PapersWithCode Client

Release 0.3.1

Viktor Kerkez <alefnula@gmail.com>

Dec 02, 2021

CONTENTS

This documentation details how to use the `paperswithcode` library to connect with PapersWithCode.

CHAPTER

ONE

CONTENTS

1.1 Installation

The library requires Python 3.6+ and can be installed via pip:

```
$ pip install paperswithcode-client
```

Usually you would want to create a virtual environment before installing the library:

```
$ python3 -m venv pwc
$ source pwc/bin/activate
$ pip install paperswithcode-client
```

1.2 Quickstart

Library is designed to work with python objects. All of the models used as return values in the library are described in the API part of the documentation.

To use the library you only need to import and instantiate the client and start calling methods on it:

```
>>> from paperswithcode import PapersWithCodeClient
>>> client = PapersWithCodeClient()
>>> papers_page = client.paper_list()
>>> papers_page.count
175834
>>> papers_page.next_page
2
>>> paper = papers_page.results[0]
>>> paper.id
'efficient-methods-for-incorporating-knowledge'
>>> paper.title
'Efficient Methods for Incorporating Knowledge into Topic Models'
```

Same principle is used for all models.

```
>>> dataset_page = client.dataset_list()
>>> dataset_page.count
2782
>>> dataset_page.results[0].id
```

(continues on next page)

(continued from previous page)

```
'hci'
>>> dataset_page.results[0].name
'HCI'
```

For nested queries you will need to provide the required id's:

```
>>> conference_page = client.conference_list()
>>> conference = conference_page.results[0]
>>> conference.id
'eccv'
>>> proceedings_page = client.proceeding_list(conference_id=conference.id)
>>> proceeding = proceedings_page.results[0]
>>> proceeding.id
'eccv-2018'
>>> papers = client.proceeding_paper_list(
    conference_id=conference.id, proceeding_id=proceeding.id
)
>>> papers[0].title
'Person Search by Multi-Scale Matching'
```

1.3 Mirroring your competition on Papers with Code

For more information on how to mirror your competition, please refer to the README file in the paperswithcode-client repository.

CHAPTER

TWO

API DOCUMENTATION

2.1 PapersWithCode Client api documentation

2.1.1 PapersWithCode Client Models

Paper Models

class paperswithcode.models.paper.**Paper**(*, *id: str*, *arxiv_id: str = None*, *nips_id: str = None*, *url_abs: str*, *url_pdf: str*, *title: str*, *abstract: str*, *authors: List[str]*, *published: datetime.date = None*, *conference: str = None*, *conference_url_abs: str = None*, *conference_url_pdf: str = None*, *proceeding: str = None*)

Paper object.

id
Paper ID.

Type str

arxiv_id
ArXiv ID.

Type str, optional

nips_id
NIPS Conference ID.

Type str, optional

url_abs
URL to the paper abstract.

Type str

url_pdf
URL to the paper PDF.

Type str

title
Paper title.

Type str

abstract
Paper abstract.

Type str

authors
List of paper authors.

Type List[str]

published
Paper publication date.

Type date, optional

conference
ID of the conference in which the paper was published.

Type str, optional

conference_url_abs
URL to the conference paper page.

Type str, optional

conference_url_pdf
URL to the conference paper PDF.

Type str, optional

proceeding
ID of the conference proceeding in which the paper was published.

Type str, optional

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

class paperswithcode.models.paper.**Papers**(*, *count: int*, *next_page: int = None*, *previous_page: int = None*, *results: List[*paperswithcode.models.paper.Paper*]*)

Object representing a paginated page of papers.

count
Number of elements matching the query.

Type int

next_page
Number of the next page.

Type int, optional

previous_page
Number of the previous page.

Type int, optional

results
List of papers on this page.

Type List[*Paper*]

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

Repository Models

class paperswithcode.models.repository.**Repository**(*, *url: str*, *owner: str*, *name: str*, *description: str*, *stars: int*, *framework: str*, *is_official: bool = None*)

Repository object.

url
URL of the repository.

Type str

owner
Repository owner.

Type str

name
Repository name.

Type str

description
Repository description.

Type str

stars
Number of repository stars.

Type int

framework
Implementation framework (TensorFlow, PyTorch, MXNet, Torch, Jax, Caffee2…).

Type str

is_official
Is this an official implementation of the paper. Available only when listing repositories for a specific paper.

Type bool

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

class paperswithcode.models.repository.**Repositories**(*, *count: int*, *next_page: int = None*, *previous_page: int = None*, *results: List[*paperswithcode.models.repository.Repository*]*)

Object representing a paginated page of repositories.

count
Number of elements matching the query.

Type int

next_page
Number of the next page.

Type int, optional

previous_page
Number of the previous page.

Type int, optional

results
List of repositories on this page.

Type List[*Repository*]

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

Paper Repository Models

class paperswithcode.models.paper_repo.**PaperRepo**(*, *paper:* paperswithcode.models.paper.Paper, *repository:* paperswithcode.models.repository.Repository = *None*, *is_official: bool*)

Paper <-> Repository object.

paper
Paper objects.

Type *Paper*

repository
Repository object.

Type *Repository*, optional

is_official
Is this the official implementation.

Type bool

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

class paperswithcode.models.paper_repo.**PaperRepos**(*, *count: int*, *next_page: int = None*, *previous_page: int = None*, *results: List[*paperswithcode.models.paper_repo.PaperRepo*]*)

Object representing a paginated page of paper<->repos.

count
Number of elements matching the query.

