

---

# Deep Lake

*Release 3.9.2*

**Activeloop**

**Apr 26, 2024**



# GETTING STARTED

<b>1</b>	<b>Installation</b>	<b>3</b>
<b>2</b>	<b>Datasets</b>	<b>5</b>
2.1	Creating Datasets . . . . .	5
2.2	Loading Datasets . . . . .	5
2.3	Deleting and Renaming Datasets . . . . .	5
2.4	Copying Datasets . . . . .	6
2.5	Dataset Operations . . . . .	6
2.6	Dataset Visualization . . . . .	7
2.7	Dataset Credentials . . . . .	7
2.8	Dataset Properties . . . . .	7
2.9	Dataset Version Control . . . . .	7
2.10	Dataset Views . . . . .	8
<b>3</b>	<b>Vector Store</b>	<b>9</b>
3.1	Creating a Deep Lake Vector Store . . . . .	9
3.2	Vector Store Operations . . . . .	9
3.3	Vector Store Properties . . . . .	9
<b>4</b>	<b>Tensors</b>	<b>11</b>
4.1	Creating Tensors . . . . .	11
4.2	Deleting and Renaming Tensors . . . . .	11
4.3	Adding and deleting samples . . . . .	11
4.4	Retrieving samples . . . . .	12
4.5	Tensor Properties . . . . .	12
4.6	Info . . . . .	12
4.7	Video features . . . . .	12
<b>5</b>	<b>Htypes</b>	<b>13</b>
5.1	Image Htype . . . . .	14
5.2	Video Htype . . . . .	15
5.3	Audio Htype . . . . .	16
5.4	Class Label Htype . . . . .	16
5.5	Tag Htype . . . . .	18
5.6	Bounding Box Htype . . . . .	18
5.7	3D Bounding Box Htype . . . . .	20
5.8	Intrinsics Htype . . . . .	21
5.9	Segmentation Mask Htype . . . . .	22
5.10	Binary Mask Htype . . . . .	23
5.11	COCO Keypoints Htype . . . . .	24

5.12	Point Htype . . . . .	25
5.13	Polygon Htype . . . . .	26
5.14	Nifti Htype . . . . .	27
5.15	Point Cloud Htype . . . . .	28
5.16	Mesh Htype . . . . .	29
5.17	Embedding Htype . . . . .	30
5.18	Sequence htype . . . . .	30
5.19	Link htype . . . . .	31
<b>6</b>	<b>Compressions</b>	<b>33</b>
6.1	Sample Compression . . . . .	33
6.2	Chunk Compression . . . . .	34
<b>7</b>	<b>PyTorch and Tensorflow Support</b>	<b>35</b>
<b>8</b>	<b>Utility Functions</b>	<b>37</b>
8.1	General Functions . . . . .	37
8.2	Making Deep Lake Samples . . . . .	37
8.3	Parallelism . . . . .	37
<b>9</b>	<b>Weights and Biases</b>	<b>39</b>
9.1	Logging Dataset Creation . . . . .	39
9.2	Logging Dataset Read . . . . .	39
<b>10</b>	<b>MMDetection</b>	<b>41</b>
<b>11</b>	<b>Dataloader</b>	<b>43</b>
11.1	DeepLakeDataLoader . . . . .	43
<b>12</b>	<b>Sampler</b>	<b>51</b>
<b>13</b>	<b>Tensor Query Language</b>	<b>53</b>
13.1	Resources . . . . .	53
<b>14</b>	<b>Random Split</b>	<b>55</b>
<b>15</b>	<b>Deep Memory API</b>	<b>57</b>
15.1	Creating a Deep Memory . . . . .	57
15.2	Deep Memory Operations . . . . .	57
15.3	Deep Memory Properties . . . . .	57
15.4	Syntax . . . . .	57
<b>16</b>	<b>deeplake</b>	<b>61</b>
<b>17</b>	<b>deeplake.VectorStore</b>	<b>87</b>
<b>18</b>	<b>deeplake.core</b>	<b>99</b>
18.1	deeplake.core.sample . . . . .	99
18.2	deeplake.core.linked_sample . . . . .	100
18.3	deeplake.core.partial_sample . . . . .	101
18.4	deeplake.core.linked_tiled_sample . . . . .	101
18.5	deeplake.core.storage . . . . .	101
18.6	deeplake.core.index . . . . .	118
<b>19</b>	<b>deeplake.core.dataset</b>	<b>123</b>
19.1	Dataset . . . . .	123

19.2	DeepLakeCloudDataset	149
19.3	ViewEntry	152
<b>20</b>	<b>deeplake.core.tensor</b>	<b>155</b>
20.1	Tensor	155
<b>21</b>	<b>deeplake.api</b>	<b>163</b>
21.1	deeplake.api.dataset	163
21.2	deeplake.api.info	165
21.3	deeplake.api.link	165
21.4	deeplake.api.read	165
21.5	deeplake.api.tiled	166
21.6	deeplake.api.link_tiled	166
<b>22</b>	<b>deeplake.auto</b>	<b>167</b>
22.1	deeplake.auto.structured	167
22.2	deeplake.auto.unstructured	167
<b>23</b>	<b>deeplake.util</b>	<b>169</b>
23.1	deeplake.util.shape_interval	169
23.2	deeplake.util.remove_cache	169
23.3	deeplake.util.notebook	170
23.4	deeplake.util.exceptions	170
<b>24</b>	<b>deeplake.client.log</b>	<b>177</b>
<b>25</b>	<b>deeplake.core.transform</b>	<b>179</b>
<b>26</b>	<b>deeplake.core.vectorstore.deep_memory</b>	<b>183</b>
26.1	DeepMemory	183
<b>27</b>	<b>deeplake.random.seed</b>	<b>189</b>
27.1	Background	189
27.2	Random number generators in other libraries	189
<b>28</b>	<b>Indices and tables</b>	<b>191</b>
	<b>Python Module Index</b>	<b>193</b>
	<b>Index</b>	<b>195</b>



Deep Lake is an open-source database for AI. This page contains the Deep Lake API reference, mostly focusing on codebase documentation. For tutorials, code examples, and high-level information, please check out the [Deep Lake Docs](#).





## INSTALLATION

Deep Lake can be installed with pip

```
pip install deeplake
```

Deep Lake has the following extras that you can choose to install according to your needs.

Table 1: Installation commands

Install command	Description	Dependencies installed
<code>pip install "deeplake[av]"</code>	Audio and video support via PyAV	av
<code>pip install "deeplake[visualizer]"</code>	Visualize Deep Lake datasets within notebooks. This is required for <code>Dataset.visualize</code> to work.	IPython, flask
<code>pip install "deeplake[gcp]"</code>	GCS support	google-cloud-storage, google-auth, google-auth-oauthlib
<code>pip install "deeplake[azure]"</code>	Azure Blob Storage support	azure-storage-blob, azure-cli, azure-identity
<code>pip install "deeplake[medical]"</code>	DICOM and NIFTI data support	pydicom, nibabel
<code>pip install "deeplake[gdrive]"</code>	Google Drive support	google-api-python-client, oauth2client, google-auth, google-auth-oauthlib
<code>pip install "deeplake[point_cloud]"</code>	Support for LiDAR point cloud data	laspy
<code>pip install "deeplake[all]"</code>	Installs all of the above	



## DATASETS

### 2.1 Creating Datasets

<code>deeplake.dataset</code>	Returns a <code>Dataset</code> object referencing either a new or existing dataset.
<code>deeplake.empty</code>	Creates an empty dataset
<code>deeplake.like</code>	Creates a new dataset by copying the source dataset's structure to a new location.
<code>deeplake.ingest_classification</code>	Ingest a dataset of images from a local folder to a Deep Lake Dataset.
<code>deeplake.ingest_coco</code>	Ingest images and annotations in COCO format to a Deep Lake Dataset.
<code>deeplake.ingest_yolo</code>	Ingest images and annotations (bounding boxes or polygons) in YOLO format to a Deep Lake Dataset.
<code>deeplake.ingest_kaggle</code>	Download and ingest a kaggle dataset and store it as a structured dataset to destination.
<code>deeplake.ingest_dataframe</code>	Convert pandas dataframe to a Deep Lake Dataset.
<code>deeplake.ingest_huggingface</code>	Converts Hugging Face datasets to Deep Lake format.

### 2.2 Loading Datasets

<code>deeplake.load</code>	Loads an existing dataset
----------------------------	---------------------------

### 2.3 Deleting and Renaming Datasets

<code>deeplake.delete</code>	Deletes a dataset at a given path.
<code>deeplake.rename</code>	Renames dataset at <code>old_path</code> to <code>new_path</code> .

## 2.4 Copying Datasets

<code>deeplake.copy</code>	Copies dataset at <code>src</code> to <code>dest</code> .
<code>deeplake.deepcopy</code>	Copies dataset at <code>src</code> to <code>dest</code> including version control history.

## 2.5 Dataset Operations

<code>Dataset.summary</code>	Prints a summary of the dataset.
<code>Dataset.append</code>	Append samples to multiple tensors at once.
<code>Dataset.extend</code>	Appends multiple rows of samples to multiple tensors at once.
<code>Dataset.update</code>	Update existing samples in the dataset with new values.
<code>Dataset.query</code>	Returns a sliced <code>Dataset</code> with given query results.
<code>Dataset.copy</code>	Copies this dataset or dataset view to <code>dest</code> .
<code>Dataset.delete</code>	Deletes the entire dataset from the cache layers (if any) and the underlying storage.
<code>Dataset.rename</code>	Renames the dataset to <code>path</code> .
<code>Dataset.connect</code>	Connect a Deep Lake cloud dataset through a deeplake path.
<code>Dataset.visualize</code>	Visualizes the dataset in the Jupyter notebook.
<code>Dataset.pop</code>	Removes a sample from all the tensors of the dataset.
<code>Dataset.rechunk</code>	Rewrites the underlying chunks to make their sizes optimal.
<code>Dataset.flush</code>	Necessary operation after writes if caches are being used.
<code>Dataset.clear_cache</code>	<ul style="list-style-type: none"> <li>Flushes (see <code>Dataset.flush()</code>) the contents of the cache layers (if any) and then deletes contents of all the layers of it.</li> </ul>
<code>Dataset.size_approx</code>	Estimates the size in bytes of the dataset.
<code>Dataset.random_split</code>	Splits the dataset into non-overlapping <code>Dataset</code> objects of given lengths.

## 2.6 Dataset Visualization

<code>Dataset.visualize</code>	Visualizes the dataset in the Jupyter notebook.
--------------------------------	---

## 2.7 Dataset Credentials

<code>Dataset.add_creds_key</code>	Adds a new creds key to the dataset.
<code>Dataset.populate_creds</code>	Populates the creds key added in <code>add_creds_key</code> with the given creds.
<code>Dataset.update_creds_key</code>	Updates the name and/or management status of a creds key.
<code>Dataset.get_creds_keys</code>	Returns the set of creds keys added to the dataset.

## 2.8 Dataset Properties

<code>Dataset.tensors</code>	All tensors belonging to this group, including those within sub groups.
<code>Dataset.groups</code>	All sub groups in this group
<code>Dataset.num_samples</code>	Returns the length of the smallest tensor.
<code>Dataset.read_only</code>	Returns True if dataset is in read-only mode and False otherwise.
<code>Dataset.info</code>	Returns the information about the dataset.
<code>Dataset.max_len</code>	Return the maximum length of the tensor.
<code>Dataset.min_len</code>	Return the minimum length of the tensor.

## 2.9 Dataset Version Control

<code>Dataset.commit</code>	Stores a snapshot of the current state of the dataset.
<code>Dataset.diff</code>	Returns/displays the differences between commits/branches.
<code>Dataset.checkout</code>	Checks out to a specific commit_id or branch.
<code>Dataset.merge</code>	Merges the target_id into the current dataset.
<code>Dataset.log</code>	Displays the details of all the past commits.
<code>Dataset.reset</code>	Resets the uncommitted changes present in the branch.
<code>Dataset.get_commit_details</code>	Get details of a particular commit.
<code>Dataset.commit_id</code>	The lastest committed commit id of the dataset.
<code>Dataset.branch</code>	The current branch of the dataset
<code>Dataset.pending_commit_id</code>	The commit_id of the next commit that will be made to the dataset.
<code>Dataset.has_head_changes</code>	Returns True if currently at head node and uncommitted changes are present.
<code>Dataset.commits</code>	Lists all the commits leading to the current dataset state.
<code>Dataset.branches</code>	Lists all the branches of the dataset.

## 2.10 Dataset Views

A dataset view is a subset of a dataset that points to specific samples (indices) in an existing dataset. Dataset views can be created by indexing a dataset, filtering a dataset with `Dataset.filter()`, querying a dataset with `Dataset.query()` or by sampling a dataset with `Dataset.sample_by()`. Filtering is done with user-defined functions or simplified expressions whereas query can perform SQL-like queries with our Tensor Query Language. See the full TQL spec [here](#).

Dataset views can only be saved when a dataset has been committed and has no changes on the HEAD node, in order to preserve data lineage and prevent the underlying data from changing after the query or filter conditions have been evaluated.

### Example

```
>>> import deeplake
>>> # load dataset
>>> ds = deeplake.load("hub://activeloop/mnist-train")
>>> # filter dataset
>>> zeros = ds.filter("labels == 0")
>>> # save view
>>> zeros.save_view(id="zeros")
>>> # load_view
>>> zeros = ds.load_view(id="zeros")
>>> len(zeros)
5923
```

<code>Dataset.query</code>	Returns a sliced <code>Dataset</code> with given query results.
<code>Dataset.sample_by</code>	Returns a sliced <code>Dataset</code> with given weighted sampler applied.
<code>Dataset.filter</code>	Filters the dataset in accordance of filter function <code>f(x: sample) -&gt; bool</code>
<code>Dataset.save_view</code>	Saves a dataset view as a virtual dataset (VDS)
<code>Dataset.get_view</code>	Returns the dataset view corresponding to <code>id</code> .
<code>Dataset.load_view</code>	Loads the view and returns the <code>Dataset</code> by <code>id</code> .
<code>Dataset.delete_view</code>	Deletes the view with given view <code>id</code> .
<code>Dataset.get_views</code>	Returns list of views stored in this Dataset.
<code>Dataset.is_view</code>	Returns True if this dataset is a view and False otherwise.
<code>Dataset.min_view</code>	Returns a view of the dataset in which all tensors are sliced to have the same length as the shortest tensor.
<code>Dataset.max_view</code>	Returns a view of the dataset in which shorter tensors are padded with None s to have the same length as the longest tensor.

## VECTOR STORE

### 3.1 Creating a Deep Lake Vector Store

---

<i>VectorStore.__init__</i>	Creates an empty VectorStore or loads an existing one if it exists at the specified path.
-----------------------------	---

---

### 3.2 Vector Store Operations

---

<i>VectorStore.add</i>	Adding elements to deeplake vector store.
<i>VectorStore.search</i>	VectorStore search method that combines embedding search, metadata search, and custom TQL search.
<i>VectorStore.delete</i>	Delete the data in the Vector Store.
<i>VectorStore.delete_by_path</i>	Deleted the Vector Store at the specified path.
<i>VectorStore.update_embedding</i>	Recompute existing embeddings of the VectorStore, that match either query, filter, ids or row_ids.

---

### 3.3 Vector Store Properties

---

<i>VectorStore.summary</i>	Prints a summary of the dataset
<i>VectorStore.tensors</i>	Returns the list of tensors present in the dataset
<i>VectorStore.__len__</i>	Length of the dataset

---





## TENSORS

## 4.1 Creating Tensors

<i>Dataset.create_tensor</i>	Creates a new tensor in the dataset.
<i>Dataset.create_group</i>	Creates a tensor group.
<i>Dataset.create_tensor_like</i>	Copies the source tensor's meta information and creates a new tensor with it.

## 4.2 Deleting and Renaming Tensors

<i>Dataset.delete_tensor</i>	Delete a tensor from the dataset.
<i>Dataset.delete_group</i>	Delete a tensor group from the dataset.
<i>Dataset.rename_tensor</i>	Renames tensor with name <i>name</i> to <i>new_name</i>
<i>Dataset.rename_group</i>	Renames group with name <i>name</i> to <i>new_name</i>

## 4.3 Adding and deleting samples

<i>Tensor.append</i>	Appends a single sample to the end of the tensor.
<i>Tensor.extend</i>	Extends the end of the tensor by appending multiple elements from a sequence.
<i>Tensor.pop</i>	Removes element(s) at the given index / indices.
<i>Tensor.clear</i>	Deletes all samples from the tensor
<i>Tensor.__setitem__</i>	Update samples with new values.

## 4.4 Retrieving samples

<i>Tensor.numpy</i>	Computes the contents of the tensor in numpy format.
<i>Tensor.data</i>	Returns data in the tensor in a format based on the tensor's base htype.
<i>Tensor.tobytes</i>	Returns the bytes of the tensor.
<i>Tensor.text</i>	Return text data.
<i>Tensor.dict</i>	Return json data.
<i>Tensor.list</i>	Return list data.
<i>Tensor._linked_sample</i>	Returns the linked sample at the given index.

## 4.5 Tensor Properties

<i>Tensor.htype</i>	Htype of the tensor.
<i>Tensor.base_htype</i>	Base htype of the tensor.
<i>Tensor.dtype</i>	Dtype of the tensor.
<i>Tensor.shape</i>	Get the shape of this tensor.
<i>Tensor.shape_interval</i>	Returns a <i>ShapeInterval</i> object that describes this tensor's shape more accurately.
<i>Tensor.ndim</i>	Number of dimensions of the tensor.
<i>Tensor.num_samples</i>	Returns the length of the primary axis of the tensor.
<i>Tensor.__len__</i>	Returns the length of the primary axis of the tensor.
<i>Tensor.is_dynamic</i>	Will return True if samples in this tensor have shapes that are unequal.
<i>Tensor.is_sequence</i>	Whether this tensor is a sequence tensor.
<i>Tensor.is_link</i>	Whether this tensor is a link tensor.
<i>Tensor.verify</i>	Whether linked data will be verified when samples are added.

## 4.6 Info

<i>Tensor.info</i>	Returns the information about the tensor.
<i>Tensor.sample_info</i>	Returns info about particular samples in a tensor.

## 4.7 Video features

<i>Tensor.play</i>	Play video sample.
<i>Tensor.timestamps</i>	Returns timestamps (in seconds) for video sample as numpy array.

## HTYPES

Htype is the class of a tensor: image, bounding box, generic tensor, etc.

It is highly recommended to specify the htype during tensor creation. For numeric tensors, if htype is not specified, it defaults to “generic”.

```
>>> ds.create_tensor("my_tensor", htype="...")
```

Specifying an htype allows for strict settings and error handling, and it is critical for increasing the performance of Deep Lake datasets containing rich data such as images and videos.

Supported htypes and their respective defaults are:

Table 1: Htype configs

HTYPE	DTYPE	COMPRESSION
generic	None	None
<i>image</i>	uint8	Required arg
<i>image.rgb</i>	uint8	Required arg
<i>image.gray</i>	uint8	Required arg
<i>video</i>	uint8	Required arg
<i>audio</i>	float64	Required arg
<i>class_label</i>	uint32	None
<i>tag</i>	str	None
<i>bbox</i>	float32	None
<i>bbox.3d</i>	float32	None
<i>intrinsic</i>	float32	None
<i>segment_mask</i>	uint32	None
<i>binary_mask</i>	bool	None
<i>keypoints_coco</i>	int32	None
<i>point</i>	int32	None
<i>polygon</i>	float32	None
text	str	None
json	Any	None
list	List	None
dicom	None	dcm
<i>nifti</i>	None	Required arg
<i>point_cloud</i>	None	las
<i>mesh</i>	None	ply
instance_label	uint32	None
<i>embedding</i>	None	None
<i>link</i>	str	None
<i>sequence</i>	None	None

## 5.1 Image Htype

- Sample dimensions: (height, width, # channels) or (height, width).

Images can be stored in Deep Lake as compressed bytes or as raw arrays. Due to the high compression ratio for most image formats, it is highly recommended to store compressed images using the `sample_compression` input to the `create_tensor` method.

### 5.1.1 Creating an image tensor

An image tensor can be created using

```
>>> ds.create_tensor("images", htype="image", sample_compression="jpg")
```

OR

```
>>> ds.create_tensor("images", htype="image", chunk_compression="jpg")
```

- **Optional args:**
  - `dtype`: Defaults to `uint8`.
- Supported compressions:

```
>>> [None, "bmp", "dib", "gif", "ico", "jpeg", "jpeg2000", "pcx", "png", "ppm", "sgi",  
→ "tga", "tiff",  
... "webp", "wmf", "xbm", "eps", "fli", "im", "msp", "mpo"]
```

### 5.1.2 Appending image samples

- Image samples can be of type `np.ndarray` or Deep Lake *Sample* which can be created using `deeplake.read()`.

Examples

Appending pixel data with array

```
>>> ds.images.append(np.zeros((5, 5, 3), dtype=np.uint8))
```

Appening Deep Lake image sample

```
>>> ds.images.append(deeplake.read("images/0001.jpg"))
```

You can append multiple samples at the same time using `extend()`.

```
>>> ds.images.extend([deeplake.read(f"images/000{i}.jpg") for i in range(10)])
```

---

**Note:** If the compression format of the input sample does not match the `sample_compression` of the tensor, Deep Lake will decompress and recompress the image for storage, which may significantly slow down the upload process. The upload process is fastest when the image compression matches the `sample_compression`.

---

### 5.1.3 image.rgb and image.gray htypes

`image.rgb` and `image.gray` htypes can be used to force your samples to be of RGB or grayscale type. i.e., if RGB images are appended to an `image.gray` tensor, Deep Lake will convert them to grayscale and if grayscale images are appended to an `image.rgb` tensor, Deep Lake will convert them to RGB format.

`image.rgb` and `image.gray` tensors can be created using

```
>>> ds.create_tensor("rgb_images", htype="image.rgb", sample_compression="...")
```

```
>>> ds.create_tensor("gray_images", htype="image.gray", sample_compression="...")
```

## 5.2 Video Htype

- Sample dimensions: (# frames, height, width, # channels) or (# frames, height, width)

### 5.2.1 Limitations

- **Visualization of videos in the Deep Lake App is limited by the video coded support for various browsers.**
  - Chrome supports the codecs shown here.
- *The Deep Lake Performant Dataloader* does not support videos.

### 5.2.2 Creating a video tensor

A video tensor can be created using

```
>>> ds.create_tensor("videos", htype="video", sample_compression="mp4")
```

- **Optional args:**
  - dtype: Defaults to `uint8`.
- Supported compressions:

```
>>> [None, "mp4", "mkv", "avi"]
```

### 5.2.3 Appending video samples

- Video samples can be of type `np.ndarray` or *Sample* which is returned by `deeplake.read()`.
- Deep Lake does not support compression of raw video frames. Therefore, array of raw frames can only be appended to tensors with `None` compression.
- Recompression of samples read with `deeplake.read` is also not supported.

Examples

Appending Deep Lake video sample

```
>>> ds.videos.append(deeplake.read("videos/0012.mp4"))
```

Extending with multiple videos

```
>>> ds.videos.extend([deeplake.read(f"videos/00{i}.mp4") for i in range(10)])
```

## 5.3 Audio Htype

- Sample dimensions: (# samples in audio, # channels) or (# samples in audio,)

### 5.3.1 Creating an audio tensor

An audio tensor can be created using

```
>>> ds.create_tensor("audios", htype="audio", sample_compression="mp3")
```

- **Optional args:**
  - dtype: Defaults to float64.
- Supported compressions:

```
>>> [None, "mp3", "wav", "flac"]
```

### 5.3.2 Appending audio samples

- Audio samples can be of type `np.ndarray` or *Sample* which is returned by `deeplake.read()`.
- Like videos, Deep Lake does not support compression or recompression of input audio samples. Thus, samples of type `np.ndarray` can only be appended to tensors with `None` compression.

Examples

Appending Deep Lake audio sample

```
>>> ds.audios.append(deeplake.read("audios/001.mp3"))
```

Extending with Deep Lake audio samples

```
>>> ds.audios.extend([deeplake.read(f"videos/00{i}.mp3") for i in range(10)])
```

## 5.4 Class Label Htype

- Sample dimensions: (# labels,)

Class labels are stored as numerical values in tensors, which are indices of the list `tensor.info.class_names`.

### 5.4.1 Creating a class label tensor

A class label tensor can be created using

```
>>> classes = ["airplanes", "cars", "birds", "cats", "deer", "dogs", "frogs", "horses",
↳ "ships", "trucks"]
>>> ds.create_tensor("labels", htype="class_label", class_names=classes, chunk_
↳ compression="lz4")
```

- **Optional args:**

- `class_names`: This must be a **list of strings**. `tensor.info.class_names` will be set to this list.
- `sample_compression` or `chunk_compression`.
- `dtype`: Defaults to `uint32`.

- Supported compressions:

```
>>> ["lz4"]
```

You can also choose to set the class names after tensor creation.

```
>>> ds.labels.info.update(class_names = ["airplanes", "cars", "birds", "cats", "deer",
↳ "dogs", "frogs", "horses", "ships", "trucks"])
```

**Note:** If specifying compression, since the number of labels in one sample will be too low, `chunk_compression` would be the better option to use.

### 5.4.2 Appending class labels

- Class labels can be appended as `int`, `str`, `np.ndarray` or `list` of `int` or `str`.
- In case of strings, `tensor.info.class_names` is updated automatically.

Examples

Appending index

```
>>> ds.labels.append(0)
>>> ds.labels.append(np.zeros((5,), dtype=np.uint32))
```

Extending with list of indices

```
>>> ds.labels.extend([[0, 1, 2], [1, 3]])
```

Appending text labels

```
>>> ds.labels.append(["cars", "airplanes"])
```

## 5.5 Tag Htype

- Sample dimensions: (# tags,)

This htype can be used to tag samples with one or more string values.

### 5.5.1 Creating a tag tensor

A tag tensor can be created using

```
>>> ds.create_tensor("tags", htype="tag", chunk_compression="lz4")
```

- **Optional args:**
  - *chunk\_compression*.
- Supported compressions:

```
>>> ["lz4"]
```

### 5.5.2 Appending tag samples

- Tag samples can be appended as str or list of str.

Examples

Appending a tag

```
>>> ds.tags.append("verified")
```

Extending with list of tags

```
>>> ds.tags.extend(["verified", "unverified"])
```

## 5.6 Bounding Box Htype

- Sample dimensions: (# bounding boxes, 4)

Bounding boxes have a variety of conventions such as those used in YOLO, COCO, Pascal-VOC and others. In order for bounding boxes to be correctly displayed by the visualizer, the format of the bounding box must be specified in the `coords` key in tensor meta information mentioned below.

### 5.6.1 Creating a bbox tensor

A bbox tensor can be created using

```
>>> ds.create_tensor("boxes", htype="bbox", coords={"type": "fractional", "mode": "CCWH"}  
↪)
```

- **Optional args:**
  - **coords:** A dictionary with keys “type” and “mode”.



\* **type:** Specifies the units of bounding box coordinates.

- “pixel”: is in unit of pixels.
- “fractional”: is in units relative to the width and height of the image, such as in YOLO format.

\* **mode:** Specifies the convention for the 4 coordinates

- “LTRB”: left\_x, top\_y, right\_x, bottom\_y
- “LTWH”: left\_x, top\_y, width, height
- “CCWH”: center\_x, center\_y, width, height

– **dtype:** Defaults to `float32`.

– *sample\_compression* or *chunk\_compression*.

- Supported compressions:

```
>>> ["lz4"]
```

You can also choose to set the class names after tensor creation.

```
>>> ds.bboxes.info.update(coords = {"type": "pixel", "mode": "LTRB"})
```

**Note:** If the bounding box format is not specified, the visualizer will assume a YOLO format (fractional + CCWH) if the box coordinates are < 1 on average. Otherwise, it will assume the COCO format (pixel + LTWH).

## 5.6.2 Appending bounding boxes

- Bounding boxes can be appended as `np.ndarrays` or `list` or `lists of arrays`.

Examples

Appending one bounding box

```
>>> box
array([[462, 123, 238, 98]])
>>> ds.bboxes.append(box)
```

Appending sample with 3 bounding boxes

```
>>> boxes
array([[965, 110, 262, 77],
       [462, 123, 238, 98],
       [688, 108, 279, 116]])
>>> boxes.shape
(3, 4)
>>> ds.bboxes.append(boxes)
```

## 5.7 3D Bounding Box Htype

In order for 3D bounding boxes to be correctly displayed by the visualizer, the format of the bounding box must be specified in the `coords` key in tensor meta information mentioned below.

### 5.7.1 Creating a 3d bbox tensor

**Note:** In order for 3D bounding boxes to be correctly displayed by the visualizer, the format of the bounding box must be specified in the `coords` key in tensor meta information mentioned below. In addition, for projecting 3D bounding boxes onto 2D data (such as an image), the *intrinsic*s tensor must exist in the dataset, or the intrinsic matrix must be specified in the `ds.img_tensor.info` dictionary, where the key is "intrinsic" and the value is the matrix.

A 3d bbox tensor can be created using

```
>>> ds.create_tensor("3d_boxes", htype="bbox.3d", coords={"mode": "center"})
```

- **Optional args:**

- **coords:** A dictionary with key “mode”.

- \* **mode:** Specifies the convention for the bbox coordinates.

- “center”: [center\_x, center\_y, center\_z, size\_x, size\_y, size\_z, rot\_x, rot\_y, rot\_z]

Sample dimensions: (# bounding boxes, 9)

size\_x - is the length of the bounding box along x direction

size\_y - is the width of the bounding box along y direction

size\_z - is the height of the bounding box along z direction

rot\_x - rotation angle along x axis, given in degrees

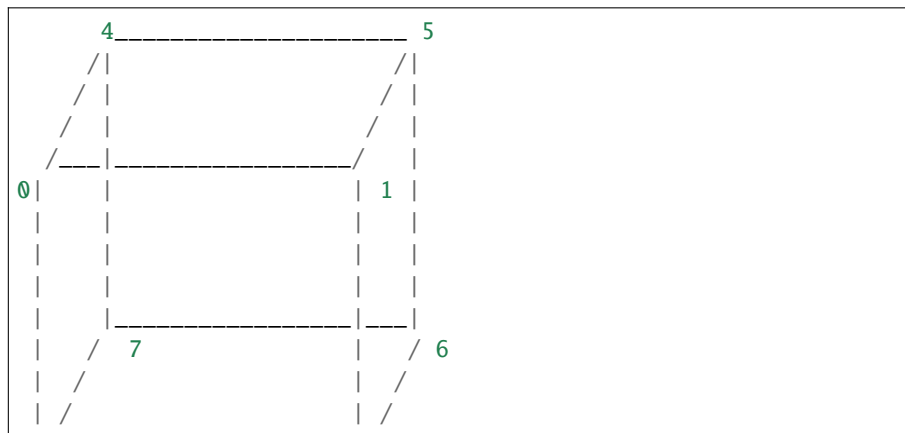
rot\_y - rotation angle along y axis, given in degrees

rot\_z - rotation angle along z axis, given in degrees

- “vertex”: 8 3D vertices - [[x0, y0, z0], [x1, y1, z1], [x2, y2, z2], ..., [x7, y7, z7]]

Sample dimensions: (# bounding boxes, 8, 3)

The vertex order is of the following form:



(continues on next page)

(continued from previous page)

$$\left| \frac{\quad}{3} \right| \frac{\quad}{2}$$

- **dtype**: Defaults to `float32`.
- `sample_compression` or `chunk_compression`.

- Supported compressions:

```
>>> ["lz4"]
```

**Note:** rotation angles are specified in degrees, not radians

## 5.7.2 Appending 3d bounding boxes

- Bounding boxes can be appended as `np.ndarrays` or `list` or `lists` of arrays.

Examples

Appending one bounding box

```
>>> box
array([[462, 123, 238, 98, 22, 36, 44, 18, 0, 36, 0]])
>>> ds.3d_boxes.append(box)
```

Appending sample with 3 bounding boxes

```
>>> boxes
array([[965, 110, 262, 77, 22, 36, 44, 18, 0, 28, 0],
       [462, 123, 238, 98, 26, 34, 24, 19, 0, -50, 0],
       [688, 108, 279, 116, 12, 32, 14, 38, 0, 30, 0]])
>>> boxes.shape
(9, 4)
>>> ds.3d_boxes.append(boxes)
```

## 5.8 Intrinsic Htype

- Sample dimensions: (# `intrinsic_matrices`, 3, 3)

The intrinsic matrix represents a projective transformation from the 3-D camera's coordinates into the 2-D image coordinates. The intrinsic parameters include the focal length, the optical center, also known as the principal point. The camera intrinsic matrix,  $K$ , is defined as:

$$\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

- $[c_x, c_y]$  - Optical center (the principal point), in pixels.
- $[f_x, f_y]$  - Focal length in pixels.
- $f_x = F/p_x$

- $f_y = F/p_y$
- $F$  - Focal length in world units, typically expressed in millimeters.
- $(p_x, p_y)$  - Size of the pixel in world units.

### 5.8.1 Creating an intrinsics tensor

An intrinsics tensor can be created using

```
>>> ds.create_tensor("intrinsics", htype="intrinsics")
```

- **Optional args:**
  - *sample\_compression* or *chunk\_compression*.
  - dtype: Defaults to float32.
- Supported compressions:

```
>>> ["lz4"]
```

### 5.8.2 Appending intrinsics matrices

```
>>> intrinsic_params = np.zeros((3, 3))
>>> ds.intrinsics.append(intrinsic_params)
```

## 5.9 Segmentation Mask Htype

- Sample dimensions: (height, width)

Segmentation masks are 2D representations of class labels where the numerical label data is encoded in an array of same shape as the image. The numerical values are indices of the list `tensor.info.class_names`.

### 5.9.1 Creating a segment\_mask tensor

A segment\_mask tensor can be created using

```
>>> classes = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle"]
>>> ds.create_tensor("masks", htype="segment_mask", class_names=classes, sample_
↪compression="lz4")
```

- **Optional args:**
  - class\_names: This must be a **list of strings**. `tensor.info.class_names` will be set to this list.
  - *sample\_compression* or *chunk\_compression*
  - dtype: Defaults to uint32.
- Supported compressions:

```
>>> ["lz4"]
```

You can also choose to set the class names after tensor creation.

```
>>> ds.labels.info.update(class_names = ["background", "aeroplane", "bicycle", "bird",
↪ "boat", "bottle"])
```

**Note:** Since segmentation masks often contain large amounts of data, it is recommended to compress them using lz4.

## 5.9.2 Appending segmentation masks

- Segmentation masks can be appended as `np.ndarray`.

Examples

```
>>> ds.masks.append(np.zeros((512, 512)))
```

**Note:** Since each pixel can only be labeled once, segmentation masks are not appropriate for datasets where objects might overlap, or where multiple objects within the same class must be distinguished. For these use cases, please use `htype = "binary_mask"`.

## 5.10 Binary Mask Htype

- Sample dimensions: (height, width, # objects in a sample)

Binary masks are similar to segmentation masks, except that each object is represented by a channel in the mask. Each channel in the mask encodes values for a single object. A pixel in a mask channel should have a value of 1 if the pixel of the image belongs to this object and 0 otherwise. The labels corresponding to the channels should be stored in an adjacent tensor of htype `class_label`, in which the number of labels at a given index is equal to the number of objects (number of channels) in the binary mask.

### 5.10.1 Creating a binary\_mask tensor

A `binary_mask` tensor can be created using

```
>>> ds.create_tensor("masks", htype="binary_mask", sample_compression="lz4")
```

- **Optional args:**
  - `ref:sample_compression <sample_compression> or chunk_compression`
  - `dtype`: Defaults to `bool`.
- Supported compressions:

```
>>> ["lz4"]
```

**Note:** Since segmentation masks often contain large amounts of data, it is recommended to compress them using lz4.

## 5.10.2 Appending binary masks

- Binary masks can be appended as `np.ndarray`.

Examples

Appending a binary mask with 5 objects

```
>>> ds.masks.append(np.zeros((512, 512, 5), dtype="bool"))
>>> ds.labels.append(["aeroplane", "aeroplane", "bottle", "bottle", "bird"])
```

## 5.11 COCO Keypoints Htype

- Sample dimensions: (3 x # keypoints, # objects in a sample)

**COCO keypoints** are a convention for storing points of interest in an image. Each keypoint consists of 3 values: `x` - coordinate, `y` - coordinate and `v` - visibility. A set of `K` keypoints of an object is represented as:

`[x1, y1, v1, x2, y2, v2, ..., xk, yk, vk]`

The visibility `v` can be one of three values:

- 0**  
keypoint not in image.
- 1**  
keypoint in image but not visible.
- 2**  
keypoint in image and visible.

### 5.11.1 Creating a keypoints\_coco tensor

A keypoints\_coco tensor can be created using

```
>>> ds.create_tensor("keypoints", htype="keypoints_coco", keypoints=["knee", "elbow",
↪ "head"], connections=[[0, 1], [1, 2]])
```

- **Optional args:**
  - `keypoints`: List of strings describing the `i` th keypoint. `tensor.info.keypoints` will be set to this list.
  - `connections`: List of strings describing which points should be connected by lines in the visualizer.
  - *sample\_compression* or *chunk\_compression*
  - `dtype`: Defaults to `int32`.
- Supported compressions:

```
>>> ["lz4"]
```

You can also choose to set `keypoints` and / or `connections` after tensor creation.

```
>>> ds.keypoints.info.update(keypoints = ['knee', 'elbow', ...])
>>> ds.keypoints.info.update(connections = [[0,1], [2,3], ...])
```

### 5.11.2 Appending keypoints

- Keypoints can be appended as `np.ndarray` or `list`.

Examples

Appending keypoints sample with 3 keypoints and 4 objects

```
>>> ds.keypoints.update(keypoints = ["left ear", "right ear", "nose"])
>>> ds.keypoints.update(connections = [[0, 2], [1, 2]])
>>> kp_arr
array([[465, 398, 684, 469],
       [178, 363, 177, 177],
       [ 2,  2,  2,  1],
       [454, 387, 646, 478],
       [177, 322, 137, 161],
       [ 2,  2,  2,  2],
       [407, 379, 536, 492],
       [271, 335, 150, 143],
       [ 2,  1,  2,  2]])
>>> kp_arr.shape
(9, 4)
>>> ds.keypoints.append(kp_arr)
```

**Warning:** In order to correctly use the keypoints and connections metadata, it is critical that all objects in every sample have the same number of K keypoints in the same order. For keypoints that are not present in an image, they can be stored with dummy coordinates of  $x = 0$ ,  $y = 0$ , and  $v = 0$ , and the visibility will prevent them from being drawn in the visualizer.

## 5.12 Point Htype

- Sample dimensions: (`# points`, 2) in case of 2-D (X, Y) co-ordinates or (`# points`, 3) in case of 3-D (X, Y, Z) co-ordinates of the point.

Points does not contain a fixed mapping across samples between the point order and real-world objects (i.e., point 0 is an elbow, point 1 is a knee, etc.). If you require such a mapping, use [COCO Keypoints Htype](#).

### 5.12.1 Creating a point tensor

A point tensor can be created using

```
>>> ds.create_tensor("points", htype="point", sample_compression=None)
```

- **Optional args:**
  - `sample_compression` or `chunk_compression`
  - `dtype`: Defaults to `int32`.
- Supported compressions:

```
>>> ["lz4"]
```

### 5.12.2 Appending point samples

- Points can be appended as `np.ndarray` or `list`.

Examples

Appending 2 2-D points

```
>>> ds.points.append([[0, 1], [1, 3]])
```

Appending 2 3-D points

```
>>> ds.points.append(np.zeros((2, 3)))
```

## 5.13 Polygon Htype

- Sample dimensions: (# polygons, # points per polygon, # co-ordinates per point)
- Each sample in a tensor of `polygon` htype is a list of polygons.
- Each polygon is a list / array of points.
- All points in a sample should have the same number of co-ordinates (eg., cannot mix 2-D points with 3-D points).
- Different samples can have different number of polygons.
- Different polygons can have different number of points.

