fluidml

Release 0.3.4

Lars Hillebrand, Rajkumar Ramamurthy

Oct 20, 2023

CONTENTS:

CHAPTER

ONE

API REFERENCE

1.1 fluidml

class fluidml.**Flow**(*tasks*, *config_ignore_prefix=None*, *config_group_prefix=None*)

Flow implements the core logic of building and expanding task graphs from task specifications.

- It automatically expands the tasks based on task specifications and task configs.
- It extends the registered dependencies to the expanded tasks.
- It provides the task graph objects for introspection and visualization.
- Finally, it executes the task graph using multiprocessing if necessary.

Parameters

- **tasks** (*List[TaskSpec]*) – List of task specifications.
- **config_ignore_prefix** (*Optional[str]*) – A config key prefix, e.g. "_". Prefixed keys will be not included in the "unique_config", which is used to determine whether a run has been executed or not.
- **config_group_prefix** (*Optional[str]*) – A config grouping prefix, to indicate that to parameters are grouped and expanded using the "zip" method. The grouping prefix enables the "zip" expansion of specific parameters, while all remaining grid parameters are expanded via "product". Example: `cfg = {"a": [1, 2, "@x"], "b": [1, 2, 3], "c": [1, 2, "@x"]` Without grouping "product" expansion would yield: *2 * 2 * 3 = 12* configs. With grouping "product" expansion yields : *2 * 3 = 6* configs, since the grouped parameters are "zip" expanded.

property num_tasks: int

The total number of expanded tasks in the task graph.

run(*num_workers=None*, *resources=None*, *start_method='spawn'*, *exit_on_error=True*, *log_to_tmux=False*, *max_panes_per_window=4*, *force=None*, *project_name='uncategorized'*, *run_name=None*, *results_store=None*, *return_results=None*)

Runs the expanded task graph sequentially or in parallel using multiprocessing and returns the results.

Parameters

- **num_workers** (*Optional[int]*) – Number of parallel processes (dolphins) used. Internally, the optimal number of processes `optimal_num_workers` is inferred from the task graph. If `num_workers is None` or `num_workers > optimal_num_workers`, `num_workers` is overwritten to `optimal_num_workers`. Defaults to `None`.

- **resources** (*Optional[Union[Any, List[Any]]]*) – A single resource object or a list of resources that are assigned to workers. Resources can hold arbitrary data, e.g. gpu or cpu device information, making sure that each worker has access to a dedicated device If `num_workers > 1` and `len(resources) == num_workers` resources are assigned to each worker. If `len(resources) < num_workers` resources are assigned randomly to workers. If `num_workers == 1` the first resource `resources[0]` is assigned to all tasks (not workers). Defaults to `None`.
- **start_method** (*str*) – Start method for multiprocessing. Defaults to `"spawn"`. Only used when `num_workers > 1`.
- **exit_on_error** (*bool*) – When an error happens all workers finish their current tasks and exit gracefully. Defaults to True. Only used when `num_workers > 1`.
- **log_to_tmux** (*bool*) – If `True` a new tmux session is created (given tmux is installed) and each worker (process) logs to a dedicated pane to avoid garbled logging output. The session's name equals the combined `f"{project_name}--{run-name}"`. Defaults to `False`. Only used when `num_workers > 1`.
- **max_panes_per_window** (*int*) – Max number of panes per tmux window. Requires `log_to_tmux` being set to `True`. Defaults to `4`. Only used when `num_workers > 1`.
- **force** (*Optional[Union[str, List[str]]]*) – Forcefully re-run tasks. Possible options are: 1) `"all"` - All the tasks are re-run. 2) A task name (e.g. "PreProcessTask") or list of task names (e.g. `["PreProcessTask1", "PreProcessTask2"]`). Additionally, each task name can have the suffix "+" to re-run also its successors (e.g. "PreProcessTask+").
- **project_name** (*str*) – Name of project. Defaults to `"uncategorized"`.
- **run_name** (*Optional[str]*) – Name of run.
- **results_store** (*Optional[ResultsStore]*) – An instance of results store for results management. If nothing is provided, a non-persistent InMemoryStore store is used.
- **return_results** (*Optional[str]*) – Return results-dictionary after `run()`. Defaults to `all`. Choices: `"all", "latest", None`.