Type int

next_page
Number of the next page.

Type int, optional

previous_page
Number of the previous page.

Type int, optional

results
List of paper<->repos on this page.

Type List[*PaperRepo*]

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

Author Models

class paperswithcode.models.author.**Author**(*, *id: str*, *full_name: str*)

Author object.

id

Author ID.

Type str

full_name

Author full name.

Type str, optional

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

class paperswithcode.models.author.**Authors**(*, *count: int*, *next_page: int = None*, *previous_page: int = None*, *results: List[*paperswithcode.models.author.Author*]*)

Object representing a paginated page of authors.

count

Number of elements matching the query.

Type int

next_page

Number of the next page.

Type int, optional

previous_page

Number of the previous page.

Type int, optional

results

List of authors on this page.

Type List[*Author*]

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

Conference Models

class paperswithcode.models.conference.**Conference**(*, *id: str*, *name: str*)

Conference object.

id

Conference ID.

Type str

name

Conerence name.

Type str

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

class paperswithcode.models.conference.**Conferences**(*, *count: int*, *next_page: int = None*, *previous_page: int = None*, *results: List[*paperswithcode.models.conference.Conference*]*)

Object representing a paginated page of conferences.

count

Number of elements matching the query.

Type int

next_page

Number of the next page.

Type int, optional

previous_page

Number of the previous page.

Type int, optional

results

List of conferences on this page.

Type List[*Conference*]

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

class paperswithcode.models.conference.**Proceeding**(*, *id: str*, *year: int = None*, *month: int = None*)

Conference proceeding object.

id

Proceeding ID.

Type str

year

Year in which the proceeding was held.

Type int, optinoal

month

Month in which the proceedingt was held.

Type int, optional

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

class paperswithcode.models.conference.**Proceedings**(*, *count: int*, *next_page: int = None*, *previous_page: int = None*, *results: List[*paperswithcode.models.conference.Proceeding*]*)

Object representing a paginated page of proceedings.

count

Number of elements matching the query.

Type int

next_page

Number of the next page.

Type int, optional

previous_page
Number of the previous page.

Type int, optional

results
List of proceedings on this page.

Type List[*Proceeding*]

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

Task Models

class paperswithcode.models.task.**Area**(*, *id: str*, *name: str*)
Area object.

Representing an area of research.

id
Area ID.

Type str

name
Area name.

Type str

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

class paperswithcode.models.task.**Areas**(*, *count: int*, *next_page: int = None*, *previous_page: int = None*, *results: List[*paperswithcode.models.task.Area*]*)
Object representing a paginated page of areas.

count
Number of elements matching the query.

Type int

next_page
Number of the next page.

Type int, optional

previous_page
Number of the previous page.

Type int, optional

results
List of areas on this page.

Type List[*Area*]

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

class paperswithcode.models.task.**Task**(*, *id: str*, *name: str*, *description: str*)
Task object.

id
Task ID.

Type str

name
Task name.

Type str

description
Task description.

Type str

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

class paperswithcode.models.task.**TaskCreateRequest**(*, *name: str*, *description: str = ''*, *area: str = None*, *parent_task: str = None*)

Task object.

name
Task name.

Type str

description
Task description.

Type str

area
Task area ID or area name.

Type str, optional

parent_task
ID of the parent task.

Type str, optional

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

class paperswithcode.models.task.**TaskUpdateRequest**(*, *name: str = None*, *description: str = None*, *area: str = None*, *parent_task: str = None*)

Evaluation table row object.

name
Task name.

Type str, optional

description
Task description.

Type str, optional

area
Task area ID.

Type str, optional

parent_task
ID of the parent task.

Type str, optional

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

class paperswithcode.models.task.**Tasks**(*, *count: int*, *next_page: int = None*, *previous_page: int = None*, *results: List[*paperswithcode.models.task.Task*]*)

Object representing a paginated page of tasks.

count
Number of elements matching the query.

Type int

next_page
Number of the next page.

Type int, optional

previous_page
Number of the previous page.

Type int, optional

results
List of tasks on this page.

Type List[*Task*]

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

Dataset Models

class paperswithcode.models.dataset.**Dataset**(*, *id: str*, *name: str*, *full_name: str = None*, *url: str = None*)

Dataset object.

id
Dataset ID.

Type str

name
Dataset name.

Type str

full_name
Dataset full name.

Type str, optional

url
URL for dataset download.

Type str, optional

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

class paperswithcode.models.dataset.**DatasetCreateRequest**(*, *name: str*, *full_name: str = None*, *url: str = None*)

Task object.

name
Dataset name.

Type str

full_name
Dataset full name.

Type str, optional

url
Dataset url.

Type str, optional

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

class paperswithcode.models.dataset.**DatasetUpdateRequest**(*, *name: str = None*, *url: str = None*)

Evaluation table row object.

name
Dataset name.

Type str, optional

url
Dataset url.

Type str, optional

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

class paperswithcode.models.dataset.**Datasets**(*, *count: int*, *next_page: int = None*, *previous_page: int = None*, *results: List[*paperswithcode.models.dataset.Dataset*]*)

Object representing a paginated page of datasets.

count
Number of elements matching the query.

Type int

next_page
Number of the next page.

