xplogger

Release 0.11.3

Shagun Sodhani

Oct 07, 2022

GETTING STARTED

CHAPTER

ONE

WHY XPLOGGER

People use different tools for logging experimental results - Tensorboard, Wandb etc to name a few. Working with different collaborators, I will have to switch my logging tool with each new project. So I made this simple tool that provides a common interface to logging results to different loggers.

CHAPTER

TWO

INSTALLATION

- `pip install "xplogger[all]"`

If you want to use only the filesystem logger, use `pip install "xplogger"`

Install from source

- `git clone git@github.com:shagunsodhani/xplogger.git`
- `cd xplogger`
- `pip install ".[all]"`

Alternatively, `pip install "git+https://git@github.com/shagunsodhani/xplogger.git@master#egg=xplogger[all]"`

If you want to use only the filesystem logger, use `pip install .` or `pip install "git+https://git@github.com/shagunsodhani/xplogger.git@master#egg=xplogger"`.

CHAPTER

THREE

USE

- Make a `logbook_config`:

```
import xplogger.logbook
logbook_config = xplogger.logbook.make_config(
    logger_dir = <path to write logs>,
    wandb_config = <wandb config or None>,
    tensorboard_config = <tensorboard config or None>,
    mlflow_config = <mlflow config or None>)
```

  The API for `make_config` can be accessed here.

- Make a `LogBook` instance:

```
logbook = xplogger.logbook.LogBook(config = logbook_config)
```

- Use the `logbook` instance:

```
log = {
    "epoch": 1,
    "loss": 0.1,
    "accuracy": 0.2
}
logbook.write_metric(log)
```

  The API for `write_metric` can be accessed here.

CHAPTER

FOUR

NOTE

- If you are writing to wandb, the `log` must have a key called `step`. If your `log` already captures the `step` but as a different key (say `epoch`), you can pass the `wandb_key_map` argument (set as `{epoch: step}`). For more details, refer the documentation here.
- If you are writing to mlflow, the `log` must have a key called `step`. If your `log` already captures the `step` but as a different key (say `epoch`), you can pass the `mlflow_key_map` argument (set as `{epoch: step}`). For more details, refer the documentation here.
- If you are writing to tensorboard, the `log` must have a key called `main_tag` or `tag` which acts as the data Identifier and another key called `global_step`. These keys are described here. If your `log` already captures these values but as different key (say `mode` for `main_tag` and `epoch` for `global_step`), you can pass the `tensorboard_key_map` argument (set as `{mode: main_tag, epoch: global_step}`). For more details, refer the documentation here.

CHAPTER

FIVE

DEV SETUP

- `pip install -e ".[dev]"`
- Install pre-commit hooks `pre-commit install`
- The code is linted using:
  - `black`
  - `flake8`
  - `mypy`
  - `isort`
- Tests can be run locally using `nox`

CHAPTER

SIX

ACKNOWLEDGEMENTS

- Config for `circleci`, `pre-commit`, `mypy` etc are borrowed/modified from Hydra

CHAPTER

SEVEN

XPLOGGER

7.1 xplogger package

7.1.1 Subpackages

xplogger.experiment_manager package

Subpackages

xplogger.experiment_manager.notebook package

Submodules

xplogger.experiment_manager.notebook.utils module

Module contents

xplogger.experiment_manager.record package

Submodules

xplogger.experiment_manager.record.base module

xplogger.experiment_manager.record.mongo module

xplogger.experiment_manager.record.omegaconf module

xplogger.experiment_manager.record.record_list module

Module contents

xplogger.experiment_manager.slurm package

Submodules

xplogger.experiment_manager.slurm.ds module

xplogger.experiment_manager.slurm.job module

xplogger.experiment_manager.slurm.utils module

Functions to interact with the SLURM system.

xplogger.experiment_manager.slurm.utils.**cancel_job**(*job_id: str*) → str
Cancel the job corresponding to the job id.

xplogger.experiment_manager.slurm.utils.**get_info_from_slurm**(*job_id: str*) → dict[str, Any]
Get info about a specific job from slurm.

Parameters **job_id** (*str*) –

Returns job info.

Return type dict[str, Any]

xplogger.experiment_manager.slurm.utils.**map_jobid_to_raw_job_id**(*job_id: str*) → str
Map job_id to raw job_id.

Module contents

xplogger.experiment_manager.store package

Submodules

xplogger.experiment_manager.store.mongo module

Module contents

xplogger.experiment_manager.utils package

Submodules

xplogger.experiment_manager.utils.enum module

Enum data-structures.

class xplogger.experiment_manager.utils.enum.**ExperimentStatus**(*value*)
Bases: enum.Enum

An enumeration.