Returns

A results dictionary with the following schema.
`{"task_name_1": List[TaskResults], "task_name_2": List[TaskResults]}`

Return type
Optional[*Dict*[str, *List*[*TaskResults*]]]

class fluidml.**Task**(*name=None*, *unique_name=None*, *project_name=None*, *run_name=None*, *state=None*, *started=None*, *ended=None*, *run_history=None*, *sweep_counter=None*, *unique_config_hash=None*, *unique_config=None*, *results_store=None*, *resource=None*, *force=None*, *expects=None*, *reduce=None*, *config=None*)

The base `Task` that provides the user with additional features such as result store and run name access.

When implementing a task as a Class

Parameters

- **name** (*Optional[str]*) – Name of the task, e.g. "Processing".
- **unique_name** (*Optional[str]*) – Unique name of the task if a run contains multiple instances of the same task, e.g. "Processing-3".
- **project_name** (*Optional[str]*) – Name of the project.

- **run_name** (*Optional[str]*) – Name of the task's current run.
- **state** (*Optional[TaskState]*) – The current state of the task, e.g. "running", "finished", etc.
- **started** (*Optional[datetime]*) – The start time of the task.
- **ended** (*Optional[datetime]*) – The end time of the task.
- **run_history** (*Optional[Dict]*) – Holds the task ids of all predecessor task including the task itself.
- **sweep_counter** (*Optional[str]*) – A dynamically created counter to distinguish different task instances with the same run name in the results store. E.g. used in the LocalFileStore to name run directories.
- **unique_config_hash** (*Optional[str]*) – An 8 character hash of the unique run config.
- **unique_config** (*Optional[Union[Dict, MetaDict]]*) – Unique config of the task. Includes all predecessor task configs as well to uniquely define a task.
- **results_store** (*Optional[ResultsStore]*) – An instance of results store for results management. If nothing is provided, a non-persistent InMemoryStore store is used.
- **resource** (*Optional[Any]*) – A resource object that can hold arbitrary data, e.g. gpu or cpu device information. Resource objects can be assigned in a multiprocessing context, so that each worker process uses a dedicated resource, e.g. cuda device.
- **force** (*Optional[bool]*) – Indicator if the task is force executed or not.
- **expects** (*Optional[Dict[str, Parameter]]*) – A dict of expected input arguments and their `inspect.Parameter()` objects of the task's run method signature.
- **reduce** (*Optional[bool]*) – A boolean indicating whether this is a reduce-task. Defaults to `None`.
- **config** (*Optional[Dict[str, Any]]*) – The config dictionary of the task. Only contains the config parameters of the task (no predecessors).

delete(*name*, *task_name=None*, *task_unique_config=None*)

Deletes object with specified name from results store

Parameters

- **name** (*str*) – A unique name given to this object.
- **task_name** (*Optional[str]*) – Task name which saved the object.
- **task_unique_config** (*Optional[Union[Dict, MetaDict]]*) – Unique config which specifies the run of the object.

delete_run(*task_name=None*, *task_unique_config=None*)

Deletes run with specified name from results store

Parameters

- **task_name** (*Optional[str]*) – Task name which saved the object.
- **task_unique_config** (*Optional[Union[Dict, MetaDict]]*) – Unique config which specifies the run of the object.

property duration: Optional[timedelta]

The current running time of the task.

classmethod from_spec(*task_spec*, *half_initialize=False*)

Initializes a Task object from a TaskSpec object.

Parameters

- **task_spec** (TaskSpec) – A task specification object.
- **half_initialize** (*bool*) – A boolean to indicate whether only the parent Task object is initialized and the child class initialization is delayed until `task._init()` is called. The half initialization is only needed internally to create a task object without directly executing the `__init__` method of the actual task implemented by the user.

Returns
A task object (fully or half initialized).

Return type
Task

get_store_context(*task_name=None*, *task_unique_config=None*)

Wrapper to get store specific storage context, e.g. the current run directory for Local File Store

Parameters

- **task_name** (*Optional[str]*) – Task name.
- **task_unique_config** (*Optional[Union[Dict, MetaDict]]*) – Unique config which specifies the run.

Returns
The store context object holding information like the current run dir.