Type int, optional

previous_page
Number of the previous page.

Type int, optional

results
List of datasets on this page.

Type List[*Dataset*]

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

Method Models

class `paperswithcode.models.method.`**`Method`**(*, *id: str*, *name: str*, *full_name: str*, *description: str*, *paper: str = None*)

Method object.

`id`

Method ID.

Type str

`name`

Method short name.

Type str

`full_name`

Method full name.

Type str

`description`

Method description.

Type str

`paper`

ID of the paper that describes the method.

Type str, optional

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

class `paperswithcode.models.method.`**`Methods`**(*, *count: int*, *next_page: int = None*, *previous_page: int = None*, *results: List[*paperswithcode.models.method.Method*]*)

Object representing a paginated page of methods.

`count`

Number of elements matching the query.

Type int

`next_page`

Number of the next page.

Type int, optional

`previous_page`

Number of the previous page.

Type int, optional

`results`

List of methods on this page.

Type List[*Method*]

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

Evaluation Table Models

class `paperswithcode.models.evaluation.`**`Metric`**(*, *id: str*, *name: str*, *description: str*, *is_loss: bool*)

Metric object.

Metric used for evaluation.

`id`

Metric id.

Type str

`name`

Metric name.

Type str

`description`

Metric description.

Type str

`is_loss`

Is this a loss metric.

Type bool

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

class `paperswithcode.models.evaluation.`**`Metrics`**(*, *count: int*, *next_page: int = None*, *previous_page: int = None*, *results: List[*paperswithcode.models.evaluation.metric.Metric*]*)

Object representing a paginated page of metrics.

`count`

Number of elements matching the query.

Type int

`next_page`

Number of the next page.

Type int, optional

`previous_page`

Number of the previous page.

Type int, optional

`results`

List of metrics on this page.

Type List[*Metric*]

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

class paperswithcode.models.evaluation.**MetricCreateRequest**(*, *name: str*, *description: str*, *is_loss: bool*)

Metric object.

Metric used for evaluation.

name
Metric name.

Type str

description
Metric description.

Type str

is_loss
Is this a loss metric.

Type bool

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

class paperswithcode.models.evaluation.**MetricUpdateRequest**(*, *name: str = None*, *description: str = None*, *is_loss: bool = None*)

Metric object.

Metric used for evaluation.

name
Metric name.

Type str, optional

description
Metric description.

Type str, optional

is_loss
Is this a loss metric.

Type bool, optional

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

class paperswithcode.models.evaluation.**Result**(*, *id: str*, *best_rank: int = None*, *metrics: dict*, *methodology: str*, *uses_additional_data: bool*, *paper: str = None*, *best_metric: str = None*, *evaluated_on: str = None*, *external_source_url: str = None*)

Evaluation table row object.

id
Result id.

Type str

best_rank
Best rank of the row.

Type int, optional

metrics
Dictionary of metrics and metric values.

Type dict

methodology
Methodology used for this implementation.

Type str

uses_additional_data
Does this evaluation uses additional data not provided in the dataset used for other evaluations.

Type bool

paper
Paper describing the evaluation.

Type str, optional

best_metric
Name of the best metric.

Type str, optional

evaluated_on
Date of the result evaluation in YYYY-MM-DD format.

Type str, optional

external_source_url
The URL to the external source (eg competition).

Type str, option

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

class paperswithcode.models.evaluation.**Results**(*, *count: int*, *next_page: int = None*, *previous_page: int = None*, *results: List[*paperswithcode.models.evaluation.result.Result*]*)

Object representing a paginated page of results.

count
Number of elements matching the query.

Type int

next_page
Number of the next page.

Type int, optional

previous_page
Number of the previous page.

Type int, optional

results
List of results on this page.

Type List[*Result*]

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

class paperswithcode.models.evaluation.**ResultCreateRequest**(*, *metrics: dict*, *methodology: str*, *uses_additional_data: bool = False*, *paper: str = None*, *evaluated_on: str = None*, *external_source_url: str = None*)

Evaluation table row object.

metrics
Dictionary of metrics and metric values.

Type dict

methodology
Methodology used for this implementation.

Type str

uses_additional_data
Does this evaluation uses additional data not provided in the dataset used for other valuations.

Type bool, optional

paper
Paper describing the evaluation.

Type str, optional

evaluated_on
Date of the result evaluation: YYYY-MM-DD format.

Type str, optional

external_source_url
The URL to the external source (eg competition).

Type str, option

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

class paperswithcode.models.evaluation.**ResultUpdateRequest**(*, *metrics: dict = None*, *methodology: str = None*, *uses_additional_data: bool = None*, *paper: str = None*, *evaluated_on: str = None*, *external_source_url: str = None*)

Evaluation table row object.

metrics
Dictionary of metrics and metric values.

Type dict, optional

methodology
Methodology used for this implementation.

Type str, optional

uses_additional_data
Does this evaluation uses additional data not provided in the dataset used for other evaluations.

Type bool, optional

paper
Paper describing the evaluation.

Type str, optional

evaluated_on
Date of the result evaluation: YYYY-MM-DD format.

Type datetime, optional

external_source_url
The URL to the external source (eg competition).

Type str, option

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

class paperswithcode.models.evaluation.**EvaluationTable**(*, *id: str*, *task: str*, *dataset: str*, *description: str = ''*, *mirror_url: str = None*)

Evaluation table object.

id
Evaluation table ID.