ANALYZED = 'ANALYZED'

COMPLETED = 'COMPLETED'

RUNNING = 'RUNNING'

Module contents

xplogger.experiment_manager.viz package

Submodules

xplogger.experiment_manager.viz.bokeh module

xplogger.experiment_manager.viz.matplotlib module

xplogger.experiment_manager.viz.utils module

Utlities functions to make bokeh plots.

xplogger.experiment_manager.viz.utils.**get_data_and_colors**(*exp_seq_dict: ExperimentSequenceDict*, *return_all_metrics_with_same_length: bool*, *kwargs_for_aggregate_metrics: dict[str, Any]*, *color_palette: list[Any]*, *colors: Optional[list[str]]*, *color_offset: int*) → tuple[dict[str, Any], list[str]]

Extract data and colors for generating the plots.

xplogger.experiment_manager.viz.utils.**validate_kwargs_for_aggregate_metrics**(*kwargs_for_aggregate_metrics: Optional[dict[str, Any]]*) → None

Validate that kwargs is not None and contains certain keys.

Module contents

Submodules

xplogger.experiment_manager.result module

Module contents

xplogger.logger package

Submodules

xplogger.logger.base module

Abstract logger class.

class xplogger.logger.base.**Logger**(*config: Dict[str, Any]*)

Bases: object

Abstract Logger Class.

abstract write(*log: Dict[str, Any]*) → None

Interface to write the log.

Parameters **log** (*LogType*) – Log to write

xplogger.logger.filesystem module

Functions to interface with the filesystem.

class xplogger.logger.filesystem.**Logger**(*config: Dict[str, Any]*)
Bases: *xplogger.logger.base.Logger*

Logger class that writes to the filesystem.

write(*log: Dict[str, Any]*) → None
Write the log to the filesystem.

Parameters **log** (*LogType*) – Log to write

xplogger.logger.filesystem.**get_logger_file_path**(*logger_dir: str*, *filename: Optional[str]*, *filename_prefix: str*, *filename_suffix: str*) → str
Get path to the file (to write logs to).

xplogger.logger.localdb module

Functions to interface with local db (a file).

class xplogger.logger.localdb.**Logger**(*config: Dict[str, Any]*)
Bases: *xplogger.logger.base.Logger*

Logger class that writes to local db (a file).

write(*log: Dict[str, Any]*) → None
Write the log to local db.

Parameters **log** (*LogType*) – Log to write

xplogger.logger.mlflow module

Logger class that writes to mlflow.

class xplogger.logger.mlflow.**Logger**(*config: Dict[str, Any]*)
Bases: *xplogger.logger.base.Logger*

Logger class that writes to mlflow.

write(*log: Dict[str, Any]*) → None
Write the log to mlflow.

Parameters **log** (*LogType*) – Log to write

write_config(*config: Dict[str, Any]*) → None
Write the config to mlflow.

Parameters **config** (*ConfigType*) – Config to write

write_metric(*metric: Dict[str, Any]*) → None
Write metric to mlflow.

Parameters **metric** (*MetricType*) – Metric to write

xplogger.logger.mongo module

Functions to interface with mongodb.

class xplogger.logger.mongo.**Logger**(*config: Dict[str, Any]*)
Bases: *xplogger.logger.base.Logger*

Logger class that writes to the mongodb.

is_connection_working() → bool
Check if the connection to the mongo server is working.

Checks the connection by issuing a dummy read query.

write(*log: Dict[str, Any]*) → None
Write the log to mongodb.

Parameters **log** (*LogType*) – Log to write

xplogger.logger.tensorboard module

Logger class that writes to tensorboard.

class xplogger.logger.tensorboard.**Logger**(*config: Dict[str, Any]*)
Bases: *xplogger.logger.base.Logger*

Logger class that writes to tensorboardX.

write(*log: Dict[str, Any]*) → None
Write the log to tensorboard.

Parameters **log** (*LogType*) – Log to write

write_config(*config: Dict[str, Any]*) → None
Write the config to tensorboard.

Parameters **config** (*ConfigType*) – Config to write

write_metric(*metric: Dict[str, Any]*) → None
Write metric to tensorboard.

Parameters **metric** (*MetricType*) – Metric to write

xplogger.logger.wandb module

Logger class that writes to wandb.

class xplogger.logger.wandb.**Logger**(*config: Dict[str, Any]*)
Bases: *xplogger.logger.base.Logger*

Logger class that writes to wandb.

write(*log: Dict[str, Any]*) → None
Write log to wandb.

Parameters **log** (*LogType*) – Log to write

write_config(*config: Dict[str, Any]*) → None
Write config to wandb.

Parameters **config** (*ConfigType*) – Config to write

write_metric(*metric: Dict[str, Any]*) → None
Write metric to wandb.

Parameters **metric** (*MetricType*) – Metric to write

Module contents

xplogger.parser package

Subpackages

xplogger.parser.experiment package

Submodules

xplogger.parser.experiment.experiment module

Container for the experiment data.

class xplogger.parser.experiment.experiment.**Experiment**(*configs: list[ConfigType]*, *metrics: experiment_utils.ExperimentMetricType*, *info: Optional[experiment_utils.ExperimentInfoType] = None*)

Bases: object

property config: Optional[Dict[str, Any]]
Access the config property.

log_to_wandb(*wandb_config: dict[str, Any]*) → LogBook
Log the experiment to wandb.

process_metrics(*metric_names: list[str]*, *x_name: str*, *x_min: int*, *x_max: int*, *mode: str*, *drop_duplicates: bool*, *dropna: bool*, *verbose: bool*) → dict[str, np.typing.NDArray[np.float32]]
Given a list of metric names, process the metrics for a given experiment.