Return type
StoreContext

property id: str

A combination of the run-name and the sweep counter or unique config hash if the former does not exist.

property info: *TaskInfo*

A `TaskInfo` object that summarizes the important task information for serialization.

load(*name*, *task_name=None*, *task_unique_config=None*, ***kwargs*)

Loads the given object from results store.

Parameters

- **name** (*str*) – A unique name given to this object.
- **task_name** (*Optional[str]*) – Task name which saved the loaded object.
- **task_unique_config** (*Optional[Union[Dict, MetaDict]]*) – Unique config which specifies the run of the loaded object.
- ****kwargs** – Additional keyword args.

Returns
The loaded object.

Return type
Any

open(*name=None*, *task_name=None*, *task_unique_config=None*, *mode=None*, *promise=None*, *type_=None*, *sub_dir=None*, ***open_kwargs*)

Wrapper to open a file from Local File Store (only available for Local File Store).

Parameters

- **name** (*Optional[str]*) – An unique name given to this object.
- **task_name** (*Optional[str]*) – Task name which saved the object.
- **task_unique_config** (*Optional[Union[Dict, MetaDict]]*) – Unique config which specifies the run of the object.
- **mode** (*Optional[str]*) – The open mode, e.g. "r", "w", etc.
- **promise** (*Optional[Promise]*) – An optional `Promise` object used for creating the returned file like object.
- **type_** (*Optional[str]*) – Additional type specification (e.g. json, which is to be passed to results store).
- **sub_dir** (*Optional[str]*) – A path of a subdirectory used for opening the file.
- ****open_kwargs** – Additional keyword arguments passed to the registered `open()` function.

Returns

A `File` object that behaves like a file like object.

Return type

Optional[File]

property predecessors: List[Union[*TaskSpec*, *Task*]]

A List of predecessor task/task spec objects, registered to this task/task spec.

required_by(*successor*)

Registers a successor task/task spec object to the current task/task spec

Parameters

successor (*Any*) – A successor task/task spec objects.

requires(**predecessors*)

Registers one or more predecessor task/task spec objects to the current task/task spec

Parameters

***predecessors** (*Union[TaskSpec, List[TaskSpec], Task, List[Task]]*) – A sequence of predecessor task/task spec objects.

abstract run(***results*)

Implementation of core logic of task.

Parameters

****results** – results from predecessors (automatically passed)

save(*obj*, *name*, *type_=None*, ***kwargs*)

Saves the given object to the results store.

Parameters

- **obj** (*Any*) – Any object that is to be saved
- **name** (*str*) – A unique name given to this object
- **type_** (*Optional[str]*) – Additional type specification (e.g. json, which is to be passed to results store). Defaults to `None`.
- ****kwargs** – Additional keyword args.

property successors: List[Union[TaskSpec, Task]]

A List of successor task/task spec objects, registered to this task/task spec.

class fluidml.TaskSpec(*task*, *config=None*, *additional_kwargs=None*, *name=None*, *reduce=None*, *expand=None*, *config_ignore_prefix=None*, *config_group_prefix=None*)

Parameters

- **task** (*Union[Type[Task], Callable]*) –
- **config** (*Optional[Dict[str, Any]]*) –
- **additional_kwargs** (*Optional[Dict[str, Any]]*) –
- **name** (*Optional[str]*) –
- **reduce** (*Optional[bool]*) –
- **expand** (*Optional[str]*) –
- **config_ignore_prefix** (*Optional[str]*) –
- **config_group_prefix** (*Optional[str]*) –

expand()

Expands a task specification based on the provided config.

This function is called internally in `Flow` when building the task graph.

Returns
A list of expanded `Task` objects.

Return type
List[Task]

property predecessors: List[Union[TaskSpec, Task]]

A List of predecessor task/task spec objects, registered to this task/task spec.

required_by(*successor*)

Registers a successor task/task spec object to the current task/task spec

Parameters
successor (*Any*) – A successor task/task spec objects.

requires(**predecessors*)

Registers one or more predecessor task/task spec objects to the current task/task spec

Parameters
***predecessors** (*Union[TaskSpec, List[TaskSpec], Task, List[Task]]*) – A sequence of predecessor task/task spec objects.

property successors: List[Union[TaskSpec, Task]]

A List of successor task/task spec objects, registered to this task/task spec.

fluidml.configure_logging(*level='INFO'*, *rich_logging=True*, *rich_traceback=True*)

A Convenience function to initialize and configure logging in the application.