Type str

task
ID of the task used in evaluation.

Type str

dataset
ID of the dataset used in evaluation.

Type str

description
Evaluation table description.

Type str

mirror_url
URL to the evaluation table that this table is a mirror of.

Type str, optional

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

class paperswithcode.models.evaluation.**EvaluationTables**(*, *count: int*, *next_page: int = None*, *previous_page: int = None*, *results: List[*paperswithcode.models.evaluation.table.EvaluationTable]

Object representing a paginated page of evaluation tables.

count
Number of elements matching the query.

Type int

next_page
Number of the next page.

Type int, optional

previous_page
Number of the previous page.

Type int, optional

results
List of evaluation tables on this page.

Type List[SotaPartial]

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

class paperswithcode.models.evaluation.**EvaluationTableCreateRequest**(*, *task: str*, *dataset: str*, *description: str = ''*, *mirror_url: str = None*)

Evaluation table create request object.

task
ID of the task used in evaluation.

Type str

dataset
ID of the dataset used in evaluation.

Type str

description
Evaluation table description.

Type str

mirror_url
URL to the evaluation table that this table is a mirror of.

Type str, optional

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

class paperswithcode.models.evaluation.**EvaluationTableUpdateRequest**(*, *task: str = None*, *dataset: str = None*, *description: str = None*, *mirror_url: str = None*)

Evaluation table update request object.

task
ID of the task used in evaluation.

Type str, optional

dataset
ID of the dataset used in evaluation.

Type str, optional

description
Evaluation table description.

Type str, optional

mirror_url
URL to the evaluation table that this table is a mirror of.f

Type str, optional

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

class paperswithcode.models.evaluation.**ResultSyncRequest**(*, *metrics: dict*, *methodology: str*, *paper: str = None*, *uses_additional_data: bool = False*, *external_id: str = ''*, *evaluated_on: str*, *external_source_url: str = None*)

Evaluation table row object.

metrics
Dictionary of metrics and metric values.

Type dict

methodology
Methodology used for this implementation.

Type str

uses_additional_data
Does this evaluation uses additional data not provided in the dataset used for other evaluations.

Type bool

paper
Paper describing the evaluation.

Type str, optional

external_id
Optional external ID used to identify rows when doing sync.

Type str, optional

evaluated_on
Evaluation date in YYYY-MM-DD format

Type str

external_source_url
The URL to the external source (eg competition).

Type str, option

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

class paperswithcode.models.evaluation.**MetricSyncRequest**(*, *name: str*, *description: str = ''*, *is_loss: bool = True*)

Metric object.

Metric used for evaluation.

name
Metric name.

Type str

description
Metric description.

Type str

is_loss
Is this a loss metric.

Type bool

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

class paperswithcode.models.evaluation.**EvaluationTableSyncRequest**(*, *task: str*, *dataset: str*, *description: str = ''*, *mirror_url: str = None*, *external_id: str = None*, *metrics: List[*paperswithcode.models.evaluation.synchron *= None*, *results: List[*paperswithcode.models.evaluation.synchron *= None*)

Evaluation table object.

task
ID of the task used in evaluation.

Type str

dataset
ID of the dataset used in evaluation.

Type str

description
Evaluation table description.

Type str

mirror_url
URL to the evaluation table that this table is a mirror of.

Type str, optional

external_id
Optional external ID used to identify rows when doing sync.

Type str, optional

metric
List of MetricSyncRequest objects used in the evaluation.

Type list

results
List of ResultSyncRequest objects - results of the evaluation.

Type list

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

class paperswithcode.models.evaluation.**ResultSyncResponse**(*, *id: str*, *metrics: dict*, *methodology: str*, *paper: str = None*, *uses_additional_data: bool = False*, *external_id: str = ''*, *evaluated_on: str = None*, *external_source_url: str = None*)

Evaluation table row object.

id
Result id.

Type str

metrics
Dictionary of metrics and metric values.

Type dict

methodology
Methodology used for this implementation.

Type str

uses_additional_data
Does this evaluation uses additional data not provided in the dataset used for other evaluations.

Type bool

paper
Paper describing the evaluation.

Type str, optional

external_id
Optional external ID used to identify rows when doing sync.

Type str, optional

evaluated_on
Evaluation date in YYYY-MM-DD format

Type str, optional

external_source_url
The URL to the external source (eg competition)

Type str, option

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

class paperswithcode.models.evaluation.**MetricSyncResponse**(*, *name: str*, *description: str = ''*, *is_loss: bool = True*)

Metric object.

Metric used for evaluation.

name
Metric name.

Type str

description
Metric description.

Type str

is_loss
Is this a loss metric.

Type bool

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

class paperswithcode.models.evaluation.**EvaluationTableSyncResponse**(*, *id: str*, *task: str*, *dataset: str*, *description: str = ''*, *mirror_url: str = None*, *external_id: str = ''*, *metrics: List[*paperswithcode.models.evaluation.synchro *= None*, *results: List[*paperswithcode.models.evaluation.synchro *= None*)

Evaluation table object.

id
Evaluation table ID.

Type str

task
ID of the task used in evaluation.

Type str

dataset
ID of the dataset used in evaluation.

Type str

description
Evaluation table description.

Type str

mirror_url
URL to the evaluation table that this table is a mirror of.