Parameters

- **metric_names** (*list[str]*) – Names of metrics to process.
- **x_name** (*str*) – The column/meric with respect to which other metrics are tracked. For example *steps* or *epochs*.
- **x_min** (*int*) – Filter the experiment where the max value of *x_name* is less than or equal to *x_min*.
- **x_max** (*int*) – Filter the metric values where value of *x_name* (corresponding to metric values) is greater than *x_max*
- **mode** (*str*) – Mode when selecting metrics. Recall that *experiment.metrics* is a dictionary mapping *modes* to dataframes.
- **drop_duplicates** (*bool*) – Should drop duplicate values in the *x_name* column
- **verbose** (*bool*) – Should print additional information

Returns

dictionary mapping metric name to 1-dimensional numpy array of metric values.

Return type dict[str, np.ndarray]

serialize(*dir_path: pathlib.Path*) → None
Serialize the experiment data and store at *dir_path*.

- configs are stored as jsonl (since there are only a few configs per experiment) in a file called *config.jsonl*.
- metrics are stored in [*feather* format](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.to_feather.html).
- info is stored in the gzip format.

xplogger.parser.experiment.experiment.**ExperimentList**
alias of *xplogger.parser.experiment.experiment.ExperimentSequence*

class xplogger.parser.experiment.experiment.**ExperimentSequence**(*experiments: list[*Experiment*]*)
Bases: collections.UserList

aggregate(*aggregate_configs: Callable[[list[list[ConfigType]]], list[ConfigType]] = <function return_first_config>, aggregate_metrics: Callable[[list[experiment_utils.ExperimentMetricType]], experiment_utils.ExperimentMetricType] = <function concat_metrics>, aggregate_infos: Callable[[list[experiment_utils.ExperimentInfoType]], experiment_utils.ExperimentInfoType] = <function return_first_infos>*) → *Experiment*
Aggregate a sequence of experiments into a single experiment.

Parameters

- **aggregate_configs** (*Callable[[list[list[ConfigType]]], list[ConfigType]], optional*) – Function to aggregate the configs. Defaults to experiment_utils.return_first_config.
- **aggregate_metrics** (*Callable[[list[experiment_utils.ExperimentMetricType]], ExperimentMetricType], optional*) – Function to aggregate the metrics. Defaults to experiment_utils.concat_metrics.
- **aggregate_infos** (*Callable[[list[experiment_utils.ExperimentInfoType]], ExperimentInfoType], optional*) – Function to aggregate the information. Defaults to experiment_utils.return_first_infos.

Returns Aggregated Experiment.

Return type *Experiment*

aggregate_metrics(***kwargs: Any*) → dict[str, np.typing.NDArray[np.float32]]
Aggregate metrics across experiment sequences.

Given a list of metric names, aggreate the metrics across different experiments in an experiment sequence.

Parameters

- **metric_names** (*list[str]*) – Names of metrics to aggregate.
- **x_name** (*str*) – The column/meric with respect to which other metrics are tracked. For example *steps* or *epochs*. This aggregated values for this metric are also returned.
- **x_min** (*int*) – Only those experiments are considered (during aggregation) where the max value of *x_name* is greater than or equal to *x_min*.
- **x_max** (*int*) – When aggregating experiments, consider metric values such that the max value of *x_name* corresponding to metric values is less than or equal to *x_max*
- **mode** (*str*) – Mode when selecting metrics. Recall that *experiment.metrics* is a dictionary mapping *modes* to dataframes.

- **drop_duplicates** (*bool*) – Should drop duplicate values in the *x_name* column
- **verbose** (*bool*) – Should print additional information

Returns

dictionary mapping metric name to 2-dimensional numpy array of metric values. The first dimension corresponds to the experiments and the second corresponds to metrics per experiment.

Return type dict[str, np.ndarray]

filter(*filter_fn: Callable[[*xplogger.parser.experiment.experiment.Experiment*], bool]*) → *ExperimentSequence*

Filter experiments in the sequence.

Parameters **filter_fn** – Function to filter an experiment

Returns A sequence of experiments for which the filter condition is true

Return type *ExperimentSequence*

get_param_groups(*params_to_exclude: Iterable[str]*) → tuple[ConfigType, dict[str, set[Any]]]

Return two groups of params, one which is fixed across the experiments and one which varies.

This function is useful when understanding the effect of different parameters on the model's performance. One could plot the performance of the different experiments, as a function of the parameters that vary.

Parameters **params_to_exclude** (*Iterable[str]*) – These parameters are not returned in either group. This is useful for ignoring parameters like *time when the experiment was started* since these parameters should not affect the performance. In absence of this argument, all such parameters will likely be returned with the group of varying parameters.

Returns

The first group/config contains the params which are fixed across the experiments. It maps these params to their *default* values, hence it should be a subset of any config. The second group/config contains the params which vary across the experiments. It maps these params to the set of values they take.

