogateModelling.src.memory

Module containing classes used to monitor various metrics related to the usage of memory.

class MEM

A static class for simple memory information monitoring.

meth get_memory_usage_MB `staticmethod`

```
get_memory_usage_MB() -> float
```

Retrieves the current RAM usage

Returns:

Name	Type	Description
`float`	`float`	returns the memory usage in MBs.

20. Module `src.mpi`

20.1 **mod** AI4SurrogateModelling.src.mpi

Module containing classes, functions and decorators related to the management and utilization of MPI resources.

class mpi

A static class containing various quality of life improvements w.r.t. MPI utilization.

class OP staticmethod

Bases: `Enum`

An internal enumerator class for reduction operation specification.

attr MAX class-attribute instance-attribute

```
MAX = MAX
```

attr MIN class-attribute instance-attribute

```
MIN = MIN
```

attr SUM class-attribute instance-attribute

```
SUM = SUM
```

meth abort staticmethod

```
abort(abort_code: int, abort_message: str) -> None
```

Terminates all MPI processes with the user provided abortion code and message..

Parameters:

Name	Type	Description	Default
`abort_code`	`int`	Code for the reason for the abortion. Defaults to 1.	*required*
`abort_message`	`str`	Message to display to the user. Defaults to "".	*required*

meth allgather staticmethod

```
allgather(data: Any) -> list[Any]
```

Gathers data from all MPI processes and creates a list out of them.

Parameters:

Name	Type	Description	Default
`data`	`Any`	Single value	*required*

Returns:

Type	Description
`list[Any]`	list[Any]: A list of values, where the i-th value corresponds to the value of data on the i-th MPI process.

meth allreduce staticmethod

```
allreduce(data: Any, op: OP) -> Any
```

Performs the reduction operation and stores the result on ALL the MPI processes. Works differently based on the data type:

- numpy arrays: returns a numpy array of the same shape as the input array
- torch tensors: returns a torch tensors of the same shape, dtype and on the same device as the input array
- other types with length: returns the same type with the same length, each entry is the product of the reduction operation specified with other entries at the same index.
- other: standard reduction implementation.

Parameters:

Name	Type	Description	Default
`data`	`Any`	Data to be reduced.	*required*
`op`	`OP`	Enumerator specifying the reduction operation.	*required*

Returns:

Name	Type	Description
`Any`	`Any`	Data of the same type and the same shape, where each entry is the product of the reduction operation specified.

meth assert_equality `staticmethod`

```
assert_equality(value: Any) -> None
```

Gathers values from all MPI processes to the master process and checks if all values are equal.

Calls MPI.Abort when some values are different.

Parameters:

Name	Type	Description	Default
`value`	`Any`	A value to be compared.	*required*

meth broadcast `staticmethod`

```
broadcast(data: Any, root: int = 0) -> Any
```

A simple broadcast wrapper using the underlying pickled broadcast version.

Parameters:

Name	Type	Description	Default
`data`	`Any`	Single value or a list of values to be broadcast from root to the rest of the MPI group	*required*
`root`	`int`	Index of the root process. Defaults to 0.	`0`

Returns:

Name	Type	Description
`Any`	`Any`	Returns the data on all MPI processes.

meth cleanup `staticmethod`

```
cleanup() -> None
```

Cleans up the MPI distributed environment.

meth gather staticmethod

```
gather(data: Any, root: int = 0) -> list[Any]
```

Gathers data from all MPI processes and creates a vector out of them.

TODO: seems complicated, maybe the underlying pickled implementation can be utilized.

Parameters:

Name	Type	Description	Default
`data`	`Any`	Single value	*required*
`root`	`int`	The rank of the process on which to gather the list of values.	`0`

Returns:

Type	Description
`list[Any]`	list[Any]: A list of values, where the i-th value corresponds to the value of data on the i-th MPI process.

meth get_comm staticmethod

```
get_comm() -> Intracomm
```

Retrieves the global intranode communicator.

Returns:

Type	Description
`Intracomm`	MPI.Intracomm: MPI communicator.

meth get_rank staticmethod

```
get_rank() -> int
```

Retrieves the global rank of this MPI process.

Returns:

Name	Type	Description
`int`	`int`	the global MPI rank.

meth get_task_timeline staticmethod

```
get_task_timeline(block_matrix: ndarray) -> Any
```

An experimental method designed to compute a communication scheme for a given computational block matrix.

Assume we are tasked to perform some work on data arranged into a 2D grid. This grid can be segmented into PxP block matrix M (computational block matrix). Let k = M[i, j], i.e. the work on the grid in the block with row index i and column index j is supposed to be calculated by MPI process k.

Furthermore, assume that for performing the calculations required by M[i, j], we need data from processes i and j. The goal of this method is to calculate a consistent sequence of computational and communication tasks so that the work is computed as prescribed with the least amount of waiting on communication.

Parameters:

Name	Type	Description	Default
`block_matrix`	`ndarray`	The computational block matrix with P rows and P columns with integer values.	*required*

Returns:

Name	Type	Description
`Any`	`Any`	*description*

meth get_world_size `staticmethod`

```
get_world_size() -> int
```

Retrieves the total number of MPI processes in this communicator context.

Returns:

Name	Type	Description
`int`	`int`	The number of MPI processes.

meth init `staticmethod`

```
init() -> None
```

Initializes the MPI environment and associated internal parameters.

meth is_master `staticmethod`

```
is_master() -> bool
```

Returns True when this process is the master process in this communicator context.

otherwise returns False.

Returns:

Name	Type	Description
`bool`	`bool`	True when master, False otherwise.

meth iterator `staticmethod`

```
iterator(obj: Iterable) -> Iterable
```

Returns an iterator over the iterable object such that all MPI processes retrieve a subset of the object's entries.

Assumes ALL MPI processes call this method on the same object, containing ALL the data.

Parameters:

Name	Type	Description	Default
`obj`	`Iterable`	An iterable object.	*required*

Returns:

Name	Type	Description
`Iterable`	`Iterable`	a generator used to iterate over the object in a distributed manner.

meth make_dir `staticmethod`

```
make_dir(tgt_dir)
```

meth print `staticmethod`

```
print(*args, end='\n')
```

Prints information only on master rank.

meth reduce `staticmethod`

```
reduce(data: Any, op: OP, root: int = 0) -> Any
```

Performs the reduction operation and stores the result on the specified process. Works differently based on the data type:

- numpy arrays: returns a numpy array of the same shape as the input array
- torch tensors: returns a torch tensors of the same shape, dtype and on the same device as the input array
- other types with length: returns the same type with the same length, each entry is the product of the reduction operation specified with other entries at the same index.
- other: standard reduction implementation.

Parameters:

Name	Type	Description	Default
`data`	`Any`	Data to be reduced.	*required*
`op`	`OP`	Enumerator specifying the reduction operation.	*required*
`root`	`int`	Target process rank. Defaults to 0.	`0`

Returns:

Name	Type	Description
`Any`	`Any`	Data of the same type and the same shape, where each entry is the product of the reduction operation specified.

meth scatter_numpy `staticmethod`

```
scatter_numpy(
    data: list[ndarray], root: int = 0
) -> ndarray
```

Scatters a list of numpy arrays from the root MPI process on all MPI processes in the context communicator.

Parameters:

Name	Type	Description	Default
`data`	`list[ndarray]`	List of numpy arrays to be scattered.	*required*
`root`	`int`	Rank of the root process, i.e. the process that holds the data to be scattered around. Defaults to 0.	`0`

Returns:

Type	Description
`ndarray`	np.ndarray: numpy array from the input list.

meth send_recv `staticmethod`

```
send_recv(
    data: Any, target_process: int, source_process: int
) -> Any
```

A 2-in-1 send/recieve wrapper using the underlying pickled send/recv version. Requires that two processes with ranks i, j call this function with the same rank arguments, i.e.: process i calls 'data_j = send_recv(..., i, j)', process j calls 'data_j = send_recv(data_j, i, j)', which sends the data_j from process j to process i

Parameters:

Name	Type	Description	Default
`data`	`Any`	Any data.	*required*
`target_process`	`int`	If my rank is the target_process, I recieve data from the source_process.	*required*
`source_process`	`int`	If my rank is the source_process, I send data to target_process.	*required*

Returns:

Name	Type	Description
`Any`	`Any`	data.

meth stdout_write `staticmethod`

```
stdout_write(msg: str)
```

meth sync `staticmethod`

```
sync() -> None
```

Compares the stack history on all MPI processes to ensure all processes call this function from the same spot. Calls MPI_Barrier.

meth tqdm `staticmethod`

```
tqdm(obj: Any, desc: str) -> Any
```

A TQDM progress bar wrapper for convenience purposes. Wraps the object in a tqdm wrapper when called from master.

Parameters:

Name	Type	Description	Default
`obj`	`Any`	Any object which can be iterated over.	*required*
`desc`	`str`	Message to be displayed in the progressbar.	*required*

Returns:

Name	Type	Description
`Any`	`Any`	TQDM progress bar if called from master, the original object otherwise.

`func` main_loop

```
main_loop(logger_level: LoggerInfoLevel) -> Any
```

A decorator for the main program function. It initializes the logging environment, the MPI environment and automatically cleans up the MPI environment upon termination.

Parameters:

Name	Type	Description	Default
`logger_level`	`LoggerInfoLevel`	An enumerator value related to the information level to be logged during the program execution.	*required*

Returns:

Name	Type	Description
`Any`	`Any`	decorator

21. Module `src.terminal_dashboard`

21.1 **mod** AI4SurrogateModelling.src.terminal_dashboard

Contains modules and functions related to the concurrent information display in the terminal.

class TrainingDashboard

```
TrainingDashboard()
```

Represents a terminal dashboard displaying progress of the training process.

attr **delim** `instance-attribute`

```
delim = '-' * header_length
```

attr **gap** `instance-attribute`

```
gap = 13
```

attr **header** `instance-attribute`

```
header = (
    " " * (gap + key_length)
    + "TRAINING"
    + " " * gap
    + "TESTING"
    + " " * gap
    + "VALIDATION"
    + " " * gap
)
```

attr **header_length** `instance-attribute`

```
header_length = len(header)
```

attr **isCursor_saved** `class-attribute` `instance-attribute`

```
isCursor_saved = False
```

attr **key_length** `instance-attribute`

```
key_length = 55
```

meth **refresh**

```
refresh(
    *,
    progress: float,
    time_per_epoch: float,
    learning_rate: float,
    time_remaining: float,
    criterion_train: float,
    criterion_test: float = None,
    criterion_validation: float = None,
    loss_train: float,
    loss_test: float = None,
    loss_validation: float = None,
    losses_train: dict,
    losses_test: dict = None,
    losses_validation: dict = None
)
```

Refreshes the dashboard information display containing the provided information.

Parameters:

Name	Type	Description	Default
`progress`	`float`	Relative progress of the training process.	*required*
`time_per_epoch`	`float`	Estimated time per one training epoch.	*required*
`learning_rate`	`float`	Last learning rate used by the optimizer.	*required*
`time_remaining`	`float`	Estimated remaining time until training finishes.	*required*
`criterion_train`	`float`	Optimality criterion w.r.t. to the training data.	*required*
`criterion_test`	`float`	Optimality criterion w.r.t. to the testing data.	`None`
`criterion_validation`	`float`	Optimality criterion w.r.t. to the validation data.	`None`
`loss_train`	`float`	Loss calculated on the training data.	*required*
`loss_test`	`float`	Loss calculated on the testing data.	`None`
`loss_validation`	`float`	Loss calculated on the validation data.	`None`
`losses_train`	`dict`	Dictionary containing individual training loss components.	*required*
`losses_test`	`dict`	Dictionary containing individual testing loss components.	`None`
`losses_validation`	`dict`	Dictionary containing individual validation loss components.	`None`

`meth` release_cursor

```
release_cursor()
```

Allows the dashboard to be placed somewhere else.

`meth` set_cursor

```
set_cursor()
```

Sets the cursor for information display at the current cursor position.

`func` flatten_dict

```
flatten_dict(d, parent_key='', sep='-')
```

`func` parse_dictionary_data

```
parse_dictionary_data(data: dict) -> dict
```

Takes the input nested dictionary and non-nested dictionary containing nested paths as keys with corresponding values.

Parameters:

Name	Type	Description	Default
`data`	`dict`	A nested dictionary to be parsed.	*required*

Returns:

Name	Type	Description
`dict`	`dict`	A non-nested dictionary containing the same data as the input.

22. Module `src.time_monitor`

22.1 `mod` AI4SurrogateModelling.src.time_monitor

Module containing classes for simple time measurement and monitoring.

`class` TimeMonitor

A static class containing methods for simple time measurements.

`meth` get `staticmethod`

```
get(key: str) -> dict
```

Retrieves the elapsed time in seconds for the CPU and WALLTIME clocks with the specified name.

Parameters:

Name	Type	Description	Default
`key`	`str`	Name of the clocks.	*required*

Returns:

Name	Type	Description
`dict`	`dict`	CPU and WALLTIME elapsed times since the clocks with the specified name were restarted.

`meth` increment `staticmethod`

```
increment(key: str, value: float) -> None
```

Increments the internal clocks by the specified amount for the given key.

Parameters:

Name	Type	Description	Default
`key`	`str`	Name of the clock to be restarted.	*required*
`value`	`float`	Amount of time to add to the internal counters.	*required*

`meth` start `staticmethod`

```
start(key: str) -> None
```

Restarts the CPU and WALLTIME clocks with the supplied name.

Parameters:

Name	Type	Description	Default
`key`	`str`	Name of the clock to be restarted.	*required*

23. Module `src.conf_rules.main_config`

23.1 **mod** AI4SurrogateModelling.src.conf_rules.main_config

A module containing the configuration specification of the main configuration script.

class Config

Bases: `BaseModel`

The main configuration specification. Expects the main configuration file to contain the fields 'main', 'aliases' and 'operations'.

attr aliases `instance-attribute`

```
aliases: dict[str, Any]
```

attr main `instance-attribute`

```
main: MainConfig
```

attr operations `instance-attribute`

```
operations: dict[str, str]
```

meth check_operation_keys

```
check_operation_keys(v)
```

class MainConfig

Bases: `BaseModel`

Configuration specifying the overall settings of the program. It constrols the level of logging, where the logs should be stored and what type of parallelization should be used.

attr logging `class-attribute` `instance-attribute`

```
logging: Literal["info", "debug"] = Field(
    "info", description="Logging level (info/debug)"
)
```

attr logging_directory `class-attribute` `instance-attribute`

```
logging_directory: str = Field(
    "logs", description="Directory for logging output"
)
```

attr parallelization `class-attribute` `instance-attribute`

```
parallelization: Literal["distributed", "isolated"] = Field(
    "distributed", description="Parallelization mode"
)
```

attr seed `class-attribute` `instance-attribute`

```
seed: int = Field(
    42, description="Seed for reproducibility"
)
```

class OperationsHub

Bases: `RootModel[dict[str, str]]`

Specifies how the sequence of operations should be performed.

meth validate_keys `classmethod`

```
validate_keys(value: dict)
```

Goes through the operation keys and checks if they have the correct naming convention and correct ordering.

Parameters:

Name	Type	Description	Default
`value`	`dict`	A dictionary containing the operation keys and files to load the specific operation configurations.	*required*

Raises:

Type	Description
`ValueError`	Throws this error when the operation key has incorrect naming.

24. Module `src.conf_rules.operation_config`

24.1 **mod** AI4SurrogateModelling.src.conf_rules.operation_config

A module containing operation configuration file validation. Each operation comprises of a set of objects to be initialized and a set of actions to be called on some existing objects.

class ActionDefinition

Bases: `BaseModel`

Defines the configuration validator for the action sequence to be perfomed in the configuration file.

attr fname `class-attribute` `instance-attribute`

```
fname: str = Field(
    description="A class member function to be called."
)
```

attr object_uid `class-attribute` `instance-attribute`

```
object_uid: str = Field(
    None,
    description="A unique ID of the object to be manipulated with.",
)
```

attr parameters `class-attribute` `instance-attribute`

```
parameters: Dict[str, Any] = Field(
    {},
    description="Parameters to be passed into the function.",
)
```

meth validate_actions `classmethod`

```
validate_actions(*, actions: dict, operation_tasks: dict)
```

Validates the series of actions in the configuration file.

Parameters:

Name	Type	Description	Default
`actions`	`dict`	A dictionary containing action keys, objects and object functions with their parameters to be called in each action.	*required*
`operation_tasks`	`dict`	To be filled with action indices.	*required*

Raises:

Type	Description
`ValueError`	This is raised when the actions do not satisfy the naming convention 'action_N' with N being an integer.
`AttributeError`	This is raised when the specified function does not exist as a member function of the object.

meth validate_object_references `classmethod`

```
validate_object_references(*, actions: dict, objects: dict)
```

Checks if the actions reference objects defined in the configuration hierarchy.

Parameters:

Name	Type	Description	Default
`actions`	`dict`	Action sequence to be validated.	*required*
`objects`	`dict`	Objects contained in the same configuration file.	*required*

Raises:

Type	Description
`ValueError`	This is raised when the object associated with some action is not specified in the same configuration file or in previously parsed configuration files.

`class` ObjectDefinition

Bases: `BaseModel`

Contains specification for object definition configuration files.

`attr` constructor `class-attribute` `instance-attribute`

```
constructor: str = Field(
    description="Name of the object class or a function returning an object           instance specified as a path in the current codebase."
)
```

`attr` parameters `class-attribute` `instance-attribute`

```
parameters: Dict[str, Any] = Field(
    {},
    description="A dictionary of parameters supplied to the constructor           of the specified object class.",
)
```

`meth` validate_objects `classmethod`

```
validate_objects(*, objects: dict, operation_tasks: dict)
```

Checks the validity of the objects defined in the configuration file.

Parameters:

Name	Type	Description	Default
`objects`	`dict`	Contains unique identifiers, class names and parameters for each object to be initialized.	*required*
`operation_tasks`	`dict`	Is filled with object configurations.	*required*

Raises:

Type	Description
`ValueError`	This is raised when multiple objects with the same identifier are being specified anywhere in the configuration hierarchy. It is also raised when an object has an identifier in the form 'action_N', wehere N is a nonnegative integer.
`AttributeError`	This is raised when the object class does not exist.

`class` OperationConfig

Bases: `BaseModel`

The main operation configuration validator. Expects the configuration file to contain the 'objects' and 'actions' fields.

attr actions `class-attribute` `instance-attribute`

```
actions: Dict[str, ActionDefinition] = Field(
    {},
    description="A set of actions to be perfomed in this operation.",
)
```

attr objects `class-attribute` `instance-attribute`

```
objects: Dict[str, ObjectDefinition] = Field(
    {}, description="A set of objects to be constructed."
)
```

meth validate_operation `classmethod`

```
validate_operation(field_value)
```

25. Module `src.dataloaders.dataloader_dictionary`

25.1 **mod** AI4SurrogateModelling.src.dataloaders.dataloader_dictionary

Contains modules and functions related to the creation of dictionary based pytorch dataloaders.

class DictionaryDataLoader

```
DictionaryDataLoader(
    *,
    dataset: DictionaryDataset,
    batch_size: int,
    shuffle: bool = False,
    device=None,
    generator=None
)
```

Parameters:

Name	Type	Description	Default
`dataset`	`DictDataset`	dataset instance	*required*
`batch_size`	`int`	number of samples per batch	*required*
`shuffle`	`bool`	whether to shuffle dataset each epoch	`False`

attr batch_size `instance-attribute`

```
batch_size = batch_size // get_world_size()
```

attr cursor `instance-attribute`

```
cursor = 0
```

attr dataset `instance-attribute`

```
dataset = dataset
```

attr device `instance-attribute`

```
device = device
```

attr dtype `instance-attribute`

```
dtype = dtype
```

attr generator `instance-attribute`

```
generator = generator
```

attr indices `instance-attribute`

```
indices = list(range(len(dataset)))
```

attr shuffle `instance-attribute`

```
shuffle = shuffle
```

meth __iter__

```
__iter__()
```

meth __len__

```
__len__()
```

meth __next__

```
__next__()
```

26. Module `src.dataloaders.dataset_dictionary`

26.1 `mod` AI4SurrogateModelling.src.dataloaders.dataset_dictionary

Contains modules and functions related to the creation of dictionary based pytorch datasets.

`class` DictionaryDataset

```
DictionaryDataset(*, data: dict)
```

Bases: `Dataset`

Parameters:

Name	Type	Description	Default
`data`	`dict`	each key points to a list of values.	*required*

`attr` data `instance-attribute`

```
data = data
```

`attr` dtype `instance-attribute`

```
dtype = dtype
```

`attr` keys `instance-attribute`

```
keys = list(keys())
```

`attr` total_size `instance-attribute`

```
total_size = allreduce(len(self), op=SUM)
```

`meth` __getitem__

```
__getitem__(idx: int) -> dict
```

Retrieves a dictionary with internal keys, each pointing to a single entry in this dataset.

Parameters:

Name	Type	Description	Default
`idx`	`int`	Index of the entry in this dataset.	*required*

Returns:

Name	Type	Description
`dict`	`dict`	Dictionary containing the entries.

`meth` __len__

```
__len__() -> int
```

Returns the number of entries in this dataset.

Returns:

Name	Type	Description
`int`	`int`	Number of entries.