### 5.13.1 Creating a polygon tensor

A polygon tensor can be created using

```
>>> ds.create_tensor("polygons", htype="polygon", sample_compression=None)
```

- **Optional args:**
  - *sample\_compression* or *chunk\_compression*
  - *dtype*: Defaults to `float32`.
- Supported compressions:

```
>>> ["lz4"]
```

### 5.13.2 Appending polygons

- Polygons can be appended as a list of list of tuples or `np.ndarray`.

Examples

Appending polygons with 2-D points

```
>>> poly1 = [(1, 2), (2, 3), (3, 4)]
>>> poly2 = [(10, 12), (14, 19)]
>>> poly3 = [(33, 32), (54, 67), (67, 43), (56, 98)]
```

(continues on next page)



(continued from previous page)

```
>>> sample = [poly1, poly2, poly3]
>>> ds.polygons.append(sample)
```

Appending polygons with 3-D points

```
>>> poly1 = [(10, 2, 9), (12, 3, 8), (12, 10, 4)]
>>> poly2 = [(10, 1, 8), (5, 17, 11)]
>>> poly3 = [(33, 33, 31), (45, 76, 13), (60, 24, 17), (67, 87, 83)]
>>> sample = [poly1, poly2, poly3]
>>> ds.polygons.append(sample)
```

Appending polygons with numpy arrays

```
>>> import numpy as np
>>> sample = np.random.randint(0, 10, (5, 7, 2)) # 5 polygons with 7 points
>>> ds.polygons.append(sample)
```

```
>>> import numpy as np
>>> poly1 = np.random.randint(0, 10, (5, 2))
>>> poly2 = np.random.randint(0, 10, (8, 2))
>>> poly3 = np.random.randint(0, 10, (3, 2))
>>> sample = [poly1, poly2, poly3]
>>> ds.polygons.append(sample)
```

## 5.14 Nifti Htype

- Sample dimensions: (# height, # width, # slices) or (# height, # width, # slices, # time unit) in case of time-series data.

### 5.14.1 Limitations

- *The Deep Lake Performant Dataloader* does not support Nifti data.

### 5.14.2 Creating a nifti tensor

A nifti tensor can be created using

```
>>> ds.create_tensor("patients", htype="nifti", sample_compression="nii.gz")
```

- Supported compressions:

```
>>> ["nii.gz", "nii", None]
```

### 5.14.3 Appending nifti data

- Nifti samples can be of type `np.ndarray` or *Sample* which is returned by `deeplake.read()`.
- Deep Lake does not support compression of raw nifti data. Therefore, array of raw frames can only be appended to tensors with `None` compression.

Examples

```
>>> ds.patients.append(deeplake.read("data/patient0.nii.gz"))
```

```
>>> ds.patients.extend([deeplake.read(f"data/patient{i}.nii.gz") for i in range(10)])
```

## 5.15 Point Cloud Htype

- Sample dimensions: `(# num_points, 3)`
- Point cloud samples can be of type `np.ndarray` or *Sample* which is returned by `deeplake.read()`.
- Each point cloud is a list / array of points.
- All points in a sample should have the same number of co-ordinates.
- Different point clouds can have different number of points.

### 5.15.1 Creating a point cloud tensor

A point cloud tensor can be created using

```
>>> ds.create_tensor("point_clouds", htype="point_cloud", sample_compression="las")
```

- **Optional args:**
  - *sample\_compression*
- Supported compressions:

```
>>> [None, "las"]
```

### 5.15.2 Appending point clouds

- Point clouds can be appended as a `np.ndarray`.

Examples

Appending point clouds with numpy arrays

```
>>> import numpy as np
>>> point_cloud1 = np.random.randint(0, 10, (5, 3))
>>> ds.point_clouds.append(point_cloud1)
>>> point_cloud2 = np.random.randint(0, 10, (15, 3))
>>> ds.point_clouds.append(point_cloud2)
>>> ds.point_clouds.shape
>>> (2, None, 3)
```

Or we can use `deeplake.read()` method to add samples

```
>>> import deeplake as dp
>>> sample = dp.read("example.las") # point cloud with 100 points
>>> ds.point_cloud.append(sample)
>>> ds.point_cloud.shape
>>> (1, 100, 3)
```

## 5.16 Mesh Htype

- Sample dimensions: (# num\_points, 3)
- Mesh samples can be of type `np.ndarray` or `Sample` which is returned by `deeplake.read()`.
- Each sample in a tensor of mesh htype is a mesh array (3-D object data).
- Each mesh is a list / array of points.
- Different meshes can have different number of points.

### 5.16.1 Creating a mesh tensor

A mesh tensor can be created using

```
>>> ds.create_tensor("mesh", htype="mesh", sample_compression="ply")
```

- **Optional args:**
  - `sample_compression`
- Supported compressions:

```
>>> ["ply"]
```

### 5.16.2 Appending meshes

Examples

Appending a ply file containing a mesh data to tensor

```
>>> import deeplake as dp
>>> sample = dp.read("example.ply") # mesh with 100 points and 200 faces
>>> ds.mesh.append(sample)
```

```
>>> ds.mesh.shape
>>> (1, 100, 3)
```

## 5.17 Embedding Htype

- Sample dimensions: (# elements in the embedding,)

### 5.17.1 Creating an embedding tensor

An embedding tensor can be created using

```
>>> ds.create_tensor("embedding", htype="embedding")
```

- Supported compressions:

```
>>> ["lz4", None]
```

### 5.17.2 Appending embedding samples

- Embedding samples can be of type `np.ndarray`.

Examples

Appending Deep Lake embedding sample

```
>>> ds.embedding.append(np.random.uniform(low=-1, high=1, size=(1024)))
```

Extending with Deep Lake embeddding samples

```
>>> ds.embedding.extend([np.random.uniform(low=-1, high=1, size=(1024)) for i in
↪ range(10)])
```

## 5.18 Sequence htype

- A special meta htype for tensors where each sample is a sequence. The items in the sequence are samples of another htype.
- It is a wrapper htype that can wrap other htypes like `sequence[image]`, `sequence[video]`, `sequence[text]`, etc.

Examples

```
>>> ds.create_tensor("seq", htype="sequence")
>>> ds.seq.append([1, 2, 3])
>>> ds.seq.append([4, 5, 6])
>>> ds.seq.numpy()
array([[1],
       [2],
       [3],
       [4],
       [5],
       [6]])
```

```
>>> ds.create_tensor("image_seq", htype="sequence[image]", sample_compression="jpg")
>>> ds.image_seq.append([deeplake.read("img01.jpg"), deeplake.read("img02.jpg")])
```

## 5.19 Link htype

- Link htype is a special meta htype that allows linking of external data (files) to the dataset, without storing the data in the dataset itself.
- Moreover, there can be variations in this htype, such as `link[image]`, `link[video]`, `link[audio]`, etc. that would enable the activeloop visualizer to correctly display the data.
- No data is actually loaded until you try to read the sample from a dataset.
- **There are a few exceptions to this:-**
  - If `create_shape_tensor=True` was specified during `create_tensor` of the tensor to which this is being added, the shape of the sample is read. This is `True` by default.
  - If `create_sample_info_tensor=True` was specified during `create_tensor` of the tensor to which this is being added, the sample info is read. This is `True` by default.
  - If `verify=True` was specified during `create_tensor` of the tensor to which this is being added, some metadata is read from them to verify the integrity of the link samples. This is `True` by default.
  - If you do not want to verify your links, all three of `verify`, `create_shape_tensor` and `create_sample_info_tensor` have to be set to `False`.

Examples

```
>>> ds = deeplake.dataset(".....")
```

Adding credentials to the dataset

You can add the names of the credentials you want to use (not needed for http/local urls)

```
>>> ds.add_creds_key("MY_S3_KEY")
>>> ds.add_creds_key("GCS_KEY")
```

and populate the added names with credentials dictionaries

```
>>> ds.populate_creds("MY_S3_KEY", {}) # add creds here
>>> ds.populate_creds("GCS_KEY", {})  # add creds here
```

These creds are only present temporarily and will have to be repopulated on every reload.

For datasets connected to Activeloop Platform, [you can store your credentials on the platform](#) as Managed Credentials and use them just by adding the keys to your dataset. For example if you have managed credentials with names `"my_s3_creds"`, `"my_gcs_creds"`, you can add them to your dataset using `Dataset.add_creds_key` without having to populate them.

```
>>> ds.add_creds_key("my_s3_creds", managed=True)
>>> ds.add_creds_key("my_gcs_creds", managed=True)
```

Create a link tensor

```
>>> ds.create_tensor("img", htype="link[image]", sample_compression="jpg")
```

Populate the tensor with links

```
>>> ds.img.append(deeplake.link("s3://abc/def.jpeg", creds_key="my_s3_key"))
>>> ds.img.append(deeplake.link("gcs://ghi/jkl.png", creds_key="GCS_KEY"))
>>> ds.img.append(deeplake.link("https://picsum.photos/200/300")) # http path doesn't need creds
>>> ds.img.append(deeplake.link("./path/to/cat.jpeg")) # local path doesn't need creds
>>> ds.img.append(deeplake.link("s3://abc/def.jpeg")) # this will throw an exception as cloud paths always need creds_key
:bluebold:`Accessing the data`
```

```
>>> for i in range(5):
...     ds.img[i].numpy()
...
```

Updating a sample

```
>>> ds.img[0] = deeplake.link("./data/cat.jpeg")
```

## COMPRESSIONS

Deep Lake can read, compress, decompress and recompress data to different formats. The supported htype-compression configurations are given below.

Sample Type	Htype	Compressions
Image	image	bmp, dib, gif, ico, jpeg, jpeg2000, pcx, png, ppm, sgi, tga, tiff, webp, wmf, xbm, eps, fli, im, msp, mpo, apng
Video	video	mp4, mkv, avi
Audio	audio	flac, mp3, wav
Dicom	dicom	dcm
Point Cloud	point_cloud	las
Mesh	mesh	ply
Other	bbox, text, list, json, generic, etc.	lz4

### 6.1 Sample Compression

If sample compression is specified when *creating tensors*, samples will be compressed to the given format if possible. If given data is already compressed and matches the provided `sample_compression`, it will be stored as is. If left as `None`, given samples are uncompressed.

---

**Note:** For audio and video, we don't support compressing raw frames but only reading compressed audio and video data.

---

Examples:

```
>>> ds.create_tensor("images", htype="image", sample_compression="jpg")
>>> ds.create_tensor("videos", htype="video", sample_compression="mp4")
>>> ds.create_tensor("point_clouds", htype="point_cloud", sample_compression="las")
```

Fig. 1: Structure of sample-wise compressed tensor.

## 6.2 Chunk Compression

If chunk compression is specified when *creating tensors*, added samples will be clubbed together and compressed to the given format chunk-wise. If given data is already compressed, it will be uncompressed and then recompressed chunk-wise.

---

**Note:** Chunk-wise compression is not supported for audio, video and point\_cloud htypes.

---

Examples:

```
>>> ds.create_tensor("images", htype="image", chunk_compression="jpg")
>>> ds.create_tensor("boxes", htype="bbox", chunk_compression="lz4")
```

Fig. 2: Structure of chunk-wise compressed tensor.

---

**Note:** See `deeplake.read()` to learn how to read data from files and populate these tensors.

---



## PYTORCH AND TENSORFLOW SUPPORT

Deep Lake datasets can be easily converted to Torch dataloaders or Tensorflow datasets for training.

<i><a href="#">Dataset.pytorch</a></i>	Converts the dataset into a pytorch Dataloader.
<i><a href="#">Dataset.tensorflow</a></i>	Converts the dataset into a tensorflow compatible format.



## UTILITY FUNCTIONS

### 8.1 General Functions

---

<i>exists</i>	Checks if a dataset exists at the given path.
---------------	---

---

### 8.2 Making Deep Lake Samples

---

<i>read</i>	Utility that reads raw data from supported files into Deep Lake format.
<i>link</i>	Utility that stores a link to raw data.
<i>link_tiled</i>	Utility that stores links to multiple images that act as tiles and together form a big image.

---

### 8.3 Parallelism

---

<i>compute</i>	Compute is a decorator for functions.
<i>compose</i>	Takes a list of functions decorated using <i>deeplake.compute()</i> and creates a pipeline that can be evaluated using <i>.eval</i>

---

Transform pipelines returned by *compute()* and *compose()* are evaluated using *eval*:

---

<i>eval</i>	Evaluates the pipeline on <i>data_in</i> to produce an output dataset <i>ds_out</i> .
-------------	---

---



## WEIGHTS AND BIASES

Deep Lake's Weights and Biases integration allows you to track and improve reproducibility of your machine learning experiments. Deep Lake will automatically push all information required to reproduce the snapshot of the data like your dataset's URI, commit ID, and view IDs of any views that you have used in your training workflow.

Learn more about Weights and Biases [here](#).

## 9.1 Logging Dataset Creation

If you create a Deep Lake dataset using any of the functions mentioned in *Creating Datasets*, just perform a commit on the dataset to log its creation on W&B.

```
>>> run = wandb.init(project="deeplake_wandb", job_type="dataset_upload")
>>> ds = deeplake.empty("hub://fayazrahman4u/my_dataset") # create dataset
>>> ds.create_tensor("images", htype="image", sample_compression="jpg") # create a tensor
>>> ds.images.append(deeplake.read("files/images/dog.jpg")) # add a sample
>>> ds.commit("creation") # commit -> trigger logging
>>> run.finish()
```

**Note:** If you created your dataset using `deeplake.deeppcopy()`, perform the commit only if you have head changes.

**Note:** If you make changes to an existing dataset, commit the changes with an active Weights and Biases run to log it's state.

## 9.2 Logging Dataset Read

A dataset read will be logged if you iterate over a dataset or call `Dataset.pytorch()` or `Tensor.numpy()` on its tensors.

```
>>> run = wandb.init(project="deeplake_wandb", job_type="torch dataloader")
>>> train_loader = ds.pytorch()
>>> run.finish()
```

```
>>> run = wandb.init(project="deeplake_wandb", job_type="iteration")
>>> for sample in ds:
>>>     print(sample["images"].shape)
>>> run.finish()
```

## MMDETECTION





## DATALOADER

Train your models using the high performance C++ dataloader, which is only available for registered users and datasets that are connected to Deep Lake. This dataloader cannot be used with local datasets.

See the `dataloader` method on how to create dataloaders from your datasets:

---

*`Dataset.dataloader`*

Returns a *`DeepLakeDataLoader`* object.

---

### 11.1 DeepLakeDataLoader

**class** `deeplake.enterprise.dataloader.DeepLakeDataLoader`

**batch**(*batch\_size: int, drop\_last: bool = False*)

Returns a batched *`DeepLakeDataLoader`* object.

**Parameters**

- **batch\_size** (*int*) – Number of samples in each batch.
- **drop\_last** (*bool*) – If True, the last batch will be dropped if its size is less than `batch_size`. Defaults to False.

**Returns**

A *`DeepLakeDataLoader`* object.

**Return type**

*`DeepLakeDataLoader`*

**Raises**

**ValueError** – If `.batch()` has already been called.

**close**()

Shuts down the workers and releases the resources.

**numpy**(*num\_workers: int = 0, tensors: Optional[List[str]] = None, num\_threads: Optional[int] = None, prefetch\_factor: int = 2, decode\_method: Optional[Dict[str, str]] = None, persistent\_workers: bool = False*)

Returns a *`DeepLakeDataLoader`* object.

**Parameters**

- **num\_workers** (*int*) – Number of workers to use for transforming and processing the data. Defaults to 0.
- **tensors** (*List[str], Optional*) – List of tensors to load. If None, all tensors are loaded. Defaults to None.

- **num\_threads** (*int*, *Optional*) – Number of threads to use for fetching and decompressing the data. If None, the number of threads is automatically determined. Defaults to None.
- **prefetch\_factor** (*int*) – Number of batches to transform and collate in advance per worker. Defaults to 2.
- **persistent\_workers** (*bool*) – If True, the data loader will not shutdown the worker processes after a dataset has been consumed once. Defaults to False.
- **decode\_method** (*Dict[str, str]*, *Optional*) – A dictionary of decode methods for each tensor. Defaults to None.
  - Supported decode methods are:-

**'numpy'**

Default behaviour. Returns samples as numpy arrays.

**'tobytes'**

Returns raw bytes of the samples.

**'pil'**

Returns samples as PIL images. Especially useful when transformation use torchvision transforms, that require PIL images as input. Only supported for tensors with sample\_compression='jpeg' or 'png'.

#### Returns

A [DeepLakeDataLoader](#) object.

#### Return type

[DeepLakeDataLoader](#)

#### Raises

**ValueError** – If .pytorch() or .tensorflow() or .numpy() has already been called.

**offset**(*off*: *int* = 0)

Returns a shifted [DeepLakeDataLoader](#) object.

#### Parameters

**off** (*int*) – index that the dataloader will start to iterate.

#### Returns

A [DeepLakeDataLoader](#) object.

#### Return type

[DeepLakeDataLoader](#)

#### Raises

**ValueError** – If .offset() has already been called.

**pytorch**(*num\_workers*: *int* = 0, *collate\_fn*: *Optional*[*Callable*] = None, *tensors*: *Optional*[*List*[*str*]] = None, *num\_threads*: *Optional*[*int*] = None, *prefetch\_factor*: *int* = 2, *distributed*: *bool* = False, *return\_index*: *bool* = True, *decode\_method*: *Optional*[*Dict*[*str*, *str*]] = None, *persistent\_workers*: *bool* = False)

Returns a [DeepLakeDataLoader](#) object.

#### Parameters

- **num\_workers** (*int*) – Number of workers to use for transforming and processing the data. Defaults to 0.
- **collate\_fn** (*Callable*, *Optional*) – merges a list of samples to form a mini-batch of Tensor(s).
- **tensors** (*List*[*str*], *Optional*) – List of tensors to load. If None, all tensors are loaded. Defaults to None.

- **num\_threads** (*int*, *Optional*) – Number of threads to use for fetching and decompressing the data. If *None*, the number of threads is automatically determined. Defaults to *None*.
- **prefetch\_factor** (*int*) – Number of batches to transform and collate in advance per worker. Defaults to 2.
- **distributed** (*bool*) – Used for DDP training. Distributes different sections of the dataset to different ranks. Defaults to *False*.
- **return\_index** (*bool*) – Used to identify where loader needs to return sample index or not. Defaults to *True*.
- **persistent\_workers** (*bool*) – If *True*, the data loader will not shutdown the worker processes after a dataset has been consumed once. Defaults to *False*.
- **decode\_method** (*Dict[str, str]*, *Optional*) – A dictionary of decode methods for each tensor. Defaults to *None*.

– Supported decode methods are:

**'numpy'**

Default behaviour. Returns samples as numpy arrays.

**'tobytes'**

Returns raw bytes of the samples.

**'pil'**

Returns samples as PIL images. Especially useful when transformation use torchvision transforms, that require PIL images as input. Only supported for tensors with `sample_compression='jpeg'` or `'png'`.

#### Returns

A *DeepLakeDataLoader* object.

#### Return type

*DeepLakeDataLoader*

#### Raises

**ValueError** – If `.pytorch()` or `.tensorflow()` or `.numpy()` has already been called.

### Examples

```
>>> import deeplake
>>> from torchvision import transforms
>>> ds_train = deeplake.load('hub://activeloop/fashion-mnist-train')
>>> tform = transforms.Compose([
...     transforms.RandomRotation(20), # Image augmentation
...     transforms.ToTensor(), # Must convert to pytorch tensor for subsequent
→operations to run
...     transforms.Normalize([0.5], [0.5]),
... ])
...
>>> batch_size = 32
>>> # create dataloader by chaining with transform function and batch size and
→returns batch of pytorch tensors
>>> train_loader = ds_train.dataloader()\
...     .transform({'images': tform, 'labels': None})\
...     .batch(batch_size)\
```

(continues on next page)

(continued from previous page)

```

...     .shuffle()\
...     .pytorch(decode_method={'images': 'pil'}) # return samples as PIL
→ images for transforms
...
>>> # iterate over dataloader
>>> for i, sample in enumerate(train_loader):
...     pass
...

```

**query**(query\_string: str)

Returns a sliced *DeepLakeDataLoader* object with given query results. It allows to run SQL like queries on dataset and extract results. See supported keywords and the Tensor Query Language documentation [here](#).

**Parameters**

**query\_string** (str) – An SQL string adjusted with new functionalities to run on the dataset object

**Returns**

A *DeepLakeDataLoader* object.

**Return type**

*DeepLakeDataLoader*

## Examples

```

>>> import deeplake
>>> ds = deeplake.load('hub://activeloop/fashion-mnist-train')
>>> query_ds_train = ds_train.dataloader().query("select * where labels != 5")

```

```

>>> import deeplake
>>> ds_train = deeplake.load('hub://activeloop/coco-train')
>>> query_ds_train = ds_train.dataloader().query("(select * where
→ contains(categories, 'car') limit 1000) union (select * where
→ contains(categories, 'motorcycle') limit 1000)")

```

**sample\_by**(weights: Union[str, list, tuple, ndarray], replace: Optional[bool] = True, size: Optional[int] = None)

Returns a sliced *DeepLakeDataLoader* with given weighted sampler applied

**Parameters**

- **weights** – (Union[str, list, tuple, np.ndarray]): If it's string then tql will be run to calculate the weights based on the expression. list, tuple and ndarray will be treated as the list of the weights per sample
- **replace** – Optional[bool] If true the samples can be repeated in the result view. (default: True).
- **size** – Optional[int] The length of the result view. (default: len(dataset))

**Returns**

A *DeepLakeDataLoader* object.

**Return type**

*DeepLakeDataLoader*

## Examples

Sample the dataloader with `labels == 5` twice more than `labels == 6`

```
>>> ds = deeplake.load('hub://activeloop/fashion-mnist-train')
>>> sampled_ds = ds.dataloader().sample_by("max_weight(labels == 5: 10, labels_
↳ == 6: 5)")
```

Sample the dataloader treating `labels` tensor as weights.

```
>>> ds = deeplake.load('hub://activeloop/fashion-mnist-train')
>>> sampled_ds = ds.dataloader().sample_by("labels")
```

Sample the dataloader with the given weights;

```
>>> ds_train = deeplake.load('hub://activeloop/coco-train')
>>> weights = list()
>>> for i in range(0, len(ds_train)):
...     weights.append(i % 5)
...
>>> sampled_ds = ds.dataloader().sample_by(weights, replace=False)
```

**shuffle**(*shuffle: bool = True, buffer\_size: int = 2048*)

Returns a shuffled [DeepLakeDataLoader](#) object.

### Parameters

- **shuffle** (*bool*) – shows wheter we need to shuffle elements or not. Defaults to `True`.
- **buffer\_size** (*int*) – The size of the buffer used to shuffle the data in MBs. Defaults to 2048 MB. Increasing the `buffer_size` will increase the extent of shuffling.

### Returns

A [DeepLakeDataLoader](#) object.

### Return type

[DeepLakeDataLoader](#)

### Raises

- **ValueError** – If `.shuffle()` has already been called.
- **ValueError** – If dataset is view and shuffle is `True`

**tensorflow**(*num\_workers: int = 0, collate\_fn: Optional[Callable] = None, tensors: Optional[List[str]] = None, num\_threads: Optional[int] = None, prefetch\_factor: int = 2, return\_index: bool = True, decode\_method: Optional[Dict[str, str]] = None, persistent\_workers: bool = False*)

Returns a [DeepLakeDataLoader](#) object.

### Parameters

- **num\_workers** (*int*) – Number of workers to use for transforming and processing the data. Defaults to 0.
- **collate\_fn** (*Callable, Optional*) – merges a list of samples to form a mini-batch of Tensor(s).
- **tensors** (*List[str], Optional*) – List of tensors to load. If `None`, all tensors are loaded. Defaults to `None`.
- **num\_threads** (*int, Optional*) – Number of threads to use for fetching and decompressing the data. If `None`, the number of threads is automatically determined. Defaults to `None`.

- **prefetch\_factor** (*int*) – Number of batches to transform and collate in advance per worker. Defaults to 2.
- **return\_index** (*bool*) – Used to identify where loader needs to return sample index or not. Defaults to True.
- **persistent\_workers** (*bool*) – If True, the data loader will not shutdown the worker processes after a dataset has been consumed once. Defaults to False.
- **decode\_method** (*Dict[str, str]*, *Optional*) – A dictionary of decode methods for each tensor. Defaults to None.

– Supported decode methods are:

**'numpy'**

Default behaviour. Returns samples as numpy arrays.

**'tobytes'**

Returns raw bytes of the samples.

**'pil'**

Returns samples as PIL images. Especially useful when transformation use torchvision transforms, that require PIL images as input. Only supported for tensors with `sample_compression='jpeg'` or `'png'`.

#### Returns

A *DeepLakeDataLoader* object.

#### Return type

*DeepLakeDataLoader*

#### Raises

**ValueError** – If `.pytorch()` or `.tensorflow()` or `.numpy()` has already been called.

### Examples

```
>>> import deeplake
>>> from torchvision import transforms
>>> ds_train = deeplake.load('hub://activeloop/fashion-mnist-train')
>>> batch_size = 32
>>> # create dataloader by chaining with transform function and batch size and
    ↪ returns batch of pytorch tensors
>>> train_loader = ds_train.dataloader()\
...     .batch(batch_size)\
...     .shuffle()\
...     .tensorflow() # return samples as PIL images for transforms
>>> # iterate over dataloader
>>> for i, sample in enumerate(train_loader):
...     pass
... 
```

**transform**(*transform: Union[Callable, Dict[str, Optional[Callable]]]*, *\*\*kwargs: Dict*)

Returns a transformed *DeepLakeDataLoader* object.

#### Parameters

- **transform** (*Callable* or *Dict[Callable]*) – A function or dictionary of functions to apply to the data.

- **kwargs** – Additional arguments to be passed to *transform*. Only applicable if *transform* is a callable. Ignored if *transform* is a dictionary.

**Returns**

A *DeepLakeDataLoader* object.

**Return type**

*DeepLakeDataLoader*

**Raises**

**ValueError** – If *.transform()* has already been called.





## SAMPLER

The sampler applies weighted sampling on the dataset and returns the sampled view. It creates a discrete distribution with given weights and randomly picks samples based on it. The resulting view is generated in such a way that when creating a dataloader from the view and training on it, the performance impact is minimal. See the `sample_by` method on how to use this feature:

---

*`Dataset.sample_by`*

Returns a sliced *`Dataset`* with given weighted sampler applied.

---



## TENSOR QUERY LANGUAGE

This page describes the Tensor Query Language (TQL), an SQL-like language used for querying Deep Lake Datasets. In addition to standard SQL operations, TQL supports array arithmetic, user-defined-functions, and other features specific to Deep Lake's diverse data formats.

### 13.1 Resources

- [TQL Overview](#)
- [TQL Syntax](#)



## RANDOM SPLIT

Splits the dataset into non overlapping new datasets of given lengths. The resulting datasets are generated in such a way that when creating a dataloader from the view and training on it, the performance impact is minimal. Using the outputs of this function with `.pytorch` method of dataset (instead of `.dataloader`) may result in poor performance. See the `random_split` method on how to use this feature:

---

*`Dataset.random_split`*

Splits the dataset into non-overlapping *`Dataset`* objects of given lengths.

---



## DEEP MEMORY API

### 15.1 Creating a Deep Memory

If Deep Memory is available on your plan, it will be automatically initialized when you create a Vector Store.

<i>DeepMemory.__init__</i>	Base Deep Memory class to train and evaluate models on DeepMemory managed service.
----------------------------	--

### 15.2 Deep Memory Operations

<i>DeepMemory.train</i>	Train a model on DeepMemory managed service.
<i>DeepMemory.cancel</i>	Cancel a training job on DeepMemory managed service.
<i>DeepMemory.delete</i>	Delete a training job on DeepMemory managed service.

### 15.3 Deep Memory Properties

<i>DeepMemory.status</i>	Get the status of a training job on DeepMemory managed service.
<i>DeepMemory.list_jobs</i>	List all training jobs on DeepMemory managed service.

### 15.4 Syntax

This page describes `ds.query`. DeepMemory is a deep learning model that is trained on the dataset to improve the search results, by aligning queries with the corpus dataset. It gives up to +22% of recall improvement on an eval dataset. To use `deep_memory`, please subscribe to our waitlist.

### 15.4.1 Training

To start training you should first create a vectostore object, and then preprocess the data and use deep memory with it:

```
>>> from deeplake import VectorStore
>>> db = VectorStore(
...     path="hub://{ $ORG_ID }/{ $DATASET_ID }",
...     token=token, # or you can be signed in with CLI
...     runtime={"tensor_db": True},
...     embedding_function=embedding_function, # function that takes converts texts into
↳ embeddings, it is optional and can be provided later
... )
```

To train a deepmemory model you need to preprocess the dataset so that, corpus, will become a list of list of tuples, where outer list corresponds to the query and inner list to the relevant documents. Each tuple should contain the document id (id tensor from the corpus dataset) and the relevance score (range is 0-1, where 0 represents unrelated document and 1 related). queries should be a list of strings.

```
>>> job_id = db.deep_memory.train(
...     corpus: List[List[Tuple[str, float]]] = corpus,
...     queries: List[str] = queries,
...     embedding_function = embedding_function, # function that takes converts texts
↳ into embeddings, it is optional and can be skipped if provided during initialization
... )
```

### 15.4.2 Tracking the training progress

job\_id is string, which can be used to track the training progress. You can use `db.deep_memory.status(job_id)` to get the status of the job.

when the model is still in pending state (not started yet) you will see the following:

```
>>> db.deep_memory.status(job_id)
| 6508464cd80cab681bfcfff3 |
| pending |
| results | not available | progress | None | status |
```

After some time the model will start training and you will see the following:

```
>>> db.deep_memory.status(job_id)
-----
| 6508464cd80cab681bfcfff3 |
| status | training |
| progress | eta: 2.5 seconds |
| | recall@10: 0.62% (+0.62%) |
| results | not available yet |
-----
```

If you want to get all training jobs you can use `db.deep_memory.list_jobs()` which will show all jobs that happened on this dataset.



```
>>> db.deep_memory.list_jobs()
ID                                STATUS    RESULTS                                PROGRESS
65198efcd28df3238c49a849         completed recall@10: 0.62% (+0.62%)         eta: 2.5 seconds
                                   recall@10: 0.62% (+0.62%)
↪62%)
651a4d41d05a21a5a6a15f67         completed recall@10: 0.62% (+0.62%)         eta: 2.5 seconds
                                   recall@10: 0.62% (+0.62%)
↪62%)
```

### 15.4.3 Deep Memory Evaluation

Once the training is completed, you can use `db.deep_memory.evaluate` to evaluate the model performance on the custom dataset. Once again you would need to preprocess the dataset so that, `corpus`, will become a list of list of tuples, where outer list corresponds to the query and inner list to the relevant documents. Each tuple should contain the document id (id tensor from the corpus dataset) and the relevance score (range is 0-1, where 0 represents unrelated document and 1 related). `queries` should be a list of strings.

```
>>> recalls = db.deep_memory.evaluate(
...     corpus: List[List[Tuple[str, float]]] = corpus,
...     queries: List[str] = queries,
...     embedding_function = embedding_function, # function that takes converts texts_
↪into embeddings, it is optional and can be skipped if provided during initialization
...     qvs_params = {"enabled": True}
... )
```

`recalls` is a dictionary with the following keys: `with_model` contains a dictionary with recall metrics for the naive vector search on the custom dataset for different k values `without_model` contains a dictionary with recall metrics for the naive vector search on the custom dataset for different k values `qvs_params` when specified creates a separate vectorstore that tracks all evaluation queries and documents, so that you can use it to compare the performance of `deep_memory` to naive vector search. By default, it is turned off. If enabled the dataset will be created at `hub://{ORG_ID}/{DATASET_ID}_eval_queries`

### 15.4.4 Deep Memory Search

After the model is trained you also can search using it:

```
>>> results = db.search(
...     embedding_data: List[str] = queries,
...     embedding_function = embedding_function, # function that takes converts texts_
↪into embeddings, it is optional and can be skipped if provided during initialization
...     k = 4, # number of results to return
...     deep_memory = True, # use deep memory model
... )
```



## DEEPLAKE

The deeplake package provides a database which stores data as compressed chunked arrays that can be stored anywhere and later streamed to deep learning models.

```
deeplake.dataset(path: Union[str, Path], runtime: Optional[Dict] = None, read_only: Optional[bool] = None,
                 overwrite: bool = False, public: bool = False, memory_cache_size: int = 2000,
                 local_cache_size: int = 0, creds: Optional[Union[Dict, str]] = None, token: Optional[str] =
                 None, org_id: Optional[str] = None, verbose: bool = True, access_method: str = 'stream',
                 unlink: bool = False, reset: bool = False, check_integrity: Optional[bool] = False,
                 lock_enabled: Optional[bool] = True, lock_timeout: Optional[int] = 0, index_params:
                 Optional[Dict[str, Union[int, str]]] = None, indra: bool = False)
```

Returns a `Dataset` object referencing either a new or existing dataset.

### Examples

```
>>> ds = deeplake.dataset("hub://username/dataset")
>>> ds = deeplake.dataset("s3://mybucket/my_dataset")
>>> ds = deeplake.dataset("./datasets/my_dataset", overwrite=True)
```

Loading to a specific version:

```
>>> ds = deeplake.dataset("hub://username/dataset@new_branch")
>>> ds = deeplake.dataset("hub://username/
↳ dataset@3e49cded62b6b335c74ff07e97f8451a37aca7b2")
```

```
>>> my_commit_id = "3e49cded62b6b335c74ff07e97f8451a37aca7b2"
>>> ds = deeplake.dataset(f"hub://username/dataset@{my_commit_id}")
```

### Parameters

- `path` (`str`, `pathlib.Path`) –
  - The full path to the dataset. Can be:
    - a Deep Lake cloud path of the form `hub://username/datasetname`. To write to Deep Lake cloud datasets, ensure that you are authenticated to Deep Lake (pass in a token using the ‘token’ parameter).
    - an s3 path of the form `s3://bucketname/path/to/dataset`. Credentials are required in either the environment or passed to the `creds` argument.
    - a local file system path of the form `./path/to/dataset` or `~/path/to/dataset` or `path/to/dataset`.

- a memory path of the form `mem://path/to/dataset` which doesn't save the dataset but keeps it in memory instead. Should be used only for testing as it does not persist.
- Loading to a specific version:
  - \* You can also specify a `commit_id` or `branch` to load the dataset to that version directly by using the `@` symbol.
  - \* The path will then be of the form `hub://username/dataset@{branch}` or `hub://username/dataset@{commit_id}`.
  - \* See examples above.
- **runtime** (*dict*) – Parameters for Activeloop DB Engine. Only applicable for `hub://` paths.
- **read\_only** (*bool*, *optional*) – Opens dataset in read only mode if this is passed as `True`. Defaults to `False`. Datasets stored on Deep Lake cloud that your account does not have write access to will automatically open in read mode.
- **overwrite** (*bool*) – If set to `True` this overwrites the dataset if it already exists. Defaults to `False`.
- **public** (*bool*) – Defines if the dataset will have public access. Applicable only if Deep Lake cloud storage is used and a new Dataset is being created. Defaults to `True`.
- **memory\_cache\_size** (*int*) – The size of the memory cache to be used in MB.
- **local\_cache\_size** (*int*) – The size of the local filesystem cache to be used in MB.
- **creds** (*dict*, *str*, *optional*) – The string ENV or a dictionary containing credentials used to access the dataset at the path. - If `'aws_access_key_id'`, `'aws_secret_access_key'`, `'aws_session_token'` are present, these take precedence over credentials present in the environment or in credentials file. Currently only works with s3 paths. - It supports `'aws_access_key_id'`, `'aws_secret_access_key'`, `'aws_session_token'`, `'endpoint_url'`, `'aws_region'`, `'profile_name'` as keys. - If `'ENV'` is passed, credentials are fetched from the environment variables. This is also the case when creds is not passed for cloud datasets. For datasets connected to hub cloud, specifying `'ENV'` will override the credentials fetched from Activeloop and use local ones.
- **token** (*str*, *optional*) – Activeloop token, used for fetching credentials to the dataset at path if it is a Deep Lake dataset. This is optional, tokens are normally auto-generated.
- **org\_id** (*str*, *Optional*) – Organization id to be used for enabling high-performance features. Only applicable for local datasets.
- **verbose** (*bool*) – If `True`, logs will be printed. Defaults to `True`.
- **access\_method** (*str*) – The access method to use for the dataset. Can be:
  - `'stream'`
    - \* Streams the data from the dataset i.e. only fetches data when required. This is the default value.
  - `'download'`
    - \* Downloads the data to the local filesystem to the path specified in environment variable `DEEPLAKE_DOWNLOAD_PATH`. This will overwrite `DEEPLAKE_DOWNLOAD_PATH`.
    - \* Raises an exception if `DEEPLAKE_DOWNLOAD_PATH` environment variable is not set or if the dataset does not exist.
    - \* The `'download'` access method can be modified to specify `num_workers` and/or `scheduler`. For example: `'download:2:processed'` will use 2 workers and use processed scheduler, while `'download:3'` will use 3 workers and

default scheduler (threaded), and 'download:processed' will use a single worker and use processed scheduler.

– 'local'

- \* Downloads the dataset if it doesn't already exist, otherwise loads from local storage.
- \* Raises an exception if DEEPLAKE\_DOWNLOAD\_PATH environment variable is not set.
- \* The 'local' access method can be modified to specify num\_workers and/or scheduler to be used in case dataset needs to be downloaded. If dataset needs to be downloaded, 'local:2:processed' will use 2 workers and use processed scheduler, while 'local:3' will use 3 workers and default scheduler (threaded), and 'local:processed' will use a single worker and use processed scheduler.
- **unlink** (*bool*) – Downloads linked samples if set to True. Only applicable if access\_method is download or local. Defaults to False.
- **reset** (*bool*) – If the specified dataset cannot be loaded due to a corrupted HEAD state of the branch being loaded, setting reset=True will reset HEAD changes and load the previous version.
- **check\_integrity** (*bool*, *Optional*) – Performs an integrity check by default (None) if the dataset has 20 or fewer tensors. Set to True to force integrity check, False to skip integrity check.
- **lock\_timeout** (*int*) – Number of seconds to wait before throwing a LockException. If None, wait indefinitely
- **lock\_enabled** (*bool*) – If true, the dataset manages a write lock. NOTE: Only set to False if you are managing concurrent access externally
- **index\_params** – Optional[Dict[str, Union[int, str]]] = None : The index parameters used while creating vector store is passed down to dataset.
- **indra** (*bool*) – Flag indicating whether indra api should be used to create the dataset. Defaults to false

#### Returns

Dataset created using the arguments provided.

#### Return type

*Dataset*

#### Raises

- **AgreementError** – When agreement is rejected
- **UserNotLoggedInException** – When user is not authenticated
- **InvalidTokenException** – If the specified token is invalid
- **TokenPermissionError** – When there are permission or other errors related to token
- **CheckoutError** – If version address specified in the path cannot be found
- **DatasetCorruptError** – If loading the dataset failed due to corruption and reset is not True
- **ValueError** – If version is specified in the path when creating a dataset or If the org id is provided but dataset is not local, or If the org id is provided but dataset is not local
- **ReadOnlyModeError** – If reset is attempted in read-only mode
- **LockedException** – When attempting to open a dataset for writing when it is locked by another machine
- **DatasetHandlerError** – If overwriting the dataset fails
- **Exception** – Re-raises caught exception if reset cannot fix the issue

**Danger:** Setting `overwrite` to `True` will delete all of your data if it exists! Be very careful when setting this parameter.

**Warning:** Setting `access_method` to `download` will overwrite the local copy of the dataset if it was previously downloaded.