Return type tuple[ConfigType, dict[str, set[Any]]]

groupby(*group_fn: Callable[[*Experiment*], str]*) → dict[str, 'ExperimentSequence']

Group experiments in the sequence.

Parameters **group_fn** – Function to assign a string group id to the experiment

Returns A dictionary mapping the sring group id to a sequence of experiments

Return type dict[str, *ExperimentSequence*]

class xplogger.parser.experiment.experiment.**ExperimentSequenceDict**(*experiment_sequence_dict: dict[Any,* ExperimentSequence*]*)

Bases: collections.UserDict

aggregate_metrics(*return_all_metrics_with_same_length: bool = True*, ***kwargs: Any*) → dict[str, np.typing.NDArray[np.float32]]

Aggreate metrics across experiment sequences.

Given a list of metric names, aggreate the metrics across different experiment sequences in a dictionary indexed by the metric name.

Parameters

- **get_experiment_name** (*Callable[[str], str]*) – Function to map the given key with a name.
- **metric_names** (*list[str]*) – Names of metrics to aggregate.
- **x_name** (*str*) – The column/meric with respect to which other metrics are tracked. For example *steps* or *epochs*. This aggregated values for this metric are also returned.
- **mode** (*str*) – Mode when selecting metrics. Recall that *experiment.metrics* is a dictionary mapping *modes* to dataframes.

Returns

dictionary mapping metric name to 2-dimensional numpy array of metric values. The first dimension corresponds to the experiments and the second corresponds to metrics per experiment.

Return type dict[str, np.typing.NDArray[np.float32]]

filter(*filter_fn: Callable[[str,* xplogger.parser.experiment.experiment.Experiment*], bool]*) → *ExperimentSequenceDict*

Filter experiment sequences in the dict.

Parameters **filter_fn** – Function to filter an experiment sequence

Returns A dict of sequence of experiments for which the filter condition is true

Return type *ExperimentSequenceDict*

xplogger.parser.experiment.experiment.**deserialize**(*dir_path: str*) → *xplogger.parser.experiment.experiment.Experiment*

Deserialize the experiment data stored at *dir_path* and return an Experiment object.

xplogger.parser.experiment.parser module

Implementation of Parser to parse experiment from the logs.

class xplogger.parser.experiment.parser.**Parser**(*parse_config_line: Callable[[str], Optional[Dict[str, Any]]] = <function parse_json_and_match_value>, parse_metric_line: Callable[[str], Optional[Dict[str, Any]]] = <function parse_json_and_match_value>, parse_info_line: Callable[[str], Optional[Dict[str, Any]]] = <function parse_json>*)

Bases: *xplogger.parser.base.Parser*

Class to parse an experiment from the log dir.

parse(*filepath_pattern: Union[str, pathlib.Path]*) → *xplogger.parser.experiment.experiment.Experiment*

Load one experiment from the log dir.

Parameters **filepath_pattern** (*Union[str, Path]*) – filepath pattern to glob or instance of Path (directory) object.

Returns Experiment

xplogger.parser.experiment.utils module

Utilit functions to work with the experiment data.

xplogger.parser.experiment.utils.**concat_metrics**(*metric_list: list[ExperimentMetricType]*) → ExperimentMetricType

Concatenate the metrics.

Parameters **metric_list** (*list[ExperimentMetricType]*) –

Returns ExperimentMetricType

xplogger.parser.experiment.utils.**mean_metrics**(*metric_list: list[ExperimentMetricType]*) → ExperimentMetricType

Compute the mean of the metrics.

Parameters **metric_list** (*list[ExperimentMetricType]*) –

Returns ExperimentMetricType

xplogger.parser.experiment.utils.**return_first_config**(*config_lists: list[list[ConfigType]]*) → list[ConfigType]

Return the first config list, from a list of list of configs, else return empty list.

Parameters **config_lists** (*list[list[ConfigType]]*) –

Returns list[ConfigType]

xplogger.parser.experiment.utils.**return_first_infos**(*info_list: list[ExperimentInfoType]*) → ExperimentInfoType

Return the first info, from a list of infos. Otherwise return empty info.

Parameters **info_list** (*list[ExperimentInfoType]*) –

Returns ExperimentInfoType

xplogger.parser.experiment.utils.**sum_metrics**(*metric_list: list[ExperimentMetricType]*) → ExperimentMetricType

Compute the sum of the metrics.

Parameters **metric_list** (*list[ExperimentMetricType]*) –

Returns ExperimentMetricType

Module contents

Module to interact with the experiment data.

Submodules

xplogger.parser.base module

Base class that all parsers extend.

class xplogger.parser.base.**Parser**(*parse_line: Callable[[str], Optional[Dict[str, Any]]] = <function parse_json>*)

Bases: abc.ABC

Base class that all parsers extend.

xplogger.parser.config module

Implementation of Parser to parse config from logs.

class xplogger.parser.config.**Parser**(*parse_line: Callable[[str], Optional[Dict[str, Any]]] = <function parse_json_and_match_value>*)

Bases: *xplogger.parser.log.Parser*

Class to parse config from the logs.

xplogger.parser.config.**parse_json_and_match_value**(*line: str*) → Optional[Dict[str, Any]]

Parse a line as JSON log and check if it a valid config log.

xplogger.parser.log module

Implementation of Parser to parse the logs.

class xplogger.parser.log.**Parser**(*parse_line: Callable[[str], Optional[Dict[str, Any]]] = <function parse_json>*)

Bases: *xplogger.parser.base.Parser*

Class to parse the log files.

parse(*filepath_pattern: str*) → Iterator[Dict[str, Any]]

Open a log file, parse its contents and return *logs*.