`meth` get_total_size

```
get_total_size() -> int
```

27. Module `src.enums.enums_dtypes`

27.1 **mod** AI4SurrogateModelling.src.enums.enums_dtypes

A module containing datatypes enums.

class ENUM_Dtypes

Bases: `Enum`

attr float16 `class-attribute` `instance-attribute`

```
float16 = float16
```

attr float32 `class-attribute` `instance-attribute`

```
float32 = float32
```

attr float64 `class-attribute` `instance-attribute`

```
float64 = float64
```

28. Module `src.enums.enums_losses`

28.1 **mod** AI4SurrogateModelling.src.enums.enums_losses

Contains enumerators for various Loss functions.

class ENUM_Losses

Bases: `Enum`

attr L1 `class-attribute` `instance-attribute`

```
L1 = {'class': Loss, 'metric': L1}
```

attr L1_RELATIVE `class-attribute` `instance-attribute`

```
L1_RELATIVE = {'class': Loss, 'metric': L1_RELATIVE}
```

attr L2 `class-attribute` `instance-attribute`

```
L2 = {'class': Loss, 'metric': L2}
```

attr L2_RELATIVE `class-attribute` `instance-attribute`

```
L2_RELATIVE = {'class': Loss, 'metric': L2_RELATIVE}
```

29. Module `src.enums.enums_metrics`

29.1 **mod** AI4SurrogateModelling.src.enums.enums_metrics

Contains enumerators for various Metrics measuring the state of the models.

class ENUM_Metrics

Bases: `Enum`

attr Histogram `class-attribute` `instance-attribute`

```
Histogram = {'class': Histogram}
```

attr L1 `class-attribute` `instance-attribute`

```
L1 = {'class': L1}
```

attr L1_RELATIVE `class-attribute` `instance-attribute`

```
L1_RELATIVE = {'class': L1_RELATIVE}
```

attr L2 `class-attribute` `instance-attribute`

```
L2 = {'class': L2}
```

attr L2_RELATIVE `class-attribute` `instance-attribute`

```
L2_RELATIVE = {'class': L2_RELATIVE}
```

attr Store `class-attribute` `instance-attribute`

```
Store = {'class': Store}
```

30. Module `src.enums.enums_models`

30.1 **mod** AI4SurrogateModelling.src.enums.enums_models

Contains various enumerators specifying Activation functions, models and layers

class ENUM_Activations

Bases: `Enum`

For a single vertex.

attr COSINE `class-attribute` `instance-attribute`

```
COSINE = {
    "short_string": "activation_cosine",
    "description": "Cosine activation function",
    "function": cos,
}
```

attr NONE `class-attribute` `instance-attribute`

```
NONE = {
    "short_string": "none",
    "description": "No activation",
    "function": Identity(),
}
```

attr RELU `class-attribute` `instance-attribute`

```
RELU = {
    "short_string": "activation_relu",
    "description": "ReLU activation function",
    "function": ReLU(),
}
```

attr SIGMOID `class-attribute` `instance-attribute`

```
SIGMOID = {
    "short_string": "activation_sigmoid",
    "description": "Sigmoidal activation function",
    "function": Sigmoid(),
}
```

attr SINE `class-attribute` `instance-attribute`

```
SINE = {
    "short_string": "activation_sine",
    "description": "Sine activation function",
    "function": sin,
}
```

class ENUM_Losses

Bases: `Enum`

attr L1 `class-attribute` `instance-attribute`

```
L1 = 1
```

attr L2 `class-attribute` `instance-attribute`

```
L2 = 2
```

attr MSE `class-attribute` `instance-attribute`

```
MSE = 0
```

class ENUM_Metrics

Bases: Enum

attr MSE class-attribute instance-attribute

```
MSE = 0
```

attr MSE_ENTRY_WISE class-attribute instance-attribute

```
MSE_ENTRY_WISE = 1
```

class ENUM_Schedulers

Bases: Enum

Schedulers and their configurations

attr NONE class-attribute instance-attribute

```
NONE = 0
```

class ENUM_TransferLayers

Bases: Enum

Betwen a pair of vertices.

attr LINEAR class-attribute instance-attribute

```
LINEAR = {
    "short_string": "layer_linear",
    "description": "Linear Layer",
    "parameters": [],
}
```

attr RESIDUAL class-attribute instance-attribute

```
RESIDUAL = {
    "short_string": "layer_residual",
    "description": "Residual Layer",
    "parameters": [],
}
```

class ENUM_VertexProcessingLayers

Bases: Enum

Transforms the values of a vertex in some way.

attr FOURIER_FEATURE class-attribute instance-attribute

```
FOURIER_FEATURE = {
    "short_string": "layer_fourier_features",
    "description": "Fourier Feature Layer",
    "parameters": ["dim_inputs"],
}
```

attr NOISE_GAUSSIAN class-attribute instance-attribute

```
NOISE_GAUSSIAN = {
    "short_string": "layer_noise_gaussian",
    "description": "Gaussian noise Layer",
    "parameters": ["stds", "musapplication_intervals"],
}
```

31. Module `src.enums.enums_objectives`

31.1 **mod** AI4SurrogateModelling.src.enums.enums_objectives

Contains enumerators related to the training objective functions.

class ENUM_Objectives

Bases: `Enum`

Contains the following enumerators: - PREDICTION: used for the model outputs. - INPUT_GRADIENT: used for stats related to the model inputs. - MODEL: used for stats related to the model parameters (weights and biases). - MODEL_WEIGHTS: used for stats related solely to model weights. - MODEL_BIASES: used for stats related solely to model biases.

attr INPUT_GRADIENT `class-attribute` `instance-attribute`

```
INPUT_GRADIENT = {
    "id": 1,
    "data_stat": True,
    "model_stat": False,
}
```

attr MODEL `class-attribute` `instance-attribute`

```
MODEL = {'id': 2, 'data_stat': False, 'model_stat': True}
```

attr MODEL_BIASES `class-attribute` `instance-attribute`

```
MODEL_BIASES = {
    "id": 4,
    "data_stat": False,
    "model_stat": True,
}
```

attr MODEL_WEIGHTS `class-attribute` `instance-attribute`

```
MODEL_WEIGHTS = {
    "id": 3,
    "data_stat": False,
    "model_stat": True,
}
```

attr PREDICTION `class-attribute` `instance-attribute`

```
PREDICTION = {
    "id": 0,
    "data_stat": True,
    "model_stat": False,
}
```

32. Module `src.enums.enums_optimizers`

32.1 **mod** AI4SurrogateModelling.src.enums.enums_optimizers

Contains enumerators specifying the optimizers to be used.

class ENUM_Optimizers

Bases: `Enum`

attr ADAM `class-attribute` `instance-attribute`

```
ADAM = {
    "name": "Adaptive Moment Estimation (ADAM)",
    "class": Adam,
}
```

attr CUSTOM_ADAM `class-attribute` `instance-attribute`

```
CUSTOM_ADAM = {
    "name": "Custom Adaptive Moment Estimation (ADAM)",
    "class": Adam,
}
```

attr LBFGS `class-attribute` `instance-attribute`

```
LBFGS = {
    "name": "Limited-memory Broyden-Fletcher-Goldfarb-Shanno (LBFGS)",
    "class": LBFGS,
}
```

attr SGD `class-attribute` `instance-attribute`

```
SGD = {
    "name": "Stochastic Gradient Descent (SGD)",
    "class": SGD,
}
```

33. Module `src.enums.enums_schedulers`

33.1 **mod** AI4SurrogateModelling.src.enums.enums_schedulers

Contains enumerators specifying the schedulers to be used.

class ENUM_Schedulers

Bases: `Enum`

attr CONSTANT `class-attribute` `instance-attribute`

```
CONSTANT = {
    "name": "Constant Scheduler",
    "class": ConstantLR,
}
```

attr COSINE_ANNEALING `class-attribute` `instance-attribute`

```
COSINE_ANNEALING = {
    "name": "Cosine Annealing Scheduler",
    "class": CosineAnnealingLR,
}
```

attr COSINE_ANNEALING_WARM_RESTARTS `class-attribute` `instance-attribute`

```
COSINE_ANNEALING_WARM_RESTARTS = {
    "name": "Cosine Annealing with Warm Restarts",
    "class": CosineAnnealingWarmRestarts,
}
```

attr CYCLIC `class-attribute` `instance-attribute`

```
CYCLIC = {'name': 'Cyclic Scheduler', 'class': CyclicLR}
```

attr EXPONENTIAL `class-attribute` `instance-attribute`

```
EXPONENTIAL = {
    "name": "Exponential Scheduler",
    "class": ExponentialLR,
}
```

attr LAMBDA `class-attribute` `instance-attribute`

```
LAMBDA = {'name': 'Lambda Scheduler', 'class': LambdaLR}
```

attr LINEAR `class-attribute` `instance-attribute`

```
LINEAR = {'name': 'Linear Scheduler', 'class': LinearLR}
```

attr MULTIPLICATIVE `class-attribute` `instance-attribute`

```
MULTIPLICATIVE = {
    "name": "Multiplicative Scheduler",
    "class": MultiplicativeLR,
}
```

attr ONE_CYCLE `class-attribute` `instance-attribute`

```
ONE_CYCLE = {
    "name": "One Cycle Scheduler",
    "class": OneCycleLR,
}
```

attr POLYNOMIAL `class-attribute` `instance-attribute`

```
POLYNOMIAL = {
    "name": "Polynomial Scheduler",
    "class": PolynomialLR,
}
```

attr REDUCE_ON_PLATEAU `class-attribute` `instance-attribute`

```
REDUCE_ON_PLATEAU = {
    "name": "Reduce on Plateau Scheduler",
    "class": ReduceLROnPlateau,
}
```

attr SEQUENTIAL `class-attribute` `instance-attribute`

```
SEQUENTIAL = {
    "name": "Sequential Scheduler",
    "class": SequentialLR,
}
```

attr STEP `class-attribute` `instance-attribute`

```
STEP = {'name': 'Step Scheduler', 'class': StepLR}
```

34. Module `src.enums.enums_tabular_stats`

34.1 **mod** AI4SurrogateModelling.src.enums.enums_tabular_stats

Contains enumerators and function hub related to the calculation of statistics related to tabular databases.

class ENUM_StatisticsCode_Tabular

Bases: `Enum`

Contains the enumerators specifying which statistic and how should it be calculated.

Each enumerator has a value with 5 entries

- value[0]: string code of the statistic, used for io and dependency calculation.
- value[1]: determines the type of the statistic, used for io.
- value[2]: a string description of the statistic, used for logging.
- value[3]: a function to calculate the statistic.
- value[4]: a dictionary containing required parameters. Contains a dependency field.

attr CORRELATION_PEARSON `class-attribute` `instance-attribute`

```
CORRELATION_PEARSON = (
    "correlation_pearson",
    COL,
    "Pearson's Correlation Matrices",
    calculate_statistic_correlation_pearson,
    {},
)
```

attr COVARIANCE `class-attribute` `instance-attribute`

```
COVARIANCE = (
    "covariance",
    COL,
    "Covariances",
    calculate_statistic_covariance,
    {},
)
```

attr HISTOGRAM `class-attribute` `instance-attribute`

```
HISTOGRAM = (
    "histogram",
    COL,
    "Histogram",
    calculate_statistic_histogram,
    {},
)
```

attr IQR `class-attribute` `instance-attribute`

```
IQR = ('iqr', COL, 'IQR', calculate_statistic_iqr, {})
```

attr KURTOSIS `class-attribute` `instance-attribute`

```
KURTOSIS = (
    "kurtosis",
    COL,
    "Kurtosis",
    calculate_statistic_kurtosis,
    {},
)
```

attr MAX `class-attribute` `instance-attribute`

```
MAX = (
    "max",
    COL,
    "Maximal values",
    calculate_statistic_max,
    {},
)
```

attr MEAN `class-attribute` `instance-attribute`

```
MEAN = (
    "mean",
    COL,
    "Mean values",
    calculate_statistic_mean,
    {},
)
```

attr MEDIAN `class-attribute` `instance-attribute`

```
MEDIAN = (
    "median",
    COL,
    "Median values",
    calculate_statistic_median,
    {},
)
```

attr MIN `class-attribute` `instance-attribute`

```
MIN = (
    "min",
    COL,
    "Minimal values",
    calculate_statistic_min,
    {},
)
```

attr NONE `class-attribute` `instance-attribute`

```
NONE = ('none', NONE, '', None, None)
```

attr P25 `class-attribute` `instance-attribute`

```
P25 = (
    "percentile_25",
    COL,
    "25% percentiles",
    calculate_statistic_percentile,
    {"p": 25},
)
```

attr P50 `class-attribute` `instance-attribute`

```
P50 = (
    "percentile_50",
    COL,
    "50% percentiles",
    calculate_statistic_percentile,
    {"p": 50},
)
```

attr P75 `class-attribute` `instance-attribute`

```
P75 = (
    "percentile_75",
    COL,
    "75% percentiles",
    calculate_statistic_percentile,
    {"p": 75},
)
```

attr PAIR_HISTOGRAM `class-attribute` `instance-attribute`

```
PAIR_HISTOGRAM = (
    "pair-histogram",
    COL,
    "Pair-wise Distribution Histogram",
    calculate_statistic_histogram_pair_wise,
```

```
    {},
)
```

attr PCA `class-attribute` `instance-attribute`

```
PCA = (
    "pca",
    COL,
    "Principal Component Analysis",
    calculate_statistic_pca,
    {},
)
```

attr RANKS `class-attribute` `instance-attribute`

```
RANKS = (
    "ranks",
    ROW,
    "Ranks",
    calculate_statistic_ranks,
    {},
)
```

attr SKEWNESS `class-attribute` `instance-attribute`

```
SKEWNESS = (
    "skewness",
    COL,
    "Skewness",
    calculate_statistic_skewness,
    {},
)
```

attr STD `class-attribute` `instance-attribute`

```
STD = (
    "std",
    COL,
    "Standard deviations",
    calculate_statistic_std,
    {},
)
```

attr VARIANCE `class-attribute` `instance-attribute`

```
VARIANCE = (
    "var",
    COL,
    "Variances",
    calculate_statistic_variance,
    {},
)
```

attr ZSCORE `class-attribute` `instance-attribute`

```
ZSCORE = (
    "z-score",
    ROW,
    "Z-Scores",
    calculate_statistic_zscore,
    {},
)
```

class ENUM_StatisticsType

Bases: `Enum`

Determines the type of the statistic. - ROW: calculates the statistics for each entry in the database. - COL: calculates the statistics for each column of the database.

attr COL `class-attribute` `instance-attribute`

```
COL = 2
```

attr NONE `class-attribute` `instance-attribute`

```
NONE = 0
```

attr ROW `class-attribute` `instance-attribute`

```
ROW = 1
```

35. Module `src.export.export_config_rules`

35.1 **mod** AI4SurrogateModelling.src.export.export_config_rules

A module containing functionality exporting the configuration file specifications into a markdown file.

func export

```
export(*, config: BaseModel, tgt_fn: str)
```

Takes the supplied configuration model and stores its markdown specification to the supplied file.

Parameters:

Name	Type	Description	Default
`config`	`BaseModel`	A configuration specification.	*required*
`tgt_fn`	`str`	A filename of the resulting markdown file.	*required*

func schema_to_markdown

```
schema_to_markdown(
    model: type[BaseModel], level: int = 1
) -> str
```

Takes the supplied model schema and constructs a markdown string to be stored to a file.

Parameters:

Name	Type	Description	Default
`model`	`type[BaseModel]`	Model schema to be analyzed.	*required*
`level`	`int`	Recursion level for indentation purposes. Defaults to 1.	`1`

Returns:

Name	Type	Description
`str`	`str`	A string representing the markdown file.

36. Module `src.export.plots.tabular`

36.1 **mod** AI4SurrogateModelling.src.export.plots.tabular

summary

class Plotting

summary

meth __calculate_histograms_2d_asymmetric `staticmethod`

```
__calculate_histograms_2d_asymmetric(
    indices1: list[int],
    indices2: list[int],
    bin_edges_list1: list[ndarray],
    bin_edges_list2: list[ndarray],
    data1: ndarray,
    data2: ndarray,
)
```

meth __calculate_histograms_2d_symmetric `staticmethod`

```
__calculate_histograms_2d_symmetric(
    indices1: list[int],
    indices2: list[int],
    bin_edges_list1: list[ndarray],
    bin_edges_list2: list[ndarray],
    data1: ndarray,
    data2: ndarray,
)
```

meth __export_box_plot_helper `staticmethod`

```
__export_box_plot_helper(
    categories: list[str],
    values_min: ndarray,
    values_q1: ndarray,
    values_median: ndarray,
    values_q3: ndarray,
    values_max: ndarray,
    target_dir: str,
    prefix: str,
    indices: list[int],
)
```

meth __export_pair_plot_helper `staticmethod`

```
__export_pair_plot_helper(
    compact,
    bin_edges_list_1,
    bin_edges_list_2,
    weights_list_1,
    weights_list_2,
    labels_1,
    labels_2,
    density_matrix_list,
    indices1,
    indices2,
    max_y_lim_1d,
    max_y_lim_2d,
    bin_centers_list_1,
    bin_centers_list_2,
    target_dir,
    prefix,
)
```

meth __export_value_distribution_helper `staticmethod`

```
__export_value_distribution_helper(
    target_dir: str,
    prefix: str,
    nbins: int,
    data: ndarray,
    data_labels: list[str],
    indices: list[int],
    values_min: list[float],
```

```
    values_max: list[float],
)
```

summary

Parameters:

Name	Type	Description	Default
`target_dir`	`str`	*description*	*required*
`prefix`	`str`	*description*	*required*
`nbins`	`int`	*description*	*required*
`data`	`ndarray`	*description*	*required*
`data_labels`	`list[str]`	*description*	*required*
`indices`	`list[int]`	*description*	*required*
`values_min`	`list[float]`	*description*	*required*
`values_max`	`list[float]`	*description*	*required*

meth calculate_cummulative_distribution `staticmethod`

```
calculate_cummulative_distribution(
    *, data: array, nbins: int
) -> tuple[array, array]
```

Takes the input flattened numpy array and calculates a cummulative value distribution curve with 'nbins' values at the x-axis. The curve represents the answer to a question "What portion of data (y axis) has value v (x axis) or lower?"