**Note:** Any changes made to the dataset in `download` / `local` mode will only be made to the local copy and will not be reflected in the original dataset.

```
deeplake.empty(path: Union[str, Path], runtime: Optional[dict] = None, overwrite: bool = False, public: bool =
    False, memory_cache_size: int = 2000, local_cache_size: int = 0, creds: Optional[Union[Dict,
    str]] = None, token: Optional[str] = None, org_id: Optional[str] = None, lock_enabled:
    Optional[bool] = True, lock_timeout: Optional[int] = 0, verbose: bool = True, index_params:
    Optional[Dict[str, Union[int, str]]] = None) → Dataset
```

Creates an empty dataset

#### Parameters

- **path** (*str*, *pathlib.Path*) –
  - The full path to the dataset. It can be:
    - a Deep Lake cloud path of the form `hub://org_id/dataset_name`. Requires registration with Deep Lake.
    - an s3 path of the form `s3://bucketname/path/to/dataset`. Credentials are required in either the environment or passed to the `creds` argument.
    - a local file system path of the form `./path/to/dataset` or `~/path/to/dataset` or `path/to/dataset`.
    - a memory path of the form `mem://path/to/dataset` which doesn't save the dataset but keeps it in memory instead. Should be used only for testing as it does not persist.
- **runtime** (*dict*) – Parameters for creating a dataset in the Deep Lake Tensor Database. Only applicable for paths of the form `hub://org_id/dataset_name` and runtime must be `{"tensor_db": True}`.
- **overwrite** (*bool*) – If set to `True` this overwrites the dataset if it already exists. Defaults to `False`.
- **public** (*bool*) – Defines if the dataset will have public access. Applicable only if Deep Lake cloud storage is used and a new Dataset is being created. Defaults to `False`.
- **memory\_cache\_size** (*int*) – The size of the memory cache to be used in MB.
- **local\_cache\_size** (*int*) – The size of the local filesystem cache to be used in MB.
- **creds** (*dict*, *str*, *optional*) – The string `ENV` or a dictionary containing credentials used to access the dataset at the path.
  - If `'aws_access_key_id'`, `'aws_secret_access_key'`, `'aws_session_token'` are present, these take precedence over credentials present in the environment or in credentials file. Currently only works with s3 paths.
  - It supports `'aws_access_key_id'`, `'aws_secret_access_key'`, `'aws_session_token'`, `'endpoint_url'`, `'aws_region'`, `'profile_name'` as keys.
  - If `'ENV'` is passed, credentials are fetched from the environment variables. This is also the case when `creds` is not passed for cloud datasets. For datasets connected to hub cloud, specifying `'ENV'` will override the credentials fetched from Activeloop and use local ones.

- **token** (*str*, *optional*) – Activeloop token, used for fetching credentials to the dataset at path if it is a Deep Lake dataset. This is optional, tokens are normally auto-generated.
- **org\_id** (*str*, *Optional*) – Organization id to be used for enabling high-performance features. Only applicable for local datasets.
- **verbose** (*bool*) – If True, logs will be printed. Defaults to True.
- **lock\_timeout** (*int*) – Number of seconds to wait before throwing a LockException. If None, wait indefinitely
- **lock\_enabled** (*bool*) – If true, the dataset manages a write lock. NOTE: Only set to False if you are managing concurrent access externally.
- **index\_params** – *Optional[Dict[str, Union[int, str]]]*: Index parameters used while creating vector store, passed down to dataset.

**Returns**

Dataset created using the arguments provided.

**Return type**

*Dataset*

**Raises**

- ***DatasetHandlerError*** – If a Dataset already exists at the given path and overwrite is False.
- ***UserNotLoggedInException*** – When user is not authenticated
- ***InvalidTokenException*** – If the specified token is invalid
- ***TokenPermissionError*** – When there are permission or other errors related to token
- ***ValueError*** – If version is specified in the path

**Danger:** Setting `overwrite` to `True` will delete all of your data if it exists! Be very careful when setting this parameter.

```
deeplake.like(dest: Union[str, Path], src: Union[str, Dataset, Path], runtime: Optional[Dict] = None, tensors:
Optional[List[str]] = None, overwrite: bool = False, creds: Optional[Union[dict, str]] = None,
token: Optional[str] = None, org_id: Optional[str] = None, public: bool = False, verbose: bool =
True) → Dataset
```

Creates a new dataset by copying the source dataset's structure to a new location. No samples are copied, only the meta/info for the dataset and it's tensors.

**Parameters**

- **dest** – Empty Dataset or Path where the new dataset will be created.
- **src** (*Union[str, Dataset]*) – Path or dataset object that will be used as the template for the new dataset.
- **runtime** (*dict*) – Parameters for Activeloop DB Engine. Only applicable for hub:// paths.
- **tensors** (*List[str]*, *optional*) – Names of tensors (and groups) to be replicated. If not specified all tensors in source dataset are considered.
- **overwrite** (*bool*) – If True and a dataset exists at *destination*, it will be overwritten. Defaults to False.
- **creds** (*dict*, *str*, *optional*) – The string ENV or a dictionary containing credentials used to access the dataset at the path. - If 'aws\_access\_key\_id', 'aws\_secret\_access\_key', 'aws\_session\_token' are present, these take precedence over credentials present in the environment or in credentials file. Currently only works with s3 paths. - It supports 'aws\_access\_key\_id', 'aws\_secret\_access\_key', 'aws\_session\_token', 'endpoint\_url', 'aws\_region', 'profile\_name' as keys. - If 'ENV' is passed, credentials are fetched from the environment variables. This is also the case when creds is not passed for cloud datasets. For datasets connected to hub cloud, specifying 'ENV' will override the credentials fetched from Activeloop and use local ones.
- **token** (*str*, *optional*) – Activeloop token, used for fetching credentials to the

dataset at path if it is a Deep Lake dataset. This is optional, tokens are normally auto-generated.

- **org\_id** (*str*, *Optional*) – Organization id to be used for enabling high-performance features. Only applicable for local datasets.
- **public** (*bool*) – Defines if the dataset will have public access. Applicable only if Deep Lake cloud storage is used and a new Dataset is being created. Defaults to False.
- **verbose** (*bool*) – If True, logs will be printed. Defaults to True.

**Returns**

New dataset object.

**Return type**

*Dataset*

**Raises**

**ValueError** – If **org\_id** is specified for a non-local dataset.

```
deeplake.ingest_classification(src: Union[str, Path], dest: Union[str, Path], image_params: Optional[Dict]
                              = None, label_params: Optional[Dict] = None, dest_creds:
                              Optional[Union[Dict, str]] = None, progressbar: bool = True, summary:
                              bool = True, num_workers: int = 0, shuffle: bool = True, token:
                              Optional[str] = None, connect_kwargs: Optional[Dict] = None, indra: bool
                              = False, **dataset_kwargs) → Dataset
```

Ingest a dataset of images from a local folder to a Deep Lake Dataset. Images should be stored in subfolders by class name.

**Parameters**

- **src** (*str*, *pathlib.Path*) – Local path to where the unstructured dataset of images is stored or path to csv file.
- **dest** (*str*, *pathlib.Path*) –
  - The full path to the dataset. Can be:
  - a Deep Lake cloud path of the form `hub://org_id/datasetname`. To write to Deep Lake cloud datasets, ensure that you are authenticated to Deep Lake (pass in a token using the ‘token’ parameter).
  - an s3 path of the form `s3://bucketname/path/to/dataset`. Credentials are required in either the environment or passed to the `creds` argument.
  - a local file system path of the form `./path/to/dataset` or `~/path/to/dataset` or `path/to/dataset`.
  - a memory path of the form `mem://path/to/dataset` which doesn’t save the dataset but keeps it in memory instead. Should be used only for testing as it does not persist.
- **image\_params** (*Optional[Dict]*) – A dictionary containing parameters for the images tensor.
- **label\_params** (*Optional[Dict]*) – A dictionary containing parameters for the labels tensor.
- **dest\_creds** (*Optional[Union[str, Dict]]*) – The string ENV or a dictionary containing credentials used to access the destination path of the dataset.
- **progressbar** (*bool*) – Enables or disables ingestion progress bar. Defaults to True.
- **summary** (*bool*) – If True, a summary of skipped files will be printed after completion. Defaults to True.
- **num\_workers** (*int*) – The number of workers to use for ingestion. Set to 0 by default.
- **shuffle** (*bool*) – Shuffles the input data prior to ingestion. Since data arranged in folders by class is highly non-random, shuffling is important in order to produce optimal results when training. Defaults to True.
- **token** (*Optional[str]*) – The token to use for accessing the dataset.



- **connect\_kwargs** (*Optional[Dict]*) – If specified, the dataset will be connected to Deep Lake, and connect\_kwargs will be passed to [Dataset.connect](#).
- **indra** (*bool*) – Flag indicating whether indra api should be used to create the dataset. Defaults to false
- **\*\*dataset\_kwargs** – Any arguments passed here will be forwarded to the dataset creator function see [deeplake.empty\(\)](#).

**Returns**

New dataset object with structured dataset.

**Return type**

[Dataset](#)

**Raises**

- [InvalidPathException](#) – If the source directory does not exist.
- [SamePathException](#) – If the source and destination path are same.
- [AutoCompressionError](#) – If the source director is empty or does not contain a valid extension.
- [InvalidFileExtension](#) – If the most frequent file extension is found to be 'None' during auto-compression.

**Note:**

- Currently only local source paths and image classification datasets / csv files are supported for automatic ingestion.
- Supported filetypes: png/jpeg/jpg/csv.
- All files and sub-directories with unsupported filetypes are ignored.
- Valid source directory structures for image classification look like:

```
data/
  img0.jpg
  img1.jpg
  ...
```

- or:

```
data/
  class0/
    cat0.jpg
    ...
  class1/
    dog0.jpg
    ...
  ...
```

- or:

```
data/
  train/
    class0/
      img0.jpg
      ...
    ...
  val/
    class0/
      img0.jpg
      ...
    ...
  ...
```

- Classes defined as sub-directories can be accessed at `ds["test/labels"].info.class_names`.
- Support for train and test sub directories is present under `ds["train/images"]`, `ds["train/labels"]` and `ds["test/images"]`, `ds["test/labels"]`.
- Mapping filenames to classes from an external file is currently not supported.

```
deeplake.ingest_coco(images_directory: Union[str, Path], annotation_files: Union[str, Path, List[str]], dest:
    Union[str, Path], key_to_tensor_mapping: Optional[Dict] = None,
    file_to_group_mapping: Optional[Dict] = None, ignore_one_group: bool = True,
    ignore_keys: Optional[List[str]] = None, image_params: Optional[Dict] = None,
    image_creds_key: Optional[str] = None, src_creds: Optional[Union[Dict, str]] = None,
    dest_creds: Optional[Union[Dict, str]] = None, inspect_limit: int = 1000000,
    progressbar: bool = True, shuffle: bool = False, num_workers: int = 0, token:
    Optional[str] = None, connect_kwargs: Optional[Dict] = None, **dataset_kwargs) →
    Dataset
```

Ingest images and annotations in COCO format to a Deep Lake Dataset. The source data can be stored locally or in the cloud.

## Examples

```
>>> # Ingest local data in COCO format to a Deep Lake dataset stored in Deep Lake_
↳storage.
>>> ds = deeplake.ingest_coco(
>>>     "<path/to/images/directory>",
>>>     ["path/to/annotation/file1.json", "path/to/annotation/file2.json"],
>>>     dest="hub://org_id/dataset",
>>>     key_to_tensor_mapping={"category_id": "labels", "bbox": "boxes"},
>>>     file_to_group_mapping={"file1.json": "group1", "file2.json": "group2"},
>>>     ignore_keys=["area", "image_id", "id"],
>>>     num_workers=4,
>>> )
>>> # Ingest data from your cloud into another Deep Lake dataset in your cloud, and_
↳connect that dataset to the Deep Lake backend.
>>> ds = deeplake.ingest_coco(
>>>     "s3://bucket/images/directory",
>>>     "s3://bucket/annotation/file1.json",
>>>     dest="s3://bucket/dataset_name",
>>>     ignore_one_group=True,
>>>     ignore_keys=["area", "image_id", "id"],
>>>     image_settings={"name": "images", "htype": "link[image]", "sample_
↳compression": "jpeg"},
>>>     image_creds_key="my_s3_managed_credentials",
>>>     src_creds=aws_creds, # Can also be inferred from environment
>>>     dest_creds=aws_creds, # Can also be inferred from environment
>>>     connect_kwargs={"creds_key": "my_s3_managed_credentials", "org_id": "org_id
↳"},
>>>     num_workers=4,
>>> )
```

## Parameters

- **images\_directory** (*str*, *pathlib.Path*) – The path to the directory containing images.

- **annotation\_files** (*str*, *pathlib.Path*, *List[str]*) – Path to JSON annotation files in COCO format.
- **dest** (*str*, *pathlib.Path*) –
  - The full path to the dataset. Can be:
    - a Deep Lake cloud path of the form `hub://org_id/datasetname`. To write to Deep Lake cloud datasets, ensure that you are authenticated to Deep Lake (pass in a token using the ‘token’ parameter).
    - an s3 path of the form `s3://bucketname/path/to/dataset`. Credentials are required in either the environment or passed to the creds argument.
    - a local file system path of the form `./path/to/dataset` or `~/path/to/dataset` or `path/to/dataset`.
    - a memory path of the form `mem://path/to/dataset` which doesn’t save the dataset but keeps it in memory instead. Should be used only for testing as it does not persist.
- **key\_to\_tensor\_mapping** (*Optional[Dict]*) – A one-to-one mapping between COCO keys and Dataset tensor names.
- **file\_to\_group\_mapping** (*Optional[Dict]*) – A one-to-one mapping between COCO annotation file names and Dataset group names.
- **ignore\_one\_group** (*bool*) – Skip creation of group in case of a single annotation file. Set to `False` by default.
- **ignore\_keys** (*List[str]*) – A list of COCO keys to ignore.
- **image\_params** (*Optional[Dict]*) – A dictionary containing parameters for the images tensor.
- **image\_creds\_key** (*Optional[str]*) – The name of the managed credentials to use for accessing the images in the linked tensor (is applicable).
- **src\_creds** (*Optional[Union[str, Dict]]*) – Credentials to access the source data. If not provided, will be inferred from the environment.
- **dest\_creds** (*Optional[Union[str, Dict]]*) – The string ENV or a dictionary containing credentials used to access the destination path of the dataset.
- **inspect\_limit** (*int*) – The maximum number of samples to inspect in the annotations json, in order to generate the set of COCO annotation keys. Set to `1000000` by default.
- **progressbar** (*bool*) – Enables or disables ingestion progress bar. Set to `True` by default.
- **shuffle** (*bool*) – Shuffles the input data prior to ingestion. Set to `False` by default.
- **num\_workers** (*int*) – The number of workers to use for ingestion. Set to `0` by default.
- **token** (*Optional[str]*) – The token to use for accessing the dataset and/or connecting it to Deep Lake.
- **connect\_kwargs** (*Optional[Dict]*) – If specified, the dataset will be connected to Deep Lake, and `connect_kwargs` will be passed to [Dataset.connect](#).
- **\*\*dataset\_kwargs** – Any arguments passed here will be forwarded to the dataset creator function. See [deeplake.empty\(\)](#).

**Returns**

The Dataset created from images and COCO annotations.

**Return type**

[Dataset](#)

**Raises**

**IngestionError** – If either `key_to_tensor_mapping` or `file_to_group_mapping` are not one-to-one.

```
deeplake.ingest_yolo(data_directory: Union[str, Path], dest: Union[str, Path], class_names_file:
Optional[Union[str, Path]] = None, annotations_directory: Optional[Union[str, Path]] =
None, allow_no_annotation: bool = False, image_params: Optional[Dict] = None,
label_params: Optional[Dict] = None, coordinates_params: Optional[Dict] = None,
src_creds: Optional[Union[Dict, str]] = None, dest_creds: Optional[Union[Dict, str]] =
None, image_creds_key: Optional[str] = None, inspect_limit: int = 1000, progressbar:
bool = True, shuffle: bool = False, num_workers: int = 0, token: Optional[str] = None,
connect_kwargs: Optional[Dict] = None, **dataset_kwargs) → Dataset
```

Ingest images and annotations (bounding boxes or polygons) in YOLO format to a Deep Lake Dataset. The source data can be stored locally or in the cloud.

## Examples

```
>>> # Ingest local data in YOLO format to a Deep Lake dataset stored in Deep Lake_
↳ storage.
>>> ds = deeplake.ingest_yolo(
>>>     "path/to/data/directory",
>>>     dest="hub://org_id/dataset",
>>>     allow_no_annotation=True,
>>>     token="my_activeloop_token",
>>>     num_workers=4,
>>> )
>>> # Ingest data from your cloud into another Deep Lake dataset in your cloud, and_
↳ connect that dataset to the Deep Lake backend.
>>> ds = deeplake.ingest_yolo(
>>>     "s3://bucket/data_directory",
>>>     dest="s3://bucket/dataset_name",
>>>     image_params={"name": "image_links", "htype": "link[image]"},
>>>     image_creds_key="my_s3_managed_credentials",
>>>     src_creds=aws_creds, # Can also be inferred from environment
>>>     dest_creds=aws_creds, # Can also be inferred from environment
>>>     connect_kwargs={"creds_key": "my_s3_managed_credentials", "org_id": "org_id
↳"},
>>>     num_workers=4,
>>> )
```

## Parameters

- **data\_directory** (*str*, *pathlib.Path*) – The path to the directory containing the data (images files and annotation files(see ‘annotations\_directory’ input for specifying annotations in a separate directory).
- **dest** (*str*, *pathlib.Path*) –
  - The full path to the dataset. Can be:
    - a Deep Lake cloud path of the form `hub://org_id/datasetname`. To write to Deep Lake cloud datasets, ensure that you are authenticated to Deep Lake (pass in a token using the ‘token’ parameter).
    - an s3 path of the form `s3://bucketname/path/to/dataset`. Credentials are required in either the environment or passed to the creds argument.
    - a local file system path of the form `./path/to/dataset` or `~/path/to/dataset` or `path/to/dataset`.

- a memory path of the form `mem://path/to/dataset` which doesn't save the dataset but keeps it in memory instead. Should be used only for testing as it does not persist.
- **class\_names\_file** – Path to the file containing the class names on separate lines. This is typically a file titled `classes.names`.
- **annotations\_directory** (*Optional[Union[str, pathlib.Path]]*) – Path to directory containing the annotations. If specified, the 'data\_directory' will not be examined for annotations.
- **allow\_no\_annotation** (*bool*) – Flag to determine whether missing annotations files corresponding to an image should be treated as empty annotations. Set to `False` by default.
- **image\_params** (*Optional[Dict]*) – A dictionary containing parameters for the images tensor.
- **label\_params** (*Optional[Dict]*) – A dictionary containing parameters for the labels tensor.
- **coordinates\_params** (*Optional[Dict]*) – A dictionary containing parameters for the coordinates tensor. This tensor either contains bounding boxes or polygons.
- **src\_creds** (*Optional[Union[str, Dict]]*) – Credentials to access the source data. If not provided, will be inferred from the environment.
- **dest\_creds** (*Optional[Union[str, Dict]]*) – The string ENV or a dictionary containing credentials used to access the destination path of the dataset.
- **image\_creds\_key** (*Optional[str]*) – `creds_key` for linked tensors, applicable if the `htype` for the images tensor is specified as 'link[image]' in the 'image\_params' input.
- **inspect\_limit** (*int*) – The maximum number of annotations to inspect, in order to infer whether they are bounding boxes or polygons. This input is ignored if the `htype` is specified in the 'coordinates\_params'.
- **progressbar** (*bool*) – Enables or disables ingestion progress bar. Set to `True` by default.
- **shuffle** (*bool*) – Shuffles the input data prior to ingestion. Set to `False` by default.
- **num\_workers** (*int*) – The number of workers to use for ingestion. Set to `0` by default.
- **token** (*Optional[str]*) – The token to use for accessing the dataset and/or connecting it to Deep Lake.
- **connect\_kwargs** (*Optional[Dict]*) – If specified, the dataset will be connected to Deep Lake, and `connect_kwargs` will be passed to `Dataset.connect`.
- **\*\*dataset\_kwargs** – Any arguments passed here will be forwarded to the dataset creator function. See `deeplake.empty()`.

**Returns**

The Dataset created from the images and YOLO annotations.

**Return type**

`Dataset`

**Raises**

**IngestionError** – If annotations are not found for all the images and 'allow\_no\_annotation' is `False`

```
deeplake.ingest_kaggle(tag: str, src: Union[str, Path], dest: Union[str, Path], exist_ok: bool = False,
                      images_compression: str = 'auto', dest_creds: Optional[Union[Dict, str]] = None,
                      kaggle_credentials: Optional[dict] = None, progressbar: bool = True, summary: bool
                      = True, shuffle: bool = True, indra: bool = False, **dataset_kwargs) → Dataset
```

Download and ingest a kaggle dataset and store it as a structured dataset to destination.

**Parameters**

- **tag** (*str*) – Kaggle dataset tag. Example: "coloradokb/dandelionimages" points to <https://www.kaggle.com/coloradokb/dandelionimages>
- **src** (*str, pathlib.Path*) – Local path to where the raw kaggle dataset will be downloaded to.

- **dest** (*str*, *pathlib.Path*) –
  - The full path to the dataset. Can be:
    - a Deep Lake cloud path of the form `hub://username/datasetname`. To write to Deep Lake cloud datasets, ensure that you are authenticated to Deep Lake (pass in a token using the ‘token’ parameter).
    - an s3 path of the form `s3://bucketname/path/to/dataset`. Credentials are required in either the environment or passed to the creds argument.
    - a local file system path of the form `./path/to/dataset` or `~/path/to/dataset` or `path/to/dataset`.
    - a memory path of the form `mem://path/to/dataset` which doesn’t save the dataset but keeps it in memory instead. Should be used only for testing as it does not persist.
- **exist\_ok** (*bool*) – If the kaggle dataset was already downloaded and `exist_ok` is True, ingestion will proceed without error.
- **images\_compression** (*str*) – For image classification datasets, this compression will be used for the `images` tensor. If `images_compression` is “auto”, compression will be automatically determined by the most common extension in the directory.
- **dest\_creds** (*Optional[Union[str, Dict]]*) – The string ENV or a dictionary containing credentials used to access the destination path of the dataset.
- **kaggle\_credentials** (*dict*) – A dictionary containing kaggle credentials {“username”: “YOUR\_USERNAME”, “key”: “YOUR\_KEY”}. If None, environment variables/the `kaggle.json` file will be used if available.
- **progressbar** (*bool*) – Enables or disables ingestion progress bar. Set to True by default.
- **summary** (*bool*) – Generates ingestion summary. Set to True by default.
- **shuffle** (*bool*) – Shuffles the input data prior to ingestion. Since data arranged in folders by class is highly non-random, shuffling is important in order to produce optimal results when training. Defaults to True.
- **indra** (*bool*) – Flag indicating whether indra api should be used to create the dataset. Defaults to false
- **\*\*dataset\_kwargs** – Any arguments passed here will be forwarded to the dataset creator function. See [deeplake.dataset\(\)](#).

**Returns**

New dataset object with structured dataset.

**Return type**

[Dataset](#)

**Raises**

[SamePathException](#) – If the source and destination path are same.

---

**Note:** Currently only local source paths and image classification datasets are supported for automatic ingestion.

---

```
deeplake.ingest_dataframe(src, dest: Union[str, Path], column_params: Optional[Dict] = None, src_creds:
    Optional[Union[Dict, str]] = None, dest_creds: Optional[Union[Dict, str]] =
    None, creds_key: Optional[Dict] = None, progressbar: bool = True, token:
    Optional[str] = None, connect_kwargs: Optional[Dict] = None, indra: bool =
    False, **dataset_kwargs)
```

Convert pandas dataframe to a Deep Lake Dataset. The contents of the dataframe can be parsed literally, or can be treated as links to local or cloud files.

## Examples

```
>>> # Ingest data from a DataFrame into a Deep Lake dataset stored in Deep Lake
↳storage.
>>> ds = deeplake.ingest_dataframe(
>>>     df,
>>>     dest="hub://org_id/dataset",
>>> )
>>> # Ingest data from a DataFrame into a Deep Lake dataset stored in Deep Lake
↳storage. The filenames in `df_column_with_cloud_paths` will be used as the
↳filenames for loading data into the dataset.
>>> ds = deeplake.ingest_dataframe(
>>>     df,
>>>     dest="hub://org_id/dataset",
>>>     column_params={"df_column_with_cloud_paths": {"name": "images", "htype":
↳"image"}},
>>>     src_creds=aws_creds
>>> )
>>> # Ingest data from a DataFrame into a Deep Lake dataset stored in Deep Lake
↳storage. The filenames in `df_column_with_cloud_paths` will be used as the
↳filenames for linked data in the dataset.
>>> ds = deeplake.ingest_dataframe(
>>>     df,
>>>     dest="hub://org_id/dataset",
>>>     column_params={"df_column_with_cloud_paths": {"name": "image_links", "htype":
↳": "link[image]"}},
>>>     creds_key="my_s3_managed_credentials"
>>> )
>>> # Ingest data from a DataFrame into a Deep Lake dataset stored in your cloud,
↳and connect that dataset to the Deep Lake backend. The filenames in `df_column_
↳with_cloud_paths` will be used as the filenames for linked data in the dataset.
>>> ds = deeplake.ingest_dataframe(
>>>     df,
>>>     dest="s3://bucket/dataset_name",
>>>     column_params={"df_column_with_cloud_paths": {"name": "image_links", "htype":
↳": "link[image]"}},
>>>     creds_key="my_s3_managed_credentials"
>>>     connect_kwargs={"creds_key": "my_s3_managed_credentials", "org_id": "org_id
↳"},
>>> )
```

### Parameters

- **src** (*pd.DataFrame*) – The pandas dataframe to be converted.
- **dest** (*str, pathlib.Path*) –
  - A Dataset or The full path to the dataset. Can be:
    - a Deep Lake cloud path of the form `hub://username/datasetname`. To write to Deep Lake cloud datasets, ensure that you are authenticated to Deep Lake (pass in a token using the ‘token’ parameter).
    - an s3 path of the form `s3://bucketname/path/to/dataset`. Credentials are required in either the environment or passed to the creds argument.
    - a local file system path of the form `./path/to/dataset` or `~/path/to/dataset` or `path/to/dataset`.



- a memory path of the form `mem://path/to/dataset` which doesn't save the dataset but keeps it in memory instead. Should be used only for testing as it does not persist.
- **column\_params** (*Optional[Dict]*) – A dictionary containing parameters for the tensors corresponding to the dataframe columns.
- **src\_creds** (*Optional[Union[str, Dict]]*) – Credentials to access the source data. If not provided, will be inferred from the environment.
- **dest\_creds** (*Optional[Union[str, Dict]]*) – The string ENV or a dictionary containing credentials used to access the destination path of the dataset.
- **creds\_key** (*Optional[str]*) – creds\_key for linked tensors, applicable if the htype any tensor is specified as 'link[...]' in the 'column\_params' input.
- **progressbar** (*bool*) – Enables or disables ingestion progress bar. Set to True by default.
- **token** (*Optional[str]*) – The token to use for accessing the dataset.
- **connect\_kwargs** (*Optional[Dict]*) – A dictionary containing arguments to be passed to the dataset connect method. See `Dataset.connect()`.
- **indra** (*bool*) – Flag indicating whether indra api should be used to create the dataset. Defaults to false
- **\*\*dataset\_kwargs** – Any arguments passed here will be forwarded to the dataset creator function. See `deeplake.empty()`.

**Returns**

New dataset created from the dataframe.

**Return type**

*Dataset*

**Raises**

**Exception** – If `src` is not a valid pandas dataframe object.

`deeplake.ingest_huggingface(src, dest, use_progressbar=True, token: Optional[str] = None, connect_kwargs: Optional[Dict] = None, **dataset_kwargs) → Dataset`

Converts Hugging Face datasets to Deep Lake format.

**Parameters**

- **src** (*hfDataset, DatasetDict*) – Hugging Face Dataset or DatasetDict to be converted. Data in different splits of a DatasetDict will be stored under respective tensor groups.
- **dest** (*Dataset, str, pathlib.Path*) – Destination dataset or path to it.
- **use\_progressbar** (*bool*) – Defines if progress bar should be used to show conversion progress.
- **token** (*Optional[str]*) – The token to use for accessing the dataset and/or connecting it to Deep Lake.
- **connect\_kwargs** (*Optional[Dict]*) – If specified, the dataset will be connected to Deep Lake, and connect\_kwargs will be passed to `Dataset.connect`.
- **\*\*dataset\_kwargs** – Any arguments passed here will be forwarded to the dataset creator function. See `deeplake.empty()`.

**Returns**

The destination Deep Lake dataset.

**Return type**

*Dataset*

**Raises**

**ValueError** – If `dest` is not a path or a Deep Lake *Dataset*.

---

**Note:**

- if DatasetDict looks like:



```
>>> {
...   train: Dataset({
...     features: ['data']
...   }),
...   validation: Dataset({
...     features: ['data']
...   }),
...   test: Dataset({
...     features: ['data']
...   }),
... }
```

it will be converted to a Deep Lake Dataset with tensors ['train/data', 'validation/data', 'test/data'].

Features of the type Sequence(feature=Value(dtype='string')) are not supported. Columns of such type are skipped.

```
deeplake.load(path: Union[str, Path], read_only: Optional[bool] = None, memory_cache_size: int = 2000,
local_cache_size: int = 0, creds: Optional[Union[dict, str]] = None, token: Optional[str] = None,
org_id: Optional[str] = None, verbose: bool = True, access_method: str = 'stream', unlink: bool
= False, reset: bool = False, indra: bool = False, check_integrity: Optional[bool] = None,
lock_timeout: Optional[int] = 0, lock_enabled: Optional[bool] = True, index_params:
Optional[Dict[str, Union[int, str]]] = None) → Dataset
```

Loads an existing dataset

## Examples

```
>>> ds = deeplake.load("hub://username/dataset")
>>> ds = deeplake.load("s3://mybucket/my_dataset")
>>> ds = deeplake.load("./datasets/my_dataset", overwrite=True)
```

Loading to a specific version:

```
>>> ds = deeplake.load("hub://username/dataset@new_branch")
>>> ds = deeplake.load("hub://username/
↳ dataset@3e49cded62b6b335c74ff07e97f8451a37aca7b2)
```

```
>>> my_commit_id = "3e49cded62b6b335c74ff07e97f8451a37aca7b2"
>>> ds = deeplake.load(f"hub://username/dataset@{my_commit_id}")
```

## Parameters

- **path** (str, pathlib.Path) –
  - The full path to the dataset. Can be:
    - a Deep Lake cloud path of the form `hub://username/datasetname`. To write to Deep Lake cloud datasets, ensure that you are authenticated to Deep Lake (pass in a token using the ‘token’ parameter).
    - an s3 path of the form `s3://bucketname/path/to/dataset`. Credentials are required in either the environment or passed to the creds argument.
    - a local file system path of the form `./path/to/dataset` or `~/path/to/dataset` or `path/to/dataset`.

- a memory path of the form `mem://path/to/dataset` which doesn't save the dataset but keeps it in memory instead. Should be used only for testing as it does not persist.
- Loading to a specific version:
  - \* You can also specify a `commit_id` or `branch` to load the dataset to that version directly by using the `@` symbol.
  - \* The path will then be of the form `hub://username/dataset@{branch}` or `hub://username/dataset@{commit_id}`.
  - \* See examples above.
- **read\_only** (*bool*, *optional*) – Opens dataset in read only mode if this is passed as `True`. Defaults to `False`. Datasets stored on Deep Lake cloud that your account does not have write access to will automatically open in read mode.
- **memory\_cache\_size** (*int*) – The size of the memory cache to be used in MB.
- **local\_cache\_size** (*int*) – The size of the local filesystem cache to be used in MB.
- **creds** (*dict*, *str*, *optional*) – The string ENV or a dictionary containing credentials used to access the dataset at the path. - If `'aws_access_key_id'`, `'aws_secret_access_key'`, `'aws_session_token'` are present, these take precedence over credentials present in the environment or in credentials file. Currently only works with s3 paths. - It supports `'aws_access_key_id'`, `'aws_secret_access_key'`, `'aws_session_token'`, `'endpoint_url'`, `'aws_region'`, `'profile_name'` as keys. - If `'ENV'` is passed, credentials are fetched from the environment variables. This is also the case when creds is not passed for cloud datasets. For datasets connected to hub cloud, specifying `'ENV'` will override the credentials fetched from Activeloop and use local ones.
- **token** (*str*, *optional*) – Activeloop token, used for fetching credentials to the dataset at path if it is a Deep Lake dataset. This is optional, tokens are normally auto-generated.
- **org\_id** (*str*, *Optional*) – Organization id to be used for enabling high-performance features. Only applicable for local datasets.
- **verbose** (*bool*) – If `True`, logs will be printed. Defaults to `True`.
- **access\_method** (*str*) – The access method to use for the dataset. Can be:
  - `'stream'`
    - \* Streams the data from the dataset i.e. only fetches data when required. This is the default value.
  - `'download'`
    - \* Downloads the data to the local filesystem to the path specified in environment variable `DEEPLAKE_DOWNLOAD_PATH`. This will overwrite `DEEPLAKE_DOWNLOAD_PATH`.
    - \* Raises an exception if `DEEPLAKE_DOWNLOAD_PATH` environment variable is not set or if the dataset does not exist.
    - \* The `'download'` access method can be modified to specify `num_workers` and/or scheduler. For example: `'download:2:processed'` will use 2 workers and use processed scheduler, while `'download:3'` will use 3 workers and default scheduler (threaded), and `'download:processed'` will use a single worker and use processed scheduler.
  - `'local'`
    - \* Downloads the dataset if it doesn't already exist, otherwise loads from local storage.

- \* Raises an exception if `DEEPLAKE_DOWNLOAD_PATH` environment variable is not set.
- \* The 'local' access method can be modified to specify `num_workers` and/or scheduler to be used in case dataset needs to be downloaded. If dataset needs to be downloaded, 'local:2:processed' will use 2 workers and use processed scheduler, while 'local:3' will use 3 workers and default scheduler (threaded), and 'local:processed' will use a single worker and use processed scheduler.
- **unlink** (*bool*) – Downloads linked samples if set to True. Only applicable if `access_method` is `download` or `local`. Defaults to False.
- **reset** (*bool*) – If the specified dataset cannot be loaded due to a corrupted HEAD state of the branch being loaded, setting `reset=True` will reset HEAD changes and load the previous version.
- **check\_integrity** (*bool, Optional*) – Performs an integrity check by default (None) if the dataset has 20 or fewer tensors. Set to True to force integrity check, False to skip integrity check.
- **indra** (*bool*) – Flag indicating whether indra api should be used to create the dataset. Defaults to false

**Returns**

Dataset loaded using the arguments provided.

**Return type**

*Dataset*

**Raises**

- ***DatasetHandlerError*** – If a Dataset does not exist at the given path.
- ***AgreementError*** – When agreement is rejected
- ***UserNotLoggedInException*** – When user is not authenticated
- ***InvalidTokenException*** – If the specified token is invalid
- ***TokenPermissionError*** – When there are permission or other errors related to token
- ***CheckoutError*** – If version address specified in the path cannot be found
- ***DatasetCorruptError*** – If loading the dataset failed due to corruption and `reset` is not True
- ***ReadOnlyModeError*** – If reset is attempted in read-only mode
- ***LockedException*** – When attempting to open a dataset for writing when it is locked by another machine
- ***ValueError*** – If `org_id` is specified for a non-local dataset
- ***Exception*** – Re-raises caught exception if reset cannot fix the issue
- ***ValueError*** – If the org id is provided but the dataset is not local

**Warning:** Setting `access_method` to `download` will overwrite the local copy of the dataset if it was previously downloaded.

**Note:** Any changes made to the dataset in `download` / `local` mode will only be made to the local copy and will not be reflected in the original dataset.

`deeplake.delete(path: Union[str, Path], force: bool = False, large_ok: bool = False, creds: Optional[Union[dict, str]] = None, token: Optional[str] = None, verbose: bool = False) → None`

Deletes a dataset at a given path.

**Parameters**

- **path** (*str, pathlib.Path*) – The path to the dataset to be deleted.

- **force** (*bool*) – Delete data regardless of whether it looks like a deeplake dataset. All data at the path will be removed if set to True.
- **large\_ok** (*bool*) – Delete datasets larger than 1GB. Disabled by default.
- **creds** (*dict, str, optional*) – The string ENV or a dictionary containing credentials used to access the dataset at the path. - If 'aws\_access\_key\_id', 'aws\_secret\_access\_key', 'aws\_session\_token' are present, these take precedence over credentials present in the environment or in credentials file. Currently only works with s3 paths. - It supports 'aws\_access\_key\_id', 'aws\_secret\_access\_key', 'aws\_session\_token', 'endpoint\_url', 'aws\_region', 'profile\_name' as keys. - If 'ENV' is passed, credentials are fetched from the environment variables. This is also the case when creds is not passed for cloud datasets. For datasets connected to hub cloud, specifying 'ENV' will override the credentials fetched from Activeloop and use local ones.
- **token** (*str, optional*) – Activeloop token, used for fetching credentials to the dataset at path if it is a Deep Lake dataset. This is optional, tokens are normally auto-generated.
- **verbose** (*bool*) – If True, logs will be printed. Defaults to True.

**Raises**

- **DatasetHandlerError** – If a Dataset does not exist at the given path and force = False.
- **UserNotLoggedInException** – When user is not authenticated.
- **NotImplementedError** – When attempting to delete a managed view.
- **ValueError** – If version is specified in the path

**Warning:** This is an irreversible operation. Data once deleted cannot be recovered.

```
deeplake.rename(old_path: Union[str, Path], new_path: Union[str, Path], creds: Optional[Union[dict, str]] = None, token: Optional[str] = None) → Dataset
```

Renames dataset at old\_path to new\_path.

**Examples**

```
>>> deeplake.rename("hub://username/image_ds", "hub://username/new_ds")
>>> deeplake.rename("s3://mybucket/my_ds", "s3://mybucket/renamed_ds")
```

**Parameters**

- **old\_path** (*str, pathlib.Path*) – The path to the dataset to be renamed.
- **new\_path** (*str, pathlib.Path*) – Path to the dataset after renaming.
- **creds** (*dict, str, optional*) – The string ENV or a dictionary containing credentials used to access the dataset at the path. - If 'aws\_access\_key\_id', 'aws\_secret\_access\_key', 'aws\_session\_token' are present, these take precedence over credentials present in the environment or in credentials file. Currently only works with s3 paths. - It supports 'aws\_access\_key\_id', 'aws\_secret\_access\_key', 'aws\_session\_token', 'endpoint\_url', 'aws\_region', 'profile\_name' as keys. - If 'ENV' is passed, credentials are fetched from the environment variables. This is also the case when creds is not passed for cloud datasets. For datasets connected to hub cloud, specifying 'ENV' will override the credentials fetched from Activeloop and use local ones.
- **token** (*str, optional*) – Activeloop token, used for fetching credentials to the dataset at path if it is a Deep Lake dataset. This is optional, tokens are normally auto-generated.

**Returns**

The renamed Dataset.

**Return type***Dataset***Raises**

**DatasetHandlerError** – If a Dataset does not exist at the given path or if new path is to a different directory.

`deeplake.copy(src: Union[str, Path, Dataset], dest: Union[str, Path], runtime: Optional[dict] = None, tensors: Optional[List[str]] = None, overwrite: bool = False, src_creds=None, dest_creds=None, token=None, num_workers: int = 0, scheduler='threaded', progressbar=True, **kwargs)`

Copies dataset at src to dest. Version control history is not included.