Parameters

- **level** (*Union[str, int]*) – Logging level to use, e.g. “DEBUG”, “INFO”, etc.
- **rich_logging** (*bool*) – Whether to use the *rich* library to prettify logging.
- **rich_traceback** (*bool*) – Whether to use the *rich* library to prettify error tracebacks.

1.2 fluidml.storage

class fluidml.storage.**File**(*path*, *mode*, *save_fn*, *load_fn*, *open_fn=None*, *load_kwargs=None*, ***open_kwargs*)

Bases: object

A file like wrapper class to support opening files using the LocalFileStore.

Parameters

- **path** (*str*) – The path to the file to open.
- **mode** (*str*) – The open mode, e.g. "r", "w", etc.
- **save_fn** (*Callable*) – A callable used for file.save() calls.
- **load_fn** (*Callable*) – A callable used for file.load() calls.
- **open_fn** (*Optional[Callable]*) – An optional callable used for file opening. The default is the inbuilt open() function.
- **load_kwargs** (*Optional[Dict]*) – Additional keyword arguments passed to the open function.

close()

property closed

flush()

classmethod from_promise(*promise*)

Creates a File object from a *FilePromise*.

Parameters
promise (FilePromise) –

load(***kwargs*)

read(*size=-1*)

Parameters
size (*int*) –

readable()

readline(*size=-1*)

Parameters
size (*int*) –

readlines(*hint=-1*)

Parameters
hint (*int*) –

save(*obj*, ***kwargs*)

seek(*offset*, *whence=0*)

Parameters
offset (*int*) –

seekable()

tell()

truncate(*size=None*)

Parameters
size (*Optional[int]*) –

writable()

write(*obj*)

Parameters
obj (*AnyStr*) –

writelines(*lines*)

Parameters
lines (*List*) –

class fluidml.storage.**FilePromise**(*name*, *path*, *save_fn*, *load_fn*, *open_fn=None*, *mode=None*, *load_kwargs=None*, ***open_kwargs*)

Bases: *Promise*

Parameters

- **name** (*str*) –
- **path** (*str*) –
- **save_fn** (*Callable*) –
- **load_fn** (*Callable*) –
- **open_fn** (*Optional[Callable]*) –
- **mode** (*Optional[str]*) –
- **load_kwargs** (*Optional[Dict]*) –

load(***kwargs*)

Loads the actual object.

class fluidml.storage.**InMemoryStore**(*manager=None*)

Bases: *ResultsStore*

An in-memory results store implemented using a dict.

When multiprocessing is used a `manager.dict()` is used for inter process communication. If no result store is provided to `flow.run()` this in-memory store is the default.

Parameters
manager (*Optional[Manager]*) –

delete(*name*, *task_name*, *task_unique_config*)

Method to delete any artifact.

Parameters

- **name** (*str*) – The object name.
- **task_name** (*str*) – The task name that saved the object.
- **task_unique_config** (*Dict*) – Unique config which specifies the run of the object.

delete_run(*task_name*, *task_unique_config*)

Method to delete all task results from a given run config

Parameters

- **task_name** (*str*) – Task name which saved the object.
- **task_unique_config** (*Dict*) – Unique config which specifies the run of the object.

get_context(*task_name*, *task_unique_config*)

Wrapper to get store specific storage context, e.g. the current run directory for Local File Store

Parameters

- **task_name** (*str*) – Task name.
- **task_unique_config** (*Dict*) – Unique config which specifies the run.

Returns
The store context object holding information like the current run dir.

Return type
StoreContext

load(*name*, *task_name*, *task_unique_config*, ***kwargs*)

Loads the given object from results store based on its name, task_name and task_config if it exists.

Parameters

- **name** (*str*) – A unique name given to this object.
- **task_name** (*str*) – Task name which saved the loaded object.
- **task_unique_config** (*Dict*) – Unique config which specifies the run of the loaded object.
- ****kwargs** – Additional keyword argumentss.

Returns
The loaded object.