Parameters:

Name	Type	Description	Default
`data`	`array`	A flattened numpy array containing the values.	*required*
`nbins`	`int`	Resolution of the function	*required*

Returns:

Type	Description
`tuple[array, array]`	tuple[np.array, np.array]: x and y values

meth calculate_histograms `staticmethod`

```
calculate_histograms(
    *,
    predefined_min: float = None,
    predefined_max: float = None,
    data: ndarray,
    nbins: int
)
```

meth export_box_plot `staticmethod`

```
export_box_plot(
    database: DatabaseTabular,
    indices: dict[list[int]] = None,
    target_dir: str = "exports/tabular/plots",
)
```

summary

Parameters:

Name	Type	Description	Default
`database`	`DatabaseTabular`	*description*	*required*
`input_indices`	`list[int]`	*description*. Defaults to None.	*required*
`output_indices`	`list[int]`	*description*. Defaults to None.	*required*
`target_dir`	`str`	*description*. Defaults to "exports/tabular/plots".	`'exports/tabular/plots'`

meth export_correlation_matrix `staticmethod`

```
export_correlation_matrix(
    database: DatabaseTabular,
    indices: dict[list[int]] = None,
    target_dir: str = "exports/tabular/plots",
)
```

summary

Parameters:

Name	Type	Description	Default
`database`	`DatabaseTabular`	*description*	*required*
`input_indices`	`list[int]`	*description*. Defaults to None.	*required*
`output_indices`	`list[int]`	*description*. Defaults to None.	*required*
`target_dir`	`str`	*description*. Defaults to "exports/tabular/plots".	`'exports/tabular/plots'`

meth export_covariance_matrix `staticmethod`

```
export_covariance_matrix(
    database: DatabaseTabular,
    indices: dict[list[int]] = None,
    target_dir: str = "exports/tabular/plots",
)
```

summary

Parameters:

Name	Type	Description	Default
`database`	`DatabaseTabular`	*description*	*required*
`input_indices`	`list[int]`	*description*. Defaults to None.	*required*
`output_indices`	`list[int]`	*description*. Defaults to None.	*required*
`target_dir`	`str`	*description*. Defaults to "exports/tabular/plots".	`'exports/tabular/plots'`

meth export_pair_plots `staticmethod`

```
export_pair_plots(
    database: DatabaseTabular,
    nbins: int,
    indices: dict[list[int]] = None,
    target_dir: str = "exports/tabular/plots",
    compact: bool = False,
)
```

summary

Parameters:

Name	Type	Description	Default
`database`	`DatabaseTabular`	*description*	*required*
`nbins`	`int`	*description*	*required*
`input_indices`	`list[int]`	*description*. Defaults to None.	*required*
`output_indices`	`list[int]`	*description*. Defaults to None.	*required*
`target_dir`	`str`	*description*. Defaults to "exports/tabular/plots".	`'exports/tabular/plots'`
`compact`	`bool`	*description*. Defaults to False.	`False`

`meth` export_sensitivity_matrix `staticmethod`

```
export_sensitivity_matrix(
    database: DatabaseTabular,
    input_indices: list[int] = None,
    output_indices: list[int] = None,
    target_dir: str = "exports/tabular/plots",
)
```

summary

Parameters:

Name	Type	Description	Default
`database`	`DatabaseTabular`	*description*	*required*
`input_indices`	`list[int]`	*description*. Defaults to None.	`None`
`output_indices`	`list[int]`	*description*. Defaults to None.	`None`
`target_dir`	`str`	*description*. Defaults to "exports/tabular/plots".	`'exports/tabular/plots'`

`meth` export_stability_matrix `staticmethod`

```
export_stability_matrix(
    database: DatabaseTabular,
    input_indices: list[int] = None,
    output_indices: list[int] = None,
    target_dir: str = "exports/tabular/plots",
)
```

summary

Parameters:

Name	Type	Description	Default
`database`	`DatabaseTabular`	*description*	*required*
`input_indices`	`list[int]`	*description*. Defaults to None.	`None`
`output_indices`	`list[int]`	*description*. Defaults to None.	`None`
`target_dir`	`str`	*description*. Defaults to "exports/tabular/plots".	`'exports/tabular/plots'`

`meth` export_value_distribution `staticmethod`

```
export_value_distribution(
    database: DatabaseTabular,
    nbins: int,
    indices: dict[list[int]] = None,
    target_dir: str = "exports/tabular/plots",
)
```

summary

Parameters:

Name	Type	Description	Default
`database`	`DatabaseTabular`	*description*	*required*
`nbins`	`int`	*description*	*required*
`input_indices`	`list[int]`	*description*. Defaults to None.	*required*
`output_indices`	`list[int]`	*description*. Defaults to None.	*required*
`target_dir`	`str`	*description*. Defaults to "exports/tabular/plots".	`'exports/tabular/plots'`

37. Module `src.export.reports.general_rajko`

37.1 **mod** AI4SurrogateModelling.src.export.reports.general_rajko

func generate_dashboard

```
generate_dashboard(
    dashboard_structure,
    output_file="dashboard.html",
    title="Data Dashboard",
)
```

func make_bar

```
make_bar(subset, label)
```

func make_correlation_sub

```
make_correlation_sub(index, total_subs)
```

func make_histogram

```
make_histogram(
    *, subset: ndarray, label: str, nbins: int, title: str
)
```

func make_line

```
make_line(subset, label)
```

func make_scatter

```
make_scatter(subset, label)
```

func serialize_dashboard

```
serialize_dashboard(dashboard_structure)
```

38. Module `src.export.reports.report_general`

38.1 **mod** AI4SurrogateModelling.src.export.reports.report_general

class ReporterTraining

meth __add_description staticmethod

```
__add_description(training_progress: TrainingManager)
```

meth __create_dirs staticmethod

```
__create_dirs(tgt_fn: str)
```

meth __generate_image staticmethod

```
__generate_image(stat, image_subdir, nepochs)
```

meth __load_template staticmethod

```
__load_template()
```

meth __update_body staticmethod

```
__update_body(code: str, value: str)
```

meth export_to_html staticmethod

```
export_to_html(
    training_manager: TrainingManager, tgt_fn: str
)
```

func create_report

```
create_report(*, fn: str, data: list[dict])
```

39. Module `src.export.reports.tabular`

39.1 **mod** AI4SurrogateModelling.src.export.reports.tabular

class ReporterTabular

meth __HTML_add_data_correlations `staticmethod`

```
__HTML_add_data_correlations(
    database: DatabaseTabular, target_dir: str
)
```

meth __HTML_add_data_covariances `staticmethod`

```
__HTML_add_data_covariances(
    database: DatabaseTabular, target_dir: str
)
```

meth __HTML_add_data_pairwise_distributions `staticmethod`

```
__HTML_add_data_pairwise_distributions(
    database: DatabaseTabular, target_dir: str
)
```

meth __HTML_add_data_sensitivity `staticmethod`

```
__HTML_add_data_sensitivity(
    database: DatabaseTabular, target_dir: str
)
```

meth __HTML_add_data_stability_coefficients `staticmethod`

```
__HTML_add_data_stability_coefficients(
    database: DatabaseTabular, target_dir: str
)
```

meth __HTML_add_dropdown_options `staticmethod`

```
__HTML_add_dropdown_options(
    database: DatabaseTabular, target_dir: str
)
```

meth __HTML_add_submenu `staticmethod`

```
__HTML_add_submenu(database: DatabaseTabular)
```

meth __add_boxplots `staticmethod`

```
__add_boxplots(
    database: DatabaseTabular, target_dir: str, prefix: str
)
```

meth __add_correlations `staticmethod`

```
__add_correlations(
    database: DatabaseTabular, target_dir: str, prefix: str
)
```

meth __add_covariances `staticmethod`

```
__add_covariances(
    database: DatabaseTabular, target_dir: str, prefix: str
)
```

meth __add_description `staticmethod`

```
__add_description(database: DatabaseTabular)
```

meth __add_distributions staticmethod

```
__add_distributions(
    database: DatabaseTabular, target_dir: str, prefix: str
)
```

meth __add_distributions_pairwise staticmethod

```
__add_distributions_pairwise(
    database: DatabaseTabular,
    target_dir: str,
    prefix: str,
    compact: bool = True,
)
```

meth __add_properties staticmethod

```
__add_properties(database: DatabaseTabular)
```

meth export_to_html staticmethod

```
export_to_html(*, database: DatabaseTabular, tgt_fn: str)
```

meth export_to_md staticmethod

```
export_to_md(
    database: DatabaseTabular,
    tgt_fn: str,
    prefix: str = "../",
)
```

meth export_to_pdf staticmethod

```
export_to_pdf(database: DatabaseTabular, tgt_fn: str)
```

40. Module `src.importer.database_importer`

40.1 `mod` AI4SurrogateModelling.src.importer.database_importer

Contains the parent class for all database importers related to tabular data.

`class` Importer

```
Importer(*, paths_tgt: list[str])
```

Serves as a parent class for all future tabular data importers.

Initializes the Importer with the path to the file/directory containing the database to be imported. Also retrieves the information about the MPI environment.

Parameters:

Name	Type	Description	Default
`paths_tgt`	`list[str]`	A list of paths to the source files containing the data to be imported.	*required*

`attr` data_paths `instance-attribute`

```
data_paths = []
```

`meth` get_data

```
get_data() -> tuple[list]
```

Returns data specified by self.data_path property, throws an error if not defined by the child class.

`meth` get_importer_code

```
get_importer_code() -> str
```

Returns the user specified unique code attached to this importer. This is used in databases for simple determination of whether some data should be imported or not.

Returns:

Name	Type	Description
`str`	`str`	A code attached to this importer.

41. Module `src.importer.tabular.database_importer_tabular`

41.1 **mod** AI4SurrogateModelling.src.importer.tabular.database_importer_tabular

Contains a tabular database importer parent class.

class ImporterTabular

```
ImporterTabular(*, paths_tgt: str)
```

Bases: `Importer`

Specialized Importer to load tabular data with input + output pairs.

Calls the parental initializer and performs various other operations specific to this type of Importer.

Parameters:

Name	Type	Description	Default
`paths_tgt`	`list[str]`	A list of paths to the source files containing the data to be imported.	*required*

42. Module `src.importer.tabular.database_importer_tabular_csv`

42.1 **mod** AI4SurrogateModelling.src.importer.tabular.database_importer_tabular_csv

Contains a general purpose importer to be used for text data in comma-separated-value (CSV) format.

class ImporterTabularCSV

```
ImporterTabularCSV(
    *,
    data_files: list[str],
    delim: str,
    headers: int,
    data_column_groups: dict
)
```

Bases: `ImporterTabular`

A general purpose CSV importer class.

Constructs the importer object to be used later.

Parameters:

Name	Type	Description	Default
`data_files`	`list[str]`	A list of paths to load the data from.	*required*
`delim`	`str`	A delimiting character sequence to form the columns.	*required*
`headers`	`int`	Number of rows to be skipped and used as column labels instead.	*required*
`data_column_groups`	`dict`	contains key-list[int] pairs determining the column groups defined upon this dataset.	*required*

attr data_code `instance-attribute`

```
data_code = join(
    [(replace("/", ".")) for v in (data_paths)]
)
```

attr data_column_groups `instance-attribute`

```
data_column_groups = data_column_groups
```

attr delim `instance-attribute`

```
delim = delim
```

attr headers `instance-attribute`

```
headers = headers
```

attr ncolumns `instance-attribute`

```
ncolumns = max(
    ncolumns, max(data_column_groups[column_group]) + 1
)
```

meth get_data

```
get_data() -> tuple[dict]
```

Loads and processes the data associated with this importer. All the data is loaded on one process.

Must be called by ALL MPI processes.

Returns:

Type	Description
`tuple[dict]`	tuple[dict]: Returns 2 dictionaries in the following order: - data, contains the groups and corresponding data - labels, contains the groups and corresponding data

`meth` get_importer_code

```
get_importer_code() -> str
```

Returns the user specified unique code attached to this importer. This is used in databases for simple determination of whether some data should be imported or not.

Returns:

Name	Type	Description
`str`	`str`	A code attached to this importer.

43. Module `src.importer.tabular.database_importer_tabular_dummy`

43.1 **mod** AI4SurrogateModelling.src.importer.tabular.database_importer_tabular_dummy

Contains a dummy importer class for experimentation.

class ImporterTabularDummy

```
ImporterTabularDummy(
    *,
    data_inputs: list,
    data_outputs: list,
    labels_inputs: list,
    labels_outputs: list,
    path_tgt: str = None
)
```

Bases: `ImporterTabular`

Takes already generated data and constructs an artifial importer on these data points. Used for testing of associated importer functionality.

Constructs this dummy importer.

Parameters:

Name	Type	Description	Default
`data_inputs`	`list`	A list of ALL input data points.	*required*
`data_outputs`	`list`	A list of ALL output data points.	*required*
`labels_inputs`	`list`	A list of labels associated with the input columns.	*required*
`labels_outputs`	`list`	A list of labels associated with the output columns.	*required*
`path_tgt`	`str`	Unnecessary in this object. Defaults to None.	`None`

attr data_inputs `instance-attribute`

```
data_inputs = data_inputs
```

attr data_outputs `instance-attribute`

```
data_outputs = data_outputs
```

attr labels_inputs `instance-attribute`

```
labels_inputs = labels_inputs
```

attr labels_outputs `instance-attribute`

```
labels_outputs = labels_outputs
```

attr nentries `instance-attribute`

```
nentries = len(data_inputs)
```

meth get_data

```
get_data() -> tuple[list]
```

Loads and processes the data associated with this importer. All the data is loaded on one process.

Must be called by ALL MPI processes.

Returns:

Type	Description
`tuple[list]`	tuple[list]: Returns 4 lists in the following order: - input data - output data - input labels - output labels

44. Module `src.importer.tabular.database_importer_tabular_random`

44.1 `mod` AI4SurrogateModelling.src.importer.tabular.database_importer_tabular_random

Contains an importer class generating random data for experimentation.

`class` ImporterTabularRandom

```
ImporterTabularRandom(
    *, nentries: int, data_column_groups: dict
)
```

Bases: `ImporterTabular`

Importer class generating random tabular data for experimentation.

Constructs this random tabular data importer.

Parameters:

Name	Type	Description	Default
`nentries`	`int`	Number of rows to be generated.	*required*
`data_column_groups`	`dict`	contains key-list[int] pairs determining the column groups defined upon this dataset.	*required*
`seed`	`int`	Random number generator seed.	*required*
`data_fn`	`str`	Unused parameter. Defaults to None.	*required*

`attr` data_code `instance-attribute`

```
data_code = 'random_importer'
```

`attr` data_column_groups `instance-attribute`

```
data_column_groups = data_column_groups
```

`attr` ncolumns `instance-attribute`

```
ncolumns = max(
    ncolumns, max(data_column_groups[column_group]) + 1
)
```

`attr` nentries `instance-attribute`

```
nentries = nentries
```

`meth` get_data

```
get_data() -> tuple[list]
```

Generates and returns the random data. Uses "numpy.random.rand".

Must be called by ALL MPI processes.

Returns:

Type	Description
`tuple[list]`	tuple[list]: Returns 4 lists in the following order: - input data - output data - input labels - output labels

`meth` get_importer_code

```
get_importer_code() -> str
```

Returns the user specified unique code attached to this importer. This is used in databases for simple determination of whether some data should be imported or not.

Returns:

Name	Type	Description
`str`	`str`	A code attached to this importer.

45. Module `src.database.labeled.database_labeled`

45.1 **mod** AI4SurrogateModelling.src.database.labeled.database_labeled

A module containing classes and method usable by all labeled dataset classes, i.e. datasets where inputs and outputs are known beforehand.

class DatabaseLabeled

```
DatabaseLabeled(path_tgt: str)
```

Bases: `ABC`

Serves as an interface for child classes.

Methods not implemented by children should throw an error.

Initializes the database in a target directory. If it exists, loads relevant metadata. If it doesnt exist, creates new directory and metadata.

Parameters:

Name	Type	Description	Default
`path_tgt`	`str`	defines a path to the top directory of the database	*required*

attr data `instance-attribute`

```
data = {}
```

attr data_loaded `instance-attribute`

```
data_loaded = {}
```

attr database_file_controller `instance-attribute`

```
database_file_controller = ParallelFile(f"{path_main}/data")
```

attr global2local_index_map `instance-attribute`

```
global2local_index_map = {}
```

attr local2global_index_map `instance-attribute`

```
local2global_index_map = {}
```

attr metadata `instance-attribute`

```
metadata = get_metadata()
```

attr path_main `instance-attribute`

```
path_main = path_tgt
```

meth add_data `abstractmethod`

```
add_data(*, importer: Importer) -> None
```

An abstract method to be implemented in child classes. Adds data to this database utilizing the importer object.

Parameters:

Name	Type	Description	Default
`importer`	`Importer`	Importer used to load, parse and retrieve data.	*required*

`meth` add_data_code

```
add_data_code(data_code: str) -> None
```

Adds a new code to the list of database sources.

Must be called by ALL MPI processes.

Parameters:

Name	Type	Description	Default
`data_code`	`str`	A code of the database being added.	*required*

`meth` add_data_key

```
add_data_key(key: str) -> None
```

Adds a new database prefix key to the list of known prefixes.

Parameters:

Name	Type	Description	Default
`key`	`str`	A new key to be added.	*required*

`meth` add_n_entries

```
add_n_entries(value: int) -> None
```

Increments the number of currently stored data entries by the supplied amount. Negative value causes a subtraction.

Parameters:

Name	Type	Description	Default
`value`	`int`	Amount by which to alter the current value.	*required*

`meth` add_transformation

```
add_transformation(f: Callable, kwargs: dict) -> None
```

Adds a transformation to a list of transformations performed on the entries in this database.

Parameters:

Name	Type	Description	Default
`f`	`Callable`	A function to be stored in the sequence of transformations.	*required*
`kwargs`	`dict`	Arguments to the transformation function.	*required*

`meth` append_new_data

```
append_new_data(
    data_local: dict[str], source_data_code: str
)
```

Takes the data provided in the data_local dictionary and appends them to this database. Adds the data source code to the list of already known sources.

Parameters:

Name	Type	Description	Default
`data_local`	`dict[str]`	A dictionary containing (prefix -> data) entries. For each prefix, append the data to the end of the database with the corresponding prefix key.	*required*
`source_data_code`	`str`	Code of the data source.	*required*

`meth` clear_database

```
clear_database() -> None
```

Deletes all data entries and metadata.

Should be called by ALL MPI processes.

`meth` contains_data

```
contains_data(data_code: str) -> bool
```

Checks whether this database already contains data entries with the specified data code.

Parameters:

Name	Type	Description	Default
`data_code`	`str`	A code uniquely determining the source of the data.	*required*

Returns:

Name	Type	Description
`bool`	`bool`	Returns True if data is contained, otherwise returns False.

`meth` contains_key

```
contains_key(key: str) -> bool
```

Determines whether this database contains the specified prefix key.

Parameters:

Name	Type	Description	Default
`key`	`str`	A database prefix key to be checked.	*required*

Returns:

Name	Type	Description
`bool`	`bool`	True if the supplied prefix exists in this database, False otherwise.

`meth` copy_database `staticmethod`

```
copy_database(
    *, database_dir_src: str, database_dir_tgt: str
) -> None
```

Copies the contents of an existing Database into this Database. This Database will be overwritten by this process.

Should be called by ALL MPI processes.

Parameters:

Name	Type	Description	Default
`database_dir_src`	`str`	Path to the directory of the existing database	*required*
`database_dir_tgt`	`str`	Path to the directory of the new database	*required*

`meth` data_changed

```
data_changed() -> None
```

Clears all chaches and supplementary structures. Called after the contents of the database has been effectivelly changed, so when required, the correct data is reloaded and/or recalculated.

Must be called by ALL MPI processes.

`meth` find_databases `staticmethod`

```
find_databases(dir_tgt: str) -> list[str]
```

Looks inside the specified directory and recursively determines all subdirectories containing a database.

If a path A/B/C contains a database, then the recursion stops, i.e. potential databases in A/B/C/* will not be discovered.

Parameters:

Name	Type	Description	Default
`dir_tgt`	`str`	Specifies which directory should be searched for for the databases.	*required*

Returns:

Type	Description
`list[str]`	list[str]: A list of strings representing absolute paths to the databases so they can be used as an argument to a database constructor

`meth` get_data

```
get_data(key: str) -> list[Any]
```

Loads (if not loaded) and returns the local list of objects associated with the supplied database prefix key.

Parameters:

Name	Type	Description	Default
`key`	`str`	A database prefix key to be queried.	*required*

Returns:

Type	Description
`list[Any]`	list[Any]: A local list of objects corresponding to the supplied prefix key.

`meth` get_data_keys

```
get_data_keys() -> list[str]
```

Retrives and returns the list of database prefix keys.

Returns:

Type	Description
`list[str]`	list[str]: A list of ALL the prefix keys contained in this database.

`meth` get_database_path

```
get_database_path() -> str
```

Generates a filename for a file containing the input data.

Returns:

Name	Type	Description
`str`	`str`	Name of the input file.

`meth` get_dataloaders `abstractmethod`

```
get_dataloaders(
    *,
    train_ratio: float,
    validation_ratio: float,
    batch_size: int,
    dtype: ENUM_Dtypes,
    inputs_require_gradient: bool = False
) -> tuple
```

An abstract method to be implemented in child classes. The method is supposed to construct a pair of dataloaders ready to be used in a training process in a distributed environment.

Parameters:

Name	Type	Description	Default
`train_ratio`	`float`	A number between 0 and 1, the portion of this database to be used in training. 0 < train_ratio + validation_ratio <= 1.	*required*
`validation_ratio`	`float`	A number between 0 and 1, the portion of this database to be used in validation. 0 < train_ratio + validation_ratio <= 1.	*required*
`batch_size`	`int`	Batch size.	*required*
`dtype`	`ENUM_Dtypes`	Data type of theentries sampled via the provided dataloaders.	*required*
`inputs_require_gradient`	`bool`	Makes it possible to calculate partial derivatives w.r.t. the inputs. Defaults to False.	`False`

Returns:

Name	Type	Description
`tuple`	`tuple`	Training and Validation dataloaders.

`meth` get_default_metadata

```
get_default_metadata() -> dict[str:Any]
```

Constructs and returns default metadata associated with this database.

Returns:

Type	Description
`dict[str:Any]`	dict[str: Any]: default empty metadata dictionary.

`meth` get_directory_plots

```
get_directory_plots() -> str
```

Returns the path to a directory containing the exported figures related to this database.

Returns:

Name	Type	Description
`str`	`str`	Path to the directory containing all figures.

`meth` get_directory_stats

```
get_directory_stats(
    *, key: str, stat_type: ENUM_StatisticsType
) -> str
```

Constructs and returns the directory for the storage of database related statistics.

Parameters:

Name	Type	Description	Default
`key`	`str`	database prefix key, e.g. "inputs" or "outputs".	*required*
`stat_type`	`ENUM_StatisticsType`	Enumerator of the statistics type	*required*

Returns:

Name	Type	Description
`str`	`str`	Path to the directory used to store the related statistics.

`meth` get_label

```
get_label(*, key: str, idx: int) -> str
```

Retrieves the specified column label in the part of the database with the supplied prefix key.

Parameters:

Name	Type	Description	Default
`key`	`str`	Database prefix key to be queried.	*required*
`idx`	`int`	Index of the label to be retrieved.	*required*

Returns:

Name	Type	Description
`str`	`str`	The label.

`meth` get_label_index

```
get_label_index(*, label: str, key: str) -> int
```

Retrievs the index of the supplied label in the database prefix.

Parameters:

Name	Type	Description	Default
`label`	`str`	Label to be queried.	*required*
`key`	`str`	The label's database prefix key.	*required*

Returns:

Name	Type	Description
`int`	`int`	Index of the label among all the labels within the same prefix group.

`meth` get_label_key

```
get_label_key(label: str) -> str
```

Retrieves the database prefix key in which the supplied label is occurring.

Parameters:

Name	Type	Description	Default
`label`	`str`	Label existing in one of the database prefixes.	*required*

Returns:

Name	Type	Description
`str`	`str`	Database prefix key containing the supplied label.

`meth` get_metadata

```
get_metadata() -> dict[str:Any]
```

Returns metadata associated with this Database. If no metadata exist, constructs an empty metadata structure.