Type str, optional

external_id
Optional external ID used to identify rows when doing sync.

Type str, optional

metric
List of metrics sync objects used in the evaluation.

Type list

results
List of result sync objects - results of the evaluation.

Type list

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

2.1.2 PapersWithCode Client Class

class paperswithcode.client.**PapersWithCodeClient**(*token=None*, *url=None*)

PapersWithCode client.

__init__(*token=None*, *url=None*)

Initialize self. See help(type(self)) for accurate signature.

search(*q: Optional[str] = None*, *page: int = 1*, *items_per_page: int = 50*) → *paperswithcode.models.paper_repo.PaperRepos*

Search in a similar fashion to the frontpage search.

Parameters

- **q** (*str, optional*) – Filter papers by querying the paper title and abstract.
- **page** (*int*) – Desired page.
- **items_per_page** (*int*) – Desired number of items per page. Default: 50.

Returns PaperRepos object.

Return type *PaperRepos*

paper_list(*q: Optional[str] = None*, *arxiv_id: Optional[str] = None*, *title: Optional[str] = None*, *abstract: Optional[str] = None*, *page: int = 1*, *items_per_page: int = 50*) → *paperswithcode.models.paper.Papers*

Return a paginated list of papers.

Parameters

- **q** (*str, optional*) – Filter papers by querying the paper title and abstract.
- **arxiv_id** (*str, optional*) – Filter papers by arxiv id.
- **title** (*str, optional*) – Filter papers by part of the title.
- **abstract** (*str, optional*) – Filter papers by part of the abstract.
- **page** (*int*) – Desired page.
- **items_per_page** (*int*) – Desired number of items per page. Default: 50.

Returns Papers object.

Return type *Papers*

paper_get(*paper_id: str*) → *paperswithcode.models.paper.Paper*

Return a paper by it's ID.

Parameters **paper_id** (*str*) – ID of the paper.

Returns Paper object.

Return type *Paper*

paper_dataset_list(*paper_id: str*, *page: int = 1*, *items_per_page: int = 50*) → *paperswithcode.models.repository.Repositories*

Return a list of datasets mentioned in the paper..

Parameters

- **paper_id** (*str*) – ID of the paper.
- **page** (*int*) – Desired page.
- **items_per_page** (*int*) – Desired number of items per page. Default: 50.

Returns Datasets object.

Return type *Datasets*

paper_repository_list(*paper_id: str, page: int = 1, items_per_page: int = 50*) → *paperswithcode.models.repository.Repositories*

Return a list of paper implementations.

Parameters

- **paper_id** (*str*) – ID of the paper.
- **page** (*int*) – Desired page.
- **items_per_page** (*int*) – Desired number of items per page. Default: 50.

Returns Repositories object.

Return type *Repositories*

paper_task_list(*paper_id: str, page: int = 1, items_per_page: int = 50*) → *paperswithcode.models.task.Tasks*

Return a list of tasks mentioned in the paper.

Parameters

- **paper_id** (*str*) – ID of the paper.
- **page** (*int*) – Desired page.
- **items_per_page** (*int*) – Desired number of items per page. Default: 50.

Returns Tasks object.

Return type *Tasks*

paper_method_list(*paper_id: str, page: int = 1, items_per_page: int = 50*) → *paperswithcode.models.method.Methods*

Return a list of methods mentioned in the paper.

Parameters

- **paper_id** (*str*) – ID of the paper.
- **page** (*int*) – Desired page.
- **items_per_page** (*int*) – Desired number of items per page. Default: 50.

Returns Methods object.

Return type *Methods*

paper_result_list(*paper_id: str, page: int = 1, items_per_page: int = 50*) → *paperswithcode.models.evaluation.result.Results*

Return a list of evaluation results for the paper.

Parameters

- **paper_id** (*str*) – ID of the paper.
- **page** (*int*) – Desired page.
- **items_per_page** (*int*) – Desired number of items per page. Default: 50.

Returns Results object.

Return type *Results*

repository_list(*q: Optional[str] = None, owner: Optional[str] = None, name: Optional[str] = None, stars: Optional[int] = None, framework: Optional[str] = None, page: int = 1, items_per_page: int = 50*) → *paperswithcode.models.paper.Papers*

Return a paginated list of repositories.

Parameters

- **q** (*str, optional*) – Search all searchable fields.
- **owner** (*str, optional*) – Filter repositories by owner.
- **name** (*str, optional*) – Filter repositories by name.
- **stars** (*int, optional*) – Filter repositories by minimum number of stars.
- **framework** (*str, optional*) – Filter repositories by framework. Available values: tf, pytorch, mxnet, torch, caffe2, jax, paddle, mindspore.
- **page** (*int*) – Desired page.
- **items_per_page** (*int*) – Desired number of items per page. Default: 50.

Returns Repositories object.

Return type *Repositories*

repository_owner_list(*owner: str*) → *paperswithcode.models.repository.Repositories*

List all repositories for a specific repo owner.

Parameters **owner** (*str*) – Repository owner.

Returns Repositories object.

Return type *Repositories*

repository_get(*owner: str, name: str*) → *paperswithcode.models.repository.Repository*

Return a repository by it's owner/name pair.

Parameters

- **owner** (*str*) – Owner name.
- **name** (*str*) – Repository name.

Returns Repository object.