Return type
Optional[*Any*]

save(*obj*, *name*, *task_name*, *task_unique_config*, *type_=None*, ***kwargs*)

In-memory save function. Adds individual object to in-memory store.

Parameters

- **obj** (*Any*) –
- **name** (*str*) –
- **task_name** (*str*) –
- **task_unique_config** (*Dict*) –
- **type_** (*Optional[str]*) –

class fluidml.storage.**LazySweep**(*value*, *config*)

Bases: object

A lazy variation of the Sweep class.

Parameters

- **value** (Promise) –

- **config** (*MetaDict*) –

config: MetaDict

The unique config of the object#s task.

value: *Promise*

The lazy value of the object.

class fluidml.storage.**LocalFileStore**(*base_dir*)

Bases: *ResultsStore*

A local file store that allows to easily save and load task results to/from a base directory in a file system.

Out of the box the local file store supports three common file types, "json", "pickle" and "text". It can be easily extended to arbitrary file types by subclassing the LocalFileStore and registering new Types to the *self._type_registry* dictionary. A new type needs to register a load and save function using the `TypeInfo` data class.

Parameters
base_dir (*str*) – The base directory that is used to store results from tasks.

base_dir

The base directory that is used to store results from tasks.

type_registry

The dictionary to register new types with a save and load function.

delete(*name*, *task_name*, *task_unique_config*)

Deletes an object from the local file store.

The object is deleted based on the name and the provided task name and unique task config.

Parameters

- **name** (*str*) – The name of the to be deleted object.
- **task_name** (*str*) – Task name which saved the object.
- **task_unique_config** (*Dict*) – Unique config which specifies the run of the object.

delete_run(*task_name*, *task_unique_config*)

Deletes an entire task run directory from the local file store.

The run is deleted based on the task name and the unique task config.

Parameters

- **task_name** (*str*) – The name of the task.
- **task_unique_config** (*Dict*) – Unique config which specifies the run of the object.

get_context(*task_name*, *task_unique_config*)

Method to get the current task's storage context.

E.g. the current run directory in case of LocalFileStore. Creates a new run dir if none exists.

Parameters

- **task_name** (*str*) – Task name.
- **task_unique_config** (*Dict*) – Unique config which specifies the run.

Return type
StoreContext

load(*name*, *task_name*, *task_unique_config*, *lazy=False*, ***kwargs*)

Loads an object from the local file store.

The object is retrieved based on the name and the provided task name and unique task config.

Parameters

- **name** (*str*) – An unique name given to this object.
- **task_name** (*str*) – Task name which saves the object.
- **task_unique_config** (*Dict*) – Unique config which specifies the run of the object.
- **lazy** (*bool*) – A boolean whether the object should be lazily loaded. If True, a `FilePromise` object will be returned, that can be loaded into memory on demand with the `promise.load()` function.
- ****kwargs** – Additional keyword arguments passed to the registered `load()` function.

Returns

The specified object if it is found.

Return type

Optional[*Any*]

open(*name=None*, *task_name=None*, *task_unique_config=None*, *mode=None*, *promise=None*, *type_=None*, *sub_dir=None*, ***open_kwargs*)

Wrapper to open a file from Local File Store (only available for Local File Store).

It returns a file like object that has additional `save()` and `load()` methods that can be used to save/load objects with a registered type to/from the file store. The `File` like object allows for an incremental write or read process of objects that for example don't fit into memory.

Parameters

- **name** (*Optional[str]*) – An unique name given to this object.
- **task_name** (*Optional[str]*) – Task name which saved the object.
- **task_unique_config** (*Optional[Dict]*) – Unique config which specifies the run of the object.
- **mode** (*Optional[str]*) – The open mode, e.g. "r", "w", etc.
- **promise** (*Optional[FilePromise]*) – An optional `Promise` object used for creating the returned file like object.
- **type_** (*Optional[str]*) – Additional type specification (e.g. json, which is to be passed to results store).
- **sub_dir** (*Optional[str]*) – A path of a subdirectory used for opening the file.
- ****open_kwargs** – Additional keyword arguments passed to the registered `open()` function.

Returns

A `File` object that wraps a file like object and enables incremental result store reading and writing.