**Parameters**

- **src** (*str*, *Dataset*, *pathlib.Path*) – The Dataset or the path to the dataset to be copied.
- **dest** (*str*, *pathlib.Path*) – Destination path to copy to.
- **runtime** (*dict*) – Parameters for Activeloop DB Engine. Only applicable for hub:// paths.
- **tensors** (*List[str]*, *optional*) – Names of tensors (and groups) to be copied. If not specified all tensors are copied.
- **overwrite** (*bool*) – If True and a dataset exists at **dest**, it will be overwritten. Defaults to False.
- **src\_creds** (*dict*, *str*, *optional*) – The string ENV or a dictionary containing credentials used to access the dataset at the path. - If 'aws\_access\_key\_id', 'aws\_secret\_access\_key', 'aws\_session\_token' are present, these take precedence over credentials present in the environment or in credentials file. Currently only works with s3 paths. - It supports 'aws\_access\_key\_id', 'aws\_secret\_access\_key', 'aws\_session\_token', 'endpoint\_url', 'aws\_region', 'profile\_name' as keys. - If 'ENV' is passed, credentials are fetched from the environment variables. This is also the case when creds is not passed for cloud datasets. For datasets connected to hub cloud, specifying 'ENV' will override the credentials fetched from Activeloop and use local ones.
- **dest\_creds** (*dict*, *optional*) – creds required to create / overwrite datasets at **dest**.
- **token** (*str*, *optional*) – Activeloop token, used for fetching credentials to the dataset at path if it is a Deep Lake dataset. This is optional, tokens are normally auto-generated.
- **num\_workers** (*int*) – The number of workers to use for copying. Defaults to 0. When set to 0, it will always use serial processing, irrespective of the scheduler.
- **scheduler** (*str*) – The scheduler to be used for copying. Supported values include: 'serial', 'threaded', 'processed' and 'ray'. Defaults to 'threaded'.
- **progressbar** (*bool*) – Displays a progress bar if True (default).
- **\*\*kwargs** (*dict*) – Additional keyword arguments

**Returns**

New dataset object.

**Return type***Dataset***Raises**

- **DatasetHandlerError** – If a dataset already exists at destination path and overwrite is False.
- **UnsupportedParameterException** – If a parameter that is no longer supported is specified.
- **DatasetCorruptError** – If loading source dataset fails with DatasetCorruptedError.

`deeplake.deepcopy(src: Union[str, Path, Dataset], dest: Union[str, Path], runtime: Optional[Dict] = None, tensors: Optional[List[str]] = None, overwrite: bool = False, src_creds=None, dest_creds=None, token=None, num_workers: int = 0, scheduler='threaded', progressbar=True, public: bool = False, verbose: bool = True, **kwargs)`

Copies dataset at `src` to `dest` including version control history.

**Parameters**

- **src** (*str*, *pathlib.Path*, *Dataset*) – The Dataset or the path to the dataset to be copied.
- **dest** (*str*, *pathlib.Path*) – Destination path to copy to.
- **runtime** (*dict*) – Parameters for Activeloop DB Engine. Only applicable for hub:// paths.
- **tensors** (*List[str]*, *optional*) – Names of tensors (and groups) to be copied. If not specified all tensors are copied.
- **overwrite** (*bool*) – If True and a dataset exists at *destination*, it will be overwritten. Defaults to False.
- **src\_creds** (*dict*, *str*, *optional*) – The string ENV or a dictionary containing credentials used to access the dataset at the path. - If ‘aws\_access\_key\_id’, ‘aws\_secret\_access\_key’, ‘aws\_session\_token’ are present, these take precedence over credentials present in the environment or in credentials file. Currently only works with s3 paths. - It supports ‘aws\_access\_key\_id’, ‘aws\_secret\_access\_key’, ‘aws\_session\_token’, ‘endpoint\_url’, ‘aws\_region’, ‘profile\_name’ as keys. - If ‘ENV’ is passed, credentials are fetched from the environment variables. This is also the case when creds is not passed for cloud datasets. For datasets connected to hub cloud, specifying ‘ENV’ will override the credentials fetched from Activeloop and use local ones.
- **dest\_creds** (*dict*, *optional*) – creds required to create / overwrite datasets at *dest*.
- **token** (*str*, *optional*) – Activeloop token, used for fetching credentials to the dataset at path if it is a Deep Lake dataset. This is optional, tokens are normally auto-generated.
- **num\_workers** (*int*) – The number of workers to use for copying. Defaults to 0. When set to 0, it will always use serial processing, irrespective of the scheduler.
- **scheduler** (*str*) – The scheduler to be used for copying. Supported values include: ‘serial’, ‘threaded’, ‘processed’ and ‘ray’. Defaults to ‘threaded’.
- **progressbar** (*bool*) – Displays a progress bar if True (default).
- **public** (*bool*) – Defines if the dataset will have public access. Applicable only if Deep Lake cloud storage is used and a new Dataset is being created. Defaults to False.
- **verbose** (*bool*) – If True, logs will be printed. Defaults to True.
- **\*\*kwargs** – Additional keyword arguments

**Returns**

New dataset object.

**Return type**

*Dataset*

**Raises**

- **DatasetHandlerError** – If a dataset already exists at destination path and overwrite is False.
- **TypeError** – If source is not a dataset.
- **UnsupportedParameterException** – If parameter that is no longer supported is being called.
- **DatasetCorruptError** – If loading source dataset fails with DatasetCorruptError

`deeplake.connect(src_path: str, creds_key: str, dest_path: Optional[str] = None, org_id: Optional[str] = None, ds_name: Optional[str] = None, token: Optional[str] = None) → Dataset`

Connects dataset at `src_path` to Deep Lake via the provided path.

## Examples

```
>>> # Connect an s3 dataset
>>> ds = deeplake.connect(src_path="s3://bucket/dataset", dest_path="hub://my_org/
↳dataset", creds_key="my_managed_credentials_key", token="my_activeloop_token")
>>> # or
>>> ds = deeplake.connect(src_path="s3://bucket/dataset", org_id="my_org", creds_
↳key="my_managed_credentials_key", token="my_activeloop_token")
```

### Parameters

- **src\_path** (*str*) – Cloud path to the source dataset. Can be: an s3 path like `s3://bucket/path/to/dataset`. a gcs path like `gcs://bucket/path/to/dataset`. an azure path like `az://account_name/container/path/to/dataset`.
- **creds\_key** (*str*) – The managed credentials to be used for accessing the source path.
- **dest\_path** (*str, optional*) – The full path to where the connected Deep Lake dataset will reside. Can be: a Deep Lake path like `hub://organization/dataset`
- **org\_id** (*str, optional*) – The organization to where the connected Deep Lake dataset will be added.
- **ds\_name** (*str, optional*) – The name of the connected Deep Lake dataset. Will be inferred from `dest_path` or `src_path` if not provided.
- **token** (*str, optional*) – Activeloop token used to fetch the managed credentials.

### Returns

The connected Deep Lake dataset.

### Return type

*Dataset*

### Raises

- **InvalidSourcePathError** – If the `src_path` is not a valid s3, gcs or azure path.
- **InvalidDestinationPathError** – If `dest_path`, or `org_id` and `ds_name` do not form a valid Deep Lake path.
- **TokenPermissionError** – If the user does not have permission to create a dataset in the specified organization.

`deeplake.exists(path: Union[str, Path], creds: Optional[Union[Dict, str]] = None, token: Optional[str] = None) → bool`

Checks if a dataset exists at the given path.

### Parameters

- **path** (*str, pathlib.Path*) – the path which needs to be checked.
- **creds** (*dict, str, optional*) – The string ENV or a dictionary containing credentials used to access the dataset at the path. - If 'aws\_access\_key\_id', 'aws\_secret\_access\_key', 'aws\_session\_token' are present, these take precedence over credentials present in the environment or in credentials file. Currently only works with s3 paths. - It supports 'aws\_access\_key\_id', 'aws\_secret\_access\_key', 'aws\_session\_token', 'endpoint\_url', 'aws\_region', 'profile\_name' as keys. - If 'ENV' is passed, credentials are fetched from the environment variables. This is also the case when creds is not passed for cloud datasets. For datasets connected to hub cloud, specifying 'ENV' will override the credentials fetched from Activeloop and use local ones.
- **token** (*str, optional*) – Activeloop token, used for fetching credentials to the dataset at path if it is a Deep Lake dataset. This is optional, tokens are normally auto-generated.

### Returns

A boolean confirming whether the dataset exists or not at the given path.

### Raises

**ValueError** – If version is specified in the path

`deeplake.read(path: Union[str, Path], verify: bool = False, creds: Optional[Dict] = None, compression: Optional[str] = None, storage: Optional[StorageProvider] = None, timeout: Optional[float] = None) → Sample`

Utility that reads raw data from supported files into Deep Lake format.

- Recompresses data into format required by the tensor if permitted by the tensor htype.
- Simply copies the data in the file if file format matches sample\_compression of the tensor, thus maximizing upload speeds.

## Examples

```
>>> ds.create_tensor("images", htype="image", sample_compression="jpeg")
>>> ds.images.append(deeplake.read("path/to/cat.jpg"))
>>> ds.images.shape
(1, 399, 640, 3)
```

```
>>> ds.create_tensor("videos", htype="video", sample_compression="mp4")
>>> ds.videos.append(deeplake.read("path/to/video.mp4"))
>>> ds.videos.shape
(1, 136, 720, 1080, 3)
```

```
>>> ds.create_tensor("images", htype="image", sample_compression="jpeg")
>>> ds.images.append(deeplake.read("https://picsum.photos/200/300"))
>>> ds.images[0].shape
(300, 200, 3)
```

Supported file types:

```
Image: "bmp", "dib", "gif", "ico", "jpeg", "jpeg2000", "pcx", "png", "ppm", "sgi",
      ↪ "tga", "tiff", "webp", "wmf", "xbm"
Audio: "flac", "mp3", "wav"
Video: "mp4", "mkv", "avi"
Dicom: "dcm"
Nifti: "nii", "nii.gz"
```

## Parameters

- **path** (*str*) – Path to a supported file.
- **verify** (*bool*) – If True, contents of the file are verified.
- **creds** (*optional, Dict*) – Credentials for s3, gcp and http urls.
- **compression** (*optional, str*) – Format of the file. Only required if path does not have an extension.
- **storage** (*optional, StorageProvider*) – Storage provider to use to retrieve remote files. Useful if multiple files are being read from same storage to minimize overhead of creating a new provider.
- **timeout** (*optional, float*) – Timeout in seconds for reading the file. Applicable only for http(s) urls.

## Returns

Sample object. Call `sample.array` to get the `np.ndarray`.

## Return type

*Sample*

**Note:** No data is actually loaded until you try to get a property of the returned `Sample`. This is useful for passing



along to `Tensor.append` and `Tensor.extend`.

`deeplake.link(path: str, creds_key: Optional[str] = None) → LinkedSample`

Utility that stores a link to raw data. Used to add data to a Deep Lake Dataset without copying it. See [Link htype](#).

Supported file types:

```
Image: "bmp", "dib", "gif", "ico", "jpeg", "jpeg2000", "pcx", "png", "ppm", "sgi",
↪ "tga", "tiff", "webp", "wmf", "xbm"
Audio: "flac", "mp3", "wav"
Video: "mp4", "mkv", "avi"
Dicom: "dcm"
Nifti: "nii", "nii.gz"
```

#### Parameters

- **path** (`str`) – Path to a supported file.
- **creds\_key** (`optional, str`) – The credential key to use to read data for this sample. The actual credentials are fetched from the dataset.

#### Returns

LinkedSample object that stores path and creds.

#### Return type

[LinkedSample](#)

### Examples

```
>>> ds = deeplake.dataset("test/test_ds")
>>> ds.create_tensor("images", htype="link[image]", sample_compression="jpeg")
>>> ds.images.append(deeplake.link("https://picsum.photos/200/300"))
```

See more examples [here](#).

`deeplake.link_tiled(path_array: ndarray, creds_key: Optional[str] = None) → LinkedTiledSample`

Utility that stores links to multiple images that act as tiles and together form a big image. These images must all have the exact same dimensions. Used to add data to a Deep Lake Dataset without copying it. See [Link htype](#).

Supported file types:

```
Image: "bmp", "dib", "gif", "ico", "jpeg", "jpeg2000", "pcx", "png", "ppm", "sgi",
↪ "tga", "tiff", "webp", "wmf", "xbm"
```

#### Parameters

- **path\_array** (`np.ndarray`) – N dimensional array of paths to the data, with paths corresponding to respective tiles. The array must have dtype=object and have string values. Each string must point to an image file with the same dimensions.
- **creds\_key** (`optional, str`) – The credential key to use to read data for this sample. The actual credentials are fetched from the dataset.

#### Returns

LinkedTiledSample object that stores path\_array and creds.

#### Return type

[LinkedTiledSample](#)

## Examples

```
>>> ds = deeplake.dataset("test/test_ds")
>>> ds.create_tensor("images", htype="link[image]", sample_compression="jpeg")
>>> arr = np.empty((10, 10), dtype=object)
>>> for j, i in itertools.product(range(10), range(10)):
...     arr[j, i] = f"s3://my_bucket/my_image_{j}_{i}.jpeg"
...
>>> ds.images.append(deeplake.link_tiled(arr, creds_key="my_s3_key"))
>>> # If all images are 1000x1200x3, we now have a 10000x12000x3 image in our
↳dataset.
```

`deeplake.tiled(sample_shape: Tuple[int, ...], tile_shape: Optional[Tuple[int, ...]] = None, dtype: Union[str, dtype] = dtype('uint8'))`

Allocates an empty sample of shape `sample_shape`, broken into tiles of shape `tile_shape` (except for edge tiles).

## Example

```
>>> with ds:
...     ds.create_tensor("image", htype="image", sample_compression="png")
...     ds.image.append(deeplake.tiled(sample_shape=(1003, 1103, 3), tile_shape=(10,
↳10, 3)))
...     ds.image[0][-217:, :212, 1:] = np.random.randint(0, 256, (217, 212, 2),
↳dtype=np.uint8)
```

### Parameters

- **sample\_shape** (*Tuple[int, ...]*) – Full shape of the sample.
- **tile\_shape** (*Optional, Tuple[int, ...]*) – The sample will be stored as tiles where each tile will have this shape (except edge tiles). If not specified, it will be computed such that each tile is close to half of the tensor's *max\_chunk\_size* (after compression).
- **dtype** (*Union[str, np.dtype]*) – Dtype for the sample array. Default `uint8`.

### Returns

A `PartialSample` instance which can be appended to a `Tensor`.

### Return type

*PartialSample*

`deeplake.compute(fn, name: Optional[str] = None) → Callable[[...], ComputeFunction]`

Compute is a decorator for functions.

The functions should have at least 2 argument, the first two will correspond to `sample_in` and `samples_out`.

There can be as many other arguments as required.

The output should be appended/extended to the second argument in a deeplake like syntax.

Any value returned by the `fn` will be ignored.

Example:

```
@deeplake.compute
def my_fn(sample_in: Any, samples_out, my_arg0, my_arg1=0):
    samples_out.my_tensor.append(my_arg0 * my_arg1)
```

(continues on next page)

(continued from previous page)

```
# This transform can be used using the eval method in one of these 2 ways:-

# Directly evaluating the method
# here arg0 and arg1 correspond to the 3rd and 4th argument in my_fn
my_fn(arg0, arg1).eval(data_in, ds_out, scheduler="threaded", num_workers=5)

# As a part of a Transform pipeline containing other functions
pipeline = deeplake.compose([my_fn(a, b), another_function(x=2)])
pipeline.eval(data_in, ds_out, scheduler="processed", num_workers=2)
```

The eval method evaluates the pipeline/transform function.

It has the following arguments:

- **data\_in**: Input passed to the transform to generate output dataset.
  - It should support `__getitem__` and `__len__`. This can be a Deep Lake dataset.
- **ds\_out** (Dataset, optional): The dataset object to which the transform will get written.
  - If this is not provided, data\_in will be overwritten if it is a Deep Lake dataset, otherwise error will be raised.
  - It should have all keys being generated in output already present as tensors.
  - It's initial state should be either:
    - \* Empty i.e. all tensors have no samples. In this case all samples are added to the dataset.
    - \* All tensors are populated and have same length. In this case new samples are appended to the dataset.
- **num\_workers** (int): The number of workers to use for performing the transform.
  - Defaults to 0. When set to 0, it will always use serial processing, irrespective of the scheduler.
- **scheduler** (str): The scheduler to be used to compute the transformation.
  - Supported values include: 'serial', 'threaded', 'processed' and 'ray'. Defaults to 'threaded'.
- **progressbar** (bool): Displays a progress bar if True (default).
- **skip\_ok** (bool): If True, skips the check for output tensors generated.
  - This allows the user to skip certain tensors in the function definition.
  - This is especially useful for inplace transformations in which certain tensors are not modified. Defaults to False.
- **check\_lengths** (bool): If True, checks whether ds\_out has tensors of same lengths initially.
- **pad\_data\_in** (bool): If True, pads tensors of data\_in to match the length of the largest tensor in data\_in. Defaults to False.
- **ignore\_errors** (bool): If True, input samples that causes transform to fail will be skipped and the errors will be ignored **if possible**.

---

**Note:** pad\_data\_in is only applicable if data\_in is a Deep Lake dataset.

---

It raises the following errors:

- **InvalidInputDataError**: If data\_in passed to transform is invalid. It should support `__getitem__` and `__len__` operations. Using scheduler other than "threaded" with deeplake dataset having base storage as memory as data\_in will also raise this.
- **InvalidOutputDatasetError**: If all the tensors of ds\_out passed to transform don't have the same length. Using scheduler other than "threaded" with deeplake dataset having base storage as memory as ds\_out will also raise this.
- **TensorMismatchError**: If one or more of the outputs generated during transform contain different tensors than the ones present in ds\_out provided to transform.
- **UnsupportedSchedulerError**: If the scheduler passed is not recognized. Supported values include: 'serial', 'threaded', 'processed' and 'ray'.
- **TransformError**: All other exceptions raised if there are problems while running the pipeline.

`deeplake.compose(functions: List[ComputeFunction])`

Takes a list of functions decorated using `deeplake.compute()` and creates a pipeline that can be evaluated using `.eval`

Example:

```
pipeline = deeplake.compose([my_fn(a=3), another_function(b=2)])
pipeline.eval(data_in, ds_out, scheduler="processed", num_workers=2)
```

The `eval` method evaluates the pipeline/transform function.

It has the following arguments:

- `data_in`: Input passed to the transform to generate output dataset.
  - It should support `__getitem__` and `__len__`. This can be a Deep Lake dataset.
- `ds_out` (Dataset, optional): The dataset object to which the transform will get written.
  - If this is not provided, `data_in` will be overwritten if it is a Deep Lake dataset, otherwise error will be raised.
  - It should have all keys being generated in output already present as tensors.
  - It's initial state should be either:
    - \* Empty i.e. all tensors have no samples. In this case all samples are added to the dataset.
    - \* All tensors are populated and have same length. In this case new samples are appended to the dataset.
- `num_workers` (int): The number of workers to use for performing the transform.
  - Defaults to 0. When set to 0, it will always use serial processing, irrespective of the scheduler.
- `scheduler` (str): The scheduler to be used to compute the transformation.
  - Supported values include: 'serial', 'threaded', 'processed' and 'ray'. Defaults to 'threaded'.
- `progressbar` (bool): Displays a progress bar if True (default).
- `skip_ok` (bool): If True, skips the check for output tensors generated.
  - This allows the user to skip certain tensors in the function definition.
  - This is especially useful for inplace transformations in which certain tensors are not modified. Defaults to False.
- `ignore_errors` (bool): If True, input samples that causes transform to fail will be skipped and the errors will be ignored **if possible**.

It raises the following errors:

- `InvalidInputDataError`: If `data_in` passed to transform is invalid. It should support `__getitem__` and `__len__` operations. Using scheduler other than "threaded" with deeplake dataset having base storage as memory as `data_in` will also raise this.
- `InvalidOutputDatasetError`: If all the tensors of `ds_out` passed to transform don't have the same length. Using scheduler other than "threaded" with deeplake dataset having base storage as memory as `ds_out` will also raise this.
- `TensorMismatchError`: If one or more of the outputs generated during transform contain different tensors than the ones present in 'ds\_out' provided to transform.
- `UnsupportedSchedulerError`: If the scheduler passed is not recognized. Supported values include: 'serial', 'threaded', 'processed' and 'ray'.
- `TransformError`: All other exceptions raised if there are problems while running the pipeline.

## DEEPLAKE.VECTORSTORE

**class** deeplake.core.vectorstore.deeplake\_vectorstore.VectorStore

Base class for VectorStore

```
__init__(path: ~typing.Optional[~typing.Union[str, ~pathlib.Path]] = None, dataset:
~typing.Optional[~deeplake.core.dataset.dataset.Dataset] = None, tensor_params:
~typing.List[~typing.Dict[str, object]] = [{ 'name': 'text', 'htype': 'text', 'create_id_tensor': False,
'create_sample_info_tensor': False, 'create_shape_tensor': False}, { 'name': 'metadata', 'htype':
'json', 'create_id_tensor': False, 'create_sample_info_tensor': False, 'create_shape_tensor': False},
{ 'name': 'embedding', 'htype': 'embedding', 'dtype': <class 'numpy.float32'>, 'create_id_tensor':
False, 'create_sample_info_tensor': False, 'create_shape_tensor': True, 'max_chunk_size':
64000000}, { 'name': 'id', 'htype': 'text', 'create_id_tensor': False, 'create_sample_info_tensor':
False, 'create_shape_tensor': False}], embedding_function: ~typing.Optional[~typing.Any] =
None, read_only: ~typing.Optional[bool] = None, ingestion_batch_size: int = 1000,
index_params: ~typing.Optional[~typing.Dict[str, ~typing.Union[int, str]]] = None, exec_option:
str = 'auto', token: ~typing.Optional[str] = None, overwrite: bool = False, verbose: bool = True,
runtime: ~typing.Optional[~typing.Dict] = None, creds:
~typing.Optional[~typing.Union[~typing.Dict, str]] = None, org_id: ~typing.Optional[str] =
None, logger: ~logging.Logger = <Logger deeplake.core.vectorstore.deeplake_vectorstore
(INFO)>, branch: str = 'main', indra: bool = False, **kwargs: ~typing.Any) → None
```

Creates an empty VectorStore or loads an existing one if it exists at the specified path.

### Examples

```
>>> # Create a vector store with default tensors
>>> data = VectorStore(
...     path = "./my_vector_store",
... )
```

```
>>> # Create a vector store in the Deep Lake Managed Tensor Database
>>> data = VectorStore(
...     path = "hub://org_id/dataset_name",
...     runtime = {"tensor_db": True},
... )
```

```
>>> # Create a vector store with custom tensors
>>> data = VectorStore(
...     path = "./my_vector_store",
...     tensor_params = [{"name": "text", "htype": "text"},
...                       {"name": "embedding_1", "htype": "embedding"}],
... )
```

(continues on next page)

(continued from previous page)

```

...         {"name": "embedding_2", "htype": "embedding"},
...         {"name": "source", "htype": "text"},
...         {"name": "metadata", "htype": "json"}
...     ]
... )

```

### Parameters

- **path** (*str*, *pathlib.Path*) –
  - The full path for storing to the Deep Lake Vector Store. It can be:
    - a Deep Lake cloud path of the form `hub://org_id/dataset_name`. Requires registration with Deep Lake.
    - an s3 path of the form `s3://bucketname/path/to/dataset`. Credentials are required in either the environment or passed to the `creds` argument.
    - a local file system path of the form `./path/to/dataset` or `~/path/to/dataset` or `path/to/dataset`.
    - a memory path of the form `mem://path/to/dataset` which doesn't save the dataset but keeps it in memory instead. Should be used only for testing as it does not persist.
- **tensor\_params** (*List[Dict[str, dict]]*, *optional*) – List of dictionaries that contains information about tensors that user wants to create. See `create_tensor` in Deep Lake API docs for more information. Defaults to `DEFAULT_VECTORSTORE_TENSORS`.
- **embedding\_function** (*Optional[Any]*, *optional*) – Function or class that converts the embeddable data into embeddings. Input to `embedding_function` is a list of data and output is a list of embeddings. Defaults to `None`.
- **read\_only** (*bool*, *optional*) – Opens dataset in read-only mode if `True`. Defaults to `False`.
- **ingestion\_batch\_size** (*int*) – Batch size to use for parallel ingestion.
- **index\_params** (*Dict[str, Union[int, str]]*) – Dictionary containing information about vector index that will be created. Defaults to `None`, which will utilize `DEFAULT_VECTORSTORE_INDEX_PARAMS` from `deeplake.constants`. The specified key-values override the default ones:
  - `'threshold'`: The threshold for the dataset size above which an index will be created for the embedding tensor. When the threshold value is set to `-1`, index creation is turned off. Defaults to `-1`, which turns off the index.
  - `'distance_metric'`: This key specifies the method of calculating the distance between vectors when creating the vector database (VDB) index. It can either be a string that corresponds to a member of the `DistanceType` enumeration, or the string value itself.
    - \* If no value is provided, it defaults to `"L2"`.
    - \* `"L2"` corresponds to `DistanceType.L2_NORM`.
    - \* `"COS"` corresponds to `DistanceType.COSINE_SIMILARITY`.
  - `'additional_params'`: Additional parameters for fine-tuning the index.

- **exec\_option** (*str*) – Default method for search execution. It could be either "auto", "python", "compute\_engine" or "tensor\_db". Defaults to "auto". If None, it's set to "auto". - **auto**- Selects the best execution method based on the storage location of the Vector Store. It is the default option. - **python** - Pure-python implementation that runs on the client and can be used for data stored anywhere. WARNING: using this option with big datasets is discouraged because it can lead to memory issues. - **compute\_engine** - Performant C++ implementation of the Deep Lake Compute Engine that runs on the client and can be used for any data stored in or connected to Deep Lake. It cannot be used with in-memory or local datasets. - **tensor\_db** - Performant and fully-hosted Managed Tensor Database that is responsible for storage and query execution. Only available for data stored in the Deep Lake Managed Database. Store datasets in this database by specifying `runtime = {"tensor_db": True}` during dataset creation.
- **token** (*str*, *optional*) – Activeloop token, used for fetching user credentials. This is Optional, tokens are normally autogenerated. Defaults to None.
- **overwrite** (*bool*) – If set to True this overwrites the Vector Store if it already exists. Defaults to False.
- **verbose** (*bool*) – Whether to print summary of the dataset created. Defaults to True.
- **creds** (*dict*, *str*, *optional*) – The string ENV or a dictionary containing credentials used to access the dataset at the path. - If 'aws\_access\_key\_id', 'aws\_secret\_access\_key', 'aws\_session\_token' are present, these take precedence over credentials present in the environment or in credentials file. Currently only works with s3 paths. - It supports 'aws\_access\_key\_id', 'aws\_secret\_access\_key', 'aws\_session\_token', 'endpoint\_url', 'aws\_region', 'profile\_name' as keys. - If 'ENV' is passed, credentials are fetched from the environment variables. This is also the case when creds is not passed for cloud datasets. For datasets connected to hub cloud, specifying 'ENV' will override the credentials fetched from Activeloop and use local ones.
- **runtime** (*Dict*, *optional*) – Parameters for creating the Vector Store in Deep Lake's Managed Tensor Database. Not applicable when loading an existing Vector Store. To create a Vector Store in the Managed Tensor Database, set `runtime = {"tensor_db": True}`.
- **branch** (*str*) – Branch name to use for the Vector Store. Defaults to "main".
- **indra** (*bool*) – Flag indicating whether indra api should be used to create the underlying dataset. Defaults to false
- **\*\*kwargs** (*dict*) – Additional keyword arguments.

**Danger:** Setting `overwrite` to True will delete all of your data if the Vector Store exists! Be very careful when setting this parameter.

**add**(*embedding\_function: Optional[Union[Callable, List[Callable]]] = None, embedding\_data: Optional[Union[List, List[List]]] = None, embedding\_tensor: Optional[Union[str, List[str]]] = None, return\_ids: bool = False, rate\_limiter: Dict = {'batch\_byte\_size': 10000, 'bytes\_per\_minute': 1800000.0, 'enabled': False}, \*\**tensors*) → Optional[List[str]]*

Adding elements to deeplake vector store.

Tensor names are specified as parameters, and data for each tensor is specified as parameter values. All data must of equal length.

## Examples

```

>>> # Dummy data
>>> texts = ["Hello", "World"]
>>> embeddings = [[1, 2, 3], [4, 5, 6]]
>>> metadatas = [{"timestamp": "01:20"}, {"timestamp": "01:22"}]
>>> embedding_fn = lambda x: [[1, 2, 3]] * len(x)
>>> embedding_fn_2 = lambda x: [[4, 5]] * len(x)
>>> # Directly upload embeddings
>>> deeplake_vector_store.add(
...     text = texts,
...     embedding = embeddings,
...     metadata = metadatas,
... )
>>> # Upload embedding via embedding function
>>> deeplake_vector_store.add(
...     text = texts,
...     metadata = metadatas,
...     embedding_function = embedding_fn,
...     embedding_data = texts,
... )
>>> # Upload embedding via embedding function to a user-defined embedding_
→tensor
>>> deeplake_vector_store.add(
...     text = texts,
...     metadata = metadatas,
...     embedding_function = embedding_fn,
...     embedding_data = texts,
...     embedding_tensor = "embedding_1",
... )
>>> # Multiple embedding functions (user defined embedding tensors must be_
→specified)
>>> deeplake_vector_store.add(
...     embedding_tensor = ["embedding_1", "embedding_2"]
...     embedding_function = [embedding_fn, embedding_fn_2],
...     embedding_data = [texts, texts],
... )
>>> # Alternative syntax for multiple embedding functions
>>> deeplake_vector_store.add(
...     text = texts,
...     metadata = metadatas,
...     embedding_tensor_1 = (embedding_fn, texts),
...     embedding_tensor_2 = (embedding_fn_2, texts),
... )
>>> # Add data to fully custom tensors
>>> deeplake_vector_store.add(
...     tensor_A = [1, 2],
...     tensor_B = ["a", "b"],
...     tensor_C = ["some", "data"],
...     embedding_function = embedding_fn,
...     embedding_data = texts,
...     embedding_tensor = "embedding_1",
... )

```



**Parameters**

- **embedding\_function** (*Optional[Callable]*) – embedding function used to convert `embedding_data` into embeddings. Input to `embedding_function` is a list of data and output is a list of embeddings. Overrides the `embedding_function` specified when initializing the Vector Store.
- **embedding\_data** (*Optional[List]*) – Data to be converted into embeddings using the provided `embedding_function`. Defaults to `None`.
- **embedding\_tensor** (*Optional[str]*) – Tensor where results from the embedding function will be stored. If `None`, the embedding tensor is automatically inferred (when possible). Defaults to `None`.
- **return\_ids** (*bool*) – Whether to return added ids as an output of the method. Defaults to `False`.
- **rate\_limiter** (*Dict*) – Rate limiter configuration. Defaults to `{"enabled": False, "bytes_per_minute": MAX_BYTES_PER_MINUTE, "batch_byte_size": TARGET_BYTE_SIZE}`.
- **\*\*tensors** – Keyword arguments where the key is the tensor name, and the value is a list of samples that should be uploaded to that tensor.

**Returns**

List of ids if `return_ids` is set to `True`. Otherwise, `None`.

**Return type**

`Optional[List[str]]`

**checkout** (*branch: str = 'main', create=False*) → `None`

Checkout the Vector Store to a specific branch.

**Parameters**

- **branch** (*str*) – Branch name to checkout. Defaults to “main”.
- **create** (*bool*) – Whether to create the branch if it doesn’t exist. Defaults to `False`.

**commit** (*allow\_empty: bool = True*) → `None`

Commits the Vector Store.

**Parameters**

**allow\_empty** (*bool*) – Whether to allow empty commits. Defaults to `True`.

**property dataset**

Returns the dataset

**delete** (*row\_ids: Optional[List[int]] = None, ids: Optional[List[str]] = None, filter: Optional[Union[Dict, Callable]] = None, query: Optional[str] = None, exec\_option: Optional[str] = None, delete\_all: Optional[bool] = None*) → `bool`

Delete the data in the Vector Store. Does not delete the tensor definitions. To delete the vector store completely, first run `VectorStore.delete_by_path()`.

## Examples

```
>>> # Delete using ids:
>>> data = vector_store.delete(ids)
>>> # Delete data using filter
>>> data = vector_store.delete(
...     filter = {"json_tensor_name": {"key": value}, "json_tensor_name_2":
...     ↪ {"key_2": value_2}},
... )
>>> # Delete data using TQL
>>> data = vector_store.delete(
...     query = "select * where ..... <add TQL syntax>",
...     exec_option = "compute_engine",
... )
```

## Parameters

- **ids** (*Optional[List[str]]*) – List of unique ids. Defaults to None.
- **row\_ids** (*Optional[List[int]]*) – List of absolute row indices from the dataset. Defaults to None.
- **filter** (*Union[Dict, Callable]*, *optional*) – Filter for finding samples for deletion. - **Dict** - Key-value search on tensors of htype json, evaluated on an AND basis (a sample must satisfy all key-value filters to be True) Dict = {"tensor\_name\_1": {"key": value}, "tensor\_name\_2": {"key": value}} - **Function** - Any function that is compatible with *deeplake.filter*.
- **query** (*Optional[str]*) – TQL Query string for direct evaluation for finding samples for deletion, without application of additional filters.
- **exec\_option** (*Optional[str]*) – Method for search execution. It could be either "python", "compute\_engine" or "tensor\_db". Defaults to None, which inherits the option from the Vector Store initialization. - **python** - Pure-python implementation that runs on the client and can be used for data stored anywhere. WARNING: using this option with big datasets is discouraged because it can lead to memory issues. - **compute\_engine** - Performant C++ implementation of the Deep Lake Compute Engine that runs on the client and can be used for any data stored in or connected to Deep Lake. It cannot be used with in-memory or local datasets. - **tensor\_db** - Performant and fully-hosted Managed Tensor Database that is responsible for storage and query execution. Only available for data stored in the Deep Lake Managed Database. Store datasets in this database by specifying runtime = {"tensor\_db": True} during dataset creation.
- **delete\_all** (*Optional[bool]*) – Whether to delete all the samples and version history of the dataset. Defaults to None.

## Returns

Returns True if deletion was successful, otherwise it raises a ValueError.

## Return type

bool

## Raises

**ValueError** – If neither `ids`, `filter`, `query`, nor `delete_all` are specified, or if an invalid `exec_option` is provided.

```
static delete_by_path(path: Union[str, Path], token: Optional[str] = None, force: bool = False, creds:
    Optional[Union[Dict, str]] = None) → None
```

Deleted the Vector Store at the specified path.

#### Parameters

- **path** (*str*, *pathlib.Path*) – The full path to the Deep Lake Vector Store.
- **token** (*str*, *optional*) – Activeloop token, used for fetching user credentials. This is optional, as tokens are normally autogenerated. Defaults to *None*.
- **creds** (*dict*, *str*, *optional*) – The string ENV or a dictionary containing credentials used to access the dataset at the path. - If 'aws\_access\_key\_id', 'aws\_secret\_access\_key', 'aws\_session\_token' are present, these take precedence over credentials present in the environment or in credentials file. Currently only works with s3 paths. - It supports 'aws\_access\_key\_id', 'aws\_secret\_access\_key', 'aws\_session\_token', 'endpoint\_url', 'aws\_region', 'profile\_name' as keys. - If 'ENV' is passed, credentials are fetched from the environment variables. This is also the case when creds is not passed for cloud datasets. For datasets connected to hub cloud, specifying 'ENV' will override the credentials fetched from Activeloop and use local ones.
- **force** (*bool*) – delete the path in a forced manner without rising an exception. Defaults to *True*.

**Danger:** This method permanently deletes all of your data if the Vector Store exists! Be very careful when using this method.

**search**(*embedding\_data: Optional[Union[str, List[str]]] = None, embedding\_function: Optional[Callable] = None, embedding: Optional[Union[List[float], ndarray]] = None, k: int = 4, distance\_metric: Optional[str] = None, query: Optional[str] = None, filter: Optional[Union[Dict, Callable]] = None, exec\_option: Optional[str] = None, embedding\_tensor: str = 'embedding', return\_tensors: Optional[List[str]] = None, return\_view: bool = False, deep\_memory: bool = False, return\_tql: bool = False) → Union[Dict, Dataset]*

VectorStore search method that combines embedding search, metadata search, and custom TQL search.

#### Examples

```
>>> # Search using an embedding
>>> data = vector_store.search(
...     embedding = [1, 2, 3],
...     exec_option = "python",
... )
>>> # Search using an embedding function and data for embedding
>>> data = vector_store.search(
...     embedding_data = "What does this chatbot do?",
...     embedding_function = query_embedding_fn,
...     exec_option = "compute_engine",
... )
>>> # Add a filter to your search
>>> data = vector_store.search(
...     embedding = np.ones(3),
...     exec_option = "python",
...     filter = {"json_tensor_name": {"key: value"}, "json_tensor_name_2":
→ {"key_2: value_2"}, ...}, # Only valid for exec_option = "python"
```

(continues on next page)

(continued from previous page)

```

... )
>>> # Search using TQL
>>> data = vector_store.search(
...     query = "select * where ..... <add TQL syntax>",
...     exec_option = "tensor_db", # Only valid for exec_option = "compute_
→engine" or "tensor_db"
... )

```

### Parameters

- **embedding** (*Union[np.ndarray, List[float]], optional*) – Embedding representation for performing the search. Defaults to None. The *embedding\_data* and *embedding* cannot both be specified.
- **embedding\_data** (*List[str]*) – Data against which the search will be performed by embedding it using the *embedding\_function*. Defaults to None. The *embedding\_data* and *embedding* cannot both be specified.
- **embedding\_function** (*Optional[Callable], optional*) – function for converting *embedding\_data* into embedding. Only valid if *embedding\_data* is specified. Input to *embedding\_function* is a list of data and output is a list of embeddings.
- **k** (*int*) – Number of elements to return after running query. Defaults to 4.
- **distance\_metric** (*str*) – Distance metric to use for sorting the data. Available options are: "L1", "L2", "COS", "MAX". Defaults to None, which uses same distance metric specified in *index\_params*. If there is no index, it performs linear search using `DEFAULT_VECTORSTORE_DISTANCE_METRIC`.
- **query** (*Optional[str]*) – TQL Query string for direct evaluation, without application of additional filters or vector search.
- **filter** (*Union[Dict, Callable], optional*) – Additional filter evaluated prior to the embedding search.
  - Dict - Key-value search on tensors of htype json, evaluated on an AND basis (a sample must satisfy all key-value filters to be True) Dict = {"tensor\_name\_1": {"key": value}, "tensor\_name\_2": {"key": value}}
  - Function - Any function that is compatible with [Dataset.filter](#).
- **exec\_option** (*Optional[str]*) – Method for search execution. It could be either "python", "compute\_engine" or "tensor\_db". Defaults to None, which inherits the option from the Vector Store initialization.
  - python - Pure-python implementation that runs on the client and can be used for data stored anywhere. **WARNING:** using this option with big datasets is discouraged because it can lead to memory issues.
  - compute\_engine - Performant C++ implementation of the Deep Lake Compute Engine that runs on the client and can be used for any data stored in or connected to Deep Lake. It cannot be used with in-memory or local datasets.
  - tensor\_db - Performant and fully-hosted Managed Tensor Database that is responsible for storage and query execution. Only available for data stored in the Deep Lake Managed Database. Store datasets in this database by specifying `runtime = {"tensor_db": True}` during dataset creation.