Returns:

Type	Description
`dict[str:Any]`	dict[str: Any]: Dictionary containing the metadata

`meth` get_metadata_path

```
get_metadata_path() -> str
```

Returns the path to the file containing metadata associated with this Database.

Returns:

Name	Type	Description
`str`	`str`	path to the metadata file.

`meth` get_n_entries_global

```
get_n_entries_global() -> int
```

Returns the number of currently stored entries in this database.

Returns:

Name	Type	Description
`int`	`int`	Number of currently stored entries in this database

`meth` get_n_entries_local

```
get_n_entries_local(*, key: str) -> int
```

Retrieves the number of dataset entries loaded by this process.

Parameters:

Name	Type	Description	Default
`key`	`str`	Database prefix key to be queried.	*required*

Returns:

Name	Type	Description
`int`	`int`	Number of entries on this MPI process associated with the prefix key.

`meth` get_stats_path

```
get_stats_path(
    *,
    stat_code_value: tuple[str, ENUM_StatisticsType],
    key: str
) -> str
```

Provided with the statistics type (either per dataset entry or per dataset category), constructs a relative path to a file containing the specified statistics.

Parameters:

Name	Type	Description	Default
`stat_code_value`	`tuple[str, ENUM_StatisticsType]`	statistics code name and type.	*required*
`key`	`str`	database prefix key, i.g. "inputs" or "outputs".	*required*

Returns:

Name	Type	Description
`str`	`str`	Path to a file containing the statistic.

`meth` get_transformation

```
get_transformation(
    t: dict[object, dict],
) -> tuple[Callable, dict]
```

Takes the stored transformation and converts it to a callable function and its arguments, then returns it.

Parameters:

Name	Type	Description	Default
`t`	`dict[object, dict]`	A dictionary created via the add_transformation function, i.e. it contains the "function" and "kwargs" fields.	*required*

Returns:

Type	Description
`tuple[Callable, dict]`	tuple[Callable, dict]: *description*

`meth` is_database_directory `staticmethod`

```
is_database_directory(dir_path: str) -> bool
```

A static method which checks whether the provided directory contains a Database or not.

Assumes the directory exists.

Parameters:

Name	Type	Description	Default
`dir_path`	`str`	Path to the directory	*required*

Returns:

Name	Type	Description
`bool`	`bool`	True if the directory contains a database, False otherwise.

`meth` load_database

```
load_database(*, keys: list[str]) -> None
```

Loads the database from the disk for each of the provide database prefixes.

Should be called on ALL MPI processes.

Parameters:

Name	Type	Description	Default
`keys`	`list[str]`	A list of database prefixes to load.	*required*

`meth` load_stats

```
load_stats(
    *,
    stat_code_value: tuple[str, ENUM_StatisticsType],
    data_keys: list[str]
) -> dict[str:Any]
```

Loads the supplied database statistic for the specified database prefixes.

Parameters:

Name	Type	Description	Default
`stat_code_value`	`tuple[str, ENUM_StatisticsType]`	statistics code name and type.	*required*
`data_keys`	`list[str]`	A list of database prefix keys to be loaded.	*required*

Returns:

Type	Description
`dict[str:Any]`	dict[str: Any]: Calculated statistics in a (key -> value(s)) dictionary.

`meth` mark_stats_outdated

```
mark_stats_outdated() -> None
```

Deletes all files containing calculated dataset statistics, effectivelly forcing a recalculation when queried.

Must be called by ALL MPI processes.

`meth` metadata_get_value

```
metadata_get_value(key: str) -> Any
```

Retrieves the metadata associated with the specified key.

Parameters:

Name	Type	Description	Default
`key`	`str`	Key for the metadata field	*required*

Returns:

Name	Type	Description
`Any`	`Any`	Value of the metadata field

`meth` metadata_set_value

```
metadata_set_value(key: str, value: Any) -> None
```

Sets a value of a metadata field and stores the updated state in the metadata file.

Must be called by ALL MPI processes.

Parameters:

Name	Type	Description	Default
`key`	`str`	Name of the metadata field	*required*
`value`	`Any`	Desired value of the corresponding metadata field	*required*

`meth` metadata_set_values

```
metadata_set_values(values: dict[str:Any]) -> None
```

`meth` prune_columns

```
prune_columns(
    kept_columns: dict[str],
) -> tuple[dict[str], dict[str], dict[str], dict[str]]
```

For each database prefix, takes the list of kept columns and removes the rest.

Parameters:

Name	Type	Description	Default
`kept_columns`	`dict[str]`	A (Prefix -> kept column indices) map.	*required*

Returns:

Type	Description
`dict[str]`	dict[str]: Local data with pruned columns.
`dict[str]`	dict[str]: New label lists for each database prefix.
`dict[str]`	dict[str]: New dimensions for each database prefix (i.e. number of columns).
`dict[str]`	dict[str]: Updated column transformations.

`meth` recalculate_global2local_indexing

```
recalculate_global2local_indexing(
    *, keys: list[str]
) -> None
```

Constructs global -> local map from the known local -> global map.

Parameters:

Name	Type	Description	Default
`keys`	`list[str]`	List of prefix keys for which the map should be constructed.	*required*

`meth` save_metadata

```
save_metadata() -> None
```

Saves metadata associated with this Database into a binary file via Pickle.

Should be called by ALL MPI processes.

`meth` set_description

```
set_description(msg: str) -> None
```

Sets the description of this database.

Parameters:

Name	Type	Description	Default
`msg`	`str`	Text of the new description.	*required*

`meth` stats_calculated

```
stats_calculated(
    *,
    stat_code_value: tuple[str, ENUM_StatisticsType],
    data_keys: list[str]
) -> dict[str:bool]
```

For each database prefix, determines whether the supplied statistic is up-to date or not.

Parameters:

Name	Type	Description	Default
`stat_code_value`	`tuple`	statistics code name and type.	*required*
`data_keys`	`list[str]`	Database prefix keys for which to check the statistics.	*required*

Returns:

Type	Description
`dict[str:bool]`	dict[str: bool]: output[prefix] is True when the statistic is up-to date, False otherwise.

`meth` train_test_split

```
train_test_split(
    train_ratio: float,
    test_ratio: float,
    validation_ratio: float,
) -> None
```

Randomly splits the entries in this database into three disjoint sets for training, testing and validation. Assumes the training ratio is greater than zero. When the ratios sum up to values greater than one, clips the values.

Parameters:

Name	Type	Description	Default
`train_ratio`	`float`	What portion of this database should be used for training.	*required*
`test_ratio`	`float`	What portion of this database should be used for testing.	*required*
`validation_ratio`	`float`	What portion of this database should be used for validation.	*required*

Raises:

Type	Description
`ValueError`	When the supplied ratios are lower than zero.

`meth` update_metadata_value

```
update_metadata_value(key: str, values: dict) -> None
```

Retrives the current value of the metadata field with the given key and updates it so it contains the information contained the the values dictionary. That means existing values are overwritten and other values are added. The updated metadata is then saved.

Must be called by ALL MPI processes.

Parameters:

Name	Type	Description	Default
`key`	`str`	Metadata field key to be updated.	*required*
`values`	`dict`	The new state of values for the given metadata field.	*required*

`meth` update_stats

```
update_stats(
    *,
    stat_code_value: tuple[str, ENUM_StatisticsType],
    data: dict[str]
) -> None
```

Takes the supplied calculated statistics dictionary and stores them in the corresponding statistics file(s).

Should be called by ALL MPI processes.

Parameters:

Name	Type	Description	Default
`stat_code_value`	`tuple[str, ENUM_StatisticsType]`	statistics code name and type.	*required*
`data`	`dict[str]`	Calculated statistics in a (key -> value(s)) manner to be saved to disk.	*required*

46. Module `src.database.labeled.tabular.aux`

46.1 **mod** AI4SurrogateModelling.src.database.labeled.tabular.aux

A module containing auxilliary methods used in the tabular dataset workflow.

func stat_function_decorator

```
stat_function_decorator(func: Callable) -> Callable
```

A decorator function checking the presence of arguments for functions used for tabular statistics calculations.

Parameters:

Name	Type	Description	Default
`func`	`Callable`	Function which calculates the tabular statistic.	*required*

Returns:

Name	Type	Description
`Callable`	`Callable`	decorator

47. Module `src.database.labeled.tabular.database_tabular`

47.1 mod AI4SurrogateModelling.src.database.labeled.tabular.database_tabular

Module containing methods and classes related to the inner workings of Tabular Datasets (Derived from Labeled Datasets)

class DatabaseTabular

```
DatabaseTabular(path_tgt: str)
```

Bases: `DatabaseLabeled`

Constructs a database containing Tabular data.

Contains methods suitable to Tabular data analysis, filtering/cleaning up noise and plotting functionality.

Loads the existing Database. If it does not exist, creates the neccessary directory structures and metadata files.

Parameters:

Name	Type	Description	Default
`path_tgt`	`str`	Path to where the Database is stored	*required*

meth add_data

```
add_data(importers: list[ImporterTabular]) -> None
```

Uses the provided Importers to parse the Tabular data from the specified file/directory. The Database is then updated. Applies all transformations already included in the history of this database

Parameters:

Name	Type	Description	Default
`importers`	`list[ImporterTabular]`	User provided importer instances for reading new data. Assumes the order of importing does not matter.	*required*

meth add_new_column

```
add_new_column(
    tgt_data_key: str,
    new_column_name: str,
    src_data_columns: list[tuple[str, int]],
    op_func: Callable,
) -> None
```

Augments the the data of this database by adding a new column as a function of existing columns.

Parameters:

Name	Type	Description	Default
`tgt_data_key`	`str`	The database prefix key for the augmentation to be added.	*required*
`new_column_name`	`str`	The label associated with the newly created column.	*required*
`src_data_columns`	`list[tuple[str, int]]`	A list of database prefix and index pairs to be used as inputs to the augmentation function.	*required*
`op_func`	`Callable`	The augmentation function.	*required*

meth calculate_statistic

```
calculate_statistic(
    *,
    stat_code: ENUM_StatisticsCode_Tabular,
    data_keys: list[str]
) -> None
```

Method calculating the supplied statistic depending on the data retrieved from the StatCode Enumerator for each of the database prefix keys specified.

Parameters:

Name	Type	Description	Default
`stat_code`	`ENUM_StatisticsCode_Tabular`	Enumerator containing instructions for calculating the statistic.	*required*
`data_keys`	`list[str]`	Database prefix keys for which to calculate the statistic.	*required*

meth cleanup_columns

```
cleanup_columns(
    *,
    keys_to_consider: list[str],
    correlation_eps: float,
    variance_eps: float
) -> None
```

Calls a suite of column cleaning methods. As of now, calls two methods, one based on column variance and the other on the column correlation.

Parameters:

Name	Type	Description	Default
`keys_to_consider`	`list[str]`	Database prefix keys for which the cleaning process is executed.	*required*
`correlation_eps`	`float`	Epsilon value for determining when a column is correlated to other columns.	*required*
`variance_eps`	`float`	Epsilon value for determining when a column has zero variance or not.	*required*

meth cleanup_rows_iqr

```
cleanup_rows_iqr(
    *, keys_to_consider: list[str], iqr_coefficient: float
) -> None
```

Determines which rows fall outside the scope of the IQR test with the supplied iqr_coefficient and removes the rows which fail this test.

Parameters:

Name	Type	Description	Default
`keys_to_consider`	`list[str]`	Database prefix keys for which the cleaning process is executed.	*required*
`iqr_coefficient`	`float`	A value greater than zero determining the interval from the mean value.	*required*

meth cleanup_rows_subspace

```
cleanup_rows_subspace(
    simplex_points: ndarray,
```

```
    simplices: list[list[int]],
    simplex_point_labels: list[str],
    tol: float,
) -> None
```

Cleans up the rows of this database. The criterion for removal is based on the proximity of the row to a set of simplices defined by their coordinates and adjacency. When the shortest distance of the row to any point in the simplices is less then the supplied tolerance, it remains in the set.

Not all columns are taken into account for the proximity calculation. The subspace in which the comparison is calculated is defined by the labels of the simplex coordinates. For example, when the dataset contains column labels A, B, C, D and the simplex coordinates contain labels D, B, then the proximity calculation is based off of the columns B, D.

Parameters:

Name	Type	Description	Default
`simplex_points`	`ndarray`	An array of shape (n, m), where n is the number of simplex coordinates and m is the dimension of each simplex point.	*required*
`simplices`	`list[list[int]]`	An incidence list forming the simplex set. For example: simplices = [[0, 1, 4]] represents one 3d simplices with coordinates from the simplex_points parameter at rows: 0, 1, 4.	*required*
`simplex_point_labels`	`list[str]`	Each dimension in the simplex coordinate system correspond to one column in one of this database's prefix keys.	*required*
`tol`	`float`	The distance treshold.	*required*

`meth` convert_external_data_to_csv

```
convert_external_data_to_csv(
    *, data: dict, fn: str, delimiter: str
)
```

Takes the user provided data and uses labeling from this database and exports it to a comma separated file using the supplied delimitting character.

Parameters:

Name	Type	Description	Default
`data`	`dict`	Dictionary containing the data to be exported. The keys in the dictionary must correspond to the keys in this database.	*required*
`fn`	`str`	Target filename.	*required*
`delimiter`	`str`	Delimiting character.	*required*

`meth` convert_to_csv

```
convert_to_csv(
    *, fn: str, delimiter: str, denormalize: bool = False
) -> None
```

Takes the content of this tabular database and stores it in a comma separated file with the supplied delimiting character.

Must be called by ALL MPI processes.

Parameters:

Name	Type	Description	Default
`fn`	`str`	Filename of the data to be saved in.	*required*
`delimiter`	`str`	Delimiting character to be used.	*required*
`denormalize`	`bool`	If true, converts the data back to the original values.	`False`

`meth` get_data_range

```
get_data_range(key: str) -> ndarray
```

Retrives the minimal and maximal values associated with the supplied database prefix key.

Parameters:

Name	Type	Description	Default
`key`	`str`	A database prefix key for which to retrieve the stats.	*required*

Returns:

Type	Description
`ndarray`	np.ndarray: A tensor fo shape (n, 2), where the first column represents the minimal values and the second column represent the maximal values.

`meth` get_data_ranges

```
get_data_ranges() -> dict[tensor]
```

Retrives the minimal and maximal values associated with all database prefix keys.

Returns:

Type	Description
`dict[tensor]`	dict[torch.tensor]: A dictionary containing tensors with minimal and maximal values.

`meth` get_dataloader_random

```
get_dataloader_random(
    *, n: int, batch_size: int, dtype: dtype
) -> DictionaryDataLoader
```

Constructs and returns pytorch Dataloader to be used outside of the training process for various testing purposes.

Parameters:

Name	Type	Description	Default
`batch_size`	`int`	Size of each batch.	*required*
`dtype`	`dtype`	Data type of the objects generated by the dataloader.	*required*

Returns:

Name	Type	Description
`DictionaryDataLoader`	`DictionaryDataLoader`	Random data uniformly distributed.

meth get_dataloaders

```
get_dataloaders(
    *,
    batch_size: int,
    dtype: dtype,
    data_require_gradient: dict
) -> tuple[
    DictionaryDataLoader,
    DictionaryDataLoader,
    DictionaryDataLoader,
]
```

Constructs and returns pytorch Dataloaders to be used in the training process. Assumes that database split has been called previously.

Parameters:

Name	Type	Description	Default
`batch_size`	`int`	Size of each batch.	*required*
`dtype`	`dtype`	Data type of the objects generated by the dataloaders.	*required*
`data_require_gradient`	`dict`	Specifies whether the generated objects should contain gradient information or not. Defaults to False.	*required*

Returns:

Type	Description
`tuple[DictionaryDataLoader, DictionaryDataLoader, DictionaryDataLoader]`	tuple[DictionaryDataLoader, DictionaryDataLoader, DictionaryDataLoader]: Training, Testing and Validation Dataloaders.

meth get_default_metadata

```
get_default_metadata() -> dict
```

Constructs and returns default metadata associated with this type of database. Calls the parent method and adds new entries, namely: - dimensions of inputs and outputs - labels of inputs and outputs - trasnforms of inputs and outputs

Returns:

Name	Type	Description
`dict`	`dict`	Constructed metadata

meth get_dim

```
get_dim(key: str) -> int
```

Retrives the number of columns associated with the supplied database prefix key.

Parameters:

Name	Type	Description	Default
`key`	`str`	A database prefix key for which to retrieve the number of columns.	*required*

Returns:

Name	Type	Description
`int`	`int`	Number of columns.

meth get_statistic

```
get_statistic(
    *,
    data_keys: list[str],
    stat_code: ENUM_StatisticsCode_Tabular = NONE
) -> dict[str:Any]
```

If not already calculated, calculates the corresponding statistic and stores it on disk. Then returns it for the database prefix keys supplied by the user.

Parameters:

Name	Type	Description	Default
`data_keys`	`list[str]`	Database prefix keys for which to retrieve the statistic.	*required*
`stat_code`	`ENUM_StatisticsCode_Tabular`	Statistic identifier. Defaults to `StatCode.NONE`.	`NONE`

Returns:

Type	Description
`dict[str:Any]`	dict[str, Any] | None: Calculated statistics keyed by prefix, or
`dict[str:Any]`	`None` when `stat_code` is `NONE`.

meth remove_duplicate_rows

```
remove_duplicate_rows(duplicate_row_eps: float) -> None
```

Iterates over all pairs of rows in this database and detects duplicate entries. Then it removes the duplicities from the filesystem.

Must be called by ALL MPI processes.

Parameters:

Name	Type	Description	Default
`duplicate_row_eps`	`float`	Duplicity detection epsilon (|A-B| <= epsilon ⇒ duplicate).	*required*

meth remove_rows_simplex_proximity

```
remove_rows_simplex_proximity(
    *,
    data: dict[str],
    simplex_points: ndarray,
    simplices: list[list[int]],
    tol: float,
    simplex_point_labels: list[str],
    local2global_index_map: dict = None
) -> dict
```

Takes the input tabular data and removes some of its rows. The criterion for removal is based on the proximity of the row to a set of simplices defined by their coordinates and adjacency. When the shortest distance of the row to any point in the simplices is less then the supplied tolerance, it remains in the set.

Not all columns are taken into account for the proximity calculation. The subspace in which the comparison is calculated is defined by the labels of the simplex coordinates. For example, when the dataset contains column labels A, B, C, D and the simplex coordinates contain labels D, B, then the proximity calculation is based off of the columns B, D.

Parameters:

Name	Type	Description	Default
`data`	`dict[str]`	Dictionary containing the data to be cleaned. The keys in the dictionary correspond to the keys in this database.	*required*
`simplex_points`	`ndarray`	An array of shape (n, m), where n is the number of simplex coordinates and m is the dimension of each simplex point.	*required*
`simplices`	`list[list[int]]`	An incidence list forming the simplex set. For example: simplices = [[0, 1, 4]] represents one 3d simplices with coordinates from the simplex_points parameter at rows: 0, 1, 4.	*required*
`tol`	`float`	The distance treshold.	*required*
`simplex_point_labels`	`list[str]`	Each dimension in the simplex coordinate system correspond to one column in one of this database's prefix keys.	*required*
`local2global_index_map`	`dict`	*description*. Defaults to None.	`None`

Returns:

Name	Type	Description
`dict`	`dict`	Dictionary with the same format as the input data, but with possibly lesser number of rows per prefix key.

`meth` revert_transform_range

```
revert_transform_range(keys: list[str]) -> None
```

Reverts the normalization transformations on this database for the supplied list of database prefix keys.

Parameters:

Name	Type	Description	Default
`keys`	`list[str]`	A list of database prefix keys for which to revert the transformation operations.	*required*

`meth` revert_transform_range_external

```
revert_transform_range_external(
    *, keys: list[str], data: dict[str]
) -> dict[str]
```

Takes the input data and applies the backward transformations on it as if it were part of this database.

Parameters:

Name	Type	Description	Default
`keys`	`list[str]`	A list of prefix keys for which to apply the backward transformation.	*required*
`data`	`dict[str]`	Dictionary containing the data for some of this database's prefix keys.	*required*

Returns:

Type	Description
`dict[str]`	dict[str]: Dictionary containing the transformed data in the same format as the input data dictionary.

`meth` transform_data

```
transform_data(
    *, data: ndarray, data_labels: list[str]
) -> tuple[ndarray, dict[str:tuple]]
```

Takes the supplied data with the corresponding labels and applies all transformations in the history of this database to it.

Parameters:

Name	Type	Description	Default
`data`	`ndarray`	Raw data to be transformed.	*required*
`data_labels`	`list[str]`	Labels for each column of the supplied raw data.	*required*

Returns:

Name	Type	Description
`tuple`	`tuple[ndarray, dict[str:tuple]]`	Transformed data with the same shape as the raw data and a dictionary containing label index mapping from the respective prefix key ordering to the raw data ordering.

`meth` transform_range

```
transform_range(
    keys: list[str], target_range: tuple[float, float]
) -> None
```

For each of the supplied database prefix keys, we perform linear scaling and shift operations so the resulting values lie in the specified range.