Return type *Repository*

repository_paper_list(*owner: str, name: str, page: int = 1, items_per_page: int = 50*) → *paperswithcode.models.paper.Papers*

List all papers connected to the repository.

Parameters

- **owner** (*str*) – Owner name.
- **name** (*str*) – Repository name.
- **page** (*int*) – Desired page.
- **items_per_page** (*int*) – Desired number of items per page. Default: 50.

Returns Papers object.

Return type *Papers*

author_list(*q: Optional[str] = None, full_name: Optional[str] = None, page: int = 1, items_per_page: int = 50*) → *paperswithcode.models.author.Authors*

Return a paginated list of paper authors.

Parameters

- **q** (*str, optional*) – Search all searchable fields.
- **full_name** (*str, optional*) – Filter authors by part of their full name.
- **page** (*int*) – Desired page.
- **items_per_page** (*int*) – Desired number of items per page. Default: 50.

Returns Repositories object.

Return type *Repositories*

author_get(*author_id: str*) → *paperswithcode.models.author.Author*

Return a specific author selected by its id.

Parameters **author_id** (*str*) – Author id.

Returns Author object.

Return type *Author*

author_paper_list(*author_id: str, page: int = 1, items_per_page: int = 50*) → *paperswithcode.models.paper.Papers*

List all papers connected to the author.

Parameters

- **author_id** (*str*) – Author id.
- **page** (*int*) – Desired page.
- **items_per_page** (*int*) – Desired number of items per page. Default: 50.

Returns Papers object.

Return type *Papers*

conference_list(*q: Optional[str] = None, name: Optional[str] = None, page: int = 1, items_per_page: int = 50*) → *paperswithcode.models.conference.Conferences*

Return a paginated list of conferences.

Parameters

- **q** (*str, optional*) – Search all searchable fields.
- **name** (*str, optional*) – Filter conferences by part of the name.
- **page** (*int*) – Desired page.
- **items_per_page** (*int*) – Desired number of items per page. Default: 50.

Returns Conferences object.

Return type *Conferences*

conference_get(*conference_id: str*) → *paperswithcode.models.conference.Conference*

Return a conference by it's ID.

Parameters **conference_id** (*str*) – ID of the conference.

Returns Conference object.

Return type *Conference*

proceeding_list(*conference_id: str*, *page: int = 1*, *items_per_page: int = 50*) → *paperswithcode.models.conference.Proceedings*

Return a paginated list of conference proceedings.

Parameters

- **conference_id** (*str*) – ID of the conference.
- **page** (*int*) – Desired page.
- **items_per_page** (*int*) – Desired number of items per page. Default: 50.

Returns Proceedings object.

Return type *Proceedings*

proceeding_get(*conference_id: str*, *proceeding_id: str*) → *paperswithcode.models.conference.Proceeding*

Return a conference proceeding by it's ID.

Parameters

- **conference_id** (*str*) – ID of the conference.
- **proceeding_id** (*str*) – ID of the proceeding.

Returns Proceeding object.

Return type *Proceeding*

proceeding_paper_list(*conference_id: str*, *proceeding_id: str*, *page: int = 1*, *items_per_page: int = 50*) → *paperswithcode.models.paper.Papers*

Return a list of papers published in a confernce proceeding.

Parameters

- **conference_id** (*str*) – ID of the conference.
- **proceeding_id** (*str*) – ID of the proceding.
- **page** (*int*) – Desired page.
- **items_per_page** (*int*) – Desired number of items per page. Default: 50.

Returns Papers object.

Return type *Papers*

area_list(*q: Optional[str] = None*, *name: Optional[str] = None*, *page: int = 1*, *items_per_page: int = 50*) → *paperswithcode.models.task.Areas*

Return a paginated list of areas.

Parameters

- **q** (*str, optional*) – Filter areas by querying the area name.
- **name** (*str, optional*) – Filter areas by part of the name.
- **page** (*int*) – Desired page.
- **items_per_page** (*int*) – Desired number of items per page. Default: 50.

Returns Areas object.

Return type *Areas*

area_get(*area_id: str*) → *paperswithcode.models.task.Area*

Return an area by it's ID.

Parameters **area_id** (*str*) – ID of the area.

Returns Area object.

Return type *Area*

area_task_list(*area_id: str*, *page: int = 1*, *items_per_page: int = 50*) → *paperswithcode.models.task.Tasks*

Return a paginated list of tasks in an area.

Parameters

- **area_id** (*str*) – ID of the area.
- **page** (*int*) – Desired page.
- **items_per_page** (*int*) – Desired number of items per page. Default: 50.

Returns Tasks object.

Return type *Tasks*

task_list(*q: Optional[str] = None*, *name: Optional[str] = None*, *page: int = 1*, *items_per_page: int = 50*) → *paperswithcode.models.task.Tasks*

Return a paginated list of tasks.

Parameters

- **q** (*str, optional*) – Filter tasks by querying the task name.
- **name** (*str, optional*) – Filter tasks by part of th name.
- **page** (*int*) – Desired page.
- **items_per_page** (*int*) – Desired number of items per page. Default: 50.

Returns Tasks object.

Return type *Tasks*

task_get(*task_id: str*) → *paperswithcode.models.task.Task*

Return a task by it's ID.

Parameters **task_id** (*str*) – ID of the task.

Returns Task object.

Return type *Task*

task_add(*task:* paperswithcode.models.task.TaskCreateRequest) → *paperswithcode.models.task.Task*

Add a task.