- **embedding\_tensor** (*str*) – Name of tensor with embeddings. Defaults to “embedding”.
- **return\_tensors** (*Optional[List[str]]*) – List of tensors to return data for. Defaults to None, which returns data for all tensors except the embedding tensor (in order to minimize payload). To return data for all tensors, specify `return_tensors = “*”`.
- **return\_view** (*bool*) – Return a Deep Lake dataset view that satisfied the search parameters, instead of a dictionary with data. Defaults to False. If True `return_tensors` is set to “\*” because data is lazy-loaded and there is no cost to including all tensors in the view.
- **deep\_memory** (*bool*) – Whether to use the Deep Memory model for improving search results. Defaults to False if `deep_memory` is not specified in the Vector Store initialization. If True, the distance metric is set to “deepmemory\_distance”, which represents the metric with which the model was trained. The search is performed using the Deep Memory model. If False, the distance metric is set to “COS” or whatever distance metric user specifies.
- **return\_tql** (*bool*) – Whether to return the TQL query string used for the search. Defaults to False.

**Raises**

- **ValueError** – When invalid parameters are specified.
- **ValueError** – when `deep_memory` is True. Deep Memory is only available for datasets stored in the Deep Lake Managed Database for paid accounts.
- **DeepMemoryAccessError** – if user does not have access to `deep_memory`.

**Returns**

Dictionary where keys are tensor names and values are the results of the search

**Return type**

Dict

**summary()**

Prints a summary of the dataset

**tensors()**

Returns the list of tensors present in the dataset

**update\_embedding**(*row\_ids: Optional[List[str]] = None, ids: Optional[List[str]] = None, filter: Optional[Union[Dict, Callable]] = None, query: Optional[str] = None, exec\_option: Optional[str] = None, embedding\_function: Optional[Union[Callable, List[Callable]]] = None, embedding\_source\_tensor: Union[str, List[str]] = 'text', embedding\_tensor: Optional[Union[str, List[str]]] = None*)

Recompute existing embeddings of the VectorStore, that match either query, filter, ids or row\_ids.

## Examples

```
>>> # Update using ids:
>>> data = vector_store.update(
...     ids,
...     embedding_source_tensor = "text",
...     embedding_tensor = "embedding",
...     embedding_function = embedding_function,
... )
>>> # Update data using filter and several embedding_tensors, several_
↪ embedding_source_tensors
>>> # and several embedding_functions:
>>> data = vector_store.update(
...     embedding_source_tensor = ["text", "metadata"],
...     embedding_function = ["text_embedding_function", "metadata_embedding_
↪ function"],
...     filter = {"json_tensor_name": {"key": "value"}, "json_tensor_name_2": {
↪ "key_2": "value_2"}},
...     embedding_tensor = ["text_embedding", "metadata_embedding"]
... )
>>> # Update data using TQL, if new embedding function is not specified the_
↪ embedding_function used
>>> # during initialization will be used
>>> data = vector_store.update(
...     embedding_source_tensor = "text",
...     query = "select * where ..... <add TQL syntax>",
...     exec_option = "compute_engine",
...     embedding_tensor = "embedding_tensor",
... )
```

## Parameters

- **row\_ids** (*Optional[List[str]], optional*) – Row ids of the elements for replacement. Defaults to None.
- **ids** (*Optional[List[str]], optional*) – hash ids of the elements for replacement. Defaults to None.
- **filter** (*Optional[Union[Dict, Callable]], optional*) – Filter for finding samples for replacement. - Dict - Key-value search on tensors of htype json, evaluated on an AND basis (a sample must satisfy all key-value filters to be True) Dict = {"tensor\_name\_1": {"key": "value"}, "tensor\_name\_2": {"key": "value"}} - Function - Any function that is compatible with *deeplake.filter*
- **query** (*Optional[str], optional*) – TQL Query string for direct evaluation for finding samples for deletion, without application of additional filters. Defaults to None.
- **exec\_option** (*Optional[str]*) – Method for search execution. It could be either "python", "compute\_engine" or "tensor\_db". Defaults to None, which inherits the option from the Vector Store initialization. - python - Pure-python implementation that runs on the client and can be used for data stored anywhere. WARNING: using this option with big datasets is discouraged because it can lead to memory issues. - compute\_engine - Performant C++ implementation of the Deep Lake Compute Engine that runs on the client and can be used for any data stored in or connected to Deep Lake. It cannot be used with in-memory or local

datasets. - **tensor\_db** - Performant and fully-hosted Managed Tensor Database that is responsible for storage and query execution. Only available for data stored in the Deep Lake Managed Database. Store datasets in this database by specifying `runtime = {"tensor_db": True}` during dataset creation.

- **embedding\_function** (*Optional[Union[Callable, List[Callable]]], optional*) – function for converting *embedding\_source\_tensor* into embedding. Only valid if *embedding\_source\_tensor* is specified. Defaults to None.
- **embedding\_source\_tensor** (*Union[str, List[str]], optional*) – Name of tensor with data that needs to be converted to embeddings. Defaults to *text*.
- **embedding\_tensor** (*Optional[Union[str, List[str]]], optional*) – Name of the tensor with embeddings. Defaults to None.





## DEEPLAKE.CORE

### 18.1 deeplake.core.sample

`class deeplake.core.sample.Sample`

```
__init__(path: Optional[str] = None, array: Optional[ndarray] = None, buffer: Optional[Union[bytes,
memoryview]] = None, compression: Optional[str] = None, verify: bool = False, shape:
Optional[Tuple[int]] = None, dtype: Optional[str] = None, creds: Optional[Dict] = None,
storage: Optional[StorageProvider] = None, timeout: Optional[float] = None)
```

Represents a single sample for a tensor. Provides all important meta information in one place.

---

**Note:** If `self.is_lazy` is `True`, this [Sample](#) doesn't actually have any data loaded. To read this data, simply try to read it into a numpy array (`sample.array`)

---

#### Parameters

- **path** (*str*) – Path to a sample stored on the local file system that represents a single sample. If `path` is provided, `array` should not be. Implicitly makes `self.is_lazy == True`.
- **array** (*np.ndarray*) – Array that represents a single sample. If `array` is provided, `path` should not be. Implicitly makes `self.is_lazy == False`.
- **buffer** – (bytes): Byte buffer that represents a single sample. If compressed, `compression` argument should be provided.
- **compression** (*str*) – Specify in case of byte buffer.
- **verify** (*bool*) – If a path is provided, verifies the sample if `True`.
- **shape** (*Tuple[int]*) – Shape of the sample.
- **dtype** (*optional, str*) – Data type of the sample.
- **creds** (*optional, Dict*) – Credentials for s3, gcp and http urls.
- **storage** (*optional, StorageProvider*) – Storage provider.
- **timeout** (*optional, float*) – Timeout in seconds for reading the file. Applicable only for http(s) urls.

**property array:** `ndarray`

Return numpy array corresponding to the sample. Decompresses the sample if necessary.

### Example

```
>>> sample = deeplake.read("./images/dog.jpg")
>>> arr = sample.array
>>> arr.shape
(323, 480, 3)
```

**compressed\_bytes**(*compression: Optional[str]*) → bytes

Returns this sample as compressed bytes.

---

**Note:** If this sample is pointing to a path and the requested **compression** is the same as it's stored in, the data is returned without re-compressing.

---

#### Parameters

**compression** (*Optional[str]*) – self.array will be compressed into this format. If compression is None, return [uncompressed\\_bytes\(\)](#).

#### Returns

Bytes for the compressed sample. Contains all metadata required to decompress within these bytes.

#### Return type

bytes

#### Raises

**ValueError** – On recompression of unsupported formats.

**property pil:** Image

Return PIL image corresponding to the sample. Decompresses the sample if necessary.

### Example

```
>>> sample = deeplake.read("./images/dog.jpg")
>>> pil = sample.pil
>>> pil.size
(480, 323)
```

**uncompressed\_bytes**() → Optional[bytes]

Returns uncompressed bytes.

## 18.2 deeplake.core.linked\_sample

**class** deeplake.core.linked\_sample.**LinkedSample**(*path: str, creds\_key: Optional[str] = None*)

Represents a sample that is initialized using external links. See [deeplake.link\(\)](#).

## 18.3 deeplake.core.partial\_sample

```
class deeplake.core.partial_sample.PartialSample(sample_shape: Tuple[int, ...], tile_shape:
    Optional[Tuple[int, ...]] = None, dtype:
    Optional[Union[dtype, str]] = dtype('uint8'))
```

Represents a sample that is initialized by just shape and the data is updated later.

## 18.4 deeplake.core.linked\_tiled\_sample

```
class deeplake.core.linked_tiled_sample.LinkerTiledSample(path_array: ndarray, creds_key:
    Optional[str] = None)
```

Represents a sample that is initialized using external links. See [deeplake.link\\_tiled\(\)](#).

## 18.5 deeplake.core.storage

### 18.5.1 Base Storage Provider

```
class deeplake.core.storage.StorageProvider
```

```
    abstract __delitem__(path: str)
```

Delete the object present at the path.

**Parameters**

**path** (*str*) – the path to the object relative to the root of the provider.

**Raises**

**KeyError** – If an object is not found at the path.

```
    abstract __getitem__(path: str)
```

Gets the object present at the path within the given byte range.

**Parameters**

**path** (*str*) – The path relative to the root of the provider.

**Returns**

The bytes of the object present at the path.

**Return type**

bytes

**Raises**

**KeyError** – If an object is not found at the path.

```
    abstract __iter__()
```

Generator function that iterates over the keys of the provider.

**Yields**

*str* – the path of the object that it is iterating over, relative to the root of the provider.

```
    abstract __len__()
```

Returns the number of files present inside the root of the provider.

**Returns**

the number of files present inside the root.

**Return type**

int

**abstract** `__setitem__(path: str, value: bytes)`

Sets the object present at the path with the value

**Parameters**

- **path** (*str*) – the path relative to the root of the provider.
- **value** (*bytes*) – the value to be assigned at the path.

**\_\_weakref\_\_**

list of weak references to the object (if defined)

**abstract** `_all_keys() → Set[str]`

Generator function that iterates over the keys of the provider.

**Returns**

set of all keys present at the root of the provider.

**Return type**

set

**\_is\_hub\_path = False**

An abstract base class for implementing a storage provider.

To add a new provider using Provider, create a subclass and implement all 5 abstract methods below.

**check\_readonly()**

Raises an exception if the provider is in read-only mode.

**abstract** `clear(prefix="")`

Delete the contents of the provider.

**copy()**

Returns a copy of the provider.

**Returns**

A copy of the provider.

**Return type**

*StorageProvider*

**disable\_readonly()**

Disables read-only mode for the provider.

**enable\_readonly()**

Enables read-only mode for the provider.

**flush()**

Only needs to be implemented for caches. Flushes the data to the next storage provider. Should be a no op for Base Storage Providers like local, s3, azure, gcs, etc.

**get\_bytes**(*path: str, start\_byte: Optional[int] = None, end\_byte: Optional[int] = None*)

Gets the object present at the path within the given byte range.

**Parameters**

- **path** (*str*) – The path relative to the root of the provider.
- **start\_byte** (*int, optional*) – If only specific bytes starting from start\_byte are required.
- **end\_byte** (*int, optional*) – If only specific bytes up to end\_byte are required.

**Returns**

The bytes of the object present at the path within the given byte range.

**Return type**

bytes

**Raises**

- **`InvalidBytesRequestedError`** – If `start_byte > end_byte` or `start_byte < 0` or `end_byte < 0`.
- **`KeyError`** – If an object is not found at the path.

**`maybe_flush()`**

Flush cache if autoflush has been enabled. Called at the end of methods which write data, to ensure consistency as a default.

**`set_bytes(path: str, value: bytes, start_byte: Optional[int] = None, overwrite: Optional[bool] = False)`**

Sets the object present at the path with the value

**Parameters**

- **`path`** (`str`) – the path relative to the root of the provider.
- **`value`** (`bytes`) – the value to be assigned at the path.
- **`start_byte`** (`int`, *optional*) – If only specific bytes starting from `start_byte` are to be assigned.
- **`overwrite`** (`boolean`, *optional*) – If the value is `True`, if there is an object present at the path it is completely overwritten, without fetching it's data.

**Raises**

- **`InvalidBytesRequestedError`** – If `start_byte < 0`.
- **`ReadOnlyModeError`** – If the provider is in read-only mode.

## 18.5.2 LRU Cache

**`class deeplake.core.storage.LRUCache`**

Bases: `StorageProvider`

LRU Cache that uses `StorageProvider` for caching

**`__delitem__(path: str)`**

Deletes the object present at the path from the cache and the underlying storage.

**Parameters**

**`path`** (`str`) – the path to the object relative to the root of the provider.

**Raises**

- **`KeyError`** – If an object is not found at the path.
- **`ReadOnlyError`** – If the provider is in read-only mode.

**`__getitem__(path: str)`**

If item is in `cache_storage`, retrieves from there and returns. If item isn't in `cache_storage`, retrieves from next storage, stores in `cache_storage` (if possible) and returns.

**Parameters**

**`path`** (`str`) – The path relative to the root of the underlying storage.

**Raises**

**`KeyError`** – if an object is not found at the path.

**Returns**

The bytes of the object present at the path.

**Return type**

`bytes`

**\_\_getstate\_\_**() → Dict[str, Any]

Returns the state of the cache, for pickling

**\_\_init\_\_**(*cache\_storage*: StorageProvider, *next\_storage*: Optional[StorageProvider], *cache\_size*: int)

Initializes the LRUCache. It can be chained with other LRUCache objects to create multilayer caches.

**Parameters**

- **cache\_storage** (StorageProvider) – The storage being used as the caching layer of the cache. This should be a base provider such as MemoryProvider, LocalProvider or S3Provider but not another LRUCache.
- **next\_storage** (StorageProvider) – The next storage layer of the cache. This can either be a base provider (i.e. it is the final storage) or another LRUCache (i.e. in case of chained cache). While reading data, all misses from cache would be retrieved from here. While writing data, the data will be written to the next\_storage when cache\_storage is full or flush is called.
- **cache\_size** (int) – The total space that can be used from the cache\_storage in bytes. This number may be less than the actual space available on the cache\_storage. Setting it to a higher value than actually available space may lead to unexpected behaviors.

**\_\_iter\_\_**()

Generator function that iterates over the keys of the cache and the underlying storage.

**Yields**

*str* – the path of the object that it is iterating over, relative to the root of the provider.

**\_\_len\_\_**()

Returns the number of files present in the cache and the underlying storage.

**Returns**

the number of files present inside the root.

**Return type**

int

**\_\_setitem\_\_**(*path*: str, *value*: Union[bytes, DeepLakeMemoryObject])

Puts the item in the cache\_storage (if possible), else writes to next\_storage.

**Parameters**

- **path** (str) – the path relative to the root of the underlying storage.
- **value** (bytes) – the value to be assigned at the path.

**Raises**

**ReadOnlyError** – If the provider is in read-only mode.

**\_\_setstate\_\_**(*state*: Dict[str, Any])

Recreates a cache with the same configuration as the state.

**Parameters**

**state** (dict) – The state to be used to recreate the cache.

---

**Note:** While restoring the cache, we reset its contents. In case the cache storage was local/s3 and is still accessible when unpickled (if same machine/s3 creds present respectively), the earlier cache contents are no longer accessible.

---

**\_all\_keys**()

Helper function that lists all the objects present in the cache and the underlying storage.

**Returns**

set of all the objects found in the cache and the underlying storage.

**Return type**

set

**`_flush_if_not_read_only()`**

Flushes the cache if not in read-only mode.

**`_forward(path)`**

Forward the value at a given path to the next storage, and un-marks its key.

**`_forward_value(path, value)`**

Forwards a path-value pair to the next storage, and un-marks its key.

**Parameters**

- **path** (*str*) – the path to the object relative to the root of the provider.
- **value** (*bytes*, *DeepLakeMemoryObject*) – the value to send to the next storage.

**`_free_up_space(extra_size: int)`****Helper function that frees up space the required space in cache.**

No action is taken if there is sufficient space in the cache.

**Parameters****extra\_size** (*int*) – the space that needs is required in bytes.**`_insert_in_cache(path: str, value: Union[bytes, DeepLakeMemoryObject])`**

Helper function that adds a key value pair to the cache.

**Parameters**

- **path** (*str*) – the path relative to the root of the underlying storage.
- **value** (*bytes*) – the value to be assigned at the path.

**Raises****ReadOnlyError** – If the provider is in read-only mode.**`_pop_from_cache()`**

Helper function that pops the least recently used key, value pair from the cache

**`clear(prefix="")`**

Deletes ALL the data from all the layers of the cache and the actual storage. This is an IRREVERSIBLE operation. Data once deleted can not be recovered.

**`clear_cache()`**

Flushes the content of all the cache layers if not in read mode and then deletes contents of all the layers of it. This doesn't delete data from the actual storage.

**`clear_deeplake_objects()`**

Removes all DeepLakeMemoryObjects from the cache.

**`flush()`**

Writes data from cache\_storage to next\_storage. Only the dirty keys are written. This is a cascading function and leads to data being written to the final storage in case of a chained cache.

**`get_bytes(path: str, start_byte: Optional[int] = None, end_byte: Optional[int] = None)`**

Gets the object present at the path within the given byte range.

**Parameters**

- **path** (*str*) – The path relative to the root of the provider.
- **start\_byte** (*int*, *optional*) – If only specific bytes starting from start\_byte are required.

- **end\_byte**(*int*, *optional*) – If only specific bytes up to *end\_byte* are required.

**Returns**

The bytes of the object present at the path within the given byte range.

**Return type**

bytes

**Raises**

- **InvalidBytesRequestedError** – If *start\_byte* > *end\_byte* or *start\_byte* < 0 or *end\_byte* < 0.
- **KeyError** – If an object is not found at the path.

**get\_deeplake\_object**(*path*: str, *expected\_class*, *meta*: Optional[Dict] = None, *url*=False, *partial\_bytes*: int = 0)

If the data at *path* was stored using the output of a `DeepLakeMemoryObject`'s *tobytes* function, this function will read it back into object form & keep the object in cache.

**Parameters**

- **path** (str) – Path to the stored object.
- **expected\_class** (callable) – The expected subclass of `DeepLakeMemoryObject`.
- **meta** (dict, optional) – Metadata associated with the stored object
- **url** (bool) – Get presigned url instead of downloading chunk (only for videos)
- **partial\_bytes** (int) – Number of bytes to read from the beginning of the file. If 0, reads the whole file. Defaults to 0.

**Raises**

- **ValueError** – If the incorrect *expected\_class* was provided.
- **ValueError** – If the type of the data at *path* is invalid.
- **ValueError** – If *url* is True but *expected\_class* is not a subclass of `BaseChunk`.

**Returns**

An instance of *expected\_class* populated with the data.

**get\_items**(*paths*)

Pre-load items from next storage into cache

**register\_deeplake\_object**(*path*: str, *obj*: `DeepLakeMemoryObject`)

Registers a new object in the cache.

**remove\_deeplake\_object**(*path*: str)

Removes a `DeepLakeMemoryObject` from the cache.

### 18.5.3 S3 Storage Provider

**class** `deeplake.core.storage.S3Provider`

Bases: `StorageProvider`

Provider class for using S3 storage.

**\_\_delitem\_\_**(*path*)

Delete the object present at the path.

**Parameters**

**path** (str) – the path to the object relative to the root of the `S3Provider`.



---

**Note:** If the object is not found, s3 won't raise `KeyError`.

---

#### Raises

- **`S3DeletionError`** – Any S3 error encountered while deleting the object.
- **`ReadOnlyError`** – If the provider is in read-only mode.

#### `__getitem__(path)`

Gets the object present at the path.

##### Parameters

**path** (*str*) – the path relative to the root of the `S3Provider`.

##### Returns

The bytes of the object present at the path.

##### Return type

bytes

#### Raises

- **`KeyError`** – If an object is not found at the path.
- **`S3GetError`** – Any other error other than `KeyError` while retrieving the object.

`__init__(root: str, aws_access_key_id: Optional[str] = None, aws_secret_access_key: Optional[str] = None, aws_session_token: Optional[str] = None, endpoint_url: Optional[str] = None, aws_region: Optional[str] = None, profile_name: Optional[str] = None, token: Optional[str] = None, **kwargs)`

Initializes the `S3Provider`

#### Example

```
>>> s3_provider = S3Provider("snark-test/benchmarks")
```

#### Parameters

- **root** (*str*) – The root of the provider. All read/write request keys will be appended to root.
- **aws\_access\_key\_id** (*str*, *optional*) – Specifies the AWS access key used as part of the credentials to authenticate the user.
- **aws\_secret\_access\_key** (*str*, *optional*) – Specifies the AWS secret key used as part of the credentials to authenticate the user.
- **aws\_session\_token** (*str*, *optional*) – Specifies an AWS session token used as part of the credentials to authenticate the user.
- **endpoint\_url** (*str*, *optional*) – The complete URL to use for the constructed client. This needs to be provided for cases in which you're interacting with MinIO, Wasabi, etc.
- **aws\_region** (*str*, *optional*) – Specifies the AWS Region to send requests to.
- **profile\_name** (*str*, *optional*) – Specifies the AWS profile name to use.
- **token** (*str*, *optional*) – Activeloop token, used for fetching credentials for Deep Lake datasets (if this is underlying storage for Deep Lake dataset). This is optional, tokens are normally autogenerated.

- **\*\*kwargs** – Additional arguments to pass to the S3 client. Includes: `expiration`.

**\_\_iter\_\_()**

Generator function that iterates over the keys of the S3Provider.

**Yields**

*str* – the name of the object that it is iterating over.

**\_\_len\_\_()**

Returns the number of files present at the root of the S3Provider.

---

**Note:** This is an expensive operation.

---

**Returns**

the number of files present inside the root.

**Return type**

int

**Raises**

**S3ListError** – Any S3 error encountered while listing the objects.

**\_\_setitem\_\_(path, content)**

Sets the object present at the path with the value

**Parameters**

- **path** (*str*) – the path relative to the root of the S3Provider.
- **content** (*bytes*) – the value to be assigned at the path.

**Raises**

- **S3SetError** – Any S3 error encountered while setting the value at the path.
- **ReadOnlyError** – If the provider is in read-only mode.

**\_all\_keys()**

Helper function that lists all the objects present at the root of the S3Provider.

**Returns**

set of all the objects found at the root of the S3Provider.

**Return type**

set

**Raises**

**S3ListError** – Any S3 error encountered while listing the objects.

**\_check\_update\_creds(force=False)**

If the client has an expiration time, check if creds are expired and fetch new ones. This would only happen for datasets stored on Deep Lake storage for which temporary 12 hour credentials are generated.

**\_set\_hub\_creds\_info(hub\_path: str, expiration: str, db\_engine: bool = True, repository: Optional[str] = None)**

Sets the tag and expiration of the credentials. These are only relevant to datasets using Deep Lake storage. This info is used to fetch new credentials when the temporary 12 hour credentials expire.

**Parameters**

- **hub\_path** (*str*) – The deeplake cloud path to the dataset.
- **expiration** (*str*) – The time at which the credentials expire.
- **db\_engine** (*bool*) – Whether Activeloop DB Engine enabled.

- **repository** (*str*, *Optional*) – Backend repository where the dataset is stored.

**\_state\_keys()**

Keys used to store the state of the provider.

**clear**(*prefix=""*)

Deletes ALL data with keys having given prefix on the s3 bucket (under self.root).

**Warning:** Exercise caution!

**get\_bytes**(*path: str*, *start\_byte: Optional[int] = None*, *end\_byte: Optional[int] = None*)

Gets the object present at the path within the given byte range.

**Parameters**

- **path** (*str*) – The path relative to the root of the provider.
- **start\_byte** (*int*, *optional*) – If only specific bytes starting from start\_byte are required.
- **end\_byte** (*int*, *optional*) – If only specific bytes up to end\_byte are required.

**Returns**

The bytes of the object present at the path within the given byte range.

**Return type**

bytes

**Raises**

- ***InvalidBytesRequestedError*** – If start\_byte > end\_byte or start\_byte < 0 or end\_byte < 0.
- **KeyError** – If an object is not found at the path.
- **S3GetAccessError** – Invalid credentials for the object path storage.
- ***S3GetError*** – Any other error while retrieving the object.

**need\_to\_reload\_creds**(*err: ClientError*) → bool

Checks if the credentials need to be reloaded. This happens if the credentials were loaded from the environment and have now expired.

**rename**(*root*)

Rename root folder.

## 18.5.4 Google Cloud Storage Provider

**class** deeplake.core.storage.GCSPProvider

Bases: *StorageProvider*

Provider class for using GC storage.

**\_\_contains\_\_**(*key*)

Checks if key exists in mapping.

**\_\_delitem\_\_**(*key*)

Remove key.

**\_\_getitem\_\_**(*key*)

Retrieve data.

**\_\_init\_\_**(*root: str, token: Optional[Union[Dict, str]] = None, project: Optional[str] = None*)

Initializes the GCSProvider.

### Example

```
>>> gcs_provider = GCSProvider("gcs://my-bucket/gcs_ds")
```

#### Parameters

- **root** (*str*) – The root of the provider. All read/write request keys will be appended to root.
- **token** (*str/Dict*) – GCP token, used for fetching credentials for storage). Can be a path to the credentials file, actual credential dictionary or one of the following:
  - **google\_default**: Tries to load default credentials for the specified project.
  - **cache**: Retrieves the previously used credentials from cache if exist.
  - **anon**: Sets **credentials=None**.
  - **browser**: Generates and stores new token file using cli.
- **project** (*str*) – Name of the project from GCloud.

#### Raises

**ModuleNotFoundError** – If google cloud packages aren't installed.

**\_\_iter\_\_**()

Iterating over the structure.

**\_\_len\_\_**()

Returns length of the structure.

**\_\_setitem\_\_**(*key, value*)

Store value in key.

**\_all\_keys**()

Generator function that iterates over the keys of the provider.

#### Returns

set of all keys present at the root of the provider.

#### Return type

set

**\_set\_hub\_creds\_info**(*hub\_path: str, expiration: str, db\_engine: bool = True, repository: Optional[str] = None*)

Sets the tag and expiration of the credentials. These are only relevant to datasets using Deep Lake storage. This info is used to fetch new credentials when the temporary 12 hour credentials expire.

#### Parameters

- **hub\_path** (*str*) – The deeplake cloud path to the dataset.
- **expiration** (*str*) – The time at which the credentials expire.
- **db\_engine** (*bool*) – Whether Activeloop DB Engine enabled.
- **repository** (*str, Optional*) – Backend repository where the dataset is stored.

**clear**(*prefix=""*)

Remove all keys with given prefix below root - empties out mapping.

**Warning:** Exercise caution!

**get\_bytes**(*path: str, start\_byte: Optional[int] = None, end\_byte: Optional[int] = None*)

Gets the object present at the path within the given byte range.

**Parameters**

- **path** (*str*) – The path relative to the root of the provider.
- **start\_byte** (*int, optional*) – If only specific bytes starting from *start\_byte* are required.
- **end\_byte** (*int, optional*) – If only specific bytes up to *end\_byte* are required.

**Returns**

The bytes of the object present at the path within the given byte range.

**Return type**

bytes

**Raises**

- ***InvalidBytesRequestedError*** – If *start\_byte* > *end\_byte* or *start\_byte* < 0 or *end\_byte* < 0.
- **KeyError** – If an object is not found at the path.

**rename**(*root*)

Rename root folder.

## 18.5.5 Google Drive Storage Provider

**class** `deeplake.core.storage.GDriveProvider`

Bases: *StorageProvider*

Provider class for using Google Drive storage.

**\_\_delitem\_\_**(*path*)

Delete the object present at the path.

**Parameters**

**path** (*str*) – the path to the object relative to the root of the provider.

**Raises**

**KeyError** – If an object is not found at the path.

**\_\_getitem\_\_**(*path*)

Gets the object present at the path within the given byte range.

**Parameters**

**path** (*str*) – The path relative to the root of the provider.

**Returns**

The bytes of the object present at the path.

**Return type**

bytes

**Raises**

**KeyError** – If an object is not found at the path.

```
__init__(root: str, token: Optional[Union[Dict, str]] = None, makemap: bool = True)
```

Initializes the GDriveProvider

### Example

```
>>> gdrive_provider = GDriveProvider("gdrive://folder_name/folder_name")
```

#### Parameters

- **root** (*str*) – The root of the provider. All read/write request keys will be appended to root.
- **token** (*dict*, *str*, *optional*) – Google Drive token. Can be path to the token file or the actual credentials dictionary.
- **makemap** (*bool*) – Creates path to id map if True.

#### Raises

**ImportError** – If required packages are not installed.

---

#### Note:

- Requires `client_secrets.json` in working directory if `token` is not provided.
  - Due to limits on requests per 100 seconds on google drive api, continuous requests such as uploading many small files can be slow.
  - Users can request to increase their quotas on their google cloud platform.
- 

```
__iter__()
```

Generator function that iterates over the keys of the provider.

#### Yields

*str* – the path of the object that it is iterating over, relative to the root of the provider.

```
__len__()
```

Returns the number of files present inside the root of the provider.

#### Returns

the number of files present inside the root.

#### Return type

int

```
__setitem__(path, content)
```

Sets the object present at the path with the value

#### Parameters

- **path** (*str*) – the path relative to the root of the provider.
- **value** (*bytes*) – the value to be assigned at the path.

```
_all_keys()
```

Generator function that iterates over the keys of the provider.

#### Returns

set of all keys present at the root of the provider.

#### Return type

set

```
clear(prefix="")
```

Delete the contents of the provider.

**sync()**

Sync provider keys with actual storage

## 18.5.6 Local Storage Provider

**class** deeplake.core.storage.LocalProvider

Bases: *StorageProvider*

Provider class for using the local filesystem.

**\_\_delitem\_\_**(*path: str*)

Delete the object present at the path.

### Example

```
>>> local_provider = LocalProvider("/home/ubuntu/Documents/")
>>> del local_provider["abc.txt"]
```

#### Parameters

**path** (*str*) – the path to the object relative to the root of the provider.

#### Raises

- **KeyError** – If an object is not found at the path.
- *DirectoryAtPathException* – If a directory is found at the path.
- **Exception** – Any other exception encountered while trying to fetch the object.
- **ReadOnlyError** – If the provider is in read-only mode.

**\_\_getitem\_\_**(*path: str*)

Gets the object present at the path within the given byte range.

### Example

```
>>> local_provider = LocalProvider("/home/ubuntu/Documents/")
>>> my_data = local_provider["abc.txt"]
```

#### Parameters

**path** (*str*) – The path relative to the root of the provider.

#### Returns

The bytes of the object present at the path.

#### Return type

bytes

#### Raises

- **KeyError** – If an object is not found at the path.
- *DirectoryAtPathException* – If a directory is found at the path.
- **Exception** – Any other exception encountered while trying to fetch the object.

**\_\_init\_\_**(*root: str*)

Initializes the LocalProvider.

### Example

```
>>> local_provider = LocalProvider("/home/ubuntu/Documents/")
```

#### Parameters

**root** (*str*) – The root of the provider. All read/write request keys will be appended to root.”

#### Raises

***FileAtPathException*** – If the root is a file instead of a directory.

#### `__iter__()`

Generator function that iterates over the keys of the provider.

### Example

```
>>> local_provider = LocalProvider("/home/ubuntu/Documents/")
>>> for my_data in local_provider:
...     pass
```

#### Yields

*str* – the path of the object that it is iterating over, relative to the root of the provider.

#### `__len__()`

Returns the number of files present inside the root of the provider.

### Example

```
>>> local_provider = LocalProvider("/home/ubuntu/Documents/")
>>> len(local_provider)
```

#### Returns

the number of files present inside the root.

#### Return type

int

#### `__setitem__(path: str, value: bytes)`

Sets the object present at the path with the value

### Example

```
>>> local_provider = LocalProvider("/home/ubuntu/Documents/")
>>> local_provider["abc.txt"] = b"abcd"
```

#### Parameters

- **path** (*str*) – the path relative to the root of the provider.
- **value** (*bytes*) – the value to be assigned at the path.

#### Raises

- **Exception** – If unable to set item due to directory at path or permission or space issues.



- **`FileAtPathException`** – If the directory to the path is a file instead of a directory.
- **`ReadOnlyError`** – If the provider is in read-only mode.

**`_all_keys`**(*refresh: bool = False*) → Set[str]

Lists all the objects present at the root of the Provider.

**Parameters**

**`refresh`** (*bool*) – refresh keys

**Returns**

set of all the objects found at the root of the Provider.

**Return type**

set

**`_check_is_file`**(*path: str*)

Checks if the path is a file. Returns the full\_path to file if True.

**Parameters**

**`path`** (*str*) – the path to the object relative to the root of the provider.

**Returns**

the full path to the requested file.

**Return type**

str

**Raises**

**`DirectoryAtPathException`** – If a directory is found at the path.

**`_set_hub_creds_info`**(*hub\_path: str, expiration: str, db\_engine: bool = True, repository: Optional[str] = None*)

Sets the tag and expiration of the credentials. These are only relevant to datasets using Deep Lake storage. This info is used to fetch new credentials when the temporary 12 hour credentials expire.

**Parameters**

- **`hub_path`** (*str*) – The deeplake cloud path to the dataset.
- **`expiration`** (*str*) – The time at which the credentials expire.
- **`db_engine`** (*bool*) – Whether Activeloop DB Engine enabled.
- **`repository`** (*str, Optional*) – Backend repository where the dataset is stored.

**`clear`**(*prefix=""*)

Deletes ALL data with keys having given prefix on the local machine (under self.root). Exercise caution!

**`get_bytes`**(*path: str, start\_byte: Optional[int] = None, end\_byte: Optional[int] = None*)

Gets the object present at the path within the given byte range.

**Parameters**

- **`path`** (*str*) – The path relative to the root of the provider.
- **`start_byte`** (*int, optional*) – If only specific bytes starting from start\_byte are required.
- **`end_byte`** (*int, optional*) – If only specific bytes up to end\_byte are required.

**Returns**

The bytes of the object present at the path within the given byte range.

**Return type**

bytes

**Raises**

- **`InvalidBytesRequestedError`** – If `start_byte > end_byte` or `start_byte < 0` or `end_byte < 0`.
- **`KeyError`** – If an object is not found at the path.

**`rename(path)`**

Renames root folder

## 18.5.7 Memory Provider

**`class deeplake.core.storage.MemoryProvider`**

Bases: `StorageProvider`

Provider class for using the memory.

**`__delitem__(path: str)`**

Delete the object present at the path.

### Example

```
>>> memory_provider = MemoryProvider("xyz")
>>> del memory_provider["abc.txt"]
```

#### Parameters

**`path (str)`** – the path to the object relative to the root of the provider.

#### Raises

- **`KeyError`** – If an object is not found at the path.
- **`ReadOnlyError`** – If the provider is in read-only mode.

**`__getitem__(path: str)`**

Gets the object present at the path within the given byte range.

### Example

```
>>> memory_provider = MemoryProvider("xyz")
>>> my_data = memory_provider["abc.txt"]
```

#### Parameters

**`path (str)`** – The path relative to the root of the provider.

#### Returns

The bytes of the object present at the path.

#### Return type

bytes

#### Raises

**`KeyError`** – If an object is not found at the path.

**`__getstate__() → str`**

Does NOT save the in memory data in state.

**`__init__(root: str = "")`**

**`__iter__()`**

Generator function that iterates over the keys of the provider.

### Example

```
>>> memory_provider = MemoryProvider("xyz")
>>> for my_data in memory_provider:
...     pass
```

#### Yields

*str* – the path of the object that it is iterating over, relative to the root of the provider.

#### `__len__()`

Returns the number of files present inside the root of the provider.

### Example

```
>>> memory_provider = MemoryProvider("xyz")
>>> len(memory_provider)
```

#### Returns

the number of files present inside the root.

#### Return type

int

#### `__setitem__(path: str, value: bytes)`

Sets the object present at the path with the value

### Example

```
>>> memory_provider = MemoryProvider("xyz")
>>> memory_provider["abc.txt"] = b"abcd"
```

#### Parameters

- **path** (*str*) – the path relative to the root of the provider.
- **value** (*bytes*) – the value to be assigned at the path.

#### Raises

**ReadOnlyError** – If the provider is in read-only mode.

#### `_all_keys()`

Lists all the objects present at the root of the Provider.

#### Returns

set of all the objects found at the root of the Provider.

#### Return type

set

#### `clear(prefix="")`

Clears the provider.

## 18.6 deeplake.core.index

**class** deeplake.core.index.**IndexEntry**(value: Union[int, slice, Tuple[int, ...]] = slice(None, None, None))

**\_\_getitem\_\_**(item: Union[int, slice, Tuple[int, ...]])

Combines the given item and this IndexEntry. Returns a new IndexEntry representing the composition of the two.

### Examples

```
>>> IndexEntry()[0:100]
IndexEntry(slice(0, 100, None))
```

```
>>> IndexEntry()[100:200][5]
IndexEntry(105)
```

```
>>> IndexEntry()[(0, 1, 2, 3)]
IndexEntry((0, 1, 2, 3))
```

```
>>> IndexEntry()[1, 2, 3]
IndexEntry((0, 1, 2, 3))
```

#### Parameters

**item** – The desired sub-index to be composed with this IndexEntry. Can be an int, a slice, or a tuple of ints.

#### Returns

The new IndexEntry object.

#### Return type

*IndexEntry*

#### Raises

**TypeError** – An integer IndexEntry should not be indexed further.

**\_\_init\_\_**(value: Union[int, slice, Tuple[int, ...]] = slice(None, None, None))

**\_\_str\_\_**()

Return str(self).

**\_\_weakref\_\_**

list of weak references to the object (if defined)

**downsample**(factor: int, length: int)

Downsamples an IndexEntry by a given factor.

#### Parameters

- **factor** (int) – The factor by which to downsample.
- **length** (int) – The length of the downsampled IndexEntry.

#### Returns

The downsampled IndexEntry.

#### Return type

*IndexEntry*

#### Raises

**TypeError** – If the IndexEntry cannot be downsampled.

**indices**(*length: int*)

Generates the sequence of integer indices for a target of a given length.

**is\_trivial**()

Checks if an IndexEntry represents the entire slice

**length**(*parent\_length: int*) → int

Returns the length of an IndexEntry given the length of the parent it is indexing.

### Examples

```
>>> IndexEntry(slice(5, 100)).length(100)
5
>>> len(list(range(100))[5:10])
5
>>> IndexEntry(slice(5, 100)).length(50)
45
>>> len(list(range(50))[5:100])
45
>>> IndexEntry(0).length(10)
1
```

#### Parameters

**parent\_length** (*int*) – The length of the target that this IndexEntry is indexing.

#### Returns

The length of the index if it were applied to a parent of the given length.

#### Return type

int

**subscriptable**()

Returns whether an IndexEntry can be further subscripted.

**validate**(*parent\_length: int*)

Checks that the index is not accessing values outside the range of the parent.

```
class deeplake.core.index.Index(item: Union[int, slice, Tuple[int, ...], Index, List[IndexEntry]] =
    slice(None, None, None))
```

**\_\_getitem\_\_**(*item: Union[int, slice, List[int], Tuple[Union[int, slice, Tuple[int, ...]]], Index]*)

Returns a new Index representing a subscripting with the given item. Modeled after NumPy's advanced integer indexing.

See: <https://numpy.org/doc/stable/reference/arrays.indexing.html>

### Examples

```
>>> Index([5, slice(None)])[5]
Index([5, 5])
```

```
>>> Index([5])[5:6]
Index([5, slice(5, 6)])
```

```
>>> Index()[0, 1, 2:5, 3]
Index([0, 1, slice(2, 5), 3])
```

```
>>> Index([slice(5, 6)])([0, 1, 2:5, 3],)
Index([(5, 1, slice(2, 5), 3)])
```

**Parameters**

**item** – The contents of the subscript expression to add to this Index.

**Returns**

The Index representing the result of the subscript operation.

**Return type**

*Index*

**Raises**

**TypeError** – Given item should be another Index, or compatible with NumPy’s advanced integer indexing.

**\_\_init\_\_**(*item: Union[int, slice, Tuple[int, ...], Index, List[IndexEntry]] = slice(None, None, None)*)

Initializes an Index from an IndexValue, another Index, or the values from another Index.

Represents a list of IndexEntry objects corresponding to indexes into each axis of an ndarray.