Return type

Optional[File]

property run_info

The current run info of a task, which is used for naming newly created directories.

save(*obj*, *name*, *type_*, *task_name*, *task_unique_config*, *sub_dir=None*, *mode=None*, *open_kwargs=None*, *load_kwargs=None*, ***kwargs*)

Saves an object to the local file store.

If no task and run directory for the provided unique config exists, a new directory will be created.

Parameters

- **obj** (*Any*) – The object to save.
- **name** (*str*) – An unique name given to this object.
- **type_** (*str*) – Additional type specification (e.g. json, which is to be passed to results store).
- **task_name** (*str*) – Task name which saves the object.
- **task_unique_config** (*Dict*) – Unique config which specifies the run of the object.
- **sub_dir** (*Optional[str]*) – A path of a subdirectory used for saving the file.
- **mode** (*Optional[str]*) – The mode to save the file, e.g. "w" or "wb".
- **open_kwargs** (*Optional[Dict[str, Any]]*) – Additional keyword arguments passed to the registered `open()` function.
- **load_kwargs** (*Optional[Dict[str, Any]]*) – Additional keyword arguments passed to the registered `load()` function.
- ****kwargs** – Additional keyword arguments passed to the registered `save()` function.

class fluidml.storage.**MongoDBStore**(*db*, *collection_name=None*, *host=None*)

Bases: *ResultsStore*

A mongo db result store implementation.

Parameters

- **db** (*str*) – The database name to be used.
- **collection_name** (*Optional[str]*) – The name of the collection.
- **host** (*Optional[str]*) – The host name of the :program: *mongod* instance.

delete(*name*, *task_name*, *task_unique_config*)

Query method to delete an object based on its name, task_name and task_config if it exists

Parameters

- **name** (*str*) –
- **task_name** (*str*) –
- **task_unique_config** (*Dict*) –

delete_run(*task_name*, *task_unique_config*)

Query method to delete a run document based on its task_name and task_config if it exists

Parameters

- **task_name** (*str*) –
- **task_unique_config** (*Dict*) –

get_context(*task_name*, *task_unique_config*)

Wrapper to get store specific storage context, e.g. the current run directory for Local File Store

Parameters

- **task_name** (*str*) – Task name.
- **task_unique_config** (*Dict*) – Unique config which specifies the run.

Returns
The store context object holding information like the current run dir.

Return type
StoreContext

load(*name*, *task_name*, *task_unique_config*, *lazy=False*)

Query method to load an object based on its name, task_name and task_config if it exists

Parameters

- **name** (*str*) –
- **task_name** (*str*) –
- **task_unique_config** (*Dict*) –
- **lazy** (*bool*) –

Return type
Optional[*Any*]

save(*obj*, *name*, *type_*, *task_name*, *task_unique_config*, ***kwargs*)

Method to save/update any artifact

Parameters

- **obj** (*Any*) –
- **name** (*str*) –
- **type_** (*str*) –
- **task_name** (*str*) –
- **task_unique_config** (*Dict*) –

class fluidml.storage.**Promise**

Bases: ABC

An interface for future objects, that can be lazy loaded.

abstract load(***kwargs*)

Loads the actual object.

class fluidml.storage.**ResultsStore**

Bases: ABC

The interface of a results store.

abstract delete(*name*, *task_name*, *task_unique_config*)

Method to delete any artifact.

Parameters

- **name** (*str*) – The object name.

- **task_name** (*str*) – The task name that saved the object.
- **task_unique_config** (*Dict*) – Unique config which specifies the run of the object.

abstract delete_run(*task_name*, *task_unique_config*)

Method to delete all task results from a given run config

Parameters

- **task_name** (*str*) – Task name which saved the object.
- **task_unique_config** (*Dict*) – Unique config which specifies the run of the object.

abstract get_context(*task_name*, *task_unique_config*)

Wrapper to get store specific storage context, e.g. the current run directory for Local File Store

Parameters

- **task_name** (*str*) – Task name.
- **task_unique_config** (*Dict*) – Unique config which specifies the run.

Returns
The store context object holding information like the current run dir.

Return type
StoreContext

get_results(*task_name*, *task_unique_config*)

Collects all saved results based that have been tracked when using `Task.save()`.

Parameters

- **task_name** (*str*) – Task name which saved the object.
- **task_unique_config** (*Dict*) – Unique config which specifies the run of the object.