Parameters:

Name	Type	Description	Default
`keys`	`list[str]`	A list of database prefix keys for which to perform the transformation.	*required*
`target_range`	`tuple`	Target minimal and maximal values of the transformed data ranges.	*required*

48. Module `src.database.labeled.tabular.database_tabular_stats`

48.1 **mod** AI4SurrogateModelling.src.database.labeled.tabular.database_tabular_stats

49. Module `src.database.labeled.tabular.stat_calculations.correlation_pearson`

49.1 **mod** AI4SurrogateModelling.src.database.labeled.tabular.stat_calculations.correlation_pearson

Module containing functions related to calculating the Pearson's Correlation coefficient matrix in parallel.

https://en.wikipedia.org/wiki/Pearson_correlation_coefficient

func calculate_statistic_correlation_pearson

```
calculate_statistic_correlation_pearson(
    *,
    tabular_dataset: DatabaseTabular,
    keys: list[str],
    **kwargs: dict
) -> dict
```

Calculates the Pearson's correlation coefficient matrices for each combination of the database prefix keys in the supplied list of prefix keys.

Must be called by ALL MPI processes.

Parameters:

Name	Type	Description	Default
`tabular_dataset`	`DatabaseTabular`	An instantiation of the tabular database.	*required*
`keys`	`list[str]`	A list of some of the database's prefix key for which to calculate the matrices.	*required*
`kwargs`	`dict`	a dictionary containing supplementary parameters, like calculated dependencies.	`{}`

Returns:

Name	Type	Description
`dict`	`dict`	A dictionary containing the correlation matrices. The key to each matrix is "A-B", where A and B are from the list of keys.

50. Module `src.database.labeled.tabular.stat_calculations.covariance`

50.1 **mod** AI4SurrogateModelling.src.database.labeled.tabular.stat_calculations.covariance

Module containing functions related to calculating the covariance matrix in parallel.

https://en.wikipedia.org/wiki/Covariance

func calculate_statistic_covariance

```
calculate_statistic_covariance(
    *,
    tabular_dataset: DatabaseTabular,
    keys: list[str],
    **kwargs: dict
) -> dict
```

Calculates the covariance matrices for each combination of the database prefix keys in the supplied list of prefix keys.

Must be called by ALL MPI processes.

Parameters:

Name	Type	Description	Default
`tabular_dataset`	`DatabaseTabular`	An instantiation of the tabular database.	*required*
`keys`	`list[str]`	A list of some of the database's prefix key for which to calculate the matrices.	*required*
`kwargs`	`dict`	a dictionary containing supplementary parameters, like calculated dependencies.	`{}`

Returns:

Name	Type	Description
`dict`	`dict`	A dictionary containing the covariance matrices. The key to each matrix is "A-B", where A and B are from the list of keys.

51. Module `src.database.labeled.tabular.stat_calculations.histogram`

51.1 **mod** AI4SurrogateModelling.src.database.labeled.tabular.stat_calculations.histogram

Module containing functions related to calculating the various histogram data in parallel.

func calculate_statistic_histogram

```
calculate_statistic_histogram(
    *,
    tabular_dataset: DatabaseTabular,
    keys: list[str],
    **kwargs: dict
) -> dict
```

Calculates the 1D histograms for each database prefix key and its columns.

Must be called by ALL MPI processes.

Parameters:

Name	Type	Description	Default
`tabular_dataset`	`DatabaseTabular`	An instantiation of the tabular database.	*required*
`keys`	`list[str]`	A list of some of the database's prefix key for which to calculate the histograms.	*required*
`kwargs`	`dict`	a dictionary containing supplementary parameters, like calculated dependencies.	`{}`

Returns:

Name	Type	Description
`dict`	`dict`	A dictionary containing the histograms. The key to each histogram is "A", where A is from the list of keys.

52. Module src.database.labeled.tabular.stat_calculations.histogram_pair_wise

52.1 mod AI4SurrogateModelling.src.database.labeled.tabular.stat_calculations.histogram_pair_wise

Module containing functions related to calculating the pair-wise histograms in parallel.

func calculate_statistic_histogram_pair_wise

```
calculate_statistic_histogram_pair_wise(
    *,
    tabular_dataset: DatabaseTabular,
    keys: list[str],
    **kwargs: dict
) -> dict
```

Calculates the 2D histograms (heatmaps) for each combination of the database prefix keys in the supplied list of prefix keys.

Must be called by ALL MPI processes.

Parameters:

Name	Type	Description	Default
`tabular_dataset`	`DatabaseTabular`	An instantiation of the tabular database.	*required*
`keys`	`list[str]`	A list of some of the database's prefix key for which to calculate the histograms.	*required*
`kwargs`	`dict`	a dictionary containing supplementary parameters, like calculated dependencies.	`{}`

Returns:

Name	Type	Description
`dict`	`dict`	A dictionary containing the histograms. The key to each histogram is "A-B", where A and B are from the list of keys.

53. Module `src.database.labeled.tabular.stat_calculations.iqr`

53.1 **mod** AI4SurrogateModelling.src.database.labeled.tabular.stat_calculations.iqr

Module containing functions related to calculating the IQR tresholds in parallel.

https://en.wikipedia.org/wiki/Interquartile_range

func calculate_statistic_iqr

```
calculate_statistic_iqr(
    *, tabular_dataset, keys, **kwargs
) -> dict
```

Uses the calculated q1 and q3 values and populates the IQR interval.

Returns:

Name	Type	Description
`dict`	`dict`	A dictionary containing the data. The key to each value is "A", where A is from the list of keys.

54. Module `src.database.labeled.tabular.stat_calculations.kurtosis`

54.1 **mod** AI4SurrogateModelling.src.database.labeled.tabular.stat_calculations.kurtosis

Module containing functions related to calculating the Kurtosis metric in parallel.

https://en.wikipedia.org/wiki/Kurtosis

func calculate_statistic_kurtosis

```
calculate_statistic_kurtosis(
    *,
    tabular_dataset: DatabaseTabular,
    keys: list[str],
    **kwargs: dict
) -> dict
```

Calculates the Kurtosis coefficient for each database prefix key and its columns.

Parameters:

Name	Type	Description	Default
`tabular_dataset`	`DatabaseTabular`	An instantiation of the tabular database.	*required*
`keys`	`list[str]`	A list of some of the database's prefix key for which to calculate the coefficients.	*required*
`kwargs`	`dict`	a dictionary containing supplementary parameters, like calculated dependencies.	`{}`

Returns:

Name	Type	Description
`dict`	`dict`	A dictionary containing the coefficients. The key to each array of coefficients is "A", where A is from the list
	`dict`	of keys.

55. Module `src.database.labeled.tabular.stat_calculations.max`

55.1 **mod** AI4SurrogateModelling.src.database.labeled.tabular.stat_calculations.max

Module containing functions related to calculating maximal values over tabular datasets in parallel.

func calculate_statistic_max

```
calculate_statistic_max(
    *,
    tabular_dataset: DatabaseTabular,
    keys: list[str],
    **kwargs: dict
) -> dict
```

Calculates the maximal values for each database prefix key and its columns.

Must be called by ALL MPI processes.

Parameters:

Name	Type	Description	Default
`tabular_dataset`	`DatabaseTabular`	An instantiation of the tabular database.	*required*
`keys`	`list[str]`	A list of some of the database's prefix key for which to calculate the values.	*required*
`kwargs`	`dict`	a dictionary containing supplementary parameters, like calculated dependencies.	`{}`

Returns:

Name	Type	Description
`dict`	`dict`	A dictionary containing the calculated values. The key to each array of values is "A", where A is from the list of keys.

56. Module `src.database.labeled.tabular.stat_calculations.mean`

56.1 **mod** AI4SurrogateModelling.src.database.labeled.tabular.stat_calculations.mean

Module containing functions related to calculating mean values over tabular datasets in parallel.

https://en.wikipedia.org/wiki/Mean

func calculate_statistic_mean

```
calculate_statistic_mean(
    *,
    tabular_dataset: DatabaseTabular,
    keys: list[str],
    **kwargs: dict
) -> dict
```

Calculates the mean values for each database prefix key and its columns.

Must be called by ALL MPI processes.

Parameters:

Name	Type	Description	Default
`tabular_dataset`	`DatabaseTabular`	An instantiation of the tabular database.	*required*
`keys`	`list[str]`	A list of some of the database's prefix key for which to calculate the values.	*required*
`kwargs`	`dict`	a dictionary containing supplementary parameters, like calculated dependencies.	`{}`

Returns:

Name	Type	Description
`dict`	`dict`	A dictionary containing the calculated values. The key to each array of values is "A", where A is from the list of keys.

57. Module `src.database.labeled.tabular.stat_calculations.median`

57.1 **mod** AI4SurrogateModelling.src.database.labeled.tabular.stat_calculations.median

Module containing functions related to calculating median values over tabular datasets in parallel.

https://en.wikipedia.org/wiki/Median

func calculate_statistic_median

```
calculate_statistic_median(
    *,
    tabular_dataset: DatabaseTabular,
    keys: list[str],
    **kwargs: dict
) -> dict
```

Calculates the median values for each database prefix key and its columns.

Must be called by ALL MPI processes.

Parameters:

Name	Type	Description	Default
`tabular_dataset`	`DatabaseTabular`	An instantiation of the tabular database.	*required*
`keys`	`list[str]`	A list of some of the database's prefix key for which to calculate the values.	*required*
`kwargs`	`dict`	a dictionary containing supplementary parameters, like calculated dependencies.	`{}`

Returns:

Name	Type	Description
`dict`	`dict`	A dictionary containing the calculated values. The key to each array of values is "A", where A is from the list of keys.

58. Module `src.database.labeled.tabular.stat_calculations.min`

58.1 **mod** AI4SurrogateModelling.src.database.labeled.tabular.stat_calculations.min

Module containing functions related to calculating minimal values over tabular datasets in parallel.

func calculate_statistic_min

```
calculate_statistic_min(
    *,
    tabular_dataset: DatabaseTabular,
    keys: list[str],
    **kwargs: dict
) -> dict
```

Calculates the minimal values for each database prefix key and its columns.

Must be called by ALL MPI processes.

Parameters:

Name	Type	Description	Default
`tabular_dataset`	`DatabaseTabular`	An instantiation of the tabular database.	*required*
`keys`	`list[str]`	A list of some of the database's prefix key for which to calculate the values.	*required*
`kwargs`	`dict`	a dictionary containing supplementary parameters, like calculated dependencies.	`{}`

Returns:

Name	Type	Description
`dict`	`dict`	A dictionary containing the calculated values. The key to each array of values is "A", where A is from the list of keys.

59. Module `src.database.labeled.tabular.stat_calculations.pca`

59.1 **mod** AI4SurrogateModelling.src.database.labeled.tabular.stat_calculations.pca

Contains functions used for Principal Component Analysis over tabular datasets in parallel.

https://en.wikipedia.org/wiki/Principal_component_analysis

func calculate_statistic_pca

```
calculate_statistic_pca(
    *,
    tabular_dataset: DatabaseTabular,
    keys: list[str],
    **kwargs: dict
) -> dict
```

Calculate PCA statistics per key in a tabular dataset.

Parameters:

Name	Type	Description	Default
`tabular_dataset`	`DatabaseTabular`	Tabular database instance to analyze.	*required*
`keys`	`list[str]`	Keys identifying independent feature groups to analyze.	*required*
`**kwargs`	`dict`	Supplementary parameters; expects `dependencies_calculated['covariance']` with per-key covariance matrices.	`{}`

Returns:

Name	Type	Description
`dict`	`dict`	Mapping `"{key}-{key}"` -> PCA results for each entry in `keys`. Each result contains: • `eigen-vectors`: Sorted eigenvectors of the covariance matrix. • `eigen-values`: Sorted eigenvalues of the covariance matrix. • `pca_max_error`: List of maximal reconstruction errors when using the first `k` principal components (index `k-1`).

60. Module `src.database.labeled.tabular.stat_calculations.percentile`

60.1 **mod** AI4SurrogateModelling.src.database.labeled.tabular.stat_calculations.percentile

Contains functions used for Percentile related calculations over tabular datasets in parallel.

https://en.wikipedia.org/wiki/Percentile

func calculate_statistic_percentile

```
calculate_statistic_percentile(
    *,
    tabular_dataset: DatabaseTabular,
    keys: list[str],
    **kwargs: dict
) -> dict
```

Calculates the corresponding percentile statistic for each column and each supplied key for the given tabular database.

Must be called by all MPI processes.

Parameters:

Name	Type	Description	Default
`tabular_dataset`	`DatabaseTabular`	An instantiation of the tabular database.	*required*
`keys`	`list[str]`	A list of some of the database's prefix key for which to calculate the analysis.	*required*
`kwargs`	`dict`	a dictionary containing supplementary parameters, like calculated dependencies or additional parameters.	`{}`

Returns:

Name	Type	Description
`dict`	`dict`	A dictionary containing the analysis. The key to each percentile is "A", where A is from the list of keys. Each entry is a numpy array.

61. Module `src.database.labeled.tabular.stat_calculations.ranks`

61.1 **mod** AI4SurrogateModelling.src.database.labeled.tabular.stat_calculations.ranks

Contains functions used for Rank determination over tabular datasets in parallel.

func calculate_statistic_ranks

```
calculate_statistic_ranks(
    *,
    tabular_dataset: DatabaseTabular,
    keys: list[str],
    **kwargs: dict
) -> dict
```

Calculates the rank value for each column in each database key.

Parameters:

Name	Type	Description	Default
`tabular_dataset`	`DatabaseTabular`	An instantiation of the tabular database.	*required*
`keys`	`list[str]`	A list of some of the database's prefix key for which to calculate the analysis.	*required*
`kwargs`	`dict`	a dictionary containing supplementary parameters, like calculated dependencies.	`{}`

Returns:

Name	Type	Description
`dict`	`dict`	A dictionary containing the analysis. The key to each rank ordering is "A", where A is from the list of keys. Each entry is a numpy array.

62. Module `src.database.labeled.tabular.stat_calculations.skewness`

62.1 **mod** AI4SurrogateModelling.src.database.labeled.tabular.stat_calculations.skewness

Module containing functions related to calculating standard deviations of values over tabular datasets in parallel.

https://en.wikipedia.org/wiki/Skewness

func calculate_statistic_skewness

```
calculate_statistic_skewness(
    *,
    tabular_dataset: DatabaseTabular,
    keys: list[str],
    **kwargs: dict
) -> dict
```

Calculates the Skewness coefficients for each database prefix key and its columns.

Must be called by ALL MPI processes.

Parameters:

Name	Type	Description	Default
`tabular_dataset`	`DatabaseTabular`	An instantiation of the tabular database.	*required*
`keys`	`list[str]`	A list of some of the database's prefix key for which to calculate the values.	*required*
`kwargs`	`dict`	a dictionary containing supplementary parameters, like calculated dependencies.	`{}`

Returns:

Name	Type	Description
`dict`	`dict`	A dictionary containing the calculated values. The key to each array of values is "A", where A is from the list of keys.

63. Module `src.database.labeled.tabular.stat_calculations.std`

63.1 **mod** AI4SurrogateModelling.src.database.labeled.tabular.stat_calculations.std

Module containing functions related to calculating standard deviations of values over tabular datasets in parallel.

https://en.wikipedia.org/wiki/Standard_deviation

func calculate_statistic_std

```
calculate_statistic_std(
    *,
    tabular_dataset: DatabaseTabular,
    keys: list[str],
    **kwargs: dict
) -> dict
```

Calculates the standard deviation values for each database prefix key and its columns.

Must be called by ALL MPI processes.

Parameters:

Name	Type	Description	Default
`tabular_dataset`	`DatabaseTabular`	An instantiation of the tabular database.	*required*
`keys`	`list[str]`	A list of some of the database's prefix key for which to calculate the values.	*required*
`kwargs`	`dict`	a dictionary containing supplementary parameters, like calculated dependencies.	`{}`

Returns:

Name	Type	Description
`dict`	`dict`	A dictionary containing the calculated values. The key to each array of values is "A", where A is from the list of keys.

64. Module `src.database.labeled.tabular.stat_calculations.variance`

64.1 **mod** AI4SurrogateModelling.src.database.labeled.tabular.stat_calculations.variance

Module containing functions related to calculating standard deviations of values over tabular datasets in parallel.

https://en.wikipedia.org/wiki/Variance

func calculate_statistic_variance

```
calculate_statistic_variance(
    *,
    tabular_dataset: DatabaseTabular,
    keys: list[str],
    **kwargs: dict
) -> dict
```

Calculates the variance values for each database prefix key and its columns.

Must be called by ALL MPI processes.

Parameters:

Name	Type	Description	Default
`tabular_dataset`	`DatabaseTabular`	An instantiation of the tabular database.	*required*
`keys`	`list[str]`	A list of some of the database's prefix key for which to calculate the values.	*required*
`kwargs`	`dict`	a dictionary containing supplementary parameters, like calculated dependencies.	`{}`

Returns:

Name	Type	Description
`dict`	`dict`	A dictionary containing the calculated values. The key to each array of values is "A", where A is from the list of keys.

65. Module `src.database.labeled.tabular.stat_calculations.zscore`

65.1 **mod** AI4SurrogateModelling.src.database.labeled.tabular.stat_calculations.zscore

Contains functions used for Z-Score related calculations over tabular datasets in parallel.

https://www.statisticshowto.com/probability-and-statistics/z-score/

func calculate_statistic_zscore

```
calculate_statistic_zscore(
    *,
    tabular_dataset: DatabaseTabular,
    keys: list[str],
    **kwargs: dict
) -> dict
```

Calculates the standard deviation values for each database prefix key and its columns.

Must be called by ALL MPI processes.

Parameters:

Name	Type	Description	Default
`tabular_dataset`	`DatabaseTabular`	An instantiation of the tabular database.	*required*
`keys`	`list[str]`	A list of some of the database's prefix key for which to calculate the values.	*required*
`kwargs`	`dict`	a dictionary containing supplementary parameters, like calculated dependencies.	`{}`

Returns:

Name	Type	Description
`dict`	`dict`	A dictionary containing the calculated values. The key to each array of values is "A", where A is from the list of keys.

66. Module `src.main.data`

66.1 **mod** AI4SurrogateModelling.src.main.data

Data related end point of the main script. Takes the relevant subparts of the configuration file and performs database related operations.

67. Module `src.main.model`

67.1 **mod** AI4SurrogateModelling.src.main.model

Model specification related end point of the main script. Takes the relevant subparts of the configuration file and configures model assemblies.

68. Module `src.main.training`

68.1 **mod** AI4SurrogateModelling.src.main.training

Model training related end point of the main script. Takes the relevant subparts of the configuration file and performs training related operations.

69. Module `src.main.utilization`

69.1 **mod** AI4SurrogateModelling.src.main.utilization

Model utilization related end point of the main script. Takes the relevant subparts of the configuration file and performs utilization related operations.