**\_\_repr\_\_**()

Return repr(self).

**\_\_str\_\_**()

Return str(self).

**\_\_weakref\_\_**

list of weak references to the object (if defined)

**apply**(*samples: List[ndarray]*)

Applies an Index to a list of ndarray samples with the same number of entries as the first entry in the Index.

**apply\_squeeze**(*samples: List[ndarray]*)

Applies the primary axis of an Index to a list of ndarray samples. Will either return the list as given, or return the first sample.

**compose\_at**(*item: Union[int, slice, Tuple[int, ...]], i: Optional[int] = None*)

Returns a new Index representing the addition of an IndexValue, or the composition with a given axis.

**Examples**

```
>>> Index([slice(None), slice(None)]).compose_at(5)
Index([slice(None), slice(None), 5])
```

```
>>> Index([slice(None), slice(5, 10), slice(None)]).compose_at(3, 1)
Index([slice(None), 8, slice(None)])
```

**Parameters**

- **item** (*IndexValue*) – The value to append or compose with the Index.
- **i** (*int, optional*) – The axis to compose with the given item. Defaults to None, meaning that the item will be appended instead.

**Returns**

The result of the addition or composition.

**Return type**

*Index*

**downsample**(*factor: int, shape: Tuple[int, ...]*)

Downsamples an Index by the given factor.

**Parameters**

- **factor** (*int*) – The factor to downsample by.
- **shape** (*Tuple[int, ...]*) – The shape of the downsampled data.

**Returns**

The downsampled Index.

**Return type**

*Index*

**find\_axis**(*offset: int = 0*)

Returns the index for the nth subscriptable axis in the values of an Index.

**Parameters**

**offset** (*int*) – The number of subscriptable axes to skip before returning. Defaults to 0, meaning that the first valid axis is returned.

**Returns**

The index of the found axis, or None if no match is found.

**Return type**

*int*

**is\_trivial**()

Checks if an Index is equivalent to the trivial slice `[:]`, aka `slice(None)`.

**length**(*parent\_length: int*)

Returns the primary length of an Index given the length of the parent it is indexing. See: [IndexEntry.length\(\)](#)

*length()*

**validate**(*parent\_length*)

Checks that the index is not accessing values outside the range of the parent.

`deeplake.core.index.merge_slices(existing_slice: slice, new_slice: slice) → slice`

Compose two slice objects

Given an iterable x, the following should be equivalent:

`x[existing_slice][new_slice] == x[merge_slices(existing_slice, new_slice)]`

**Parameters**

- **existing\_slice** (*slice*) – The existing slice to be restricted.
- **new\_slice** (*slice*) – The new slice to be applied to the existing slice.

**Returns**

the composition of the given slices

**Return type**

*slice*

**Raises**

**NotImplementedError** – Composing slices with negative values is not supported. Negative indexing for slices is only supported for the first slice.

`deeplake.core.index.slice_at_int(s: slice, i: int)`

Returns the *i* th element of a slice *s*.

### Examples

```
>>> slice_at_int(slice(None), 10)
10
```

```
>>> slice_at_int(slice(10, 20, 2), 3)
16
```

#### Parameters

- **s** (*slice*) – The slice to index into.
- **i** (*int*) – The integer offset into the slice.

#### Returns

The index corresponding to the offset into the slice.

#### Return type

int

#### Raises

- **NotImplementedError** – Nontrivial slices should not be indexed with negative integers.
- **IndexError** – If step is negative and start is not greater than stop.

`deeplake.core.index.slice_length(s: slice, parent_length: int) → int`

Returns the length of a slice given the length of its parent.



## DEEPLAKE.CORE.DATASET

### 19.1 Dataset

**class** deeplake.core.dataset.Dataset

**add\_creds\_key**(creds\_key: str, managed: bool = False)

Adds a new creds key to the dataset. These keys are used for tensors that are linked to external data.

#### Examples

```
>>> # create/load a dataset
>>> ds = deeplake.empty("path/to/dataset")
>>> # add a new creds key
>>> ds.add_creds_key("my_s3_key")
```

#### Parameters

- **creds\_key** (str) – The key to be added.
- **managed** (bool) –
  - If True, the creds corresponding to the key will be fetched from Activeloop platform.
  - Defaults to False.

#### Raises

**ValueError** – If the dataset is not connected to Activeloop platform and managed is True.

---

**Note:** managed parameter is applicable only for datasets that are connected to [Activeloop platform](#).

---

**property allow\_delete:** bool

Returns True if dataset can be deleted from storage. Whether it can be deleted or not is stored in the database\_meta.json and can be changed with `allow_delete = True|False`

**append**(sample: Dict[str, Any], skip\_ok: bool = False, append\_empty: bool = False)

Append samples to multiple tensors at once. This method expects all tensors being updated to be of the same length.

#### Parameters

- **sample** (dict) – Dictionary with tensor names as keys and samples as values.

- **skip\_ok** (*bool*) – Skip tensors not in *sample* if set to *True*.
- **append\_empty** (*bool*) – Append empty samples to tensors not specified in *sample* if set to *True*. If *True*, *skip\_ok* is ignored.

**Raises**

- **KeyError** – If any tensor in the dataset is not a key in *sample* and *skip\_ok* is *False*.
- **TensorDoesNotExistError** – If tensor in *sample* does not exist.
- **ValueError** – If all tensors being updated are not of the same length.
- **NotImplementedError** – If an error occurs while writing tiles.
- **Exception** – Error while attempting to rollback appends.
- **SampleAppendingError** – Error that occurs when someone tries to append a tensor value directly to the dataset without specifying tensor name.

**Examples**

```
>>> ds = deeplake.empty("../test/test_ds")
>>> ds.create_tensor('data')
Tensor(key='data')
>>> ds.create_tensor('labels')
Tensor(key='labels')
>>> ds.append({"data": [1, 2, 3, 4], "labels": [0, 1, 2, 3]})
```

**property branch:** *str*

The current branch of the dataset

**property branches**

Lists all the branches of the dataset.

**Returns**

List of branches.

**checkout** (*address: str, create: bool = False, reset: bool = False*) → *Optional[str]*

Checks out to a specific *commit\_id* or branch. If *create = True*, creates a new branch with name *address*.

**Parameters**

- **address** (*str*) – The *commit\_id* or branch to checkout to.
- **create** (*bool*) – If *True*, creates a new branch with name as *address*.
- **reset** (*bool*) – If checkout fails due to a corrupted HEAD state of the branch, setting *reset=True* will reset HEAD changes and attempt the checkout again.

**Returns**

The *commit\_id* of the dataset after checkout.

**Return type**

*Optional[str]*

**Raises**

- **CheckoutError** – If *address* could not be found.
- **ReadOnlyModeError** – If branch creation or reset is attempted in read-only mode.

- **DatasetCorruptError** – If checkout failed due to dataset corruption and reset is not True.
- **Exception** – If the dataset is a filtered view.

### Examples

```
>>> ds = deeplake.empty("../test/test_ds")
>>> ds.create_tensor("abc")
Tensor(key='abc')
>>> ds.abc.append([1, 2, 3])
>>> first_commit = ds.commit()
>>> ds.checkout("alt", create=True)
'firstdbf9474d461a19e9333c2fd19b46115348f'
>>> ds.abc.append([4, 5, 6])
>>> ds.abc.numpy()
array([[1, 2, 3],
       [4, 5, 6]])
>>> ds.checkout(first_commit)
'firstdbf9474d461a19e9333c2fd19b46115348f'
>>> ds.abc.numpy()
array([[1, 2, 3]])
```

**Note:** Checkout from a head node in any branch that contains uncommitted data will lead to an automatic commit before the checkout.

### clear\_cache()

- Flushes (see [Dataset.flush\(\)](#)) the contents of the cache layers (if any) and then deletes contents of all the layers of it.
- This doesn't delete data from the actual storage.
- This is useful if you have multiple datasets with memory caches open, taking up too much RAM.
- Also useful when local cache is no longer needed for certain datasets and is taking up storage space.

### property client

Returns the client of the dataset.

**commit**(*message: Optional[str] = None, allow\_empty=False*) → str

Stores a snapshot of the current state of the dataset.

#### Parameters

- **message** (str, Optional) – Used to describe the commit.
- **allow\_empty** (bool) – If True, commit even if there are no changes.

#### Returns

the commit id of the saved commit that can be used to access the snapshot.

#### Return type

str

#### Raises

- **Exception** – If dataset is a filtered view.
- **EmptyCommitError** – if there are no changes and user does not forced to commit unchanged data.

**Note:**

- Committing from a non-head node in any branch, will lead to an automatic checkout to a new branch.
  - This same behaviour will happen if new samples are added or existing samples are updated from a non-head node.
- 

**property commit\_id:** `Optional[str]`

The lastest committed commit id of the dataset. If there are no commits, this returns `None`.

**property commits:** `List[Dict]`

Lists all the commits leading to the current dataset state.

**Returns**

List of dictionaries containing commit information.

**connect**(*creds\_key: str, dest\_path: Optional[str] = None, org\_id: Optional[str] = None, ds\_name: Optional[str] = None, token: Optional[str] = None*)

Connect a Deep Lake cloud dataset through a deeplake path.

**Examples**

```
>>> # create/load an s3 dataset
>>> s3_ds = deeplake.dataset("s3://bucket/dataset")
>>> ds = s3_ds.connect(dest_path="hub://my_org/dataset", creds_key="my_managed_
→ credentials_key", token="my_active_loop_token")
>>> # or
>>> ds = s3_ds.connect(org_id="my_org", creds_key="my_managed_credentials_key",
→ token="my_active_loop_token")
```

**Parameters**

- **creds\_key** (*str*) – The managed credentials to be used for accessing the source path.
- **dest\_path** (*str, optional*) – The full path to where the connected Deep Lake dataset will reside. Can be: a Deep Lake path like `hub://organization/dataset`
- **org\_id** (*str, optional*) – The organization to where the connected Deep Lake dataset will be added.
- **ds\_name** (*str, optional*) – The name of the connected Deep Lake dataset. Will be inferred from `dest_path` or `src_path` if not provided.
- **token** (*str, optional*) – Activeloop token used to fetch the managed credentials.

**Raises**

- **InvalidSourcePathError** – If the dataset's path is not a valid s3, gcs or azure path.
- **InvalidDestinationPathError** – If `dest_path`, or `org_id` and `ds_name` do not form a valid Deep Lake path.
- **TokenPermissionError** – If the user does not have permission to create a dataset in the specified organization.

**copy**(*dest: Union[str, Path], runtime: Optional[dict] = None, tensors: Optional[List[str]] = None, overwrite: bool = False, creds=None, token=None, num\_workers: int = 0, scheduler='threaded', progressbar=True, public: bool = False*)

Copies this dataset or dataset view to *dest*. Version control history is not included.

#### Parameters

- **dest** (*str, pathlib.Path*) – Destination dataset or path to copy to. If a Dataset instance is provided, it is expected to be empty.
- **tensors** (*List[str], optional*) – Names of tensors (and groups) to be copied. If not specified all tensors are copied.
- **runtime** (*dict*) – Parameters for Activeloop DB Engine. Only applicable for *hub://* paths.
- **overwrite** (*bool*) – If True and a dataset exists at *destination*, it will be overwritten. Defaults to False.
- **creds** (*dict, Optional*) – creds required to create / overwrite datasets at *dest*.
- **token** (*str, Optional*) – token used to for fetching credentials to *dest*.
- **num\_workers** (*int*) – The number of workers to use for copying. Defaults to 0. When set to 0, it will always use serial processing, irrespective of the scheduler.
- **scheduler** (*str*) – The scheduler to be used for copying. Supported values include: 'serial', 'threaded', 'processed' and 'ray'. Defaults to 'threaded'.
- **progressbar** (*bool*) – Displays a progress bar If True (default).
- **public** (*bool*) – Defines if the dataset will have public access. Applicable only if Deep Lake cloud storage is used and a new Dataset is being created. Defaults to False.

#### Returns

New dataset object.

#### Return type

*Dataset*

#### Raises

*DatasetHandlerError* – If a dataset already exists at destination path and overwrite is False.

**create\_group**(*name: str, exist\_ok=False*) → *Dataset*

Creates a tensor group. Intermediate groups in the path are also created.

#### Parameters

- **name** – The name of the group to create.
- **exist\_ok** – If True, the group is created if it does not exist. If False, an error is raised if the group already exists. Defaults to False.

#### Returns

The created group.

#### Raises

*TensorGroupAlreadyExistsError* – If the group already exists and *exist\_ok* is False.

## Examples

```
>>> ds.create_group("images")
>>> ds["images"].create_tensor("cats")
```

```
>>> ds.create_groups("images/jpg/cats")
>>> ds["images"].create_tensor("png")
>>> ds["images/jpg"].create_group("dogs")
```

**create\_tensor**(*name: str, htype: str = 'unspecified', dtype: Union[str, dtype] = 'unspecified', sample\_compression: str = 'unspecified', chunk\_compression: str = 'unspecified', hidden: bool = False, create\_sample\_info\_tensor: bool = True, create\_shape\_tensor: bool = True, create\_id\_tensor: bool = True, verify: bool = True, exist\_ok: bool = False, verbose: bool = True, downsampling: Optional[Tuple[int, int]] = None, tiling\_threshold: Optional[int] = None, \*\*kwargs*)

Creates a new tensor in the dataset.

## Examples

```
>>> # create dataset
>>> ds = deeplake.dataset("path/to/dataset")
```

```
>>> # create tensors
>>> ds.create_tensor("images", htype="image", sample_compression="jpg")
>>> ds.create_tensor("videos", htype="video", sample_compression="mp4")
>>> ds.create_tensor("data")
>>> ds.create_tensor("point_clouds", htype="point_cloud")
```

```
>>> # append data
>>> ds.images.append(np.ones((400, 400, 3), dtype='uint8'))
>>> ds.videos.append(deeplake.read("videos/sample_video.mp4"))
>>> ds.data.append(np.zeros((100, 100, 2)))
```

## Parameters

- **name** (*str*) – The name of the tensor to be created.
- **htype** (*str*) –
  - The class of data for the tensor.
  - The defaults for other parameters are determined in terms of this value.
  - For example, `htype="image"` would have `dtype` default to `uint8`.
  - These defaults can be overridden by explicitly passing any of the other parameters to this function.
  - May also modify the defaults for other parameters.
- **dtype** (*str*) – Optionally override this tensor's `dtype`. All subsequent samples are required to have this `dtype`.
- **sample\_compression** (*str*) – All samples will be compressed in the provided format. If `None`, samples are uncompressed. For `link[]` tensors, `sample_compression` is used only for optimizing dataset views.

- **chunk\_compression** (*str*) – All chunks will be compressed in the provided format. If *None*, chunks are uncompressed. For `link[]` tensors, `chunk_compression` is used only for optimizing dataset views.
- **hidden** (*bool*) – If *True*, the tensor will be hidden from `ds.tensors` but can still be accessed via `ds[tensor_name]`.
- **create\_sample\_info\_tensor** (*bool*) – If *True*, meta data of individual samples will be saved in a hidden tensor. This data can be accessed via `tensor[i].sample_info`.
- **create\_shape\_tensor** (*bool*) – If *True*, an associated tensor containing shapes of each sample will be created.
- **create\_id\_tensor** (*bool*) – If *True*, an associated tensor containing unique ids for each sample will be created. This is useful for merge operations.
- **verify** (*bool*) – Valid only for link htypes. If *True*, all links will be verified before they are added to the tensor. If *False*, links will be added without verification but note that `create_shape_tensor` and `create_sample_info_tensor` will be set to *False*.
- **exist\_ok** (*bool*) – If *True*, the group is created if it does not exist. if *False*, an error is raised if the group already exists.
- **verbose** (*bool*) – Shows warnings if *True*.
- **downsampling** (*tuple[int, int]*) – If not *None*, the tensor will be downsampled by the provided factors. For example, `(2, 5)` will downsample the tensor by a factor of 2 in both dimensions and create 5 layers of downsampled tensors. Only support for image and mask htypes.
- **tiling\_threshold** (*Optional, int*) – In bytes. Tiles large images if their size exceeds this threshold. Set to -1 to disable tiling.
- **\*\*kwargs** –
  - htype defaults can be overridden by passing any of the compatible parameters.
  - To see all htypes and their correspondent arguments, check out [Htypes](#).

**Returns**

The new tensor, which can be accessed by `dataset[name]` or `dataset.name`.

**Return type**

[Tensor](#)

**Raises**

- **TensorAlreadyExistsError** – If the tensor already exists and `exist_ok` is *False*.
- **TensorGroupAlreadyExistsError** – Duplicate tensor groups are not allowed.
- **InvalidTensorNameError** – If `name` is in dataset attributes.
- **NotImplementedError** – If trying to override `chunk_compression`.
- **TensorMetaInvalidHtype** – If invalid htype is specified.
- **ValueError** – If an illegal argument is specified.

**create\_tensor\_like**(*name: str, source: [Tensor](#), unlink: bool = False*) → [Tensor](#)

Copies the source tensor's meta information and creates a new tensor with it. No samples are copied, only the meta/info for the tensor is.

## Examples

```
>>> ds.create_tensor_like("cats", ds["images"])
```

### Parameters

- **name** (*str*) – Name for the new tensor.
- **source** (*Tensor*) – Tensor who's meta/info will be copied. May or may not be contained in the same dataset.
- **unlink** (*bool*) – Whether to unlink linked tensors.

### Returns

New Tensor object.

### Return type

*Tensor*

**dataloader**(*ignore\_errors: bool = False, verbose: bool = False*)

Returns a *DeepLakeDataLoader* object.

### Parameters

- **ignore\_errors** (*bool*) – If True, the data loader will ignore errors appeared during data iteration otherwise it will collect the statistics and report appeared errors. Default value is False
- **verbose** (*bool*) – If True, the data loader will dump verbose logs of it's steps. Default value is False

### Returns

A *deeplake.enterprise.dataloader.DeepLakeDataLoader* object.

### Return type

*DeepLakeDataLoader*

## Examples

Creating a simple dataloader object which returns a batch of numpy arrays

```
>>> import deeplake
>>> ds_train = deeplake.load('hub://activeloop/fashion-mnist-train')
>>> train_loader = ds_train.dataloader().numpy()
>>> for i, data in enumerate(train_loader):
...     # custom logic on data
...     pass
```

Creating dataloader with custom transformation and batch size

```
>>> import deeplake
>>> import torch
>>> from torchvision import datasets, transforms, models
>>>
>>> ds_train = deeplake.load('hub://activeloop/fashion-mnist-train')
>>> tform = transforms.Compose([
...     transforms.ToPILImage(), # Must convert to PIL image for subsequent
↳ operations to run
...     transforms.RandomRotation(20), # Image augmentation
...     transforms.ToTensor(), # Must convert to pytorch tensor for subsequent
↳
```

(continues on next page)



(continued from previous page)

```

→operations to run
...     transforms.Normalize([0.5], [0.5]),
... ]])
...
>>> batch_size = 32
>>> # create dataloader by chaining with transform function and batch size and
→returns batch of pytorch tensors
>>> train_loader = ds_train.dataloader()\
...     .transform({'images': tform, 'labels': None})\
...     .batch(batch_size)\
...     .shuffle()\
...     .pytorch()
...
>>> # loop over the elements
>>> for i, data in enumerate(train_loader):
...     # custom logic on data
...     pass

```

Creating dataloader and chaining with query

```

>>> ds = deeplake.load('hub://activeloop/coco-train')
>>> train_loader = ds_train.dataloader()\
...     .query("(select * where contains(categories, 'car') limit 1000) union
→(select * where contains(categories, 'motorcycle') limit 1000)")\
...     .pytorch()
...
>>> # loop over the elements
>>> for i, data in enumerate(train_loader):
...     # custom logic on data
...     pass

```

**delete**(large\_ok=False)

Deletes the entire dataset from the cache layers (if any) and the underlying storage. This is an **IRREVERSIBLE** operation. Data once deleted can not be recovered.

**Parameters**

**large\_ok** (bool) – Delete datasets larger than 1 GB. Defaults to False.

**Raises**

- **DatasetTooLargeToDelete** – If the dataset is larger than 1 GB and large\_ok is False.
- **DatasetHandlerError** – If the dataset is marked as allow\_delete=False.

**delete\_branch**(name: str) → None

Deletes the branch and cleans up any unneeded data. Branches can only be deleted if there are no sub-branches and if it has never been merged into another branch.

**Parameters**

**name** (str) – The branch to delete.

**Raises**

- **CommitError** – If branch could not be found.
- **ReadOnlyModeError** – If branch deletion is attempted in read-only mode.
- **Exception** – If you have the given branch currently checked out.

### Examples

```
>>> ds = deeplake.empty("../test/test_ds")
>>> ds.create_tensor("abc")
Tensor(key='abc')
>>> ds.abc.append([1, 2, 3])
>>> first_commit = ds.commit()
>>> ds.checkout("alt", create=True)
'firstdbf9474d461a19e9333c2fd19b46115348f'
>>> ds.abc.append([4, 5, 6])
>>> ds.abc.numpy()
array([[1, 2, 3],
       [4, 5, 6]])
>>> ds.checkout(first_commit)
'firstdbf9474d461a19e9333c2fd19b46115348f'
>>> ds.delete_branch("alt")
```

**delete\_group**(*name: str, large\_ok: bool = False*)

Delete a tensor group from the dataset.

### Examples

```
>>> ds.delete_group("images/dogs")
```

#### Parameters

- **name** (*str*) – The name of tensor group to be deleted.
- **large\_ok** (*bool*) – Delete tensor groups larger than 1 GB. Disabled by default.

#### Returns

None

#### Raises

**TensorGroupDoesNotExistError** – If tensor group of name *name* does not exist in the dataset.

**delete\_tensor**(*name: str, large\_ok: bool = False*)

Delete a tensor from the dataset.

### Examples

```
>>> ds.delete_tensor("images/cats")
```

#### Parameters

- **name** (*str*) – The name of tensor to be deleted.
- **large\_ok** (*bool*) – Delete tensors larger than 1 GB. Disabled by default.

#### Returns

None

#### Raises

- **TensorDoesNotExistError** – If tensor of name *name* does not exist in the dataset.

- **TensorTooLargeToDelete** – If the tensor is larger than 1 GB and `large_ok` is `False`.

**delete\_view**(*id: str*)

Deletes the view with given view id.

**Parameters**

**id** (*str*) – Id of the view to delete.

**Raises**

**KeyError** – if view with given id does not exist.

**diff**(*id\_1: Optional[str] = None, id\_2: Optional[str] = None, as\_dict=False*) → `Optional[Dict]`

Returns/displays the differences between commits/branches.

For each tensor this contains information about the sample indexes that were added/modified as well as whether the tensor was created.

**Parameters**

- **id\_1** (*str, Optional*) – The first commit\_id or branch name.
- **id\_2** (*str, Optional*) – The second commit\_id or branch name.
- **as\_dict** (*bool, Optional*) – If `True`, returns the diff as lists of commit wise dictionaries.

**Returns**

`Optional[Dict]`

**Raises**

**ValueError** – If `id_1` is `None` and `id_2` is not `None`.

---

**Note:**

- If both `id_1` and `id_2` are `None`, the differences between the current state and the previous commit will be calculated. If you're at the head of the branch, this will show the uncommitted changes, if any.
  - If only `id_1` is provided, the differences between the current state and `id_1` will be calculated. If you're at the head of the branch, this will take into account the uncommitted changes, if any.
  - If only `id_2` is provided, a `ValueError` will be raised.
  - If both `id_1` and `id_2` are provided, the differences between `id_1` and `id_2` will be calculated.
- 

---

**Note:** A dictionary of the differences between the commits/branches is returned if `as_dict` is `True`. The dictionary will always have 2 keys, “dataset” and “tensors”. The values corresponding to these keys are detailed below:

- If `id_1` and `id_2` are `None`, both the keys will have a single list as their value. This list will contain a dictionary describing changes compared to the previous commit.
- If only `id_1` is provided, both keys will have a tuple of 2 lists as their value. The lists will contain dictionaries describing commitwise differences between commits. The 2 lists will range from current state and `id_1` to most recent common ancestor the commits respectively.
- If only `id_2` is provided, a `ValueError` will be raised.
- If both `id_1` and `id_2` are provided, both keys will have a tuple of 2 lists as their value. The lists will contain dictionaries describing commitwise differences between commits. The 2 lists will range from `id_1` and `id_2` to most recent common ancestor the commits respectively.

`None` is returned if `as_dict` is `False`.

---

**extend**(*samples: Dict[str, Any], skip\_ok: bool = False, append\_empty: bool = False, ignore\_errors: bool = False, progressbar: bool = False*)

Appends multiple rows of samples to multiple tensors at once. This method expects all tensors being

updated to be of the same length.

#### Parameters

- **samples** (*Dict[str, Any]*) – Dictionary with tensor names as keys and samples as values.
- **skip\_ok** (*bool*) – Skip tensors not in **samples** if set to True.
- **append\_empty** (*bool*) – Append empty samples to tensors not specified in sample if set to True. If True, **skip\_ok** is ignored.
- **ignore\_errors** (*bool*) – Skip samples that cause errors while extending, if set to True.
- **progressbar** (*bool*) – Displays a progress bar if set to True.

#### Raises

- **KeyError** – If any tensor in the dataset is not a key in **samples** and **skip\_ok** is False.
- **TensorDoesNotExistError** – If tensor in **samples** does not exist.
- **ValueError** – If all tensors being updated are not of the same length.
- **NotImplementedError** – If an error occurs while writing tiles.
- **SampleExtendError** – If the extend failed while appending a sample.
- **Exception** – Error while attempting to rollback appends.

**filter**(*function: Union[Callable, str], num\_workers: int = 0, scheduler: str = 'threaded', progressbar: bool = True, save\_result: bool = False, result\_path: Optional[str] = None, result\_ds\_args: Optional[dict] = None*)

Filters the dataset in accordance of filter function `f(x: sample) -> bool`

#### Parameters

- **function** (*Callable, str*) – Filter function that takes sample as argument and returns True / False if sample should be included in result. Also supports simplified expression evaluations. See `deeplake.core.query.query.DatasetQuery` for more details.
- **num\_workers** (*int*) – Level of parallelization of filter evaluations. 0 indicates in-place for-loop evaluation, multiprocessing is used otherwise.
- **scheduler** (*str*) – Scheduler to use for multiprocessing evaluation. “threaded” is default.
- **progressbar** (*bool*) – Display progress bar while filtering. True is default.
- **save\_result** (*bool*) – If True, result of the filter will be saved to a dataset asynchronously.
- **result\_path** (*Optional, str*) – Path to save the filter result. Only applicable if **save\_result** is True.
- **result\_ds\_args** (*Optional, dict*) – Additional args for result dataset. Only applicable if **save\_result** is True.

#### Returns

View of Dataset with elements that satisfy filter function.

### Example

Return dataset view where all the samples have label equals to 2:

```
>>> dataset.filter(lambda sample: sample.labels.numpy() == 2)
```

Append one dataset onto another (only works if their structure is identical):

```
>>> @deeplake.compute
>>> def dataset_append(sample_in, sample_out):
>>>     sample_out.append(sample_in.tensors)
>>>     return sample_out
>>>
>>> dataset_append().eval(
>>>     ds_in,
>>>     ds_out,
>>>     num_workers = 2
>>> )
```

#### **fix\_vc()**

Rebuilds version control info. To be used when the version control info is corrupted.

#### **flush()**

Necessary operation after writes if caches are being used. Writes all the dirty data from the cache layers (if any) to the underlying storage. Here dirty data corresponds to data that has been changed/assigned and but hasn't yet been sent to the underlying storage.

#### **get\_commit\_details(commit\_id)** → Dict

Get details of a particular commit.

##### **Parameters**

**commit\_id** (str) – commit id of the commit.

##### **Returns**

Dictionary of details with keys - commit, author, time, message.

##### **Return type**

Dict

##### **Raises**

**KeyError** – If given commit\_id is was not found in the dataset.

#### **get\_creds\_keys()** → Set[str]

Returns the set of creds keys added to the dataset. These are used to fetch external data in linked tensors

#### **get\_managed\_creds\_keys()** → List[str]

Returns the list of creds keys added to the dataset that are managed by Activeloop platform. These are used to fetch external data in linked tensors.

#### **get\_view(id: str)** → *ViewEntry*

Returns the dataset view corresponding to id.

## Examples

```
>>> # save view
>>> ds[:100].save_view(id="first_100")
>>> # load view
>>> first_100 = ds.get_view("first_100").load()
>>> # 100
>>> print(len(first_100))
```

See `Dataset.save_view()` to learn more about saving views.

### Parameters

**id** (*str*) – id of required view.

### Returns

`ViewEntry`

### Raises

**KeyError** – If no such view exists.

**get\_views** (*commit\_id: Optional[str] = None*) → List[`ViewEntry`]

Returns list of views stored in this Dataset.

### Parameters

**commit\_id** (*str, optional*) –

- Commit from which views should be returned.
- If not specified, views from all commits are returned.

### Returns

List of `ViewEntry` instances.

### Return type

List[`ViewEntry`]

**property groups:** Dict[*str*, `Dataset`]

All sub groups in this group

**property has\_head\_changes**

Returns True if currently at head node and uncommitted changes are present.

**property info**

Returns the information about the dataset.

**property is\_head\_node**

Returns True if the current commit is the head node of the branch and False otherwise.

**property is\_view:** bool

Returns True if this dataset is a view and False otherwise.

**load\_view** (*id: str, optimize: Optional[bool] = False, tensors: Optional[List[str]] = None, num\_workers: int = 0, scheduler: str = 'threaded', progressbar: Optional[bool] = True*)

Loads the view and returns the `Dataset` by id. Equivalent to `ds.get_view(id).load()`.

### Parameters

- **id** (*str*) – id of the view to be loaded.
- **optimize** (*bool*) – If True, the dataset view is optimized by copying and rechunking the required data before loading. This is necessary to achieve fast streaming speeds when training models using the dataset view. The optimization process will take some time, depending on the size of the data.

- **tensors** (*Optional*, *List[str]*) – Tensors to be copied if *optimize* is *True*. By default all tensors are copied.
- **num\_workers** (*int*) – Number of workers to be used for the optimization process. Only applicable if *optimize=True*. Defaults to 0.
- **scheduler** (*str*) – The scheduler to be used for optimization. Supported values include: ‘serial’, ‘threaded’, ‘processed’ and ‘ray’. Only applicable if *optimize=True*. Defaults to ‘threaded’.
- **progressbar** (*bool*) – Whether to use progressbar for optimization. Only applicable if *optimize=True*. Defaults to *True*.

**Returns**

The loaded view.

**Return type**

*Dataset*

**Raises**

**KeyError** – if view with given id does not exist.

**log()**

Displays the details of all the past commits.

**property max\_len**

Return the maximum length of the tensor.

**property max\_view**

Returns a view of the dataset in which shorter tensors are padded with *None* s to have the same length as the longest tensor.

**Example**

Creating a dataset with 5 images and 4 labels. `ds.max_view` will return a view with `labels` tensor padded to have 5 samples.

```
>>> import deeplake
>>> ds = deeplake.dataset("../test/test_ds", overwrite=True)
>>> ds.create_tensor("images", htype="link[image]", sample_compression="jpg")
>>> ds.create_tensor("labels", htype="class_label")
>>> ds.images.extend([deeplake.link("https://picsum.photos/20/20") for _ in
→ range(5)])
>>> ds.labels.extend([0, 1, 2, 1])
>>> len(ds.images)
5
>>> len(ds.labels)
4
>>> for i, sample in enumerate(ds.max_view):
...     print(sample["images"].shape, sample["labels"].numpy())
...
(20, 20, 3) [0]
(20, 20, 3) [1]
(20, 20, 3) [2]
(20, 20, 3) [1]
(20, 20, 3) [None]
```

**merge**(*target\_id: str*, *conflict\_resolution: Optional[str] = None*, *delete\_removed\_tensors: bool = False*, *force: bool = False*)

Merges the `target_id` into the current dataset.

**Parameters**

- **target\_id** (*str*) – The `commit_id` or branch to merge.
- **conflict\_resolution** (*str*, *Optional*) –
  - The strategy to use to resolve merge conflicts.
  - Conflicts are scenarios where both the current dataset and the target id have made changes to the same sample/s since their common ancestor.
  - **Must be one of the following**
    - \* None - this is the default value, will raise an exception if there are conflicts.
    - \* "ours" - during conflicts, values from the current dataset will be used.
    - \* "theirs" - during conflicts, values from target id will be used.
- **delete\_removed\_tensors** (*bool*) – If True, deleted tensors will be deleted from the dataset.
- **force** (*bool*) –
  - Forces merge.
  - **force=True will have these effects in the following cases of merge conflicts:**
    - \* If tensor is renamed on target but is missing from HEAD, renamed tensor will be registered as a new tensor on current branch.
    - \* If tensor is renamed on both target and current branch, tensor on target will be registered as a new tensor on current branch.
    - \* If tensor is renamed on target and a new tensor of the new name was created on the current branch, they will be merged.

**Raises**

- **Exception** – if dataset is a filtered view.
- **ValueError** – if the conflict resolution strategy is not one of the None, "ours", or "theirs".

**property meta: DatasetMeta**

Returns the metadata of the dataset.

**property min\_len**

Return the minimum length of the tensor.

**property min\_view**

Returns a view of the dataset in which all tensors are sliced to have the same length as the shortest tensor.



## Example

Creating a dataset with 5 images and 4 labels. `ds.min_view` will return a view in which tensors are sliced to have 4 samples.

```
>>> import deeplake
>>> ds = deeplake.dataset("../test/test_ds", overwrite=True)
>>> ds.create_tensor("images", htype="link[image]", sample_compression="jpg")
>>> ds.create_tensor("labels", htype="class_label")
>>> ds.images.extend([deeplake.link("https://picsum.photos/20/20") for _ in
    ↪range(5)])
>>> ds.labels.extend([0, 1, 2, 1])
>>> len(ds.images)
5
>>> len(ds.labels)
4
>>> for i, sample in enumerate(ds.max_view):
...     print(sample["images"].shape, sample["labels"].numpy())
...
(20, 20, 3) [0]
(20, 20, 3) [1]
(20, 20, 3) [2]
(20, 20, 3) [1]
```

### property `no_view_dataset`

Returns the same dataset without slicing.

### property `num_samples: int`

Returns the length of the smallest tensor. Ignores any applied indexing and returns the total length.

### property `parent`

Returns the parent of this group. Returns None if this is the root dataset.

### property `pending_commit_id: str`

The `commit_id` of the next commit that will be made to the dataset. If you're not at the head of the current branch, this will be the same as the `commit_id`.

### `pop(index: Optional[int] = None)`

Removes a sample from all the tensors of the dataset. For any tensor if the `index >= len(tensor)`, the sample won't be popped from it.

#### Parameters

**index** (*int*, *Optional*) – The index of the sample to be removed. If it is None, the index becomes the length of the longest tensor - 1.

#### Raises

- **ValueError** – If duplicate indices are provided.
- **IndexError** – If the index is out of range.

### `populate_creds(creds_key: str, creds: Optional[dict] = None, from_environment: bool = False)`

Populates the `creds` key added in `add_creds_key` with the given `creds`. These `creds` are used to fetch the external data. This needs to be done everytime the dataset is reloaded for datasets that contain links to external data.

## Examples

```
>>> # create/load a dataset
>>> ds = deeplake.dataset("path/to/dataset")
>>> # add a new creds key
>>> ds.add_creds_key("my_s3_key")
>>> # populate the creds
>>> ds.populate_creds("my_s3_key", {"aws_access_key_id": "my_access_key", "aws_
↪secret_access_key": "my_secret_key"})
>>> # or
>>> ds.populate_creds("my_s3_key", from_environment=True)
```

**pytorch**(*transform*: Optional[Callable] = None, *tensors*: Optional[Sequence[str]] = None, *num\_workers*: int = 1, *batch\_size*: int = 1, *drop\_last*: bool = False, *collate\_fn*: Optional[Callable] = None, *pin\_memory*: bool = False, *shuffle*: bool = False, *buffer\_size*: int = 2048, *use\_local\_cache*: bool = False, *progressbar*: bool = False, *return\_index*: bool = True, *pad\_tensors*: bool = False, *transform\_kwargs*: Optional[Dict[str, Any]] = None, *decode\_method*: Optional[Dict[str, str]] = None, *cache\_size*: int = 32000000, \*args, \*\*kwargs)

Converts the dataset into a pytorch Dataloader.

### Parameters

- **\*args** – Additional args to be passed to torch\_dataset
- **\*\*kwargs** – Additional kwargs to be passed to torch\_dataset
- **transform** (Callable, Optional) – Transformation function to be applied to each sample.
- **tensors** (List, Optional) – Optionally provide a list of tensor names in the ordering that your training script expects. For example, if you have a dataset that has “image” and “label” tensors, if `tensors=["image", "label"]`, your training script should expect each batch will be provided as a tuple of (image, label).
- **num\_workers** (int) – The number of workers to use for fetching data in parallel.
- **batch\_size** (int) – Number of samples per batch to load. Default value is 1.
- **drop\_last** (bool) – Set to True to drop the last incomplete batch, if the dataset size is not divisible by the batch size. If False and the size of dataset is not divisible by the batch size, then the last batch will be smaller. Default value is False. Read torch.utils.data.DataLoader docs for more details.
- **collate\_fn** (Callable, Optional) – merges a list of samples to form a mini-batch of Tensor(s). Used when using batched loading from a map-style dataset. Read torch.utils.data.DataLoader docs for more details.
- **pin\_memory** (bool) – If True, the data loader will copy Tensors into CUDA pinned memory before returning them. Default value is False. Read torch.utils.data.DataLoader docs for more details.
- **shuffle** (bool) – If True, the data loader will shuffle the data indices. Default value is False. Details about how Deep Lake shuffles data can be found at [Shuffling in ds.pytorch\(\)](#)
- **buffer\_size** (int) – The size of the buffer used to shuffle the data in MBs. Defaults to 2048 MB. Increasing the buffer\_size will increase the extent of shuffling.

- **use\_local\_cache** (*bool*) – If True, the data loader will use a local cache to store data. The default cache location is `~/.activeloop/cache`, but it can be changed by setting the `LOCAL_CACHE_PREFIX` environment variable. This is useful when the dataset can fit on the machine and we don't want to fetch the data multiple times for each iteration. Default value is `False`
- **progressbar** (*bool*) – If True, `tqdm` will be wrapped around the returned dataloader. Default value is `True`.
- **return\_index** (*bool*) – If True, the returned dataloader will have a key “index” that contains the index of the sample(s) in the original dataset. Default value is `True`.
- **pad\_tensors** (*bool*) – If True, shorter tensors will be padded to the length of the longest tensor. Default value is `False`.
- **transform\_kwargs** (*optional, Dict[str, Any]*) – Additional kwargs to be passed to `transform`.
- **decode\_method** (*Dict[str, str], Optional*) – The method for decoding the Deep Lake tensor data, the result of which is passed to the transform. Decoding occurs outside of the transform so that it can be performed in parallel and as rapidly as possible as per Deep Lake optimizations.

– Supported decode methods are:

**'numpy'**

Default behaviour. Returns samples as numpy arrays, the same as `ds.tensor[i].numpy()`

**'tobytes'**

Returns raw bytes of the samples the same as `ds.tensor[i].tobytes()`

**'data'**

Returns a dictionary with keys, values depending on htype, the same as `ds.tensor[i].data()`

**'pil'**

Returns samples as PIL images. Especially useful when transformation use torchvision transforms, that require PIL images as input. Only supported for tensors with `sample_compression='jpeg'` or `'png'`.

- **cache\_size** (*int*) – The size of the cache per tensor in MBs. Defaults to `max(maximum chunk size of tensor, 32 MB)`.

**Returns**

A `torch.utils.data.DataLoader` object.