Returns
A dictionary of all saved result objects.

Return type
Optional[*Dict*]

is_finished(*task_name*, *task_unique_config*)

Checks if a task is finished.

Parameters

- **task_name** (*str*) – Task name which saved the object.
- **task_unique_config** (*Dict*) – Unique config which specifies the run of the object.

Returns
A boolean indicating whether the task is finished or not.

Return type
bool

abstract load(*name*, *task_name*, *task_unique_config*, ***kwargs*)

Loads the given object from results store based on its name, task_name and task_config if it exists.

Parameters

- **name** (*str*) – A unique name given to this object.
- **task_name** (*str*) – Task name which saved the loaded object.

- **task_unique_config** (*Dict*) – Unique config which specifies the run of the loaded object.
- ****kwargs** – Additional keyword argumentss.

Returns
The loaded object.

Return type
Optional[*Any*]

property lock

open(*name=None*, *task_name=None*, *task_unique_config=None*, *mode=None*, *promise=None*, *type_=None*, *sub_dir=None*, ***open_kwargs*)

Method to open a file from Local File Store (only available for file stores).

Parameters

- **name** (*Optional[str]*) – An unique name given to this object.
- **task_name** (*Optional[str]*) – Task name which saved the object.
- **task_unique_config** (*Optional[Dict]*) – Unique config which specifies the run of the object.
- **mode** (*Optional[str]*) – The open mode, e.g. "r", "w", etc.
- **promise** (*Optional[Promise]*) – An optional `Promise` object used for creating the returned file like object.
- **type_** (*Optional[str]*) – Additional type specification (e.g. json, which is to be passed to results store).
- **sub_dir** (*Optional[str]*) – A path of a subdirectory used for opening the file.
- ****open_kwargs** – Additional keyword arguments passed to the registered `open()` function.

Returns
A `File` object that behaves like a file like object.

abstract save(*obj*, *name*, *type_*, *task_name*, *task_unique_config*, ***kwargs*)

Method to save/update any artifact.

Parameters

- **obj** (*Any*) – The object to save/update
- **name** (*str*) – The object name.
- **type_** (*str*) – The type of the object. Only required for file stores.
- **task_name** (*str*) – The task name that saves/updates the object.
- **task_unique_config** (*Dict*) – The unique config of that task.
- ****kwargs** – Additional keyword arguments.

class fluidml.storage.**StoreContext**(*run_dir=None*, *sweep_counter=None*)

Bases: `object`

The store context of the current task.

Parameters

- **run_dir** (*Optional[str]*) –
- **sweep_counter** (*Optional[str]*) –

run_dir: Optional[str] = None

The run directory of the task. Relevant for File Stores.

sweep_counter: Optional[str] = None

The sweep counter of the task. Relevant for File Stores. A dynamically created counter to distinguish different task instances with the same run name in the results store.

class fluidml.storage.**Sweep**(*value*, *config*)

Bases: object

A sweep class holding the value and config of a specific task result.

List of Sweeps are the standard inputs for `reduce` tasks that gather results from grid search expanded tasks.

Parameters

- **value** (*Any*) –
- **config** (*MetaDict*) –

config: MetaDict

The unique config of the object#s task.

value: Any

The value of the object.

class fluidml.storage.**TypeInfo**(*save_fn*, *load_fn*, *extension=None*, *is_binary=None*, *open_fn=None*, *needs_path=False*)

Bases: object

Initializes saving and loading information for a specific type.

Parameters

- **save_fn** (*Callable*) –
- **load_fn** (*Callable*) –
- **extension** (*Optional[str]*) –
- **is_binary** (*Optional[bool]*) –
- **open_fn** (*Optional[Callable]*) –
- **needs_path** (*bool*) –

extension: Optional[str] = None

File extension the object is saved with.

is_binary: Optional[bool] = None

Read, write and append in binary mode.

load_fn: Callable

Load function used to load the object from store.

needs_path: bool = False

false.

Type

Whether save and load fn should operate on path and not on file like object. Default

open_fn: Optional[Callable] = None

Function used to open a file object (default is builtin open()).

save_fn: Callable

Save function used to save the object to store.