Parameters **filepath_pattern** (*str*) – filepath pattern to glob

Returns Iterator over the logs

Return type Iterator[LogType]

Yields *Iterator[LogType]* – Iterator over the logs

parse_first_log(*filepath_pattern: str*) → Optional[Dict[str, Any]]

Return the first log from a file.

The method will return after finding the first log. Unlike *parse()* method, it will not iterate over the entire log file (thus saving memory and time).

Parameters **filepath_pattern** (*str*) – filepath pattern to glob

Returns First instance of a log

Return type LogType

parse_last_log(*filepath_pattern: str*) → Optional[Dict[str, Any]]

Return the last log from a file.

Like *parse()* method, it will iterate over the entire log file but will not keep all the logs in memory (thus saving memory).

Parameters **filepath_pattern** (*str*) – filepath pattern to glob

Returns Last instance of a log

Return type LogType

xplogger.parser.log.**parse_json_and_match_value**(*line: str*, *value: str*) → Optional[Dict[str, Any]]

Parse a line as JSON log and check if it a valid log.

xplogger.parser.metric module

Implementation of Parser to parse metrics from logs.

class xplogger.parser.metric.**Parser**(*parse_line: Callable[[str], Optional[Dict[str, Any]]] = <function parse_json_and_match_value>*)

Bases: *xplogger.parser.log.Parser*

Class to parse the metrics from the logs.

parse_as_df(*filepath_pattern: str, group_metrics: Callable[[list[LogType]], dict[str, list[LogType]]] = <function group_metrics>, aggregate_metrics: Callable[[list[LogType]], list[LogType]] = <function aggregate_metrics>*) → dict[str, pd.DataFrame]

Create a dict of (metric_name, dataframe).

Method that: (i) reads metrics from the filesystem (ii) groups metrics (iii) aggregates all the metrics within a group, (iv) converts the aggregate metrics into dataframes and returns a dictionary of dataframes

Parameters

- **filepath_pattern** (*str*) – filepath pattern to glob
- **group_metrics** (*Callable[[list[LogType]], dict[str, list[LogType]]], optional*) – Function to group a list of metrics into a dictionary of (key, list of grouped metrics). Defaults to group_metrics.
- **aggregate_metrics** (*Callable[[list[LogType]], list[LogType]], optional*) – Function to aggregate a list of metrics. Defaults to aggregate_metrics.

xplogger.parser.metric.**aggregate_metrics**(*metrics: list[MetricType]*) → list[MetricType]

Aggregate a list of metrics.

Parameters **metrics** (*list[MetricType]*) – list of metrics to aggregate

Returns list of aggregated metrics

Return type list[MetricType]

xplogger.parser.metric.**group_metrics**(*metrics: list[MetricType]*) → dict[str, list[MetricType]]

Group a list of metrics.

Group a list of metrics into a dictionary of (key, list of grouped metrics)

Parameters **metrics** (*list[MetricType]*) – list of metrics to group

Returns

Dictionary of (key, list of grouped metrics)

Return type dict[str, list[MetricType]]

xplogger.parser.metric.**metrics_to_df**(*metric_logs: list[LogType], group_metrics: Callable[[list[LogType]], dict[str, list[LogType]]] = <function group_metrics>, aggregate_metrics: Callable[[list[LogType]], list[LogType]] = <function aggregate_metrics>*) → dict[str, pd.DataFrame]

Create a dict of (metric_name, dataframe).

Method that: (i) groups metrics (ii) aggregates all the metrics within a group, (iii) converts the aggregate metrics into dataframes and returns a dictionary of dataframes

Parameters

- **metric_logs** (*list[LogType]*) – list of metrics

- **group_metrics** (*Callable[[list[LogType]], dict[str, list[LogType]]], optional*) – Function to group a list of metrics into a dictionary of (key, list of grouped metrics). Defaults to group_metrics.
- **aggregate_metrics** (*Callable[[list[LogType]], list[LogType]], optional*) – Function to aggregate a list of metrics. Defaults to aggregate_metrics.

Returns [description]

Return type dict[str, pd.DataFrame]

xplogger.parser.metric.**parse_json_and_match_value**(*line: str*) → Optional[Dict[str, Any]]

Parse a line as JSON log and check if it a valid metric log.

xplogger.parser.utils module

Utility functions for the parser module.

xplogger.parser.utils.**compare_logs**(*first_log: LogType*, *second_log: LogType*, *verbose: bool = False*) → tuple[list[str], list[str], list[str]]

Compare two logs.

Return list of keys that are either missing or have different valus in the two logs.