**Raises**

***EmptyTensorError*** – If one or more tensors being passed to pytorch are empty.

---

**Note:** Pytorch does not support `uint16`, `uint32`, `uint64` dtypes. These are implicitly type casted to `int32`, `int64` and `int64` respectively. This spins up it's own workers to fetch data.

---

**query**(*query\_string: str, runtime: Optional[Dict] = None, return\_data: bool = False*)

Returns a sliced *Dataset* with given query results.

It allows to run SQL like queries on dataset and extract results. See supported keywords and the Tensor Query Language documentation [here](#).

#### Parameters

- **query\_string** (*str*) – An SQL string adjusted with new functionalities to run on the given *Dataset* object
- **runtime** (*Optional[Dict]*) – Runtime parameters for query execution. Supported keys: {"tensor\_db": True or False}.
- **return\_data** (*bool*) – Defaults to False. Whether to return raw data along with the view.

#### Raises

**ValueError** – if `return_data` is True and `runtime` is not {"tensor\_db": true}

#### Returns

A *Dataset* object.

#### Return type

*Dataset*

## Examples

Query from dataset all the samples with lables other than 5

```
>>> import deeplake
>>> ds = deeplake.load('hub://activeloop/fashion-mnist-train')
>>> query_ds = ds.query("select * where labels != 5")
```

Query from dataset first appeard 1000 samples where the categories is car and 1000 samples where the categories is motorcycle

```
>>> ds_train = deeplake.load('hub://activeloop/coco-train')
>>> query_ds_train = ds_train.query("(select * where contains(categories, 'car'
↪) limit 1000) union (select * where contains(categories, 'motorcycle')
↪ limit 1000)")
```

**random\_split**(*lengths: Sequence[Union[int, float]]*)

Splits the dataset into non-overlapping *Dataset* objects of given lengths. If a list of fractions that sum up to 1 is given, the lengths will be computed automatically as `floor(frac * len(dataset))` for each fraction provided.

After computing the lengths, if there are any remainders, 1 count will be distributed in round-robin fashion to the lengths until there are no remainders left.

## Example

```
>>> import deeplake
>>> ds = deeplake.dataset("../test/test_ds", overwrite=True)
>>> ds.create_tensor("labels", htype="class_label")
>>> ds.labels.extend([0, 1, 2, 1, 3])
>>> len(ds)
5
>>> train_ds, val_ds = ds.random_split([0.8, 0.2])
>>> len(train_ds)
4
```

(continues on next page)

(continued from previous page)

```

>>> len(val_ds)
1
>>> train_ds, val_ds = ds.random_split([3, 2])
>>> len(train_ds)
3
>>> len(val_ds)
2
>> train_loader = train_ds.pytorch(batch_size=2, shuffle=True)
>> val_loader = val_ds.pytorch(batch_size=2, shuffle=False)

```

**Parameters**

**lengths** (*Sequence[Union[int, float]]*) – lengths or fractions of splits to be produced.

**Returns**

a tuple of datasets of the given lengths.

**Return type**

Tuple[*Dataset*, ...]

**Raises**

- **ValueError** – If the sum of the lengths is not equal to the length of the dataset.
- **ValueError** – If the dataset has variable length tensors.
- **ValueError** – If lengths are floats and one or more of them are not between 0 and 1.

**property read\_only**

Returns True if dataset is in read-only mode and False otherwise.

**rechunk**(*tensors: Optional[Union[str, List[str]]] = None, num\_workers: int = 0, scheduler: str = 'threaded', progressbar: bool = True*)

Rewrites the underlying chunks to make their sizes optimal. This is usually needed in cases where a lot of updates have been made to the data.

**Parameters**

- **tensors** (*str, List[str], Optional*) – Name/names of the tensors to rechunk. If None, all tensors in the dataset are rechunked.
- **num\_workers** (*int*) – The number of workers to use for rechunking. Defaults to 0. When set to 0, it will always use serial processing, irrespective of the scheduler.
- **scheduler** (*str*) – The scheduler to be used for rechunking. Supported values include: 'serial', 'threaded', 'processed' and 'ray'. Defaults to 'threaded'.
- **progressbar** (*bool*) – Displays a progress bar If True (default).

**rename**(*path: Union[str, Path]*)

Renames the dataset to *path*.

### Example

```
>>> ds = deeplake.load("hub://username/dataset")
>>> ds.rename("hub://username/renamed_dataset")
```

#### Parameters

**path** (*str*, *pathlib.Path*) – New path to the dataset.

#### Raises

**RenameError** – If path points to a different directory.

**rename\_group**(*name: str*, *new\_name: str*) → None

Renames group with name *name* to *new\_name*

#### Parameters

- **name** (*str*) – Name of group to be renamed.

- **new\_name** (*str*) – New name of group.

#### Raises

- **TensorGroupDoesNotExistError** – If tensor group of name *name* does not exist in the dataset.

- **TensorAlreadyExistsError** – Duplicate tensors are not allowed.

- **TensorGroupAlreadyExistsError** – Duplicate tensor groups are not allowed.

- **InvalidTensorGroupNameError** – If *name* is in dataset attributes.

- **RenameError** – If *new\_name* points to a group different from *name*.

**rename\_tensor**(*name: str*, *new\_name: str*) → *Tensor*

Renames tensor with name *name* to *new\_name*

#### Parameters

- **name** (*str*) – Name of tensor to be renamed.

- **new\_name** (*str*) – New name of tensor.

#### Returns

Renamed tensor.

#### Return type

*Tensor*

#### Raises

- **TensorDoesNotExistError** – If tensor of name *name* does not exist in the dataset.

- **TensorAlreadyExistsError** – Duplicate tensors are not allowed.

- **TensorGroupAlreadyExistsError** – Duplicate tensor groups are not allowed.

- **InvalidTensorNameError** – If *new\_name* is in dataset attributes.

- **RenameError** – If *new\_name* points to a group different from *name*.

**reset**(*force: bool = False*)

Resets the uncommitted changes present in the branch.

---

**Note:** The uncommitted data is deleted from underlying storage, this is not a reversible operation.

---

**property root**

Returns the root dataset of a group.

**sample\_by**(weights: Union[str, list, tuple], replace: Optional[bool] = True, size: Optional[int] = None)

Returns a sliced [Dataset](#) with given weighted sampler applied.

**Parameters**

- **weights** – (Union[str, list, tuple]): If it's string then `sql` will be run to calculate the weights based on the expression. list and tuple will be treated as the list of the weights per sample.
- **replace** – Optional[bool] If true the samples can be repeated in the result view. Defaults to True
- **size** – Optional[int] The length of the result view. Defaults to length of the dataset.

**Returns**

A `deeplake.Dataset` object.

**Return type**

[Dataset](#)

**Examples**

Sample the dataset with `labels == 5` twice more than `labels == 6`

```
>>> from deeplake.experimental import query
>>> ds = deeplake.load('hub://activeloop/fashion-mnist-train')
>>> sampled_ds = ds.sample_by("max_weight(labels == 5: 10, labels == 6: 5)")
```

Sample the dataset treating `labels` tensor as weights.

```
>>> import deeplake
>>> ds = deeplake.load('hub://activeloop/fashion-mnist-train')
>>> sampled_ds = ds.sample_by("max_weight(labels == 5: 10, labels == 6: 5)")
```

Sample the dataset with the given weights;

```
>>> ds = deeplake.load('hub://activeloop/coco-train')
>>> weights = list()
>>> for i in range(len(ds)):
...     weights.append(i % 5)
...
>>> sampled_ds = ds.sample_by(weights, replace=False)
```

**property sample\_indices**

Returns all the indices pointed to by this dataset view.

**save\_view**(message: Optional[str] = None, path: Optional[Union[str, Path]] = None, id: Optional[str] = None, optimize: bool = False, tensors: Optional[List[str]] = None, num\_workers: int = 0, scheduler: str = 'threaded', verbose: bool = True, ignore\_errors: bool = False, \*\*ds\_args) → str

Saves a dataset view as a virtual dataset (VDS)

## Examples

```
>>> # Save to specified path
>>> vds_path = ds[:10].save_view(path="views/first_10", id="first_10")
>>> vds_path
views/first_10
```

```
>>> # Path unspecified
>>> vds_path = ds[:100].save_view(id="first_100", message="first 100 samples")
>>> # vds_path = path/to/dataset
```

```
>>> # Random id
>>> vds_path = ds[:100].save_view()
>>> # vds_path = path/to/dataset/.queries/
→ 92f41922ed0471ec2d27690b7351fc96bea060e6c5ee22b14f7ffa5f291aa068
```

See `Dataset.get_view()` to learn how to load views by id. These virtual datasets can also be loaded from their path like normal datasets.

### Parameters

- **message** (*Optional, str*) – Custom user message.
- **path** (*Optional, str, pathlib.Path*) –
  - The VDS will be saved as a standalone dataset at the specified path.
  - If not specified, the VDS is saved under `.queries` subdirectory of the source dataset's storage.
- **id** (*Optional, str*) – Unique id for this view. Random id will be generated if not specified.
- **optimize** (*bool*) –
  - If `True`, the dataset view will be optimized by copying and rechunking the required data. This is necessary to achieve fast streaming speeds when training models using the dataset view. The optimization process will take some time, depending on the size of the data.
  - You can also choose to optimize the saved view later by calling its `ViewEntry.optimize()` method.
- **tensors** (*List, optional*) – Names of tensors (and groups) to be copied. If not specified all tensors are copied.
- **num\_workers** (*int*) – Number of workers to be used for optimization process. Applicable only if `optimize=True`. Defaults to 0.
- **scheduler** (*str*) – The scheduler to be used for optimization. Supported values include: 'serial', 'threaded', 'processed' and 'ray'. Only applicable if `optimize=True`. Defaults to 'threaded'.
- **verbose** (*bool*) – If `True`, logs will be printed. Defaults to `True`.
- **ignore\_errors** (*bool*) – Skip samples that cause errors while saving views. Only applicable if `optimize=True`. Defaults to `False`.
- **ds\_args** (*dict*) – Additional args for creating VDS when path is specified. (See documentation for `deeplake.dataset()`)



**Returns**

Path to the saved VDS.

**Return type**

str

**Raises**

- **`ReadOnlyModeError`** – When attempting to save a view inplace and the user doesn't have write access.
- **`DatasetViewSavingError`** – If HEAD node has uncommitted changes.
- **`TypeError`** – If `id` is not of type `str`.

---

**Note:** Specifying `path` makes the view external. External views cannot be accessed using the parent dataset's `Dataset.get_view()`, `Dataset.load_view()`, `Dataset.delete_view()` methods. They have to be loaded using `deeplake.load()`.

---

**`set_token(new_token: str)`**

Method to set a new token

**`size_approx()`**

Estimates the size in bytes of the dataset. Includes only content, so will generally return an under-estimate.

**`summary(force: bool = False)`**

Prints a summary of the dataset.

**Parameters**

**`force (bool)`** – Dataset views with more than 10000 samples might take a long time to summarize. If `force=True`, the summary will be printed regardless. An error will be raised otherwise.

**Raises**

**`ValueError`** – If the dataset view might take a long time to summarize and `force=False`

**`tensorflow(tensors: Optional[Sequence[str]] = None, tobytes: Union[bool, Sequence[str]] = False, fetch_chunks: bool = True)`**

Converts the dataset into a tensorflow compatible format.

See [https://www.tensorflow.org/api\\_docs/python/tf/data/Dataset](https://www.tensorflow.org/api_docs/python/tf/data/Dataset)

**Parameters**

- **`tensors (List, Optional)`** – Optionally provide a list of tensor names in the ordering that your training script expects. For example, if you have a dataset that has “image” and “label” tensors, if `tensors=["image", "label"]`, your training script should expect each batch will be provided as a tuple of (image, label).
- **`tobytes (bool)`** – If `True`, samples will not be decompressed and their raw bytes will be returned instead of numpy arrays. Can also be a list of tensors, in which case those tensors alone will not be decompressed.
- **`fetch_chunks`** – See `fetch_chunks` argument in `deeplake.core.tensor.Tensor.numpy()`

**Returns**

`tf.data.Dataset` object that can be used for tensorflow training.

**property tensors:** `Dict[str, Tensor]`

All tensors belonging to this group, including those within sub groups. Always returns the sliced tensors.

**property token**

Get attached token of the dataset

**update**(*sample: Dict[str, Any]*)

Update existing samples in the dataset with new values.

**Examples**

```
>>> ds[0].update({"images": deeplake.read("new_image.png"), "labels": 1})
```

```
>>> new_images = [deeplake.read(f"new_image_{i}.png") for i in range(3)]
>>> ds[:3].update({"images": new_images, "labels": [1, 2, 3]})
```

**Parameters**

**sample** (*dict*) – Dictionary with tensor names as keys and samples as values.

**Raises**

- **ValueError** – If partial update of a sample is attempted.
- **Exception** – Error while attempting to rollback updates.

**update\_creds\_key**(*creds\_key: str, new\_creds\_key: Optional[str] = None, managed: Optional[bool] = None*)

Updates the name and/or management status of a creds key.

**Parameters**

- **creds\_key** (*str*) – The key whose name and/or management status is to be changed.
- **new\_creds\_key** (*str, optional*) – The new key to replace the old key. If not provided, the old key will be used.
- **managed** (*bool*) – The target management status. If **True**, the creds corresponding to the key will be fetched from activeloop platform.

**Raises**

- **ValueError** – If the dataset is not connected to activeloop platform.
- **ValueError** – If both **new\_creds\_key** and **managed** are **None**.
- **KeyError** – If the creds key is not present in the dataset.

**Examples**

```
>>> # create/load a dataset
>>> ds = deeplake.dataset("path/to/dataset")
>>> # add a new creds key
>>> ds.add_creds_key("my_s3_key")
>>> # Populate the name added with creds dictionary
>>> # These creds are only present temporarily and will have to be repopulated
    ↳ on every reload
>>> ds.populate_creds("my_s3_key", {})
>>> # Rename the key and change the management status of the key to True.
    ↳ Before doing this, ensure that the creds have been created on activeloop
    ↳ platform
```

(continues on next page)

(continued from previous page)

```
>>> # Now, this key will no longer use the credentials populated in the
→previous step but will instead fetch them from activeloop platform
>>> # These creds don't have to be populated again on every reload and will be
→fetched every time the dataset is loaded
>>> ds.update_creds_key("my_s3_key", "my_managed_key", True)
```

**visualize**(width: Optional[Union[int, str]] = None, height: Optional[Union[int, str]] = None)

Visualizes the dataset in the Jupyter notebook.

#### Parameters

- **width** – Union[int, str, None] Optional width of the visualizer canvas.
- **height** – Union[int, str, None] Optional height of the visualizer canvas.

#### Raises

**Exception** – If the dataset is not a Deep Lake cloud dataset and the visualization is attempted in colab.

## 19.2 DeepLakeCloudDataset

**class** deeplake.core.dataset.DeepLakeCloudDataset

Bases: [Dataset](#)

Subclass of [Dataset](#). Deep Lake cloud datasets are those datasets which are stored in or connected to Activeloop servers, their paths look like: `hub://username/dataset_name`.

**add\_creds\_key**(creds\_key: str, managed: bool = False)

Adds a new creds key to the dataset. These keys are used for tensors that are linked to external data.

#### Examples

```
>>> # create/load a dataset
>>> ds = deeplake.dataset("hub://username/dataset")
>>> # add a new creds key
>>> ds.add_creds_key("my_s3_key")
```

#### Parameters

- **creds\_key** (str) – The key to be added.
- **managed** (bool) – If True, the creds corresponding to the key will be fetched from activeloop platform. Note, this is only applicable for datasets that are connected to activeloop platform. Defaults to False.

#### property client

Returns the client of the dataset.

**connect**(\*args, \*\*kwargs)

Connect a Deep Lake cloud dataset through a deeplake path.

## Examples

```
>>> # create/load an s3 dataset
>>> s3_ds = deeplake.dataset("s3://bucket/dataset")
>>> ds = s3_ds.connect(dest_path="hub://my_org/dataset", creds_key="my_managed_
→ credentials_key", token="my_activeloop_token")
>>> # or
>>> ds = s3_ds.connect(org_id="my_org", creds_key="my_managed_credentials_key",
→ token="my_activeloop_token")
```

### Parameters

- **creds\_key** (*str*) – The managed credentials to be used for accessing the source path.
- **dest\_path** (*str, optional*) – The full path to where the connected Deep Lake dataset will reside. Can be: a Deep Lake path like `hub://organization/dataset`
- **org\_id** (*str, optional*) – The organization to where the connected Deep Lake dataset will be added.
- **ds\_name** (*str, optional*) – The name of the connected Deep Lake dataset. Will be inferred from `dest_path` or `src_path` if not provided.
- **token** (*str, optional*) – Activeloop token used to fetch the managed credentials.

### Raises

- **InvalidSourcePathError** – If the dataset's path is not a valid s3, gcs or azure path.
- **InvalidDestinationPathError** – If `dest_path`, or `org_id` and `ds_name` do not form a valid Deep Lake path.
- **TokenPermissionError** – If the user does not have permission to create a dataset in the specified organization.

**delete**(*large\_ok=False*)

Deletes the entire dataset from the cache layers (if any) and the underlying storage. This is an **IRREVERSIBLE** operation. Data once deleted can not be recovered.

### Parameters

**large\_ok** (*bool*) – Delete datasets larger than 1 GB. Defaults to False.

### Raises

- **DatasetTooLargeToDelete** – If the dataset is larger than 1 GB and `large_ok` is False.
- **DatasetHandlerError** – If the dataset is marked as `allow_delete=False`.

**get\_managed\_creds\_keys**() → Set[str]

Returns the set of creds keys added to the dataset that are managed by Activeloop platform. These are used to fetch external data in linked tensors.

**property is\_actually\_cloud:** bool

Datasets that are connected to Deep Lake cloud can still technically be stored anywhere. If a dataset is in Deep Lake cloud but stored without `hub://` prefix, it should only be used for testing.

**rename**(*path*)

Renames the dataset to *path*.

### Example

```
>>> ds = deeplake.load("hub://username/dataset")
>>> ds.rename("hub://username/renamed_dataset")
```

#### Parameters

**path** (*str*, *pathlib.Path*) – New path to the dataset.

#### Raises

**RenameError** – If path points to a different directory.

### property token

Get attached token of the dataset

**update\_creds\_key** (*creds\_key: str*, *new\_creds\_key: Optional[str] = None*, *managed: Optional[bool] = None*)

Updates the name and/or management status of a creds key.

#### Parameters

- **creds\_key** (*str*) – The key whose management status is to be changed.
- **new\_creds\_key** (*str*, *optional*) – The new key to replace the old key. If not provided, the old key will be used.
- **managed** (*bool*) – The target management status. If **True**, the creds corresponding to the key will be fetched from activeloop platform.

#### Raises

- **ValueError** – If the dataset is not connected to activeloop platform.
- **ValueError** – If both **new\_creds\_key** and **managed** are **None**.
- **KeyError** – If the creds key is not present in the dataset.
- **Exception** – All other errors such as during population of managed creds.

### Examples

```
>>> # create/load a dataset
>>> ds = deeplake.dataset("path/to/dataset")
>>> # add a new creds key
>>> ds.add_creds_key("my_s3_key")
>>> # Populate the name added with creds dictionary
>>> # These creds are only present temporarily and will have to be repopulated
    ↳ on every reload
>>> ds.populate_creds("my_s3_key", {})
>>> # Rename the key and change the management status of the key to True.
    ↳ Before doing this, ensure that the creds have been created on activeloop
    ↳ platform
>>> # Now, this key will no longer use the credentials populated in the
    ↳ previous step but will instead fetch them from activeloop platform
>>> # These creds don't have to be populated again on every reload and will be
    ↳ fetched every time the dataset is loaded
>>> ds.update_creds_key("my_s3_key", "my_managed_key", True)
```

**visualize** (*width: Optional[Union[int, str]] = None*, *height: Optional[Union[int, str]] = None*)

Visualizes the dataset in the Jupyter notebook.

**Parameters**

- **width** – Union[int, str, None] Optional width of the visualizer canvas.
- **height** – Union[int, str, None] Optional height of the visualizer canvas.

**Raises**

**Exception** – If the dataset is not a Deep Lake cloud dataset and the visualization is attempted in colab.

## 19.3 ViewEntry

**class** deeplake.core.dataset.**ViewEntry**

Represents a view saved inside a dataset.

**delete()**

Deletes the view.

**property id: str**

Returns id of the view.

**load(verbose=True)**

Loads the view and returns the *Dataset*.

**Parameters**

**verbose** (*bool*) – If True, logs will be printed. Defaults to True.

**Returns**

Loaded dataset view.

**Return type**

*Dataset*

**property message: str**

Returns the message with which the view was saved.

**optimize**(*tensors: Optional[List[str]] = None, unlink=True, num\_workers=0, scheduler='threaded', progressbar=True*)

Optimizes the dataset view by copying and rechunking the required data. This is necessary to achieve fast streaming speeds when training models using the dataset view. The optimization process will take some time, depending on the size of the data.

### Example

```
>>> # save view
>>> ds[:10].save_view(id="first_10")
>>> # optimize view
>>> ds.get_view("first_10").optimize()
>>> # load optimized view
>>> ds.load_view("first_10")
```

**Parameters**

- **tensors** (*List[str]*) – Tensors required in the optimized view. By default all tensors are copied.
- **unlink** (*bool*) –

- If `True`, this unlinks linked tensors (if any) by copying data from the links to the view.
- This does not apply to linked videos. Set `deeplake.constants._UNLINK_VIDEOS` to `True` to change this behavior.
- **num\_workers** (*int*) – Number of workers to be used for the optimization process. Defaults to 0.
- **scheduler** (*str*) – The scheduler to be used for optimization. Supported values include: ‘serial’, ‘threaded’, ‘processed’ and ‘ray’. Only applicable if `optimize=True`. Defaults to ‘threaded’.
- **progressbar** (*bool*) – Whether to display a progressbar.

**Returns**

`ViewEntry`

**Raises**

**Exception** – When query view cannot be optimized.





## DEEPLAKE.CORE.TENSOR

### 20.1 Tensor

**class** deeplake.core.tensor.**Tensor**

**\_\_len\_\_()**

Returns the length of the primary axis of the tensor. Accounts for indexing into the tensor object.

#### Examples

```
>>> len(tensor)
0
>>> tensor.extend(np.zeros((100, 10, 10)))
>>> len(tensor)
100
>>> len(tensor[5:10])
5
```

#### Returns

The current length of this tensor.

#### Return type

int

**\_\_setitem\_\_**(*item: Union[int, slice], value: Any*)

Update samples with new values.

#### Example

```
>>> tensor.append(np.zeros((10, 10)))
>>> tensor.shape
(1, 10, 10)
>>> tensor[0] = np.zeros((3, 3))
>>> tensor.shape
(1, 3, 3)
```

**\_check\_compatibility\_with\_htype**(*htype*)

Checks if the tensor is compatible with the given htype. Raises an error if not compatible.

**property \_config**

Returns a summary of the configuration of the tensor.

**\_linked\_sample()**

Returns the linked sample at the given index. This is only applicable for tensors of `link[]` htype and can only be used for exactly one sample.

```
>>> linked_sample = ds.abc[0]._linked_sample().path
'https://picsum.photos/200/300'
```

**\_pop(index: List[int])**

Removes elements at the given indices. `index` must be sorted in descending order.

**append(sample: Union[Sample, ndarray, int, float, bool, dict, list, str, integer, floating, bool\_])**

Appends a single sample to the end of the tensor. Can be an array, scalar value, or the return value from `deeplake.read()`, which can be used to load files. See examples down below.

**Examples**

Numpy input:

```
>>> len(tensor)
0
>>> tensor.append(np.zeros((28, 28, 1)))
>>> len(tensor)
1
```

File input:

```
>>> len(tensor)
0
>>> tensor.append(deeplake.read("path/to/file"))
>>> len(tensor)
1
```

**Parameters**

**sample** (*InputSample*) – The data to append to the tensor. *Sample* is generated by `deeplake.read()`. See the above examples.

**property base\_htype**

Base htype of the tensor.

**Example**

```
>>> ds.create_tensor("video_seq", htype="sequence[video]", sample_compression=
↳ "mp4")
>>> ds.video_seq.htype
sequence[video]
>>> ds.video_seq.base_htype
video
```

**clear()**

Deletes all samples from the tensor

**creds\_key()**

Return path data. Only applicable for linked tensors

**data**(*aslist: bool = False, fetch\_chunks: bool = False*) → Any

Returns data in the tensor in a format based on the tensor's base htype.

- **If tensor has text base htype**
  - Returns dict with dict["value"] = `Tensor.text()`
- **If tensor has json base htype**
  - Returns dict with dict["value"] = `Tensor.dict()`
- **If tensor has list base htype**
  - Returns dict with dict["value"] = `Tensor.list()`
- For video tensors, returns a dict with keys "frames", "timestamps" and "sample\_info":
  - Value of dict["frames"] will be same as `numpy()`.
  - Value of dict["timestamps"] will be same as `timestamps` corresponding to the frames.
  - Value of dict["sample\_info"] will be same as `sample_info`.
- For class\_label tensors, returns a dict with keys "value" and "text".
  - Value of dict["value"] will be same as `numpy()`.
  - Value of dict["text"] will be list of class labels as strings.
- For image or dicom tensors, returns dict with keys "value" and "sample\_info".
  - Value of dict["value"] will be same as `numpy()`.
  - Value of dict["sample\_info"] will be same as `sample_info`.
- For all else, returns dict with key "value" with value same as `numpy()`.

**dict**(*fetch\_chunks: bool = False*)

Return json data. Only applicable for tensors with 'json' base htype.

**property dtype: Optional[dtype]**

Dtype of the tensor.

**extend**(*samples: Union[ndarray, Sequence[Union[Sample, ndarray, int, float, bool, dict, list, str, integer, floating, bool\_]], Tensor], progressbar: bool = False, ignore\_errors: bool = False*)

Extends the end of the tensor by appending multiple elements from a sequence. Accepts a sequence, a single batched numpy array, or a sequence of `deeplake.read()` outputs, which can be used to load files. See examples down below.

## Example

Numpy input:

```
>>> len(tensor)
0
>>> tensor.extend(np.zeros((100, 28, 28, 1)))
>>> len(tensor)
100
```

File input:

```
>>> len(tensor)
0
>>> tensor.extend([
    deeplake.read("path/to/image1"),
    deeplake.read("path/to/image2"),
```

(continues on next page)

(continued from previous page)

```

    ])
>>> len(tensor)
2

```

**Parameters**

- **samples** (*np.ndarray*, *Sequence*, *Sequence[Sample]*) – The data to add to the tensor. The length should be equal to the number of samples to add.
- **progressbar** (*bool*) – Specifies whether a progressbar should be displayed while extending.
- **ignore\_errors** (*bool*) – Skip samples that cause errors while extending, if set to True.

**Raises**

*TensorDtypeMismatchError* – Dtype for array must be equal to or castable to this tensor's dtype.

**property hidden:** *bool*

Whether this tensor is a hidden tensor.

**property htype**

Htype of the tensor.

**property info:** *Info*

Returns the information about the tensor. User can set info of tensor.

**Returns**

Information about the tensor.

**Return type**

*Info*

**Example**

```

>>> # update info
>>> ds.images.info.update(large=True, gray=False)
>>> # get info
>>> ds.images.info
{'large': True, 'gray': False}

```

```

>>> ds.images.info = {"complete": True}
>>> ds.images.info
{'complete': True}

```

**invalidate\_libdeeplake\_dataset()**

Invalidates the libdeeplake dataset object.

**property is\_dynamic:** *bool*

Will return True if samples in this tensor have shapes that are unequal.

**property is\_link**

Whether this tensor is a link tensor.

**property is\_sequence**

Whether this tensor is a sequence tensor.

**list**(*fetch\_chunks: bool = False*)

Return list data. Only applicable for tensors with 'list' or 'tag' base htype.

**property meta**

Metadata of the tensor.

**modified\_samples**(*target\_id: Optional[str] = None, return\_indexes: Optional[bool] = False*)

Returns a slice of the tensor with only those elements that were modified/added. By default the modifications are calculated relative to the previous commit made, but this can be changed by providing a **target id**.

#### Parameters

- **target\_id**(*str, optional*) – The commit id or branch name to calculate the modifications relative to. Defaults to None.
- **return\_indexes**(*bool, optional*) – If True, returns the indexes of the modified elements. Defaults to False.

#### Returns

A new tensor with only the modified elements if **return\_indexes** is False. Tuple[**Tensor**, List[int]]: A new tensor with only the modified elements and the indexes of the modified elements if **return\_indexes** is True.

#### Return type

*Tensor*

#### Raises

**TensorModifiedError** – If a target id is passed which is not an ancestor of the current commit.

**property ndim: int**

Number of dimensions of the tensor.

**property num\_samples: int**

Returns the length of the primary axis of the tensor. Ignores any applied indexing and returns the total length.

**numpy**(*aslist=False, fetch\_chunks=False*) → Union[ndarray, List[ndarray]]

Computes the contents of the tensor in numpy format.

#### Parameters

- **aslist**(*bool*) – If True, a list of np.ndarrays will be returned. Helpful for dynamic tensors. If False, a single np.ndarray will be returned unless the samples are dynamically shaped, in which case an error is raised.
- **fetch\_chunks**(*bool*) – If True, full chunks will be retrieved from the storage, otherwise only required bytes will be retrieved. This will always be True even if specified as False in the following cases:
  - The tensor is **ChunkCompressed**.
  - The chunk which is being accessed has more than 128 samples.

#### Raises

- **DynamicTensorNumpyError** – If reading a dynamically-shaped array slice without **aslist=True**.
- **ValueError** – If the tensor is a link and the credentials are not populated.

#### Returns

A numpy array containing the data represented by this tensor.

---

**Note:** For tensors of htype `polygon`, `aslist` is always `True`.

---

**path**(*aslist: bool = True, fetch\_chunks: bool = False*)

Return path data. Only applicable for linked tensors.

**Parameters**

- **aslist** (*bool*) – Returns links in a list if `True`.
- **fetch\_chunks** (*bool*) – If `True`, full chunks will be retrieved from the storage, otherwise only required bytes will be retrieved.

**Returns**

A list or numpy array of links.

**Return type**

Union[np.ndarray, List]

**Raises**

**Exception** – If the tensor is not a linked tensor.

**play()**

Play video sample. Plays video in Jupyter notebook or plays in web browser. Video is streamed directly from storage. This method will fail for incompatible htypes.

**Example**

```
>>> ds = deeplake.load("./test/my_video_ds")
>>> # play second sample
>>> ds.videos[2].play()
```

---

**Note:** Video streaming is not yet supported on colab.

---

**pop**(*index: Optional[Union[int, List[int]]] = None*)

Removes element(s) at the given index / indices.

**property sample\_indices**

Returns all the indices pointed to by this tensor in the dataset view.

**property sample\_info: Union[Dict, List[Dict]]**

Returns info about particular samples in a tensor. Returns dict in case of single sample, otherwise list of dicts. Data in returned dict would depend on the tensor's htype and the sample itself.

**Example**

```
>>> ds.videos[0].sample_info
{'duration': 400400, 'fps': 29.97002997002997, 'timebase': 3.3333333333333335e-
→ 05, 'shape': [400, 360, 640, 3], 'format': 'mp4', 'filename': '../deeplake/
→ tests/dummy_data/video/samplemp4.mp4', 'modified': False}
>>> ds.images[:2].sample_info
[{'exif': {'Software': 'Google'}, 'shape': [900, 900, 3], 'format': 'jpeg',
→ 'filename': '../deeplake/tests/dummy_data/images/cat.jpeg', 'modified':
→ False}, {'exif': {}, 'shape': [495, 750, 3], 'format': 'jpeg', 'filename': '
→ ../deeplake/tests/dummy_data/images/car.jpg', 'modified': False}]
```

**property shape:** `Tuple[Optional[int], ...]`

Get the shape of this tensor. Length is included.

### Example

```
>>> tensor.append(np.zeros((10, 10)))
>>> tensor.append(np.zeros((10, 15)))
>>> tensor.shape
(2, 10, None)
```

#### Returns

Tuple where each value is either None (if that axis is dynamic) or an int (if that axis is fixed).

#### Return type

tuple

---

**Note:** If you don't want None in the output shape or want the lower/upper bound shapes, use [shape\\_interval](#) instead.

---

**property shape\_interval:** [ShapeInterval](#)

Returns a [ShapeInterval](#) object that describes this tensor's shape more accurately. Length is included.

### Example

```
>>> tensor.append(np.zeros((10, 10)))
>>> tensor.append(np.zeros((10, 15)))
>>> tensor.shape_interval
ShapeInterval(lower=(2, 10, 10), upper=(2, 10, 15))
>>> str(tensor.shape_interval)
(2, 10, 10:15)
```

#### Returns

Object containing lower and upper properties.

#### Return type

[ShapeInterval](#)

---

**Note:** If you are expecting a tuple, use [shape](#) instead.

---

**shapes()**

Get the shapes of all the samples in the tensor.

#### Returns

List of shapes of all the samples in the tensor.

#### Return type

np.ndarray

**summary()**

Prints a summary of the tensor.

**text**(*fetch\_chunks: bool = False*)

Return text data. Only applicable for tensors with 'text' base htype.

**property timestamps: ndarray**

Returns timestamps (in seconds) for video sample as numpy array.

### Example

```
>>> # Return timestamps for all frames of first video sample
>>> ds.videos[0].timestamps.shape
(400,)
>>> # Return timestamps for 5th to 10th frame of first video sample
>>> ds.videos[0, 5:10].timestamps
array([0.2002      , 0.23356667, 0.26693332, 0.33366665, 0.4004      ],
      dtype=float32)
```

**tobytes()** → bytes

Returns the bytes of the tensor.

- Only works for a single sample of tensor.
- If the tensor is uncompressed, this returns the bytes of the numpy array.
- If the tensor is sample compressed, this returns the compressed bytes of the sample.
- If the tensor is chunk compressed, this raises an error.

#### Returns

The bytes of the tensor.

#### Return type

bytes

#### Raises

**ValueError** – If the tensor has multiple samples.

**property verify**

Whether linked data will be verified when samples are added. Applicable only to tensors with htype `link[htype]`.



## DEEPLAKE.API

### 21.1 deeplake.api.dataset

**class** `deeplake.api.dataset.dataset`

**static exists**(*path*: Union[str, pathlib.Path], *creds*: Optional[dict] = None, *token*: Optional[str] = None) → bool

See `deeplake.exists()`.

**static empty**(*path*: Union[str, pathlib.Path], *overwrite*: bool = False, *public*: bool = False, *memory\_cache\_size*: int = DEFAULT\_MEMORY\_CACHE\_SIZE, *local\_cache\_size*: int = DEFAULT\_LOCAL\_CACHE\_SIZE, *creds*: Optional[dict] = None, *token*: Optional[str] = None) → Dataset

See `deeplake.empty()`.

**static load**(*path*: Union[str, pathlib.Path], *read\_only*: Optional[bool] = None, *memory\_cache\_size*: int = DEFAULT\_MEMORY\_CACHE\_SIZE, *local\_cache\_size*: int = DEFAULT\_LOCAL\_CACHE\_SIZE, *creds*: Optional[dict] = None, *token*: Optional[str] = None, *verbose*: bool = True, *access\_method*: str = 'stream') → Dataset

See `deeplake.load()`.

**static rename**(*old\_path*: Union[str, pathlib.Path], *new\_path*: Union[str, pathlib.Path], *creds*: Optional[dict] = None, *token*: Optional[str] = None) → Dataset

See `deeplake.rename()`.

**static delete**(*path*: Union[str, pathlib.Path], *force*: bool = False, *large\_ok*: bool = False, *creds*: Optional[dict] = None, *token*: Optional[str] = None, *verbose*: bool = False) → None

See `deeplake.delete()`.

**static like**(*dest*: Union[str, pathlib.Path], *src*: Union[str, Dataset, pathlib.Path], *tensors*: Optional[List[str]] = None, *overwrite*: bool = False, *creds*: Optional[dict] = None, *token*: Optional[str] = None, *public*: bool = False) → Dataset

See `deeplake.like()`.

**static copy**(*src*: Union[str, pathlib.Path, Dataset], *dest*: Union[str, pathlib.Path], *tensors*: Optional[List[str]] = None, *overwrite*: bool = False, *src\_creds*=None, *dest\_creds*=None, *token*=None, *num\_workers*: int = 0, *scheduler*='threaded', *progressbar*=True)

See `deeplake.copy()`.

**static deepcopy**(*src*: Union[str, pathlib.Path], *dest*: Union[str, pathlib.Path], *tensors*: Optional[List[str]] = None, *overwrite*: bool = False, *src\_creds*=None, *dest\_creds*=None, *token*=None, *num\_workers*: int = 0, *scheduler*='threaded', *progressbar*=True, *public*: bool = False, *verbose*: bool = True)

See [`deeplake.deeplakecopy\(\)`](#).

```
static connect(src_path: str, creds_key: str, dest_path: Optional[str], org_id: Optional[str], ds_name: Optional[str], token: Optional[str])
```

See [`deeplake.connect\(\)`](#).

```
static ingest_classification(src: Union[str, pathlib.Path], dest: Union[str, pathlib.Path],
                             image_params: Optional[Dict] = None, label_params: Optional[Dict] =
                             None, dest_creds: Optional[Dict] = None, progressbar: bool = True,
                             summary: bool = True, num_workers: int = 0, shuffle: bool = False,
                             token: Optional[str] = None, connect_kwargs: Optional[Dict] = None,
                             **dataset_kwargs) → Dataset
```

See [`deeplake.ingest\_classification\(\)`](#).

```
static ingest_coco(images_directory: Union[str, pathlib.Path], annotation_files: Union[str, pathlib.Path,
List[str]], dest: Union[str, pathlib.Path], key_to_tensor_mapping: Optional[Dict] =
None, file_to_group_mapping: Optional[Dict] = None, ignore_one_group: bool =
False, ignore_keys: Optional[List[str]] = None, image_settings: Optional[Dict] =
None, src_creds: Optional[Dict] = None, dest_creds: Optional[Dict] = None,
inspect_limit: int = 1000000, progressbar: bool = True, shuffle: bool = False,
num_workers: int = 0, token: Optional[str] = None, connect_kwargs: Optional[Dict]
= None, **dataset_kwargs) → Dataset
```

See [`deeplake.ingest\_coco\(\)`](#).

```
def ingest_yolo(data_directory: Union[str, pathlib.Path], dest: Union[str, pathlib.
Path], class_names_file: Optional[Union[str, pathlib.Path]] = None,
annotations_directory: Optional[Union[str, pathlib.Path]] = None,
allow_no_annotation: bool = False, image_params: Optional[Dict] = None,
label_params: Optional[Dict] = None, coordinates_params: Optional[Dict] = None,
src_creds: Optional[Dict] = None, dest_creds: Optional[Dict] = None,
image_creds_key: Optional[str] = None, inspect_limit: int = 1000,
progressbar: bool = True, shuffle: bool = False, num_workers: int = 0,
token: Optional[str] = None, connect_kwargs: Optional[Dict] = None,
**dataset_kwargs) → Dataset:
```

See [`deeplake.ingest\_yolo\(\)`](#).

```
static ingest_kaggle(tag: str, src: Union[str, pathlib.Path], dest: Union[str, pathlib.Path], exist_ok: bool
= False, images_compression: str = 'auto', dest_creds: dict = None,
kaggle_credentials: dict = None, progressbar: bool = True, summary: bool = True,
**dataset_kwargs) → Dataset
```

See [`deeplake.ingest\_kaggle\(\)`](#).

```
static ingest_dataframe(src, dest: Union[str, pathlib.Path], column_params: Optional[Dict] = None,
src_creds: Optional[Dict] = None, dest_creds: Optional[Dict] = None,
creds_key: Optional[Dict] = None, progressbar: bool = True, token:
Optional[str] = None, connect_kwargs: Optional[Dict] = None,
**dataset_kwargs)
```

See [`deeplake.ingest\_dataframe\(\)`](#).

```
static list(workspace: str = "", token: Optional[str] = None) → None
```

See [`deeplake.list\(\)`](#).