Parameters **task** (TaskCreateRequest) – Task create request.

Returns Created task.

Return type *Task*

task_update(*task_id: str*, *task:* paperswithcode.models.task.TaskUpdateRequest) → *paperswithcode.models.task.Task*

Update a task.

Parameters

- **task_id** (*str*) – ID of the task.
- **task** (TaskUpdateRequest) – Task update request.

Returns Updated task.

Return type *Task*

task_delete(*task_id: str*)

Delete a task.

Parameters **task_id** (*str*) – ID of the task.

task_parent_list(*task_id: str, page: int = 1, items_per_page: int = 50*) → *paperswithcode.models.task.Tasks*

Return a paginated list of parent tasks for a selected task.

Parameters

- **task_id** (*str*) – ID of the task.
- **page** (*int*) – Desired page.
- **items_per_page** (*int*) – Desired number of items per page. Default: 50.

Returns Tasks object.

Return type *Tasks*

task_child_list(*task_id: str, page: int = 1, items_per_page: int = 50*) → *paperswithcode.models.task.Tasks*

Return a paginated list of child tasks for a selected task.

Parameters

- **task_id** (*str*) – ID of the task.
- **page** (*int*) – Desired page.
- **items_per_page** (*int*) – Desired number of items per page. Default: 50.

Returns Tasks object.

Return type *Tasks*

task_paper_list(*task_id: str, page: int = 1, items_per_page: int = 50*) → *paperswithcode.models.paper.Papers*

Return a paginated list of papers for a selected task.

Parameters

- **task_id** (*str*) – ID of the task.
- **page** (*int*) – Desired page.
- **items_per_page** (*int*) – Desired number of items per page. Default: 50.

Returns Papers object.

Return type *Papers*

task_evaluation_list(*task_id: str, page: int = 1, items_per_page: int = 50*) → *paperswithcode.models.evaluation.table.EvaluationTables*

Return a list of evaluation tables for a selected task.

Parameters

- **task_id** (*str*) – ID of the task.
- **page** (*int*) – Desired page.
- **items_per_page** (*int*) – Desired number of items per page. Default: 50.

Returns EvaluationTables object.

Return type *EvaluationTables*

dataset_list(*q: Optional[str] = None, name: Optional[str] = None, full_name: Optional[str] = None, page: int = 1, items_per_page: int = 50*) → *paperswithcode.models.dataset.Datasets*

Return a paginated list of datasets.

Parameters

- **q** (*str, optional*) – Filter datasets by querying the dataset name.
- **name** (*str, optional*) – Filter datasets by their name.
- **full_name** (*str, optional*) – Filter datasets by their full name.
- **page** (*int*) – Desired page.
- **items_per_page** (*int*) – Desired number of items per page. Default: 50.

Returns Datasets object.

Return type *Datasets*

dataset_get(*dataset_id: str*) → *paperswithcode.models.dataset.Dataset*

Return a dastaset by it's ID.

Parameters **dataset_id** (*str*) – ID of the dataset.

Returns Dataset object.

Return type *Dataset*

dataset_add(*dataset:* paperswithcode.models.dataset.DatasetCreateRequest) → *paperswithcode.models.dataset.Dataset*

Add a dataset.

Parameters **dataset** (DatasetCreateRequest) – Dataset create request.

Returns Created dataset.

Return type *Dataset*

dataset_update(*dataset_id: str, dataset:* paperswithcode.models.dataset.DatasetUpdateRequest) → *paperswithcode.models.dataset.Dataset*

Update a dataset.

Parameters

- **dataset_id** (*str*) – ID of the dataset.
- **dataset** (DatasetUpdateRequest) – Dataset update request.

Returns Updated dataset.

Return type *Dataset*

dataset_delete(*dataset_id: str*)

Delete a dataset.

Parameters **dataset_id** (*str*) – ID of the dataset.

dataset_evaluation_list(*dataset_id: str, page: int = 1, items_per_page: int = 50*) → *paperswithcode.models.evaluation.table.EvaluationTables*

Return a list of evaluation tables for a selected dataset.

Parameters

- **dataset_id** (*str*) – ID of the dasaset.
- **page** (*int*) – Desired page.

- **items_per_page** (*int*) – Desired number of items per page. Default: 50.

Returns EvaluationTables object.

Return type *EvaluationTables*

method_list(*q: Optional[str] = None, name: Optional[str] = None, full_name: Optional[str] = None, page: int = 1, items_per_page: int = 50*) → *paperswithcode.models.method.Methods*

Return a paginated list of methods.

Parameters

- **q** (*str, optional*) – Search all searchable fields.
- **name** (*str, optional*) – Filter methods by part of the name.
- **full_name** (*str, optional*) – Filter methods by part of the full name.
- **page** (*int*) – Desired page.
- **items_per_page** (*int*) – Desired number of items per page. Default: 50.

Returns Methods object.

Return type *Methods*

method_get(*method_id*) → *paperswithcode.models.method.Method*

Return a method by it's ID.

Parameters **method_id** (*str*) – ID of the method.

Returns Method object.

Return type *Method*

evaluation_list(*page: int = 1, items_per_page: int = 50*) → *paperswithcode.models.evaluation.table.EvaluationTables*

Return a paginated list of evaluation tables.