70. Module `src.model.graph`

70.1 **mod** AI4SurrogateModelling.src.model.graph

class Edge

```
Edge(
    *,
    vertex_src: int,
    vertex_tgt: int,
    layer: ENUM_TransferLayers
)
```

attr layer `instance-attribute`

```
layer = layer
```

attr vertex_src `instance-attribute`

```
vertex_src = vertex_src
```

attr vertex_tgt `instance-attribute`

```
vertex_tgt = vertex_tgt
```

class Vertex

```
Vertex(
    *,
    dim: int,
    activation: ENUM_Activations,
    bias: bool,
    name: str
)
```

attr activation `instance-attribute`

```
activation = activation
```

attr bias `instance-attribute`

```
bias = bias
```

attr dim `instance-attribute`

```
dim = dim
```

attr name `instance-attribute`

```
name = name
```

func contains_cycle

```
contains_cycle(
    *, neighborhood: list[list[int]], u: int, v: int
)
```

func get_outward_vertex_groups

```
get_outward_vertex_groups(
    *,
    edges_outward: list[set[int]],
    edges_inward: list[set[int]],
    source_vertex: int,
```

```
    target_vertex: int
) -> list[set[int]]
```

71. Module `src.model.model_assembler`

71.1 **mod** AI4SurrogateModelling.src.model.model_assembler

class ModelAssembler

```
ModelAssembler()
```

attr root_model `instance-attribute`

```
root_model = None
```

attr tree_calculated `instance-attribute`

```
tree_calculated = True
```

attr tree_edges `instance-attribute`

```
tree_edges = []
```

attr tree_vertices `instance-attribute`

```
tree_vertices = []
```

meth __getitem__

```
__getitem__(item)
```

meth add_model

```
add_model(
    *,
    parent: dict = None,
    unique_id: str,
    model_class: str,
    parameters=dict
)
```

meth add_transfer

```
add_transfer(
    *,
    unique_id: str,
    model_class: str,
    parameters: dict,
    parent: dict
)
```

meth assemble

```
assemble(*, dtype: ENUM_Dtypes)
```

meth assemble_tree

```
assemble_tree()
```

meth clear_assembled_data

```
clear_assembled_data()
```

meth get_parent_2_child_param_name

```
get_parent_2_child_param_name(vertex_index)
```

meth get_parent_id

```
get_parent_id(vertex_index)
```

meth get_vertex_id

```
get_vertex_id(vertex_index)
```

meth get_vertex_model_class

```
get_vertex_model_class(vertex_index)
```

meth get_vertex_model_class_parameters

```
get_vertex_model_class_parameters(vertex_index)
```

func convert_dictionary_to_tuple

```
convert_dictionary_to_tuple(value)
```

func convert_parameters_to_tuple

```
convert_parameters_to_tuple(parameters)
```

func convert_parent_to_tuple

```
convert_parent_to_tuple(parent)
```

func convert_tuple_to_dictionary

```
convert_tuple_to_dictionary(value)
```

func get_vertex_class

```
get_vertex_class(vertex)
```

func get_vertex_id

```
get_vertex_id(vertex)
```

func get_vertex_parameters

```
get_vertex_parameters(vertex)
```

func get_vertex_parent

```
get_vertex_parent(vertex)
```

72. Module `src.model.model_base`

72.1 **mod** AI4SurrogateModelling.src.model.model_base

Contains the base pytorch model class and functions related to general manipulation with pytorch models.

class ModelBase

```
ModelBase(**kwargs)
```

Bases: `Module`

The base pytorch model from which all other pytorch models should be derived.

attr config `instance-attribute`

```
config = parse_configuration_recursive(dict(kwargs))
```

meth forward `abstractmethod`

```
forward(*, data: dict) -> dict
```

meth get_biases_group_stats

```
get_biases_group_stats()
```

meth get_biases_groups

```
get_biases_groups()
```

meth get_biases_list

```
get_biases_list()
```

meth get_biases_list_stats

```
get_biases_list_stats()
```

meth get_gradient

```
get_gradient() -> Tensor
```

meth get_input_spec `abstractmethod`

```
get_input_spec() -> dict
```

meth get_n_biases

```
get_n_biases()
```

meth get_n_parameters

```
get_n_parameters()
```

meth get_n_weights

```
get_n_weights()
```

meth get_output_spec `abstractmethod`

```
get_output_spec() -> dict
```

meth get_parameters

```
get_parameters() -> Tensor
```

meth get_parameters_group_stats

```
get_parameters_group_stats()
```

meth get_parameters_groups

```
get_parameters_groups()
```

meth get_parameters_list

```
get_parameters_list()
```

meth get_parameters_list_stats

```
get_parameters_list_stats()
```

meth get_shape_input abstractmethod

```
get_shape_input()
```

meth get_shape_output abstractmethod

```
get_shape_output()
```

meth get_weights_group_stats

```
get_weights_group_stats()
```

meth get_weights_groups

```
get_weights_groups()
```

meth get_weights_list

```
get_weights_list()
```

meth get_weights_list_stats

```
get_weights_list_stats()
```

meth load_model staticmethod

```
load_model(*, fn: str)
```

meth parse_activation

```
parse_activation(activation_code: str) -> Module
```

meth parse_activation_string

```
parse_activation_string(activation_string)
```

meth save_model

```
save_model(*, fn: str)
```

class ModelConfiguration

A class used to parse and store model configuration parameters.

meth parse_configuration_recursive staticmethod

```
parse_configuration_recursive(obj: Any) -> Any
```

73. Module `src.model.models.aux`

73.1 **mod** AI4SurrogateModelling.src.model.models.aux

func parse_model

```
parse_model(model)
```

func unparse_model

```
unparse_model(model)
```

74. Module `src.model.models.classifiers.classifier_simple`

74.1 **mod** AI4SurrogateModelling.src.model.models.classifiers.classifier_simple

class ClassifierSimple

```
ClassifierSimple(layers_cardinality, dim_input, nclasses)
```

Bases: `Module`

attr model `instance-attribute`

```
model = Sequential(*layers)
```

meth forward

```
forward(x)
```

func __criterion

```
__criterion(X, Y, W)
```

func __criterion_F1

```
__criterion_F1(X, Y, W)
```

func __criterion_SE

```
__criterion_SE(X, Y, W)
```

func __criterion_accuracy

```
__criterion_accuracy(X, Y, W)
```

func __criterion_binary_cross_entropy

```
__criterion_binary_cross_entropy(X, Y, W)
```

func __criterion_cross_entropy

```
__criterion_cross_entropy(X, Y, W)
```

func __criterion_precision

```
__criterion_precision(X, Y, W)
```

func __criterion_recall

```
__criterion_recall(X, Y, W)
```

func __get_stats

```
__get_stats(model, dataloader, nclasses: int, desc: str)
```

For each class C[i], calculate the following values:

TP: how many of the predicted values marked as C[i] are also expected to be C[i]? FP: how many of the predicted values marked as C[i] are NOT expected to be C[i]?

TN: how many of the predicted values marked as NOT C[i] are NOT expected to be C[i]? FN: how many of the predicted values marked as NOT C[i] are indeed expected to be C[i]?

func classifier_train

```
classifier_train(
    model,
    training_progress: TrainingProgress,
    dataset: DatabaseTabular,
    learning_rate: float,
    l2_lambda: float,
    l1_lambda: float,
    nepochs: int,
    batch_size: int,
    train_ratio: float,
    validation_ratio: float,
    optimizer=None,
    scheduler=None,
)
```

func get_loss

```
get_loss(
    dataloader,
    model,
    l1_lambda: float,
    W,
    desc,
    epoch,
    optimizer=None,
    scheduler=None,
)
```

func get_model_biases

```
get_model_biases(model: Module)
```

func get_model_weights

```
get_model_weights(model: Module)
```

75. Module `src.model.models.model_wrapper`

75.1 **mod** AI4SurrogateModelling.src.model.models.model_wrapper

func parallel_model_class

```
parallel_model_class(model)
```

76. Module `src.model.models.autoencoders.ae_base`

76.1 **mod** AI4SurrogateModelling.src.model.models.autoencoders.ae_base

Contains modules and functions related to general work with auto encoders.

class AEBase

```
AEBase(**kwargs)
```

Bases: `ModelBase`

Serves as the parent class for all future autoencoding models.

meth decode `abstractmethod`

```
decode(x: Tensor) -> Tensor
```

Decodes the supplied latent embedding and returns the decoded data.

Parameters:

Name	Type	Description	Default
`x`	`Tensor`	Embedding to be decoded.	*required*

Returns:

Type	Description
`Tensor`	torch.Tensor: Decoded embeddings.

meth encode `abstractmethod`

```
encode(x: Tensor) -> Tensor
```

Encodes the supplied data and returns the latent embedding.

Parameters:

Name	Type	Description	Default
`x`	`Tensor`	Data to be encoded.	*required*

Returns:

Type	Description
`Tensor`	torch.Tensor: Encoded inputs.

meth forward `abstractmethod`

```
forward(data: dict) -> dict
```

Takes the provided data encodes them to the latent embedding space then decodes this latent representation to get the decoded data.

Returns:

Name	Type	Description
`dict`	`dict`	Contains a pair of values specified by the encoding and decoding keys.

77. Module `src.model.models.autoencoders.ae_simple`

77.1 **mod** AI4SurrogateModelling.src.model.models.autoencoders.ae_simple

Contains a simple autoencoding module for supervised learning on labeled data, and related functions.

class AESimple

```
AESimple(
    *, decoder: ModelBase, encoder: ModelBase, name: str
)
```

Bases: `AEBase`

The very basic version of an autoencoder. Takes as inputs two already constructed models, one for encoding the inputs, the other for decoding.

attr decoder `instance-attribute`

```
decoder = decoder
```

attr encoder `instance-attribute`

```
encoder = encoder
```

meth decode

```
decode(*, data_in: dict, data_out: dict = None) -> dict
```

Decodes the latent embeddings contained in the provided data dictionary and returns the results in another dictionary.

Parameters:

Name	Type	Description	Default
`data_in`	`dict`	A dictionary containing the encoded data.	*required*
`data_out`	`dict`	A dictionary containing the decoded data.	`None`

Returns:

Name	Type	Description
`dict`	`dict`	Contains the decoded data.

meth encode

```
encode(*, data_in: dict, data_out: dict = None) -> dict
```

Encodes the provided data and returns a dictionary containing the results.

Parameters:

Name	Type	Description	Default
`data_in`	`dict`	Contains data to be encoded.	*required*
`data_out`	`dict`	Contains data to be encoded.	`None`

Returns:

Name	Type	Description
`dict`	`dict`	Contains the encoded data.

`meth` forward

```
forward(*, data_in: dict, data_out: dict = None) -> dict
```

Takes the input data, encodes them, decodes the encodings and returns a dictionary containing both the encodings and decodings.

Parameters:

Name	Type	Description	Default
`data_in`	`dict`	A dictionary containing the data to be auto-encoded.	*required*
`data_out`	`dict`	A dictionary containing the data to be auto-encoded.	`None`

Returns:

Name	Type	Description
`dict`	`dict`	Contains the latent embeddings and the associated decoded values.

78. Module `src.model.models.autoencoders.cvae`

78.1 **mod** AI4SurrogateModelling.src.model.models.autoencoders.cvae

class VAEConditional

```
VAEConditional(
    in_channels: int,
    num_classes: int,
    latent_dim: int,
    hidden_dims: list = None,
    **kwargs
)
```

Bases: `VAEBase`

attr decoder `instance-attribute`

```
decoder = Sequential(*modules)
```

attr decoder_input `instance-attribute`

```
decoder_input = Linear(
    latent_dim + num_classes, hidden_dims[-1] * 4
)
```

attr embed_class `instance-attribute`

```
embed_class = Linear(num_classes, img_size * img_size)
```

attr embed_data `instance-attribute`

```
embed_data = Conv2d(in_channels, in_channels, kernel_size=1)
```

attr encoder `instance-attribute`

```
encoder = Sequential(*modules)
```

attr fc_mu `instance-attribute`

```
fc_mu = Linear(hidden_dims[-1] * 4, latent_dim)
```

attr fc_var `instance-attribute`

```
fc_var = Linear(hidden_dims[-1] * 4, latent_dim)
```

attr final_layer `instance-attribute`

```
final_layer = Sequential(
    ConvTranspose2d(
        hidden_dims[-1],
        hidden_dims[-1],
        kernel_size=3,
        stride=2,
        padding=1,
        output_padding=1,
    ),
    BatchNorm2d(hidden_dims[-1]),
    LeakyReLU(),
    Conv2d(
        hidden_dims[-1],
        out_channels=3,
        kernel_size=3,
        padding=1,
    ),
    Tanh(),
)
```

attr latent_dim `instance-attribute`

```
latent_dim = latent_dim
```

meth decode

```
decode(z: tensor) -> tensor
```

meth encode

```
encode(input: tensor) -> list[tensor]
```

meth forward

```
forward(input: Tensor, **kwargs) -> List[Tensor]
```

meth generate

```
generate(x: Tensor, **kwargs) -> Tensor
```

Given an input image x, returns the reconstructed image :param x:

(Tensor) [B x C x H x W] :return: (Tensor) [B x C x H x W]

meth loss_function

```
loss_function(*args, **kwargs) -> dict
```

meth reparameterize

```
reparameterize(mu: Tensor, logvar: Tensor) -> Tensor
```

Will a single z be enough ti compute the expectation for the loss??

:param mu: (Tensor) Mean of the latent Gaussian :param logvar: (Tensor) Standard deviation of the latent Gaussian :return:

meth sample

```
sample(
    num_samples: int, current_device: int, **kwargs
) -> Tensor
```

Samples from the latent space and return the corresponding image space map.

:param num_samples: (Int) Number of samples :param current_device: (Int) Device to run the model :return: (Tensor)

79. Module `src.model.models.autoencoders.vae_base`

79.1 **mod** AI4SurrogateModelling.src.model.models.autoencoders.vae_base

class VAEBase

```
VAEBase()
```

Bases: `Module`

meth decode

```
decode(input: tensor) -> tensor
```

meth encode

```
encode(input: tensor) -> tensor
```

meth forward `abstractmethod`

```
forward(*inputs: tensor) -> tensor
```

meth generate

```
generate(x: tensor, **kwargs) -> tensor
```

meth loss_function `abstractmethod`

```
loss_function(*inputs: tensor, **kwargs) -> tensor
```

meth sample

```
sample(
    batch_size: int, current_device: int, **kwargs
) -> tensor
```

80. Module `src.model.models.layers.bias`

80.1 **mod** AI4SurrogateModelling.src.model.models.layers.bias

Contains modules and functions related to a simple bias application to vector data.

class BiasLayer

```
BiasLayer(*, dtype: dtype, n: int, name: str)
```

Bases: `ModelBase`

Constructs a simple bias parameter.

Parameters:

Name	Type	Description	Default
`dtype`	`dtype`	datatype of the parameters	*required*
`n`	`int`	number of biases to construct	*required*
`name`	`str`	Internal name of this object, for logging and reporting purposes.	*required*

attr bias `instance-attribute`

```
bias = to(dtype)
```

meth forward

```
forward(x: Tensor) -> Tensor
```

Applies the bias to the input data.

81. Module `src.model.models.layers.fourier_features`

81.1 **mod** AI4SurrogateModelling.src.model.models.layers.fourier_features

82. Module `src.model.models.layers.gaussian_noise`

82.1 **mod** AI4SurrogateModelling.src.model.models.layers.gaussian_noise

class GaussianNoise

```
GaussianNoise(
    stds: Tensor, mus: Tensor, application_interval: int = 1
)
```

Bases: `Module`

attr application_interval `instance-attribute`

```
application_interval = application_interval
```

attr mus `instance-attribute`

```
mus = unsqueeze(0)
```

attr n `instance-attribute`

```
n = 0
```

attr stds `instance-attribute`

```
stds = unsqueeze(0)
```

meth forward

```
forward(x: Tensor)
```

83. Module `src.model.models.layers.residual`

83.1 **mod** AI4SurrogateModelling.src.model.models.layers.residual

Residual layer wrappers that inject skip connections around submodules.

class ResidualLayer

```
ResidualLayer(
    *,
    name: str = "Residual Layer",
    model: ModelBase,
    dtype: dtype = float32,
    object_uid: str
)
```

Bases: `ModelBase`

Wraps a submodule with a residual skip connection.

Initialize the residual wrapper.

Parameters:

Name	Type	Description	Default
`name`	`str`	Identifier passed to the base `ModelBase`.	`'Residual Layer'`
`model`	`ModelBase`	Submodule providing `input_key`, `output_key`, and shape accessors for residual alignment.	*required*

attr dim_in `instance-attribute`

```
dim_in = dim_in
```

attr dim_out `instance-attribute`

```
dim_out = dim_out
```

attr input_keys `instance-attribute`

```
input_keys = input_keys
```

attr model `instance-attribute`

```
model = unparse_model(model)
```

attr output_keys `instance-attribute`

```
output_keys = output_keys
```

attr residual_transform `instance-attribute`

```
residual_transform = to(dtype=dtype)
```

meth forward

```
forward(*, data_in: dict, data_out: dict = None) -> dict
```

Apply the wrapped module and add the residual projection.

Parameters:

Name	Type	Description	Default
`data_in`	`dict`	Mapping containing the tensor keyed by `input_key`.	*required*
`data_out`	`dict`	Optional output dictionary to populate.	`None`

Returns:

Name	Type	Description
`dict`	`dict`	Dictionary containing the residual output at `output_key`.

84. Module `src.model.models.other.mlp`

84.1 **mod** AI4SurrogateModelling.src.model.models.other.mlp

Contains modules and functions related to simple Multi-Layer Perceptron models.

class MultiLayerPerceptron

```
MultiLayerPerceptron(
    *,
    dtype: dtype,
    layer_vertices: list[int],
    layer_activations: list[str],
    layer_biases: list[bool],
    name: str,
    input_keys: list[str],
    output_keys: list[str],
    object_uid=str
)
```

Bases: `ModelBase`

The simplest architecture representing a Multi-Layer Perceptron (MLP). MLP transforms an input vector to an output vector via a sequence of non-linear transformations.

Constructs the MLP model.

Parameters:

Name	Type	Description	Default
`dtype`	`dtype`	datatype of the parameters	*required*
`layer_vertices`	`list[int]`	Specifies the number of neurons in each layer. The first entry has the dimension of the input vectors, the last entry has the dimension of the output vectors.	*required*
`layer_activations`	`list[str]`	Each layer can perform a point-wise function on the data. This list specifies these functions for each layer.	*required*
`layer_biases`	`list[bool]`	Holds True/False flags for each layer. When true, the data in that layer will be shifted before activation function application and forwarding the result to further layers.	*required*
`name`	`str`	Internal name of the model for logging and reporting.	*required*
`input_keys`	`list[str]`	A list of data keys on which this model should be applied.	*required*
`output_keys`	`list[str]`	Data keys which this model should output.	*required*

attr dim_in `instance-attribute`

```
dim_in = layer_vertices[0]
```

attr dim_out `instance-attribute`

```
dim_out = layer_vertices[-1]
```

attr input_keys `instance-attribute`

```
input_keys = input_keys
```

attr model `instance-attribute`

```
model = Sequential(*layer_list)
```

attr output_keys `instance-attribute`

```
output_keys = output_keys
```

meth forward

```
forward(*, data_in: dict, data_out: dict = None) -> dict
```

Takes the data and applies this model to the value with the internal input key. Saves the result into a dictionary with the internal output key.

Returns:

Name	Type	Description
`dict`	`dict`	Contains a single key-value.

meth get_shape_input

```
get_shape_input() -> int
```

Returns the input dimension of this model.

Returns:

Name	Type	Description
`int`	`int`	Input dimension of this model.

meth get_shape_output

```
get_shape_output() -> int
```

Returns the output dimension of this model.

Returns:

Name	Type	Description
`int`	`int`	Output dimension of this model.

85. Module `src.model.models.other.sequential`

85.1 **mod** AI4SurrogateModelling.src.model.models.other.sequential

class Sequential

```
Sequential(
    *, layers: list[ModelBase], name: str, object_uid: str
)
```

Bases: `ModelBase`

Constructs the Sequential model.

Parameters:

Name	Type	Description	Default
`layers`	`list[ModelBase]`	List of layers to be applied sequentially.	*required*
`name`	`str`	Internal name of the model for logging and reporting.	*required*

attr dim_in `instance-attribute`

```
dim_in = dim_in
```

attr dim_out `instance-attribute`

```
dim_out = dim_out
```

attr input_keys `instance-attribute`

```
input_keys = input_keys
```

attr layers `instance-attribute`

```
layers = ModuleList(
    [(unparse_model(model)) for model in layers]
)
```

attr output_keys `instance-attribute`

```
output_keys = output_keys
```

meth forward

```
forward(*, data_in: dict, data_out: dict = None) -> dict
```

Forward pass through the Sequential model.

Parameters:

Name	Type	Description	Default
`data_in`	`dict`	Input data dictionary.	*required*
`data_out`	`dict`	Output data dictionary.	`None`

Returns:

Name	Type	Description
`dict`	`dict`	Updated output data dictionary.

86. Module `src.model.models.other.structured`

86.1 **mod** AI4SurrogateModelling.src.model.models.other.structured

summary

class StructuredNet

```
StructuredNet(
    *,
    layer_vertices: list[Vertex],
    layer_edges: list[Edge],
    name: str
)
```

Bases: `ModelBase`

attr edges `instance-attribute`

```
edges = ParameterList(edges)
```

attr input_vertex `instance-attribute`

```
input_vertex = -1
```

attr inward_edge_index `instance-attribute`

```
inward_edge_index = {u: {} for u in (range(nvertices))}
```

attr nedges `instance-attribute`

```
nedges = 0
```

attr nvertices `instance-attribute`

```
nvertices = max([k for k, _ in (items())]) + 1
```

attr output_vertex `instance-attribute`

```
output_vertex = -1
```

attr outward_edge_index `instance-attribute`

```
outward_edge_index = {u: {} for u in (range(nvertices))}
```

attr vertex_activations `instance-attribute`

```
vertex_activations = [
    (Identity()) for _ in (range(nvertices))
]
```

attr vertex_bias `instance-attribute`

```
vertex_bias = ParameterList(vertex_bias)
```

attr vertex_dim `instance-attribute`

```
vertex_dim = [0 for _ in (range(nvertices))]
```

attr vertex_labels `instance-attribute`

```
vertex_labels = [None for _ in (range(nvertices))]
```

attr vertex_neighbours_inward instance-attribute

```
vertex_neighbours_inward = [
    (set()) for _ in (range(nvertices))
]
```

attr vertex_neighbours_outward instance-attribute

```
vertex_neighbours_outward = [
    (set()) for _ in (range(nvertices))
]
```

attr vertex_processing_groups instance-attribute

```
vertex_processing_groups = get_outward_vertex_groups(
    edges_outward=vertex_neighbours_outward,
    edges_inward=vertex_neighbours_inward,
    source_vertex=input_vertex,
    target_vertex=output_vertex,
)
```

meth forward

```
forward(x: Tensor)
```

meth get_shape_input

```
get_shape_input()
```

meth get_shape_output

```
get_shape_output()
```

87. Module `src.utilization.parameter_estimation.gradient_sampler`

87.1 **mod** AI4SurrogateModelling.src.utilization.parameter_estimation.gradient_sampler

class GradientParameterSampler

Utility helpers for gradient-based parameter identification.

These helpers take a trained model together with a collection of reference samples and search for admissible parameter vectors whose model outputs match the known data. Public entrypoints are intentionally thin wrappers around the private optimisation routine so they can be reused in different experiment scripts.

meth estimate_parameters_tabular `staticmethod`

```
estimate_parameters_tabular(
    *,
    ncandidates: int,
    nepochs: int,
    dtype: dtype,
    model_uid: str,
    optimizer_uid: str,
    scheduler_uid: str,
    loss_uid: str,
    database_uid: str,
    parameter_key: str,
    known_values: dict = {},
    report_dir: str
)
```

Searches for parameter vectors that reproduce known tabular samples.