Parameters

- **first_log** (*LogType*) – First Log
- **second_log** (*LogType*) – Second Log
- **verbose** (*bool*) – Defaults to False

Returns

tuple of [list of keys with different values, list of keys with values missing in first log, list of keys with values missing in the second log,]

Return type tuple[list[str], list[str], list[str]]

xplogger.parser.utils.**flatten_log**(*d: Dict[str, Any]*, *parent_key: str = ''*, *sep: str = '#'*) → Dict[str, Any]

Flatten a log using a separator.

Taken from https://stackoverflow.com/a/6027615/1353861

Parameters

- **d** (*LogType*) – [description]
- **parent_key** (*str, optional*) – [description]. Defaults to "".
- **sep** (*str, optional*) – [description]. Defaults to "#".

Returns [description]

Return type LogType

xplogger.parser.utils.**get_param_groups**(*configs: Iterable[ConfigType]*, *params_to_exclude: Iterable[str]*) → tuple[ConfigType, dict[str, set[Any]]]

Return two groups of params, one which is fixed across the experiments and one which varies.

This function is useful when understanding the effect of different parameters on the model's performance. One could plot the performance of the different experiments, as a function of the parameters that vary.

Parameters

- **configs** (*Iterable[ConfigType]*) – Collection of configs, to extract params from.
- **params_to_exclude** (*Iterable[str]*) – These parameters are not returned in either group. This is useful for ignoring parameters like *time when the experiment was started* since these parameters should not affect the performance. In absence of this argument, all such parameters will likely be returned with the group of varying parameters.

Returns

The first group/config contains the params which are fixed across the experiments. It maps these params to their *default* values, hence it should be a subset of any config. The second group/config contains the params which vary across the experiments. It maps these params to the set of values they take.

Return type tuple[ConfigType, dict[str, set[Any]]]

xplogger.parser.utils.**parse_json**(*line: str*) → Optional[Dict[str, Any]]
Parse a line as JSON string.

Module contents

7.1.2 Submodules

7.1.3 xplogger.logbook module

Implementation of the LogBook class.

LogBook class provides an interface to persist the logs on the filesystem, tensorboard, remote backends, etc.

class xplogger.logbook.**LogBook**(*config: Dict[str, Any]*)
Bases: object

This class provides an interface to persist the logs on the filesystem, tensorboard, remote backends, etc.

write(*log: Dict[str, Any]*, *log_type: str = 'metric'*) → None
Write log to loggers.

Parameters

- **log** (*LogType*) – Log to write
- **log_type** (*str, optional*) – Type of this log. Defaults to "metric".

write_config(*config: Dict[str, Any]*) → None
Write config to loggers.

Parameters **[ConfigType]** (*config*) – Config to write.

write_message(*message: Any*, *log_type: str = 'info'*) → None
Write message string to loggers.

Parameters

- **message** (*Any*) – Message string to write
- **log_type** (*str, optional*) – Type of this message (log). Defaults to "info".

write_metadata(*metadata: Dict[str, Any]*) → None
Write metadata to loggers.

Parameters **metadata** (*LogType*) – Metadata to wite

write_metric(*metric: Dict[str, Any]*) → None
Write metric to loggers.

Parameters **metric** (*MetricType*) – Metric to write

xplogger.logbook.**make_config**(*id: str = '0'*, *name: str = 'default_logger'*, *write_to_console: bool = True*, *logger_dir: Optional[str] = None*, *filename: Optional[str] = None*, *filename_prefix: str = ''*, *create_multiple_log_files: bool = True*, *wandb_config: Optional[Dict[str, Any]] = None*, *wandb_key_map: Optional[Dict[str, str]] = None*, *wandb_prefix_key: Optional[str] = None*, *tensorboard_config: Optional[Dict[str, Any]] = None*, *tensorboard_key_map: Optional[Dict[str, str]] = None*, *tensorboard_prefix_key: Optional[str] = None*, *mlflow_config: Optional[Dict[str, Any]] = None*, *mlflow_key_map: Optional[Dict[str, str]] = None*, *mlflow_prefix_key: Optional[str] = None*, *mongo_config: Optional[Dict[str, Any]] = None*, *localdb_config: Optional[Dict[str, Any]] = None*) → Dict[str, Any]
Make the config that can be passed to the LogBook constructor.