Parameters

- **page** (*int*) – Desired page.
- **items_per_page** (*int*) – Desired number of items per page. Default: 50.

Returns Evaluation table page object.

Return type *EvaluationTables*

evaluation_get(*evaluation_id: str*) → *paperswithcode.models.evaluation.table.EvaluationTable*

Return a evaluation table by it's ID.

Parameters **evaluation_id** (*str*) – ID of the evaluation table.

Returns Evaluation table object.

Return type *EvaluationTable*

evaluation_create(*evaluation:* paperswithcode.models.evaluation.table.EvaluationTableCreateRequest) → *paperswithcode.models.evaluation.table.EvaluationTable*

Create an evaluation table.

Parameters **evaluation** (EvaluationTableCreateRequest) – Evaluation table create request object.

Returns The new created evaluation table.

Return type *EvaluationTable*

evaluation_update(*evaluation_id: str*, *evaluation:* paperswithcode.models.evaluation.table.EvaluationTableUpdateRequest) → *paperswithcode.models.evaluation.table.EvaluationTable*

Update an evaluation table.

Parameters

- **evaluation_id** (*str*) – ID of the evaluation table.
- **evaluation** (EvaluationTableUpdateRequest) – Evaluation table update request object.

Returns The updated evaluation table.

Return type *EvaluationTable*

evaluation_delete(*evaluation_id: str*)

Delete an evaluation table.

Parameters **evaluation_id** (*str*) – ID of the evaluation table.

evaluation_metric_list(*evaluation_id: str*, *page: int = 1*, *items_per_page: int = 50*) → *paperswithcode.models.evaluation.metric.Metrics*

List all metrics used in the evaluation table.

Parameters

- **evaluation_id** (*str*) – ID of the evaluation table.
- **page** (*int*) – Desired page.
- **items_per_page** (*int*) – Desired number of items per page. Default: 50.

Returns Metrics object.

Return type *Metrics*

evaluation_metric_get(*evaluation_id: str*, *metric_id: str*) → *paperswithcode.models.evaluation.metric.Metric*

Get a metrics used in the evaluation table.

Parameters

- **evaluation_id** (*str*) – ID of the evaluation table.
- **metric_id** (*str*) – ID of the metric.

Returns Requested metric.

Return type *Metric*

evaluation_metric_add(*evaluation_id: str*, *metric:* paperswithcode.models.evaluation.metric.MetricCreateRequest) → *paperswithcode.models.evaluation.metric.Metric*

Add a metrics to the evaluation table.

Parameters

- **evaluation_id** (*str*) – ID of the evaluation table.
- **metric** (MetricCreateRequest) – Metric create request.

Returns Created metric.

Return type *Metric*

evaluation_metric_update(*evaluation_id: str*, *metric_id: str*, *metric:* paperswithcode.models.evaluation.metric.MetricUpdateRequest) → *paperswithcode.models.evaluation.metric.Metric*

Update a metrics in the evaluation table.

Parameters

- **evaluation_id** (*str*) – ID of the evaluation table.
- **metric_id** (*str*) – ID of the metric.
- **metric** (MetricCreateRequest) – Metric update request.

Returns Updated metric.

Return type *Metric*

evaluation_metric_delete(*evaluation_id: str*, *metric_id: str*)

Delete a metrics from the evaluation table.

Parameters

- **evaluation_id** (*str*) – ID of the evaluation table.
- **metric_id** (*str*) – ID of the metric.

evaluation_result_list(*evaluation_id: str*, *page: int = 1*, *items_per_page: int = 50*) → *paperswithcode.models.evaluation.result.Results*

List all results from the evaluation table.

Parameters

- **evaluation_id** (*str*) – ID of the evaluation table.
- **page** (*int*) – Desired page.
- **items_per_page** (*int*) – Desired number of items per page. Default: 50.

Returns Results object.

Return type *Results*

evaluation_result_get(*evaluation_id: str*, *result_id: str*) → *paperswithcode.models.evaluation.result.Result*

Get a result from the evaluation table.

Parameters

- **evaluation_id** (*str*) – ID of the evaluation table.
- **result_id** (*str*) – ID of the result.

Returns Requested result.

Return type *Result*

evaluation_result_add(*evaluation_id: str*, *result:* paperswithcode.models.evaluation.result.ResultCreateRequest) → *paperswithcode.models.evaluation.result.Result*

Add a result to the evaluation table.

Parameters

- **evaluation_id** (*str*) – ID of the evaluation table.
- **result** (ResultCreateRequest) – Result create request.

Returns Created result.

Return type *Result*

evaluation_result_update(*evaluation_id: str*, *result_id: str*, *result:* paperswithcode.models.evaluation.result.ResultUpdateRequest) → *paperswithcode.models.evaluation.result.Result*

Update a result in the evaluation table.

Parameters

- **evaluation_id** (*str*) – ID of the evaluation table.
- **result_id** (*str*) – ID of the result.
- **result** (ResultUpdateRequest) – Result update request.

Returns Updated result.

Return type *Result*

evaluation_result_delete(*evaluation_id: str*, *result_id: str*)

Delete a result from the evaluation table.

Parameters

- **evaluation_id** (*str*) – ID of the evaluation table.
- **result_id** (*str*) – ID of the result.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex

PYTHON MODULE INDEX

p

INDEX

Symbols

A

B

C

D

E

F

I

M

N

O

P

R

S

T

U

Y