The routine samples `ncandidates` random parameter vectors, clips them to the admissible bounds provided by the tabular database, and refines them jointly via `_refine_input_optimization`. Known input/output pairs are injected through `known_values`, while `parameter_key` selects which data block is considered optimisable. Once refined, the best candidates are exported to the requested report directory together with an HTML summary.

Parameters:

Name	Type	Description	Default
`ncandidates`	`int`	Number of parameter configurations to find.	*required*
`nepochs`	`int`	Maximal number of epochs in the optimization process.	*required*
`dtype`	`dtype`	Datatype of the parameters.	*required*
`model_uid`	`str`	Unique ID of the model to use for optimization.	*required*
`optimizer_uid`	`str`	Unique ID of the optimizer to be used.	*required*
`scheduler_uid`	`str`	Unique ID of the Scheduler to be used.	*required*
`loss_uid`	`str`	Unique ID of the loss to be used.	*required*
`database_uid`	`str`	Unique ID of the database containing training data.	*required*
`parameter_key`	`str`	Tells which part of the database corresponds to the parameters to be optimized.	*required*
`known_values`	`dict[str, list[float]]`	User prescribed known values to fill in/augment the search space.	`{}`
`report_dir`	`str`	Where to store the HTML report file.	*required*

func export_results2csv

```
export_results2csv(
    *,
    labels: list[str],
    data: ndarray,
    fn: str,
    delimiter: str
)
```

Takes the user provided data and labeling and exports it to a comma separated file using the supplied delimitting character.

Parameters:

Name	Type	Description	Default
`fn`	`str`	Target filename.	*required*
`delimiter`	`str`	Delimiting character.	*required*
`data`	`ndarray`	Data array.	*required*
`labels`	`list[str]`	Labels.	*required*

88. Module `src.utilization.parameter_estimation.random_sampler`

88.1 **mod** AI4SurrogateModelling.src.utilization.parameter_estimation.random_sampler

class RandomParameterSampler

meth estimate_vertices_random `staticmethod`

```
estimate_vertices_random(
    *,
    ntests: int,
    dtype: ENUM_Dtypes,
    model: ModelBase,
    vertex_coordinates: ndarray,
    vertex_labels_local2global_map,
    **kwargs
)
```

meth sample_simplices `staticmethod`

```
sample_simplices(
    *,
    ntests: int,
    dtype: ENUM_Dtypes,
    model: ModelBase,
    simplex_coordinates: ndarray,
    simplex_vertex_labels_local2global_map,
    nsamples: int
)
```

meth sample_simplices_interpolation `staticmethod`

```
sample_simplices_interpolation(
    *,
    ntests: int,
    dtype: ENUM_Dtypes,
    model: ModelBase,
    simplex_coordinates: ndarray,
    simplex_vertex_labels_local2global_map,
    nsamples: int
)
```

89. Module `src.runtime_state.aux`

89.1 **mod** AI4SurrogateModelling.src.runtime_state.aux

A module containing auxilliary functions for the runtime state class.

attr dtype_map `module-attribute`

```
dtype_map = {
    "float16": float16,
    "float32": float32,
    "float64": float64,
    "int32": int32,
    "int64": int64,
    "bool": bool,
}
```

func is_object_lazy

```
is_object_lazy(obj: Any) -> bool
```

If the object has been decorated with the mark_lazy_function decorator, returns True.

Parameters:

Name	Type	Description	Default
`obj`	`Any`	Object to be checked.	*required*

Returns:

Name	Type	Description
`bool`	`bool`	True if lazy, False otherwise.

func mark_lazy_function

```
mark_lazy_function(obj: Any) -> Any
```

Adds an attribute to the provided object, for future checking in runtime state.

Parameters:

Name	Type	Description	Default
`obj`	`Any`	Object for which to add the attribute	*required*

Returns:

Name	Type	Description
`Any`	`Any`	The wrapped function

func parse_parameter

```
parse_parameter(value: Any) -> Any
```

Takes the input and if it is a string representing a tuple or a list, returns a tuple, or list containing integer, float or string values. Otherwise returns the input.

Parameters:

Name	Type	Description	Default
`value`	`Any`	An object to be parsed.	*required*

Returns:

Name	Type	Description
`Any`	`Any`	A list, tuple, string or the input.

90. Module `src.runtime_state.parameters`

90.1 **mod** AI4SurrogateModelling.src.runtime_state.parameters

A module containing classes and functions related to parametrization and sampling from the parameter space.

class Parametrization

```
Parametrization(parameter_key: str, parameter_code: str)
```

A simple wrapper class around parameters passed to the runtime through the configuration files. Its purpose is to decode the parameter key and construct auxilliary structure for efficient random sampling within the supplied space. The parameters can be either:

```
- a range or a union of ranges, defined with the keyword        PARAMETER_RANGE and followed by the list of intervals, i.e.
    "PARAMETER_RANGE: [(a0, b0), (a1, b1), (a2, b2)]"
    The constructor post-processes the ranges and joins intervals which            are overlapping or touching.

- a set of choices, defined with the keyword PARAMETER_CHOICE and        followed by a list of values, i.e.
    "PARAMETER_CHOICE: [a, b, c]"
```

Post processes the supplied strings and constructs all auxilliary structures.

Parameters:

Name	Type	Description	Default
`parameter_key`	`str`	Key of the parameter to be fererenced down the pipeline.	*required*
`parameter_code`	`str`	A string representation of the parameter value.	*required*

meth sample

```
sample(value: Any = None) -> Any
```

Performs random sampling. If the user supplies the value, raises an error if the value is not correct.

Parameters:

Name	Type	Description	Default
`value`	`Any`	Value to be checked. Defaults to None.	`None`

Returns:

Name	Type	Description
`Any`	`Any`	The sampled value.

91. Module `src.runtime_state.runtime_state`

91.1 **mod** AI4SurrogateModelling.src.runtime_state.runtime_state

Contains classes and functions related to the state of the runtime. Serves as a bridge between the provided configurations and the instantiated objects/actions to be taken.

class Action

```
Action(
    *,
    object_uid: str = None,
    f: Any,
    parameters: dict[str, Any] = {},
    action_type: str,
    lazy: bool
)
```

Serves as a wrapper around an action to be taken. An action can be an object initialization or a function call.

Depending on which inputs are None, determines what action should be performed.

- Object initialization and a member function call?
- A function call?

The supplied parameters are used to override the default parameters of the object construction/function call.

Parameters:

Name	Type	Description	Default
`object_uid`	`str`	An object with the function f as an attribute. Defaults to None.	`None`
`f`	`Any`	The function to be called, if object_uid refers to a configured object, calls the function as the member of the object class.	*required*
`parameters`	`dict[str, Any]`	Parameters to be passed into the function call, this overrides the default parameters supplied elsewhere in the configuration hierarchy. Defaults to {}.	`{}`
`action_type`	`str`	the type of action to be defined, possible values: - 'invalid' - 'class.static_function(**parameters)' - 'global_function(**parameters)' - 'member_function(object, **parameters)'	*required*
`lazy`	`bool`	A flag determining whether the associated objects should be initialized automatically before (False) the function call or explicitly (True) by the called function.	*required*

attr action_type `instance-attribute`

```
action_type = action_type
```

attr f `instance-attribute`

```
f = f
```

attr lazy `instance-attribute`

```
lazy = lazy
```

attr object_class `instance-attribute`

```
object_class = get_object_configuration(
    object_uid=object_uid
)
```

attr object_uid `instance-attribute`

```
object_uid = object_uid
```

attr parameters `instance-attribute`

```
parameters = parameters
```

meth __str__

```
__str__() -> str
```

Returns a string representation of this object.

Returns:

Name	Type	Description
`str`	`str`	A string containing relevant information about this object in a nicely formatted manner.

meth call

```
call(
    *, parameters: dict[str, Any] = {}, reinit: bool
) -> Any
```

Calls the function. The supplied parameters override the member parameters.

Parameters:

Name	Type	Description	Default
`parameters`	`dict[str, Any]`	Explicit override of the function call parameters to be used instead of the default ones.	`{}`

Returns:

Name	Type	Description
`Any`	`Any`	An instance of the class.

meth is_parametrized

```
is_parametrized(parameters: dict[str, Any] = {}) -> bool
```

Determines if at least one member parameter is to be sampled. If the user provides the override parameters, then the method considers the overriden parametrization.

Parameters:

Name	Type	Description	Default
`parameters`	`dict[str, Any]`	Goes through the member parameters and if any key is also contained the the supplied dictionary, uses the supplied value. Defaults to {}.	`{}`

Returns:

Name	Type	Description
`bool`	`bool`	True if at least one parameter is to be sampled, False otherwise.

`class` ObjectParametrization

```
ObjectParametrization(
    *, obj: Any, parameters: dict[str, Any], object_uid: str
)
```

Serves as a wrapper around a class and its parametrization. Handles the object initialization and parameter sampling (if the parameters are to be sampled).

Stores the object and its parametrization for future use. Processes the parameters for efficient future use. Args: obj (Any): A class representing the object to be initialized. parameters (dict[str, Any]): Object parametrization.

`attr` obj `instance-attribute`

```
obj = obj
```

`attr` object_uid `instance-attribute`

```
object_uid = object_uid
```

`attr` parameters `instance-attribute`

```
parameters = parameters
```

`meth` __str__

```
__str__() -> str
```

Returns a string representation of this object.

Returns:

Name	Type	Description
`str`	`str`	A string containing relevant information about this object in a nicely formatted manner.

`meth` get_object_class

```
get_object_class() -> Any
```

Retrieves the class to be initialized.

Returns:

Name	Type	Description
`Any`	`Any`	This object's class.

`meth` get_object_parameters

```
get_object_parameters() -> dict
```

Retrieves the parameters to be used in construction of this object.

Returns:

Name	Type	Description
`dict`	`dict`	Parameters to be passed to the constructor.

`meth` init

```
init(
    *, parameters: dict[str, Any] = {}, reinit: bool
) -> Any
```

Initializes a new instance of the class with parameters sampled from a uniform random distribution (if any parameters are to be sampled). The supplied parameters override the member parameters.

Parameters:

Name	Type	Description	Default
`parameters`	`dict[str, Any]`	Explicit override of the constructor parameters to be used instead of the default ones.	`{}`

Returns:

Name	Type	Description
`Any`	`Any`	An instance of the class.

`meth` is_parametrized

```
is_parametrized(parameters: dict[str, Any] = {}) -> bool
```

Determines if at least one member parameter is to be sampled. If the user provides the override parameters, then the method considers the overriden parametrization.

Parameters:

Name	Type	Description	Default
`parameters`	`dict[str, Any]`	Goes through the member parameters and if any key is also contained the the supplied dictionary, uses the supplied value. Defaults to {}.	`{}`

Returns:

Name	Type	Description
`bool`	`bool`	True if at least one parameter is to be sampled, False otherwise.

`class` runtime_state

Contains the following static objects which are referenced during the program execution:

```
- object_configurations: holds the parameters used in constructors of       all objects.
- objects: holds all the instantiated objects.
- parametrized_object_uids: a list of unique identifiers of objects           requiring randomg parameter sampling
- actions: for each object, holds a sequence of actions to be performed           with this object
```

Furthermore, contains the global configuration object related to this runtime.

`attr` actions `class-attribute` `instance-attribute`

```
actions = []
```

attr object_configurations class-attribute instance-attribute

```
object_configurations = {}
```

attr objects class-attribute instance-attribute

```
objects = {}
```

attr operations class-attribute instance-attribute

```
operations = []
```

attr parametrized_object_uids class-attribute instance-attribute

```
parametrized_object_uids = []
```

meth add_action staticmethod

```
add_action(
    *, object_uid: str, fname: str, parameters: dict
)
```

Adds an action associated with the specified object. The action is a class member function to be called with the specified parameters.

Parameters:

Name	Type	Description	Default
`object_uid`	`str`	A unique identifier specifying the object.	*required*
`fname`	`str`	A name of the function to be called.	*required*
`parameters`	`dict`	Parameters to override the default function parameters to the member function.	*required*

meth add_object_configuration staticmethod

```
add_object_configuration(
    *, key: str, constructor: str, parameters: dict
)
```

Given the object unique identifier, stores the class and parameters required for initialization.

Parameters:

Name	Type	Description	Default
`key`	`str`	A unique identifier of the object.	*required*
`constructor`	`str`	Name of the class representing the object or a function returing an instance of an object.	*required*
`parameters`	`dict`	Parameters for initialization.	*required*

Raises:

Type	Description
`ValueError`	If the object is already stored, raises this error.

meth add_operation staticmethod

```
add_operation(operation: dict)
```

Adds an operation dictionary to the runtime.

Parameters:

Name	Type	Description	Default
`operation`	`dict`	Contains two keys 'objects' and 'actions'. - 'objects': a list of objects UIDs to be constructed in the operation. - 'actions': a list of integers, indices of actions to be taken.	*required*

meth check_object_dependencies `staticmethod`

```
check_object_dependencies()
```

Analyzes the configured objects and actions and builds a hierarchy graph and ordering in which the objects should be initialized and actions called.

meth clear_parametrized_objects `staticmethod`

```
clear_parametrized_objects()
```

Goes through the objects which were constructed with at least one sampled parameter and frees them from memory.

meth construct_object `staticmethod`

```
construct_object(
    *,
    object_uid: str,
    reinit: bool = False,
    override_parameters: dict = {}
) -> Any
```

Takes the provided object_uid and constructs a new object referred to by that UID. If an object already exists, deletes it and uses the new object.

Parameters:

Name	Type	Description	Default
`object_uid`	`str`	A UID of the object to be created.	*required*
`reinit`	`bool`	If True, reinitializes the object even though its already initialized. Defaults to True.	`False`
`override_parameters`	`dict`	a list of parameters which should be added to/override the parameters provided by the configuration files.	`{}`

Returns:

Name	Type	Description
`Any`	`Any`	Returns the instantion of the object.

meth contains_uid `staticmethod`

```
contains_uid(key: str) -> bool
```

Determines whether the unique ID is already being used.

Parameters:

Name	Type	Description	Default
`key`	`str`	A string representing the unique identifier of an object.	*required*

Returns:

Name	Type	Description
`bool`	`bool`	True if present, otherwise False.

meth delete_object staticmethod

```
delete_object(object_uid: str) -> None
```

Deletes the instance of the object with the specified UID.

Parameters:

Name	Type	Description	Default
`object_uid`	`str`	UID of the object to be deleted.	*required*

meth get_n_actions staticmethod

```
get_n_actions() -> int
```

Returns the number of currently defined actions.

Returns:

Name	Type	Description
`int`	`int`	An integer >= 0.

meth get_object staticmethod

```
get_object(object_uid: str) -> Any
```

Attempts to retrieve an initialized instance of the object with the provided unique identifier. If the object is not initialized, returns None.

Parameters:

Name	Type	Description	Default
`object_uid`	`str`	A unique identifier referencing the intialized object.	*required*

Returns:

Name	Type	Description
`Any`	`Any`	Returns the object or None if it does not exist.

meth get_object_configuration staticmethod

```
get_object_configuration(
    object_uid: str,
) -> ObjectParametrization
```

Retrieves the object class with the given unique identifier.

Parameters:

Name	Type	Description	Default
`object_uid`	`str`	A string representing the unique identifier of the object to be retrieved.	*required*

Raises:

Type	Description
`RuntimeError`	If the unique identifier does not exist.

Returns:

Name	Type	Description
`ObjectParametrization`	`ObjectParametrization`	The class representing the queried object.

`meth` get_object_dependencies `staticmethod`

```
get_object_dependencies(object_uid: str) -> list
```

Retrieves the list of objects and actions to be taken before the supplied object can be initialized.

Parameters:

Name	Type	Description	Default
`object_uid`	`str`	A unique identifier referencing the object to be analyzed.	*required*

Returns:

Name	Type	Description
`list`	`list`	Contains object UIDs (str) and Action indices (int)

`meth` get_objects_from_parameters `staticmethod`

```
get_objects_from_parameters(parameters: Any) -> list[str]
```

`meth` perform_action `staticmethod`

```
perform_action(
    *, action_idx: int, parameters: dict = {}, reinit: bool
)
```

Performs a chosen action specified by its index.

Parameters:

Name	Type	Description	Default
`action_idx`	`int`	Index of the action to be performed.	*required*
`parameters`	`dict`	Parameters to override the parameters supplied in the action configuration. Defaults to {}.	`{}`

92. Module `src.sensitivity_analysis.goniometric`

92.1 **mod** AI4SurrogateModelling.src.sensitivity_analysis.goniometric

func construct_system

```
construct_system(basis_values, target_values)
```

func perform_sensitivity_analysis_Goniometric

```
perform_sensitivity_analysis_Goniometric(
    dataset: DatabaseTabular,
)
```

93. Module `src.sensitivity_analysis.gradient`

93.1 **mod** AI4SurrogateModelling.src.sensitivity_analysis.gradient

func gradient_sensitivity

```
gradient_sensitivity(
    *,
    dataloader: DictionaryDataLoader,
    model: ModelBase,
    labels: dict[str, list[str]]
) -> tuple[list[list[float]], list[str], list[str]]
```

Calculates mean absolute gradients of outputs w.r.t. individual inputs.

Iterates through a dataloader, performing a backward pass per output column (per output key) to extract dY/dX sensitivities. The resulting matrix is aggregated across MPI workers by averaging absolute gradients.

Parameters:

Name	Type	Description	Default
`dataloader`	`DictionaryDataLoader`	Source of evaluation batches.	*required*
`model`	`ModelBase`	Model whose gradients are analysed.	*required*
`labels`	`dict[str, list[str]]`	Mapping from key to per-column labels used to label the resulting matrix axes.	*required*

Returns:

Type	Description
`list[list[float]]`	tuple[list[list[float]], list[str], list[str]]: Matrix of sensitivities
`list[str]`	plus row/column labels.

94. Module `src.sensitivity_analysis.morris`

94.1 **mod** AI4SurrogateModelling.src.sensitivity_analysis.morris

func morris_sensitivity

```
morris_sensitivity(
    *,
    nsamples: int,
    nlevels: int,
    model: ModelBase,
    labels: dict,
    bounds: dict,
    dtype: dtype
) -> dict
```

Evaluate Morris screening indices for the provided model outputs.

The function defines a SALib problem from the model metadata, generates Morris trajectories, pushes them through the model without gradient tracking, and post-processes the results into per-output `mu_star` values.

Parameters:

Name	Type	Description	Default
`nsamples`	`int`	Number of trajectories (N) used by the Morris sampler.	*required*
`nlevels`	`int`	Discretization levels of the Morris grid.	*required*
`model`	`ModelBase`	Model that maps dictionary inputs to dictionary outputs.	*required*
`labels`	`dict`	Mapping from input/output keys to readable component labels used in SALib and the final report.	*required*
`bounds`	`dict`	Mapping from input keys to `(n, 2)` tensors containing lower and upper bounds for every labeled component.	*required*
`dtype`	`dtype`	Torch dtype used for tensors created from the sampled values.	*required*

Returns:

Type	Description
`dict`	tuple[list[list[float]], list[str], list[str]]: Morris `mu_star` values arranged per output column then transposed per parameter, the flat list of input names, and the flat list of output labels. Non-root MPI ranks return `(None, None, None)`.

95. Module `src.sensitivity_analysis.polynomial`

95.1 **mod** AI4SurrogateModelling.src.sensitivity_analysis.polynomial

class PolynomialModelPytorch

```
PolynomialModelPytorch(coefficients, exponents)
```

Bases: `Module`

attr coefficients `instance-attribute`

```
coefficients = Parameter(unsqueeze(0))
```

attr exponents `instance-attribute`

```
exponents = unsqueeze(0)
```

meth forward

```
forward(x)
```

func construct_system

```
construct_system(exponents, source_values, target_values)
```

func evaluate_polynomial

```
evaluate_polynomial(exponents, x)
```

func get_polynomial_degrees

```
get_polynomial_degrees(n, d)
```

Generate a vector of polynomial degrees for all monomials in n dimensions with total degree at most d.

Parameters: n (int): Number of dimensions (variables). d (int): Maximum total degree.

Returns: numpy.ndarray: Vector containing the degrees of all monomials.

func get_sensitivities

```
get_sensitivities(sol, exponents, x, y)
```

func get_solution_error

```
get_solution_error(sol, A, b, exponents, x, y)
```

func perform_sensitivity_analysis_Polynomial

```
perform_sensitivity_analysis_Polynomial(
    dataset: DatabaseTabular,
)
```

96. Module `src.sensitivity_analysis.sobol`

96.1 **mod** AI4SurrogateModelling.src.sensitivity_analysis.sobol

func sobol_sensitivity

```
sobol_sensitivity(
    *,
    nsamples: int,
    model: ModelBase,
    labels: dict,
    bounds: dict,
    dtype: dtype
) -> dict
```

Compute first-order Sobol indices for each model output.

The function builds a SALib problem definition from the provided model metadata, samples the corresponding input space via the Saltelli sampler, evaluates the supplied model without gradient tracking, and finally runs a Sobol analysis per output column.