1.3 fluidml.task

class fluidml.task.**TaskInfo**(*, *project_name*, *run_name*, *state=None*, *started=None*, *ended=None*, *duration=None*, *run_history=None*, *sweep_counter=None*, *unique_config_hash=None*, *id=None*)

Bases: BaseModel

A wrapper class to store important task information.

Used for serializing the information to the results store.

Parameters

- **project_name** (*str*) –
- **run_name** (*str*) –
- **state** (*Optional[TaskState]*) –
- **started** (*Optional[datetime]*) –
- **ended** (*Optional[datetime]*) –
- **duration** (*Optional[timedelta]*) –
- **run_history** (*Optional[Dict]*) –
- **sweep_counter** (*Optional[str]*) –
- **unique_config_hash** (*Optional[str]*) –
- **id** (*Optional[str]*) –

duration: Optional[timedelta]

The current running time of the task.

ended: Optional[datetime]

The end time of the task.

id: Optional[str]

A combination of the run-name and the sweep counter or unique config hash if the former does not exist.

model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True, 'use_enum_values': True}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'duration': FieldInfo(annotation=Union[timedelta, NoneType], required=False), 'ended': FieldInfo(annotation=Union[datetime, NoneType], required=False), 'id': FieldInfo(annotation=Union[str, NoneType], required=False), 'project_name': FieldInfo(annotation=str, required=True), 'run_history': FieldInfo(annotation=Union[Dict, NoneType], required=False), 'run_name': FieldInfo(annotation=str, required=True), 'started': FieldInfo(annotation=Union[datetime, NoneType], required=False), 'state': FieldInfo(annotation=Union[TaskState, NoneType], required=False), 'sweep_counter': FieldInfo(annotation=Union[str, NoneType], required=False), 'unique_config_hash': FieldInfo(annotation=Union[str, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

project_name: str

Name of the project.

run_history: Optional[Dict]

Holds the task ids of all predecessor task including the task itself.

run_name: str

Name of the task's current run.

started: Optional[datetime]

The start time of the task.

state: Optional[*TaskState*]

The current state of the task, e.g. "running", "finished", etc.

sweep_counter: Optional[str]

A dynamically created counter to distinguish different task instances with the same run name in the results store. E.g. used in the LocalFileStore to name run directories.

unique_config_hash: Optional[str]

An 8 character hash of the unique run config.

class fluidml.task.**TaskState**(*value*)

Bases: str, Enum

The state of a task.

FAILED = 'failed'

Task failed due to an unexpected error.

FINISHED = 'finished'

Task finished successfully.

KILLED = 'killed'

Task has been killed by the user via KeyBoardInterrupt.

PENDING = 'pending'

Task has not been scheduled and processed, yet.

RUNNING = 'running'

Task is currently running.

SCHEDULED = 'scheduled'

Task has been scheduled for processing.

UPSTREAM_FAILED = 'upstream_failed'

Task failed/could not be executed due to one or more upstream task failures.

1.4 fluidml.visualization

fluidml.visualization.**visualize_graph_in_console**(*graph*, *use_pager=True*, *use_unicode=False*)

Visualizes the task graph by rendering it to the console.

When using a pager the keyboard input ":q" is required to continue.

Parameters

- **graph** (*nx.DiGraph*) – A networkx directed graph object
- **use_pager** (*bool*) – If true, tries rendering via pager (defaulting to print), if false, print
- **use_unicode** (*bool*) – Renders the graph in unicode if console supports it

fluidml.visualization.**visualize_graph_interactive**(*graph*, *plot_width=500*, *plot_height=200*, *node_width=50*, *node_height=50*, *scale_width=True*, *browser=None*)

Visualizes the task graph interactively in a browser or jupyter notebook.

Parameters

- **graph** (*Graph*) – A networkx directed graph object
- **plot_width** (*int*) – The width of the plot.
- **plot_height** (*int*) – The height of the plot.
- **node_width** (*int*) – Influences the horizontal space between nodes.
- **node_height** (*int*) – Influences the vertical space between nodes.
- **scale_width** (*bool*) – If true, scales the graph to the screen width.
- **browser** (*Optional[str]*) – If provided, renders the graph in the browser, e.g. "chrome" or "firefox". Note the browser might need to be registered using Python's `webbrowser` library.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

f

INDEX

M

N

O

P

R

S

T

U

V

W