Parameters

- **id** (*str, optional*) – Id of the current LogBook instance. Defaults to "0".
- **name** (*str, optional*) – Name of the logger. Defaults to "default_logger".
- **write_to_console** (*bool, optional*) – Should write the logs to console. Defaults to True
- **logger_dir** (*str, optional*) – Path where the logs will be written. If None is passed, logs are not written to the filesystem. LogBook creates the directory, if it does not exist. Defaults to None.
- **filename** (*str, optional*) – Name to assign to the log file (eg log.jsonl). If None is passed, this argument is ignored. If the value is set, *filename_prefix* and *create_multiple_log_files* arguments are ignored. Defaults to None.
- **filename_prefix** (*str*) – String to prefix before the name of the log files. Eg if filename_prefix is "dummy", name of log files are dummymetric.jsonl, dummylog.jsonl etc. This argument is ignored if *filename* is set. Defaults to "".
- **create_multiple_log_files** (*bool, optional*) – Should multiple log files be created - for config, metric, metadata and message logs. If True, the files are named as config_log.jsonl, metric_log.jsonl etc. If False, only one file log.jsonl is created. This argument is ignored if *filename* is set. Defaults to True.
- **wandb_config** (*Optional[ConfigType], optional*) – Config for the wandb logger. If None, wandb logger is not created. The config can have any parameters that wandb.init() methods accepts (https://docs.wandb.com/library/init). Note that the wandb_config is passed as keyword arguments to the wandb.init() method. This provides a lot of flexibility to the users to configure wandb. This also means that the config should not have any parameters that wandb.init() would not accept. Defaults to None.
- **wandb_key_map** (*Optional[KeyMapType], optional*) – When using wandb logger for logging metrics, certain keys are required. This dictionary provides an easy way to map the keys in the *log* to be written) with the keys that wandb logger needs. For instance, wandb logger needs a *step* key in all the metric logs. If your logs have a key called *epoch* that you want to use as *step*, set *wandb_key_map* as *{epoch: step}*. This argument is ignored if set to None. Defaults to None.
- **wandb_prefix_key** (*Optional[str], optional*) – When a metric is logged to wandb, prefix the value (corresponding to the key) to all the remaining keys before values are logged

in the wandb logger. This argument is ignored if set to None. Defaults to None.

- **tensorboard_config** (*Optional[ConfigType], optional*) – config to initialise the tensorboardX logger. The config can have any parameters that [tensorboardX.SummaryWriter() method](https://tensorboardx.readthedocs.io/en/latest/tensorboard.html#tensorboardX.SummaryWriter) accepts. Note that the config is passed as keyword arguments to the tensorboardX.SummaryWriter() method. This provides a lot of flexibility to the users to configure tensorboard. This also means that config should not have any parameters that tensorboardX.SummaryWriter() would not accept. Defaults to None.
- **tensorboard_key_map** (*Optional[KeyMapType], optional*) – When using tensorboard logger for logging metrics, certain keys are required. This dictionary provides an easy way to map the keys in the *log* (to be written) with the keys that tensorboard logger needs. For instance, tensorboard logger needs a *main_tag* key and a *global_step* in all the metric logs. If your logs have a key called *epoch* that you want to use as *step*, and a key called *mode* that you want to use as *main_tag*, set *tensorboard_key_map* as *{epoch: global_step, mode: main_tag}*. This argument is ignored if set to None. Defaults to None.
- **tensorboard_prefix_key** (*Optional[str], optional*) – When a metric is logged to tensorboard, prefix the value (corresponding to the key) to all the remaining keys before values are logged in the tensorboard logger. This argument is ignored if set to None. Defaults to None.
- **mlflow_config** (*Optional[ConfigType], optional*) – config to initialise an mlflow experiment. The config can have any parameters that [mlflow.create_experiment() method](https://mlflow.org/docs/latest/python_api/mlflow.html#mlflow.create_experiment) accepts. Note that the config is passed as keyword arguments to the mlflow.create_experiment() method. This provides a lot of flexibility to the users to configure mlflow. This also means that config should not have any parameters that mlflow.create_experiment would not accept. Defaults to None.
- **mlflow_key_map** (*Optional[KeyMapType], optional*) – When using mlflow logger for logging metrics, certain keys are required. This dictionary provides an easy way to map the keys in the *log* (to be written) with the keys that mlflow logger needs. For instance, mlflow logger needs a *step* key in all the metric logs. If your logs have a key called *epoch* that you want to use as *step*, set *mlflow_key_map* as *{epoch: step}*. This argument is ignored if set to None. Defaults to None.
- **mlflow_prefix_key** (*Optional[str], optional*) – When a metric is logged to mlflow, prefix the value (corresponding to the key) to all the remaining keys before values are logged in the mlflow logger. This argument is ignored if set to None. Defaults to None.
- **mongo_config** (*Optional[ConfigType], optional*) – config to initialise connection to a collection in mongodb. The config supports the following required keys:

(1) host: host where mongodb is running.

(2) port: port on which mongodb is running.

(3) db: name of the db to use.

(4) collection: name of the collection to use.

The config supports the following optional keys:

(1) logger_types: list/set of types that the logger should log.

Defaults to None.

- **localdb_config** (*Optional[ConfigType], optional*) – config to initialise connection to localdb. The config supports the following keys:

(1) path: path to the localdb file.

The config supports the following optional keys:

(1) logger_types: list/set of types that the logger should log.

Defaults to None.

Returns config to construct the LogBook

Return type ConfigType

7.1.4 xplogger.metrics module

Implementation of different type of metrics.

class xplogger.metrics.**AverageMetric**(*name: str*)

Bases: *xplogger.metrics.BaseMetric*

Metric to track the average value.

This is generally used for logging strings

Parameters **BaseMetric** – Base metric class

get_val() → float
Get the current average value.

reset() → None
Reset Metric.

update(*val: Union[int, float]*, *n: int = 1*) → None
Update the metric.