Parameters:

Name	Type	Description	Default
`nsamples`	`int`	Number of draws used by the Saltelli sampler.	*required*
`model`	`ModelBase`	Model that consumes dictionary inputs and produces dictionary outputs.	*required*
`labels`	`dict`	Mapping from each input/output key to the ordered labels describing its scalar components.	*required*
`bounds`	`dict`	Mapping from each input key to a `(n, 2)` tensor that holds the lower/upper bound for every labeled component.	*required*
`dtype`	`dtype`	Torch dtype used for constructing the sampled tensors.	*required*

Returns:

Type	Description
`dict`	tuple[list[list[float]], list[str], list[str]]: First-order Sobol indices per output column, the flattened list of input names, and the flattened list of output labels. All MPI ranks different from zero return `(None, None, None)`.

97. Module `src.training.schedulers.lambdas.cyclical_lr`

97.1 **mod** AI4SurrogateModelling.src.training.schedulers.lambdas.cyclical_lr

Contains modules representing various lambda function based on cyclical behaviour.

class CyclicalLR

```
CyclicalLR(
    *,
    log_factor_low: float = -1,
    log_factor_high: float = 1,
    periods: list
)
```

A basic cyclical learning rate scheduler utilizing user provided periods and learning rate ranges. Designed to be used by the LambdaLR scheduler provided by pytorch.

Initializes the cyclical learning rate scheduler. The log10 learning rate multiplier ranges between [log_factor_low, log_factor_high] with nested cyclical behaviour with periods specified by the user.

Parameters:

Name	Type	Description	Default
`periods`	`list`	A list of integers specifying the periodical behaviour of the scheduler.	*required*
`log_factor_low`	`float`	Minimal log10 learning rate multiplier. Defaults to -1.	`-1`
`log_factor_high`	`float`	Maximal log10 learning rate multiplier. Defaults to 1.	`1`

attr log_factor_high `instance-attribute`

```
log_factor_high = log_factor_high / len(periods)
```

attr log_factor_low `instance-attribute`

```
log_factor_low = log_factor_low / len(periods)
```

attr periods `instance-attribute`

```
periods = periods
```

meth __call__

```
__call__(step: int) -> float
```

Given the training step, calculates the cyclical learning rate multiplier based on the internal multiplier range and periodical behaviour.

Parameters:

Name	Type	Description	Default
`step`	`int`	Index of the epoch.	*required*

Returns:

Name	Type	Description
`float`	`float`	Learning rate multiplier in range 10**[log_factor_low, log_factor_high]

98. Module `src.training.trainers.feed_forward`

98.1 **mod** AI4SurrogateModelling.src.training.trainers.feed_forward

Contains modules and functions used for training simple feedforward networks.

func get_model_properties

```
get_model_properties(
    model: ModelBase,
    loss: LossBase,
    database,
    dataloader_train,
    dataloader_test,
    dataloader_validation,
) -> dict
```

Collects diagnostics describing the trained model.

Builds a dictionary that contains L1-error histograms, cumulative error curves, weight/bias distributions and gradient-based sensitivity matrices. All dataloaders are evaluated in `eval()` mode, so no gradients are tracked.

Parameters:

Name	Type	Description	Default
`model`	`ModelBase`	Fully initialised model to analyse.	*required*
`loss`	`LossBase`	Loss used during training (for matching outputs to targets).	*required*
`database`	`DatabaseTabular`	Database object used for label/metadata lookups.	*required*
`dataloader_train`	`DictionaryDataLoader`	Training dataloader providing reference samples.	*required*
`dataloader_test`	`DictionaryDataLoader`	Test dataloader providing reference samples.	*required*
`dataloader_validation`	`DictionaryDataLoader`	Validation dataloader providing reference samples.	*required*

Returns:

Name	Type	Description
`dict`	`dict`	Nested data structure describing distributions/sensitivities.

func one_test_run

```
one_test_run(
    *,
    number_of_epochs: int,
    dataloader_train: DictionaryDataLoader,
    dataloader_test: DictionaryDataLoader,
    dataloader_validation: DictionaryDataLoader,
    optimizer: Optimizer,
    scheduler,
    model: ModelBase,
    loss: LossBase,
    train_manager: TrainingManager
) -> None
```

Runs a full training loop with metric logging and dashboards.

Parameters:

Name	Type	Description	Default
`number_of_epochs`	`int`	Number of epochs to train.	*required*
`dataloader_train`	`DictionaryDataLoader`	Training dataloader.	*required*
`dataloader_test`	`DictionaryDataLoader`	Test dataloader.	*required*
`dataloader_validation`	`DictionaryDataLoader`	Validation dataloader.	*required*
`optimizer`	`Optimizer`	Optimizer applied to model parameters.	*required*
`scheduler`	`_LRScheduler`	Scheduler advanced after each epoch/batch sequence.	*required*
`model`	`ModelBase`	Model instance to train.	*required*
`loss`	`LossBase`	Loss instance used for optimisation and metric aggregation.	*required*
`train_manager`	`TrainingManager`	Persistent state tracking checkpoints and histories.	*required*

`func` refine

```
refine(
    *,
    report_dir: str = None,
    checkpoint_dir: str,
    number_of_epochs: int,
    batch_size: int,
    dtype: dtype,
    data_uid: str,
    model_uid: str,
    optimizer_uid: str,
    scheduler_uid: str,
    loss_uid: str,
    reset: bool = False
) -> tuple
```

Constructs training components, runs one training cycle, and reports.

Because the optimizer depends on model parameters and the scheduler depends on the optimizer, this function is marked lazy—runtime state instantiates the components only when the function executes.

Parameters:

Name	Type	Description	Default
`report_dir`	`str | None`	Directory where the HTML training report is stored.	`None`
`checkpoint_dir`	`str`	Directory that stores checkpoints/progress metadata.	*required*
`number_of_epochs`	`int`	Number of epochs to train in the single run.	*required*
`batch_size`	`int`	Batch size used by the dataloaders.	*required*
`dtype`	`dtype`	Torch dtype for model parameters and datasets.	*required*
`data_uid`	`str`	Runtime-state identifier for the database object.	*required*
`model_uid`	`str`	Identifier for the model to instantiate.	*required*
`optimizer_uid`	`str`	Identifier for the optimizer to instantiate.	*required*
`scheduler_uid`	`str`	Identifier for the scheduler to instantiate.	*required*
`loss_uid`	`str`	Identifier for the loss function.	*required*
`reset`	`bool`	If True, wipes existing checkpoints before training.	`False`

Returns:

Name	Type	Description
`tuple`	`tuple`	`(criterion, best_model_train, best_model_validation)` as
	`tuple`	produced by downstream routines.

`func` simple

```
simple(
    *,
    checkpoint_dir: str,
    number_of_inner_tests: int = 1,
    number_of_epochs: int,
    batch_size: int,
    dtype: dtype,
    data_uid: str,
    model_uid: str,
    optimizer_uid: str,
    scheduler_uid: str,
    loss_uid: str
) -> tuple
```

Runs one or more fresh training sessions for statistical evaluation.

Parameters:

Name	Type	Description	Default
`checkpoint_dir`	`str`	Directory used to store checkpoints and histories.	*required*
`number_of_inner_tests`	`int`	How many fresh trainings to launch sequentially.	`1`
`number_of_epochs`	`int`	Number of epochs per training run.	*required*
`batch_size`	`int`	Size of batches fed to dataloaders.	*required*
`dtype`	`dtype`	Torch dtype used when creating dataloaders/models.	*required*
`data_uid`	`str`	Runtime-state identifier for the database to construct loaders.	*required*
`model_uid`	`str`	Identifier for the model factory.	*required*
`optimizer_uid`	`str`	Identifier for the optimizer factory.	*required*
`scheduler_uid`	`str`	Identifier for the scheduler factory.	*required*
`loss_uid`	`str`	Identifier for the loss function to instantiate.	*required*

Returns:

Name	Type	Description
`tuple`	`tuple`	Whatever `runtime_state.construct_object` returns for downstream
	`tuple`	consumption (historically criterion + model references).

99. Module `src.training.loss.loss_base`

99.1 `mod` AI4SurrogateModelling.src.training.loss.loss_base

Contains modules and functions related to the base class of various losses to be used during model training.

`class` LossBase

```
LossBase(
    *,
    model_parameters: dict = None,
    outputs: dict = None,
    derivatives: dict = None
)
```

A base class containing basic functionality and configuration for loss calculations. All loss classes should derive from this class.

The base loss class has the following capabilities: - loss on predicted values, model weights, model biases

`attr` coefficients `instance-attribute`

```
coefficients = {}
```

`attr` loss_keys `class-attribute` `instance-attribute`

```
loss_keys = {'mae', 'mse', 'rmae', 'rmse'}
```

`meth` __call__

```
__call__(
    *,
    model: ModelBase = None,
    data: dict = None,
    predictions: dict = None,
    total_data_size: int = 1,
    number_of_batches: int = 1
) -> Tensor
```

Goes through all provided loss dictionaries and calculates up-to date losses scaled by the user provided factors.

Parameters:

Name	Type	Description	Default
`model`	`ModelBase`	Model for which to calculate the metrics.	`None`
`data`	`dict`	Data provided by the dataloader, usually contains inputs and expected outputs.	`None`
`predictions`	`predictions`	Data provided via the model evaluating the input data, i.e. predictions.	`None`
`total_data_size`	`int`	total number of data entries over all MPI processes and across all batches.	`1`
`number_of_batches`	`int`	total number of batches.	`1`

Returns:

Type	Description
`Tensor`	torch.Tensor: A single differentiable scalar value.

`meth` get_accumulated_raw_losses

```
get_accumulated_raw_losses() -> dict
```

Goes through all the partial loss results, sums them up across all MPI processes and returns a dictionary containing non-nested loss measurements same on all MPI processes.

Returns:

Name	Type	Description
`dict`	`dict`	A non-nested dictionary containing all losses.

`meth` get_criterion

```
get_criterion() -> Tensor
```

Returns the UNSCALED loss associated with the prediction losses. Useful for measuring the real properties of the model.

Returns:

Type	Description
`Tensor`	torch.Tensor: Non-differentiable sum of all losses associated with the predictions and expectations.

`meth` get_total_criterion

```
get_total_criterion() -> float
```

Computes the total sum of all raw losses related to the predictions category and returns it to the user.

Returns:

Name	Type	Description
`float`	`float`	Total criterion summed over all MPI processes.

`meth` get_total_scaled_loss

```
get_total_scaled_loss() -> float
```

Computes the total sum of all scaled losses and returns it to the user.

Returns:

Name	Type	Description
`float`	`float`	Total loss summed over all MPI processes.

`meth` what_requires_gradients

```
what_requires_gradients() -> dict
```

If any loss requires the calculation of the partial derivatives w.r.t. to any of the data, returns a dictionary containing those data keys.

Returns:

Name	Type	Description
`dict`	`dict`	keys of database subsets which require gradients.

`func` parse_constraint

```
parse_constraint(expr: str)
```

100. Module `src.training.manager`

100.1 `mod` AI4SurrogateModelling.src.training.manager

Contains modules and functions related to the functionality of training management, i.e.: storing/loading training statistics, histories and best performing models.

`class` TrainingManager

```
TrainingManager(
    *, checkpoint_dir: str, reset: bool = False
)
```

Persists training metadata, history and model artefacts.

The manager abstracts disk I/O of checkpoints and training statistics. It can record arbitrary scalar statistics (`update_stats`), append training trajectories (`update_training_progress`), and keep track of both latest and best-performing model checkpoints.

`attr` checkpoint_dir `instance-attribute`

```
checkpoint_dir = checkpoint_dir
```

`attr` manager_fn `instance-attribute`

```
manager_fn = f'{checkpoint_dir}/data.pkl'
```

`attr` model_fn_best `instance-attribute`

```
model_fn_best = f'{checkpoint_dir}/model_best.bin'
```

`attr` model_fn_latest `instance-attribute`

```
model_fn_latest = f'{checkpoint_dir}/model_latest.bin'
```

`attr` path_manager `instance-attribute`

```
path_manager = Path(manager_fn)
```

`attr` path_model `instance-attribute`

```
path_model = Path(model_fn_best)
```

`meth` get_progress_data

```
get_progress_data(xkey: str) -> dict
```

Reformats stored history into grouped line-plot data structures.

Parameters:

Name	Type	Description	Default
`xkey`	`str`	Name of the metric to use as the common x-axis.	*required*

Returns:

Name	Type	Description
`dict`	`dict`	Mapping hierarchical metric prefixes to the plot template
	`dict`	consumed by the reporting layer.

meth get_training_epoch

```
get_training_epoch() -> int
```

meth get_training_time

```
get_training_time() -> float
```

meth load_model_best

```
load_model_best() -> ModelBase
```

Loads the best-scoring model checkpoint if it exists.

Returns:

Type	Description
`ModelBase`	ModelBase | None: Deserialised model or `None` on failure.

meth load_model_latest

```
load_model_latest() -> ModelBase
```

Loads the most recent model checkpoint if it exists.

Returns:

Type	Description
`ModelBase`	ModelBase | None: Deserialised model or `None` on failure.

meth update_model_best

```
update_model_best(*, model: ModelBase, criterion: float)
```

Serialises the model as the best checkpoint when it improves.

Parameters:

Name	Type	Description	Default
`model`	`ModelBase`	Candidate model to store.	*required*
`criterion`	`float`	Lower-is-better score compared against stored best.	*required*

meth update_model_latest

```
update_model_latest(*, model: ModelBase)
```

Serialises the model as the latest checkpoint.

meth update_stats

```
update_stats(*, stats: dict)
```

Updates cached training statistics with user-provided values.

Parameters:

Name	Type	Description	Default
`stats`	`dict[str, Any]`	Mapping of statistic name to serialisable payload.	*required*

`meth` update_training_progress

```
update_training_progress(
    *, progress: HistoryProgress
) -> None
```

Appends HistoryProgress measurements and persists them.

Parameters:

Name	Type	Description	Default
`progress`	`HistoryProgress`	History object holding batched metric series.	*required*

101. Module `src.training.__manager.manager_tabular`

101.1 **mod** AI4SurrogateModelling.src.training.__manager.manager_tabular

class ManagerTabular

```
ManagerTabular(
    target_dir: str,
    dataset: DatabaseTabular,
    train_ratio: float,
    validation_ratio: float,
)
```

attr dataset `instance-attribute`

```
dataset = dataset
```

attr project_dir `instance-attribute`

```
project_dir = target_dir
```

attr train_p `instance-attribute`

```
train_p = train_ratio
```

attr valid_p `instance-attribute`

```
valid_p = validation_ratio
```

meth perform_tests

```
perform_tests(
    ntests,
    models: list[Module],
    optimizers: list[Optimizer],
    schedulers: list[LRScheduler],
    parameter_range_batch_size: tuple[int, int],
    parameter_ranges_models: list[
        list[dict[str, tuple[int, int]]]
    ],
    parameter_ranges_optimizers: list[
        list[dict[str, tuple[int, int]]]
    ],
    parameter_ranges_schedulers: list[
        list[dict[str, tuple[int, int]]]
    ],
)
```

102. Module `src.training.measurement`

102.1 **mod** AI4SurrogateModelling.src.training.measurement

class Measurement

```
Measurement()
```

attr data `instance-attribute`

```
data = {}
```

attr iterator_data `instance-attribute`

```
iterator_data = {}
```

meth __add__

```
__add__(x: Measurement)
```

meth __getitem__

```
__getitem__(key: str)
```

meth __iter__

```
__iter__()
```

meth __str__

```
__str__()
```

meth add_measurement

```
add_measurement(*, tree_keys: list[str], values)
```

meth add_prefix

```
add_prefix(prefix: str)
```

103. Module `src.training.metric.metric_base`

103.1 **mod** AI4SurrogateModelling.src.training.metric.metric_base

Contains modules and functions related to the base class of various metrics to be calculated and stored during model training.

class MetricBase

```
MetricBase()
```

A base class containing basic functionality and configuration for metric calculations. All metric classes should derive from this class.

104. Module `src.training.scheduler.scheduler_configuration`

104.1 **mod** AI4SurrogateModelling.src.training.scheduler.scheduler_configuration

class CompositeScheduler

```
CompositeScheduler(
    *,
    schedulers: list[_LRScheduler],
    epoch_activations: list[int]
)
```

Bases: `_LRScheduler`

attr current_scheduler_index `instance-attribute`

```
current_scheduler_index = 0
```

attr epoch_activations `instance-attribute`

```
epoch_activations = []
```

attr epoch_index `instance-attribute`

```
epoch_index = 0
```

attr lr_multipliers `instance-attribute`

```
lr_multipliers = [1.0 for _ in (param_groups)]
```

attr param_groups `instance-attribute`

```
param_groups = param_groups
```

attr schedulers `instance-attribute`

```
schedulers = schedulers
```

meth get_lr

```
get_lr()
```

meth get_lr_multiplier

```
get_lr_multiplier()
```

meth load_scheduler `staticmethod`

```
load_scheduler(*, fn: str)
```

meth save_scheduler

```
save_scheduler(*, fn: str)
```

meth step

```
step(epoch=None)
```

class SchedulerAssembler

```
SchedulerAssembler()
```

attr assembled_schedulers instance-attribute

```
assembled_schedulers = {}
```

attr composite_scheduler instance-attribute

```
composite_scheduler = None
```

attr scheduler_configurations instance-attribute

```
scheduler_configurations = []
```

meth __getitem__

```
__getitem__(item)
```

meth add_scheduler

```
add_scheduler(
    *,
    scheduler_class: ENUM_Schedulers,
    parameters: dict,
    nepochs: int = 1,
    name: str
)
```

meth assemble

```
assemble(optimizer)
```

meth clear_assembled_data

```
clear_assembled_data()
```

meth get_scheduler_names

```
get_scheduler_names()
```

meth parse_scheduler_string

```
parse_scheduler_string(scheduler_string)
```

105. Module `src.training.trainer`

105.1 **mod** AI4SurrogateModelling.src.training.trainer

class Trainer

meth experiment `staticmethod`

```
experiment(model, dataloader)
```

meth gather_losses `staticmethod`

```
gather_losses(
    *,
    training_dataloader: DataLoader,
    validation_dataloader: DataLoader,
    model: ModelBase,
    criterion: CompositeLoss,
    optimizer: CompositeOptimizer = None,
    scheduler: CompositeScheduler = None
) -> tuple[Measurement, float]
```

meth gather_metrics `staticmethod`

```
gather_metrics(
    *,
    training_dataloader: DataLoader,
    validation_dataloader: DataLoader,
    model: ModelBase,
    metric: CompositeMetric
) -> Measurement
```

meth train `staticmethod`

```
train(
    *,
    reset: bool = False,
    model_assembler: ModelAssembler = None,
    optimizer_assembler: OptimizerAssembler = None,
    scheduler_assembler: SchedulerAssembler = None,
    loss_assembler: LossAssembler = None,
    metric_assembler: MetricAssembler = None,
    training_config: dict,
    dataset_config: dict,
    progress_save_dir: str
) -> ModelBase
```

106. Module `src.training.optimizer.custom_optimizers.adam`

106.1 **mod** AI4SurrogateModelling.src.training.optimizer.custom_optimizers.adam

class Adam

```
Adam(
    params,
    lr=0.0001,
    betas=(0.9, 0.999),
    eps=1e-08,
    weight_decay=0,
)
```

Bases: `Optimizer`

meth step

```
step(closure=None)
```

107. Module `src.training.optimizer.optimizer_configuration`

107.1 **mod** AI4SurrogateModelling.src.training.optimizer.optimizer_configuration

class CompositeOptimizer

```
CompositeOptimizer(
    *,
    optimizers: list[Optimizer],
    epoch_activations: list[int]
)
```

Bases: `Optimizer`

attr batch_index `instance-attribute`

```
batch_index = 0
```

attr current_optimizer_index `instance-attribute`

```
current_optimizer_index = 0
```

attr epoch_activations `instance-attribute`

```
epoch_activations = []
```

attr epoch_index `instance-attribute`

```
epoch_index = 0
```

attr optimizers `instance-attribute`

```
optimizers = copy()
```

attr param_groups `instance-attribute`

```
param_groups = param_groups
```

meth load_optimizer `staticmethod`

```
load_optimizer(*, fn: str)
```

meth next_epoch

```
next_epoch()
```

meth save_optimizer

```
save_optimizer(*, fn: str)
```

meth step

```
step(closure=None)
```

meth zero_grad

```
zero_grad()
```

class OptimizerAssembler

```
OptimizerAssembler()
```

attr assembled_optimizers instance-attribute

```
assembled_optimizers = {}
```

attr composite_optimizer instance-attribute

```
composite_optimizer = None
```

attr optimizer_configurations instance-attribute

```
optimizer_configurations = []
```

meth __getitem__

```
__getitem__(item)
```

meth add_optimizer

```
add_optimizer(
    *,
    optimizer_class: ENUM_Optimizers,
    parameters: dict = {},
    nepochs: int = 1,
    name: str
)
```

meth assemble

```
assemble(params)
```

meth clear_assembled_data

```
clear_assembled_data()
```

meth get_optimizer_names

```
get_optimizer_names()
```

108. Module `src.training.optimizer.wrapper`

108.1 **mod** AI4SurrogateModelling.src.training.optimizer.wrapper

func optimizer_wrapper

```
optimizer_wrapper(*, optimizer_constructor, params, kwargs)
```

https://shaiper-software.github.io/