Update the metric using the current average value and the number of samples used to compute the average value

Parameters

- **val** (*NumType*) – current average value
- **n** (*int, optional*) – Number of samples used to compute the average. Defaults to 1

class xplogger.metrics.**BaseMetric**(*name: str*)

Bases: object

Base Metric class. This class is not to be used directly.

get_val() → Union[str, int, float]
Get the current value of the metric.

reset() → None
Reset the metric to the default value.

update(*val: Any*) → None
Update the metric using the current val.

Parameters **val** (*Any*) – Current value. This value is used to update the metric

class xplogger.metrics.**ComparisonMetric**(*name: str*, *default_val: Union[str, int, float]*, *comparison_op: Callable[[Union[str, int, float], Union[str, int, float]], bool]*)

Bases: *xplogger.metrics.BaseMetric*

Metric to track the min/max value.

This is generally used for logging best accuracy, least loss, etc.

Parameters **BaseMetric** – Base metric class

reset() → None
Reset the metric to the default value.

update(*val: Union[str, int, float]*) → None
Use the comparison operator to decide which value to keep.

If the output of self.comparison_op(val, self)

Parameters **val** (*ValueType*) – Value to compare the current value with. If comparison_op(current_val, new_val) is true, we update the current value.

class xplogger.metrics.**ConstantMetric**(*name: str*, *val: Union[str, int, float]*)

Bases: *xplogger.metrics.BaseMetric*

Metric to track one fixed value.

This is generally used for logging strings

Parameters **BaseMetric** – Base metric class

reset() → None
Do nothing for the constant metrics.

update(*val: Optional[Union[str, int, float]] = None*) → None
Do nothing for the constant metrics.

Parameters **val** (*Any*) – This value is ignored

class xplogger.metrics.**CurrentMetric**(*name: str*)

Bases: *xplogger.metrics.BaseMetric*

Metric to track only the most recent value.

Parameters **BaseMetric** – Base metric class

update(*val: Union[str, int, float]*) → None
Update the metric using the current val.

Parameters **val** (*Any*) – Current value. The metric value is set to this value

class xplogger.metrics.**MaxMetric**(*name: str*)

Bases: *xplogger.metrics.ComparisonMetric*

Metric to track the max value.

This is generally used for logging best accuracy, etc.

Parameters **ComparisonMetric** – Comparison metric class

class xplogger.metrics.**MetricDict**(*metric_list: Iterable[*xplogger.metrics.BaseMetric*]*)

Bases: object

Class that wraps over a collection of metrics.

reset() → None
Reset all the metrics to default values.

to_dict() → Dict[str, Any]
Convert the metrics into a dictionary for *LogBook*.

Returns Metric data in as a dictionary

Return type LogType

update(*metrics_dict: Union[Dict[str, Any],* xplogger.metrics.MetricDict*]*) → None
Update all the metrics using the current values.

Parameters **metrics_dict** (*Union[LogType,* MetricDict*]*) – Current value of metrics

class xplogger.metrics.**MinMetric**(*name: str*)
Bases: *xplogger.metrics.ComparisonMetric*

Metric to track the min value.

This is generally used for logging least loss, etc.

Parameters **ComparisonMetric** – Comparison metric class

class xplogger.metrics.**SumMetric**(*name: str*)
Bases: *xplogger.metrics.AverageMetric*

Metric to track the sum value.

Parameters **BaseMetric** – Base metric class

get_val() → float
Get the current sum value.

7.1.5 xplogger.types module

Types used in the package.

7.1.6 xplogger.utils module

Utility Methods.

xplogger.utils.**compare_keys_in_dict**(*dict1: dict[Any, Any]*, *dict2: dict[Any, Any]*) → bool
Check that the two dicts have the same set of keys.

xplogger.utils.**flatten_dict**(*d: dict[str, Any]*, *parent_key: str = ''*, *sep: str = '#'*) → dict[str, Any]
Flatten a given dict using the given seperator.

Taken from https://stackoverflow.com/a/6027615/1353861

Parameters

- **d** (*dict[str, Any]*) – dictionary to flatten
- **parent_key** (*str, optional*) – Keep track of the higher level key Defaults to "".
- **sep** (*str, optional*) – string for concatenating the keys. Defaults to "#"

Returns [description]

Return type dict[str, Any]

xplogger.utils.**get_elem_from_set**(*_set: set[Any]*) → Any
Get an element from a set.

xplogger.utils.**make_dir**(*path: pathlib.Path*) → None
Make dir, if not exists.

Parameters **path** (*Path*) – dir to make

xplogger.utils.**serialize_log_to_json**(*log: Dict[str, Any]*) → str
Serialize the log into a JSON string.

Parameters **log** (*LogType*) – Log to be serialized

Returns JSON serialized string

Return type str

xplogger.utils.**to_json_serializable**(*val: Any*) → Any
Serialize values as json.

7.1.7 Module contents

CHAPTER

EIGHT

COMMUNITY

- If you have questions, open an Issue
- Or, use Github Discussions
- To contribute, open a Pull Request (PR)

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

X

INDEX

L

M

P

R

S