## 21.2 deeplake.api.info

**class** `deeplake.api.info.Info`

Contains optional key-value pairs that can be stored for datasets/tensors.

**clear()**

Clear info.

**get**(*key*, *default=None*)

Get value for key from info.

**items()**

Return all items in info.

**keys()**

Return all keys in info.

**property** `nbytes`

Returns size of info stored in bytes.

**pop**(*key*, *default=None*)

Pop item from info by key.

**popitem()**

Pop item from info.

**replace\_with**(*d*)

Replace info with another dictionary.

**setdefault**(*key*, *default=None*)

Set default value for a key in info.

**update**(\**args*, \*\**kwargs*)

Update info.

**values()**

Return all values in info.

## 21.3 deeplake.api.link

`deeplake.api.link.link`(*path*: *str*, *creds\_key*: *Optional[str] = None*) → *LinkedSample*

See [deeplake.link\(\)](#).

## 21.4 deeplake.api.read

`deeplake.api.read.read`(*path*: *str*, *verify*: *bool = False*, *creds*: *Optional[Dict] = None*, *compression*: *Optional[str] = None*, *storage*: *Optional[StorageProvider] = None*) → *Sample*

See [deeplake.read\(\)](#).

## 21.5 deeplake.api.tiled

`deeplake.api.tiled.tiled(sample_shape: Tuple[int, ...], tile_shape: Optional[Tuple[int, ...]] = None, dtype: Union[str, np.dtype] = np.dtype('uint8'))`

See `deeplake.tiled()`.

## 21.6 deeplake.api.link\_tiled

`deeplake.api.link_tiled.link_tiled(path_array: np.ndarray, creds_key: Optional[str] = None) → LinkedTiledSample:`

See `deeplake.link_tiled()`.

## DEEPLAKE.AUTO

### 22.1 deeplake.auto.structured

#### 22.1.1 deeplake.auto.structured.base

**class** deeplake.auto.structured.base.**StructuredDataset**(*source*)

Initializes a structured dataset.

**Parameters**

**source** (*str*) – The local path to folder or file containing a structured dataset and of the form `./path/to/dataset` or `~/path/to/dataset` or `path/to/dataset`.

#### 22.1.2 deeplake.auto.structured.dataframe

**class** deeplake.auto.structured.dataframe.**DataFrame**(*source*, *column\_params=None*, *creds=None*, *creds\_key=None*)

**fill\_dataset**(*ds*: [Dataset](#), *progressbar: bool = True*) → [Dataset](#)

Fill dataset with data from the dataframe - one tensor per column

**Parameters**

- **ds** ([Dataset](#)) – A Deep Lake dataset object.
- **progressbar** (*bool*) – Defines if the method uses a progress bar. Defaults to `True`.

**Returns**

A Deep Lake dataset.

### 22.2 deeplake.auto.unstructured

#### 22.2.1 deeplake.auto.unstructured.base

**class** deeplake.auto.unstructured.base.**UnstructuredDataset**(*source: str*)

### 22.2.2 deeplake.auto.unstructured.image\_classification

**class** deeplake.auto.unstructured.image\_classification.**ImageClassification**(*source: str*)

**structure**(*ds: Dataset, progressbar: bool = True, generate\_summary: bool = True, shuffle: bool = True, image\_tensor\_args: dict = {}, label\_tensor\_args: dict = {}, num\_workers: int = 0*) → *Dataset*

Create a structured dataset.

**Parameters**

- **ds** (*Dataset*) – A Deep Lake dataset object.
- **progressbar** (*bool*) – Defines if the method uses a progress bar. Defaults to True.
- **generate\_summary** (*bool*) – Defines if the method generates ingestion summary. Defaults to True.
- **shuffle** (*bool*) – Defines if the file paths should be shuffled prior to ingestion. Defaults to True.
- **image\_tensor\_args** (*dict*) – Defines the parameters for the images tensor.
- **label\_tensor\_args** (*dict*) – Defines the parameters for the class\_labels tensor.
- **num\_workers** (*int*) – The number of workers passed to compute.

**Returns**

A Deep Lake dataset.

### 22.2.3 deeplake.auto.unstructured.kaggle

deeplake.auto.unstructured.kaggle.**download\_kaggle\_dataset**(*tag: str, local\_path: str, kaggle\_credentials: Optional[dict] = None, exist\_ok: bool = False*)

Calls the kaggle API (<https://www.kaggle.com/docs/api>) to download a kaggle dataset and unzip it's contents.

**Parameters**

- **tag** (*str*) – Kaggle dataset tag. Example: “coloradokb/dandelionimages” points to <https://www.kaggle.com/coloradokb/dandelionimages>
- **local\_path** (*str*) – Path where the kaggle dataset will be downloaded and unzipped. Only local path downloading is supported.
- **kaggle\_credentials** (*dict*) – Credentials are gathered from the environment variables or `~/kaggle.json`. If those don't exist, the *kaggle\_credentials* argument will be used.
- **exist\_ok** (*bool*) – If the kaggle dataset was already downloaded, and *exist\_ok* is True, no error is thrown.

**Raises**

- **KaggleMissingCredentialsError** – If no kaggle credentials are found.
- **KaggleDatasetAlreadyDownloadedError** – If the dataset *tag* already exists in *local\_path*.

## 23.1 deeplake.util.shape\_interval

**class** deeplake.util.shape\_interval.**ShapeInterval**

**\_\_init\_\_**(*lower: Sequence[int], upper: Optional[Sequence[int]] = None*)

Shapes in Deep Lake are best represented as intervals, this is to support dynamic tensors. Instead of having a single tuple of integers representing shape, we use 2 tuples of integers to represent the lower and upper bounds of the representing shape.

- If `lower == upper` for all cases, the shape is considered “fixed”.
- If `lower != upper` for any cases, the shape is considered “dynamic”.

### Parameters

- **lower** (*sequence*) – Sequence of integers that represent the lower-bound shape.
- **upper** (*sequence*) – Sequence of integers that represent the upper-bound shape.  
If None is provided, lower is used as upper (implicitly fixed-shape).

### Raises

***InvalidShapeIntervalError*** – If the provided lower/upper bounds are incompatible to represent a shape.

## 23.2 deeplake.util.remove\_cache

deeplake.util.remove\_cache.**remove\_memory\_cache**(*storage: StorageProvider*)

Removes the memory cache.

deeplake.util.remove\_cache.**get\_base\_storage**(*storage: StorageProvider*)

Removes all layers of caching and returns the underlying storage.

deeplake.util.remove\_cache.**get\_dataset\_with\_zero\_size\_cache**(*ds*)

Returns a dataset with same storage but cache size set to zero.

deeplake.util.remove\_cache.**create\_read\_copy\_dataset**(*dataset, commit\_id: Optional[str] = None*)

Creates a read-only copy of the given dataset object, without copying underlying data.

### Parameters

- **dataset** – The Dataset object to copy.
- **commit\_id** – The commit id to checkout the new read-only copy to.

### Returns

A new Dataset object in read-only mode.

## 23.3 deeplake.util.notebook

`deeplake.util.notebook.is_notebook()`

Whether running in a notebook.

`deeplake.util.notebook.is_jupyter()`

Whether running in a Jupyter notebook.

`deeplake.util.notebook.is_colab()`

Whether running in a colab notebook.

## 23.4 deeplake.util.exceptions

`class deeplake.util.exceptions.ExternalCommandError(command: str, status: int)`

Bases: `Exception`

`class deeplake.util.exceptions.KaggleError`

Bases: `Exception`

`class deeplake.util.exceptions.KaggleMissingCredentialsError(env_var_name: str)`

Bases: `KaggleError`

`class deeplake.util.exceptions.KaggleDatasetAlreadyDownloadedError(tag: str, path: str)`

Bases: `KaggleError`

`class deeplake.util.exceptions.InvalidPathException(directory)`

Bases: `Exception`

`class deeplake.util.exceptions.AutoCompressionError(directory)`

Bases: `Exception`

`class deeplake.util.exceptions.InvalidFileExtension(directory)`

Bases: `Exception`

`class deeplake.util.exceptions.SamePathException(directory)`

Bases: `Exception`

`class deeplake.util.exceptions.TensorInvalidSampleShapeError(shape: Sequence[int],  
expected_dims: int)`

Bases: `Exception`

`class deeplake.util.exceptions.TensorMetaMissingKey(key: str, meta: dict)`

Bases: `Exception`

`class deeplake.util.exceptions.TensorDoesNotExistError(tensor_name: str)`

Bases: `KeyError`, `AttributeError`

`class deeplake.util.exceptions.TensorAlreadyExistsError(key: str)`

Bases: `Exception`

`class deeplake.util.exceptions.TensorGroupDoesNotExistError(group_name: str)`

Bases: `KeyError`



```

class deeplake.util.exceptions.TensorGroupAlreadyExistsError(key: str)
    Bases: Exception
class deeplake.util.exceptions.InvalidTensorNameError(name: str)
    Bases: Exception
class deeplake.util.exceptions.InvalidTensorGroupNameError(name: str)
    Bases: Exception
class deeplake.util.exceptions.DynamicTensorNumpyError(key: str, index, property_key: str)
    Bases: Exception
class deeplake.util.exceptions.InvalidShapeIntervalError(message: str, lower:
    Optional[Sequence[int]] = None, upper:
    Optional[Sequence[int]] = None)
    Bases: Exception
class deeplake.util.exceptions.InvalidKeyTypeError(item: Any)
    Bases: TypeError
class deeplake.util.exceptions.UnsupportedTensorTypeError(item: Any)
    Bases: TypeError
class deeplake.util.exceptions.InvalidBytesRequestedError
    Bases: Exception
class deeplake.util.exceptions.ProviderListEmptyError
    Bases: Exception
class deeplake.util.exceptions.DirectoryAtPathException
    Bases: Exception
class deeplake.util.exceptions.FileAtPathException(path)
    Bases: Exception
class deeplake.util.exceptions.ProviderSizeListMismatch
    Bases: Exception
class deeplake.util.exceptions.ModuleNotInstalledException(message)
    Bases: Exception
class deeplake.util.exceptions.LoginException(message='Error while logging in, invalid auth token.
    Please try logging in again.')
    Bases: Exception
class deeplake.util.exceptions.UserNotLoggedInException
    Bases: Exception
class deeplake.util.exceptions.InvalidHubPathException(path)
    Bases: Exception
class deeplake.util.exceptions.PathNotEmptyException(use_hub=True)
    Bases: Exception
class deeplake.util.exceptions.AuthenticationException(message='Authentication failed. Please try
    logging in again.')
    Bases: Exception

```

```
class deeplake.util.exceptions.AuthorizationException(message='You are not authorized to access
                                                    this resource on Activeloop Server.',
                                                    response=None)

    Bases: Exception

class deeplake.util.exceptions.InvalidPasswordException(message='The password you provided was
                                                                invalid.')
```

Bases: [AuthorizationException](#)

```
class deeplake.util.exceptions.CouldNotCreateNewDatasetException(path: str)

    Bases: AuthorizationException
```

```
class deeplake.util.exceptions.ResourceNotFoundException(message='The resource you are looking
                                                                for was not found. Check if the name or id
                                                                is correct.')
```

Bases: Exception

```
class deeplake.util.exceptions.BadRequestException(message)

    Bases: Exception
```

```
class deeplake.util.exceptions.OverLimitException(message='You are over the allowed limits for this
                                                            operation.')
```

Bases: Exception

```
class deeplake.util.exceptions.ServerException(message='Internal Activeloop server error.')
```

Bases: Exception

```
class deeplake.util.exceptions.BadGatewayException(message='Invalid response from Activeloop
                                                            server.')
```

Bases: Exception

```
class deeplake.util.exceptions.GatewayTimeoutException(message='Activeloop server took too long to
                                                                respond.')
```

Bases: Exception

```
class deeplake.util.exceptions.WaitTimeoutException(message='Timeout waiting for server state
                                                            update.')
```

Bases: Exception

```
class deeplake.util.exceptions.LockedException(message='The resource is currently locked.')
```

Bases: Exception

```
class deeplake.util.exceptions.UnexpectedStatusCodeException(message)

    Bases: Exception
```

```
class deeplake.util.exceptions.EmptyTokenException(message='The authentication token is empty.')
```

Bases: Exception

```
class deeplake.util.exceptions.S3Error

    Bases: Exception

    Catchall for all errors encountered while working with S3
```

```
class deeplake.util.exceptions.S3GetError

    Bases: S3Error

    Catchall for all errors encountered while working getting an object from S3
```

```
class deeplake.util.exceptions.S3SetError
```

Bases: [S3Error](#)

Catchall for all errors encountered while working setting an object in S3

```
class deeplake.util.exceptions.S3DeletionError
```

Bases: [S3Error](#)

Catchall for all errors encountered while working deleting an object in S3

```
class deeplake.util.exceptions.S3ListError
```

Bases: [S3Error](#)

Catchall for all errors encountered while retrieving a list of objects present in S3

```
class deeplake.util.exceptions.UnsupportedCompressionError(compression: Optional[str], htype:  
Optional[str] = None)
```

Bases: `CompressionError`

```
class deeplake.util.exceptions.SampleCompressionError(sample_shape: Tuple[int, ...],  
compression_format: Optional[str], message:  
str)
```

Bases: `CompressionError`

```
class deeplake.util.exceptions.SampleDecompressionError(path: Optional[str] = None)
```

Bases: `CompressionError`

```
class deeplake.util.exceptions.InvalidImageDimensions(actual_dims, expected_dims)
```

Bases: `Exception`

```
class deeplake.util.exceptions.TensorUnsupportedSampleType
```

Bases: `Exception`

```
class deeplake.util.exceptions.MetaError
```

Bases: `Exception`

```
class deeplake.util.exceptions.MetaDoesNotExistError(key: str)
```

Bases: [MetaError](#)

```
class deeplake.util.exceptions.MetaAlreadyExistsError(key: str, required_meta: dict)
```

Bases: [MetaError](#)

```
class deeplake.util.exceptions.MetaInvalidKey(name: str, available_keys: List[str])
```

Bases: [MetaError](#)

```
class deeplake.util.exceptions.MetaInvalidRequiredMetaKey(key: str, subclass_name: str)
```

Bases: [MetaError](#)

```
class deeplake.util.exceptions.TensorMetaInvalidHtype(htype: str, available_htypes: Sequence[str])
```

Bases: [MetaError](#)

```
class deeplake.util.exceptions.TensorMetaInvalidHtypeOverwriteValue(key: str, value: Any,  
explanation: str = "")
```

Bases: [MetaError](#)

```
class deeplake.util.exceptions.TensorMetaMissingRequiredValue(htype: str, key: Union[str,  
List[str]])
```

Bases: [MetaError](#)

```
class deeplake.util.exceptions.TensorMetaInvalidHtypeOverwriteKey(htype: str, key: str,
                                                                    available_keys:
                                                                    Sequence[str])

    Bases: MetaError

class deeplake.util.exceptions.TensorDtypeMismatchError(expected: Union[dtype, str], actual: str,
                                                         htype: str)

    Bases: MetaError

class deeplake.util.exceptions.InvalidTensorLinkError(msg='Invalid tensor link.')

    Bases: MetaError

class deeplake.util.exceptions.TensorMetaMutuallyExclusiveKeysError(keys: Optional[List[str]] =
                                                                    None, custom_message:
                                                                    Optional[str] = None)

    Bases: MetaError

class deeplake.util.exceptions.ReadOnlyModeError(custom_message: Optional[str] = None)

    Bases: Exception

class deeplake.util.exceptions.TransformError(index=None, sample=None, samples_processed=0,
                                              suggest=False)

    Bases: Exception

class deeplake.util.exceptions.FilterError

    Bases: Exception

class deeplake.util.exceptions.InvalidInputDataError(operation)

    Bases: TransformError

class deeplake.util.exceptions.UnsupportedSchedulerError(scheduler)

    Bases: TransformError

class deeplake.util.exceptions.TensorMismatchError(tensors, output_keys, skip_ok=False)

    Bases: TransformError

class deeplake.util.exceptions.InvalidOutputDatasetError(message='The output Dataset to transform
                                                                should not be `read_only`.')

    Bases: TransformError

class deeplake.util.exceptions.InvalidTransformDataset(message='The TransformDataset (2nd
                                                                argument to transform function) of one of the
                                                                functions is invalid. All the tensors should
                                                                have equal length for it to be valid.')

    Bases: TransformError

class deeplake.util.exceptions.HubComposeEmptyListError(message='Cannot deeplake.compose an
                                                                empty list.')

    Bases: TransformError

class deeplake.util.exceptions.HubComposeIncompatibleFunction(index: int)

    Bases: TransformError

class deeplake.util.exceptions.DatasetUnsupportedPytorch(reason)

    Bases: Exception
```

```

class deeplake.util.exceptions.CorruptedMetaError
    Bases: Exception

class deeplake.util.exceptions.ChunkEngineError
    Bases: Exception

class deeplake.util.exceptions.FullChunkError
    Bases: ChunkEngineError

class deeplake.util.exceptions.ChunkIdEncoderError
    Bases: ChunkEngineError

class deeplake.util.exceptions.ChunkSizeTooSmallError(message='If the size of the last chunk is given,
it must be smaller than the requested chunk
size.')
    Bases: ChunkEngineError

class deeplake.util.exceptions.DatasetHandlerError(message)
    Bases: Exception

class deeplake.util.exceptions.MemoryDatasetCanNotBePickledError
    Bases: Exception

class deeplake.util.exceptions.CorruptedSampleError(compression, path: Optional[str] = None)
    Bases: Exception

class deeplake.util.exceptions.VersionControlError
    Bases: Exception

class deeplake.util.exceptions.MergeError
    Bases: Exception

class deeplake.util.exceptions.MergeNotSupportedError
    Bases: MergeError

class deeplake.util.exceptions.MergeMismatchError(tensor_name, mismatch_type, original_value,
target_value)
    Bases: MergeError

class deeplake.util.exceptions.MergeConflictError(conflict_tensors=None, message="")
    Bases: MergeError

class deeplake.util.exceptions.CheckoutError
    Bases: VersionControlError

class deeplake.util.exceptions.CommitError
    Bases: VersionControlError

class deeplake.util.exceptions.EmptyCommitError
    Bases: CommitError

class deeplake.util.exceptions.TensorModifiedError
    Bases: Exception

class deeplake.util.exceptions.GCSDefaultCredsNotFound
    Bases: Exception

```

```
class deeplake.util.exceptions.InvalidOperationError(method: str, type: str)
    Bases: Exception

class deeplake.util.exceptions.AgreementError
    Bases: Exception

class deeplake.util.exceptions.AgreementNotAcceptedError(agreements=None)
    Bases: AgreementError

class deeplake.util.exceptions.RenameError(msg='Only name of the dataset can be different in new path.')
    Bases: Exception

class deeplake.util.exceptions.BufferError
    Bases: Exception

class deeplake.util.exceptions.InfoError
    Bases: Exception

class deeplake.util.exceptions.OutOfChunkCountError
    Bases: Exception

class deeplake.util.exceptions.OutOfSampleCountError
    Bases: Exception

class deeplake.util.exceptions.SampleHtypeMismatchError(htype, sample_type)
    Bases: Exception

class deeplake.util.exceptions.EmptyTensorError(message)
    Bases: Exception

class deeplake.util.exceptions.DatasetViewSavingError
    Bases: Exception

class deeplake.util.exceptions.ManagedCredentialsNotFoundError(org_id, creds_key)
    Bases: Exception

class deeplake.util.exceptions.UnableToReadFromUrlError(url, status_code)
    Bases: Exception

class deeplake.util.exceptions.InvalidTokenException
    Bases: Exception

class deeplake.util.exceptions.TokenPermissionError(message=None)
    Bases: Exception
```

## DEEPLAKE.CLIENT.LOG

Deep Lake does logging using the “deeplake” logger. Logging level is `logging.INFO` by default. See example on how to change this.

```
>>> import deeplake
>>> import logging
>>> logger = logging.getLogger("deeplake")
>>> logger.setLevel(logging.WARNING)
```





## DEEPLAKE.CORE.TRANSFORM

**class** deeplake.core.transform.Pipeline(*functions: List[ComputeFunction]*)

**eval**(*data\_in, ds\_out: Optional[Dataset] = None, num\_workers: int = 0, scheduler: str = 'threaded',  
progressbar: bool = True, skip\_ok: bool = False, check\_lengths: bool = True, pad\_data\_in: bool =  
False, read\_only\_ok: bool = False, cache\_size: int = 16, checkpoint\_interval: int = 0, ignore\_errors:  
bool = False, verbose: bool = True, \*\*kwargs*)

Evaluates the pipeline on *data\_in* to produce an output dataset *ds\_out*.

### Parameters

- **data\_in** – Input passed to the transform to generate output dataset. Should support `__getitem__` and `__len__`. Can be a Deep Lake dataset.
- **ds\_out** (*Dataset*, *optional*) –
  - The dataset object to which the transform will get written. If this is not provided, *data\_in* will be overwritten if it is a Deep Lake dataset, otherwise error will be raised.
  - It should have all keys being generated in output already present as tensors. It's initial state should be either:
    - **Empty**, i.e., all tensors have no samples. In this case all samples are added to the dataset.
    - **All tensors are populated and have same length**. In this case new samples are appended to the dataset.
- **num\_workers** (*int*) – The number of workers to use for performing the transform. Defaults to 0. When set to 0, it will always use serial processing, irrespective of the scheduler.
- **scheduler** (*str*) – The scheduler to be used to compute the transformation. Supported values include: 'serial', 'threaded', 'processed' and 'ray'. Defaults to 'threaded'.
- **progressbar** (*bool*) – Displays a progress bar if True (default).
- **skip\_ok** (*bool*) – If True, skips the check for output tensors generated. This allows the user to skip certain tensors in the function definition. This is especially useful for inplace transformations in which certain tensors are not modified. Defaults to False.
- **check\_lengths** (*bool*) – If True, checks whether *ds\_out* has tensors of same lengths initially.
- **pad\_data\_in** (*bool*) – If True, pads tensors of *data\_in* to match the length of the largest tensor in *data\_in*. Defaults to False.

- **read\_only\_ok** (*bool*) – If True and output dataset is same as input dataset, the read-only check is skipped. Defaults to False.
- **cache\_size** (*int*) – Cache size to be used by transform per worker.
- **checkpoint\_interval** (*int*) – If > 0, the transform will be checkpointed with a commit every `checkpoint_interval` input samples to avoid restarting full transform due to intermitten failures. If the transform is interrupted, the intermediate data is deleted and the dataset is reset to the last commit. If <= 0, no checkpointing is done. Checkpoint interval should be a multiple of `num_workers` if `num_workers` > 0. Defaults to 0.
- **ignore\_errors** (*bool*) – If True, input samples that causes transform to fail will be skipped and the errors will be ignored **if possible**.
- **verbose** (*bool*) – If True, prints additional information about the transform.
- **\*\*kwargs** – Additional arguments.

#### Raises

- **InvalidInputDataError** – If `data_in` passed to transform is invalid. It should support `__getitem__` and `__len__` operations. Using scheduler other than “threaded” with deeplake dataset having base storage as memory as `data_in` will also raise this.
- **InvalidOutputDatasetError** – If all the tensors of `ds_out` passed to transform don’t have the same length. Using scheduler other than “threaded” with deeplake dataset having base storage as memory as `ds_out` will also raise this.
- **TensorMismatchError** – If one or more of the outputs generated during transform contain different tensors than the ones present in ‘`ds_out`’ provided to transform.
- **UnsupportedSchedulerError** – If the scheduler passed is not recognized. Supported values include: ‘serial’, ‘threaded’, ‘processed’ and ‘ray’.
- **TransformError** – All other exceptions raised if there are problems while running the pipeline.
- **ValueError** – If `num_workers` > 0 and `checkpoint_interval` is not a multiple of `num_workers` or if `checkpoint_interval` > 0 and `ds_out` is None.

# noqa: DAR401

Example:

```
@deeplake.compute
def my_fn(sample_in: Any, samples_out, my_arg0, my_arg1=0):
    samples_out.my_tensor.append(my_arg0 * my_arg1)

# This transform can be used using the eval method in one of these 2 ways:-

# Directly evaluating the method
# here arg0 and arg1 correspond to the 3rd and 4th argument in my_fn
my_fn(arg0, arg1).eval(data_in, ds_out, scheduler="threaded", num_workers=5)

# As a part of a Transform pipeline containing other functions
pipeline = deeplake.compose([my_fn(a, b), another_function(x=2)])
pipeline.eval(data_in, ds_out, scheduler="processed", num_workers=2)
```

---

**Note:** `pad_data_in` is only applicable if `data_in` is a Deep Lake dataset.

---



## DEEPLAKE.CORE.VECTORSTORE.DEEP\_MEMORY

### 26.1 DeepMemory

**class** deeplake.core.vectorstore.deep\_memory.deep\_memory.**DeepMemory**

**\_\_init\_\_**(dataset: Dataset, path: Union[str, Path], logger: Logger, embedding\_function: Optional[Any] = None, token: Optional[str] = None, creds: Optional[Union[Dict, str]] = None)

Base Deep Memory class to train and evaluate models on DeepMemory managed service.

**Parameters**

- **dataset** (Dataset) – deeplake dataset object or path.
- **path** (Union[str, pathlib.Path]) – Path to the dataset.
- **logger** (logging.Logger) – Logger object.
- **embedding\_function** (Optional[Any], optional) – Embedding function class used to convert queries/documents to embeddings. Defaults to None.
- **token** (Optional[str], optional) – API token for the DeepMemory managed service. Defaults to None.
- **creds** (Optional[Dict[str, Any]], optional) – Credentials to access the dataset. Defaults to None.

**Raises**

**ImportError** – if indra is not installed

**cancel**(job\_id: str)

Cancel a training job on DeepMemory managed service.

**Examples**

```
>>> cancelled: bool = vectorstore.deep_memory.cancel(job_id)
```

**Parameters**

**job\_id** (str) – job\_id of the training job.

**Returns**

True if job was cancelled successfully, False otherwise.

**Return type**

bool

**delete**(job\_id: str)

Delete a training job on DeepMemory managed service.

## Examples

```
>>> deleted: bool = vectorstore.deep_memory.delete(job_id)
```

### Parameters

**job\_id** (*str*) – job\_id of the training job.

### Returns

True if job was deleted successfully, False otherwise.

### Return type

bool

**evaluate** (*relevance: List[List[Tuple[str, int]]], queries: List[str], embedding\_function: Optional[Callable[[...], List[ndarray]]] = None, embedding: Optional[Union[List[ndarray], List[List[float]]]] = None, top\_k: List[int] = [1, 3, 5, 10, 50, 100], qvs\_params: Optional[Dict[str, Any]] = None*) → Dict[str, Dict[str, float]]

Evaluate a model using the DeepMemory managed service.

## Examples

```
>>> # 1. Evaluate a model using an embedding function:
>>> relevance = [(("doc_id_1", 1), ("doc_id_2", 1)), [("doc_id_3", 1)]]
>>> queries = ["What is the capital of India?", "What is the capital of France?"]
>>> embedding_function = openai_embedding.embed_documents
>>> vectorstore.deep_memory.evaluate(
...     relevance=relevance,
...     queries=queries,
...     embedding_function=embedding_function,
... )
```

```
>>> # 2. Evaluate a model with precomputed embeddings:
>>> embeddings = [[-1.2, 12, ...], ...]
>>> vectorstore.deep_memory.evaluate(
...     relevance=relevance,
...     queries=queries,
...     embedding=embeddings,
... )
```

```
>>> # 3. Evaluate a model with precomputed embeddings and log queries:
>>> vectorstore.deep_memory.evaluate(
...     relevance=relevance,
...     queries=queries,
...     embedding=embeddings,
...     qvs_params={"log_queries": True},
... )
```

```
>>> # 4. Evaluate with precomputed embeddings, log queries, and a custom
    ↪ branch:
>>> vectorstore.deep_memory.evaluate(
...     relevance=relevance,
...     queries=queries,
```

(continues on next page)

(continued from previous page)

```

...     embedding=embeddings,
...     qvs_params={
...         "log_queries": True,
...         "branch": "queries",
...     }
... )

```

**Parameters**

- **queries** (*List[str]*) – Queries for model evaluation.
- **relevance** (*List[List[Tuple[str, int]]]*) – Relevant documents and scores for each query. - Outer list: matches the queries. - Inner list: pairs of doc\_id and relevance score. - doc\_id: Document ID from the corpus dataset, found in the *id* tensor. - relevance\_score: Between 0 (not relevant) and 1 (relevant).
- **embedding** (*Optional[np.ndarray]*, *optional*) – Query embeddings. Defaults to None.
- **embedding\_function** (*Optional[Callable[... , List[np.ndarray]]]*, *optional*) – Function to convert queries into embeddings. Defaults to None.
- **top\_k** (*List[int]*, *optional*) – Ranks for model evaluation. Defaults to [1, 3, 5, 10, 50, 100].
- **qvs\_params** (*Optional[Dict]*, *optional*) – Parameters to initialize the queries vectorstore. When specified, creates a new vectorstore to track evaluation queries, the Deep Memory response, and the naive vector search results. Defaults to None.

**Returns**

Recalls for each rank.

**Return type**

Dict[str, Dict[str, float]]

**Raises**

- **ImportError** – If *indra* is not installed.
- **ValueError** – If no *embedding\_function* is provided either during initialization or evaluation.

**get\_model()**

Get the name of the model currently being used by DeepMemory managed service.

**list\_jobs(debug=False)**

List all training jobs on DeepMemory managed service.

**set\_model(model\_name: str)**

Set model.npy to use *model\_name* instead of default model :param model\_name: name of the model to use :type model\_name: str

**status(job\_id: str)**

Get the status of a training job on DeepMemory managed service.

## Examples

```
>>> vectorstore.deep_memory.status(job_id)
```

```
-----
|                               6508464cd80cab681bfcfff3                               |
|-----|-----|
| status                               | pending                               |
|-----|-----|
| progress                             | None                               |
|-----|-----|
| results                             | not available yet                   |
|-----|-----|
```

### Parameters

**job\_id** (*str*) – job\_id of the training job.

**train**(*queries: List[str], relevance: List[List[Tuple[str, int]]], embedding\_function: Optional[Callable[[str], ndarray]] = None, token: Optional[str] = None*) → *str*

Train a model on DeepMemory managed service.

## Examples

```
>>> queries: List[str] = ["What is the capital of India?", "What is the
↪ capital of France?"]
>>> relevance: List[List[Tuple[str, int]]] = [[("doc_id_1", 1), ("doc_id_2",
↪ 1)], [("doc_id_3", 1)]]
>>> # doc_id_1, doc_id_2, doc_id_3 are the ids of the documents in the corpus
↪ dataset that is relevant to the queries. It is stored in the `id` tensor of
↪ the corpus dataset.
>>> job_id: str = vectorstore.deep_memory.train(queries, relevance)
```

### Parameters

- **queries** (*List[str]*) – List of queries to train the model on.
- **relevance** (*List[List[Tuple[str, int]]]*) – List of relevant documents for each query with their respective relevance score. The outer list corresponds to the queries and the inner list corresponds to the doc\_id, relevance\_score pair for each query. doc\_id is the document id in the corpus dataset. It is stored in the *id* tensor of the corpus dataset. relevance\_score is the relevance score of the document for the query. The value is either 0 and 1, where 0 stands for not relevant (unknown relevance) and 1 stands for relevant. Currently, only values of 1 contribute to the training, and there is no reason to provide examples with relevance of 0.
- **embedding\_function** (*Optional[Callable[[str], np.ndarray]]*, *optional*) – Embedding function used to convert queries to embeddings. Defaults to None.
- **token** (*str*, *optional*) – API token for the DeepMemory managed service. Defaults to None.

### Returns

job\_id of the training job.

### Return type

str



**Raises**

**ValueError** – if `embedding_function` is not specified either during initialization or during training.



## DEEPLAKE.RANDOM.SEED

`class deeplake.core.seed.DeeplakeRandom`

`get_seed()` → `Optional[int]`

Returns the seed which set to the deeplake to control the flows

`seed(seed: Optional[int] = None)`

Set random seed to the deeplake engines

**Parameters**

`seed(int, optional)` – Integer seed for initializing the computational engines, used for bringing reproducibility to random operations. Set to `None` to reset the seed. Defaults to `None`.

**Raises**

**TypeError** – If the provided value type is not supported.

### 27.1 Background

Specify a seed to train models and run randomized Deep Lake operations reproducibly. Features affected are:

- Dataloader shuffling
- Sampling and random operations in Tensor Query Language (TQL)
- `Dataset.random_split`

The random seed can be specified using `deeplake.random.seed`:

```
>>> import deeplake
>>> deeplake.random.seed(0)
```

### 27.2 Random number generators in other libraries

The Deep Lake random seed does not affect random number generators in other libraries such as `numpy`.

However, seeds in other libraries will affect code where Deep Lake uses those libraries, but it will not impact the methods above where Deep Lake uses its internal seed.



## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## PYTHON MODULE INDEX

### d

`deeplake`, [61](#)

`deeplake.api.info`, [165](#)

`deeplake.integrations.wandb.wandb`, [39](#)





## Symbols

`__contains__()` (*deeplake.core.storage.GCSProvider* method), 109  
`__delitem__()` (*deeplake.core.storage.GCSProvider* method), 109  
`__delitem__()` (*deeplake.core.storage.GDriveProvider* method), 111  
`__delitem__()` (*deeplake.core.storage.LRUCache* method), 103  
`__delitem__()` (*deeplake.core.storage.LocalProvider* method), 113  
`__delitem__()` (*deeplake.core.storage.MemoryProvider* method), 116  
`__delitem__()` (*deeplake.core.storage.S3Provider* method), 106  
`__delitem__()` (*deeplake.core.storage.StorageProvider* method), 101  
`__getitem__()` (*deeplake.core.index.Index* method), 119  
`__getitem__()` (*deeplake.core.index.IndexEntry* method), 118  
`__getitem__()` (*deeplake.core.storage.GCSProvider* method), 109  
`__getitem__()` (*deeplake.core.storage.GDriveProvider* method), 111  
`__getitem__()` (*deeplake.core.storage.LRUCache* method), 103  
`__getitem__()` (*deeplake.core.storage.LocalProvider* method), 113  
`__getitem__()` (*deeplake.core.storage.MemoryProvider* method), 116  
`__getitem__()` (*deeplake.core.storage.S3Provider* method), 107  
`__getitem__()` (*deeplake.core.storage.StorageProvider* method), 101  
`__getstate__()` (*deeplake.core.storage.LRUCache* method), 103  
`__getstate__()` (*deeplake.core.storage.MemoryProvider* method), 116  
`__init__()` (*deeplake.core.index.Index* method), 120  
`__init__()` (*deeplake.core.index.IndexEntry* method), 118  
`__init__()` (*deeplake.core.sample.Sample* method), 99  
`__init__()` (*deeplake.core.storage.GCSProvider* method), 110  
`__init__()` (*deeplake.core.storage.GDriveProvider* method), 111  
`__init__()` (*deeplake.core.storage.LRUCache* method), 104  
`__init__()` (*deeplake.core.storage.LocalProvider* method), 113  
`__init__()` (*deeplake.core.storage.MemoryProvider* method), 116  
`__init__()` (*deeplake.core.storage.S3Provider* method), 107  
`__init__()` (*deeplake.core.vectorstore.deep\_memory.deep\_memory.DeepMemory* method), 183  
`__init__()` (*deeplake.core.vectorstore.deeplake\_vectorstore.VectorStore* method), 87  
`__init__()` (*deeplake.util.shape\_interval.ShapeInterval* method), 169  
`__iter__()` (*deeplake.core.storage.GCSProvider* method), 110  
`__iter__()` (*deeplake.core.storage.GDriveProvider* method), 112  
`__iter__()` (*deeplake.core.storage.LRUCache* method), 104  
`__iter__()` (*deeplake.core.storage.LocalProvider* method), 114  
`__iter__()` (*deeplake.core.storage.MemoryProvider* method), 116  
`__iter__()` (*deeplake.core.storage.S3Provider* method), 108  
`__iter__()` (*deeplake.core.storage.StorageProvider* method), 101  
`__len__()` (*deeplake.core.storage.GCSProvider* method), 110  
`__len__()` (*deeplake.core.storage.GDriveProvider* method), 112  
`__len__()` (*deeplake.core.storage.LRUCache* method), 104  
`__len__()` (*deeplake.core.storage.LocalProvider* method), 114  
`__len__()` (*deeplake.core.storage.MemoryProvider* method), 116

`method`), 117  
`__len__()` (*deeplake.core.storage.S3Provider* *method*), 108  
`__len__()` (*deeplake.core.storage.StorageProvider* *method*), 101  
`__len__()` (*deeplake.core.tensor.Tensor* *method*), 155  
`__repr__()` (*deeplake.core.index.Index* *method*), 120  
`__setitem__()` (*deeplake.core.storage.GCSProvider* *method*), 110  
`__setitem__()` (*deeplake.core.storage.GDriveProvider* *method*), 112  
`__setitem__()` (*deeplake.core.storage.LRUCache* *method*), 104  
`__setitem__()` (*deeplake.core.storage.LocalProvider* *method*), 114  
`__setitem__()` (*deeplake.core.storage.MemoryProvider* *method*), 117  
`__setitem__()` (*deeplake.core.storage.S3Provider* *method*), 108  
`__setitem__()` (*deeplake.core.storage.StorageProvider* *method*), 101  
`__setitem__()` (*deeplake.core.tensor.Tensor* *method*), 155  
`__setstate__()` (*deeplake.core.storage.LRUCache* *method*), 104  
`__str__()` (*deeplake.core.index.Index* *method*), 120  
`__str__()` (*deeplake.core.index.IndexEntry* *method*), 118  
`__weakref__` (*deeplake.core.index.Index* *attribute*), 120  
`__weakref__` (*deeplake.core.index.IndexEntry* *attribute*), 118  
`__weakref__` (*deeplake.core.storage.StorageProvider* *attribute*), 102  
`_all_keys()` (*deeplake.core.storage.GCSProvider* *method*), 110  
`_all_keys()` (*deeplake.core.storage.GDriveProvider* *method*), 112  
`_all_keys()` (*deeplake.core.storage.LRUCache* *method*), 104  
`_all_keys()` (*deeplake.core.storage.LocalProvider* *method*), 115  
`_all_keys()` (*deeplake.core.storage.MemoryProvider* *method*), 117  
`_all_keys()` (*deeplake.core.storage.S3Provider* *method*), 108  
`_all_keys()` (*deeplake.core.storage.StorageProvider* *method*), 102  
`_check_compatibility_with_htype()` (*deeplake.core.tensor.Tensor* *method*), 155  
`_check_is_file()` (*deeplake.core.storage.LocalProvider* *method*), 115  
`_check_update_creds()` (*deeplake.core.storage.S3Provider* *method*), 108  
`_config` (*deeplake.core.tensor.Tensor* *property*), 155  
`_flush_if_not_read_only()` (*deeplake.core.storage.LRUCache* *method*), 105  
`_forward()` (*deeplake.core.storage.LRUCache* *method*), 105  
`_forward_value()` (*deeplake.core.storage.LRUCache* *method*), 105  
`_free_up_space()` (*deeplake.core.storage.LRUCache* *method*), 105  
`_insert_in_cache()` (*deeplake.core.storage.LRUCache* *method*), 105  
`_is_hub_path` (*deeplake.core.storage.StorageProvider* *attribute*), 102  
`_linked_sample()` (*deeplake.core.tensor.Tensor* *method*), 156  
`_pop()` (*deeplake.core.tensor.Tensor* *method*), 156  
`_pop_from_cache()` (*deeplake.core.storage.LRUCache* *method*), 105  
`_set_hub_creds_info()` (*deeplake.core.storage.GCSProvider* *method*), 110  
`_set_hub_creds_info()` (*deeplake.core.storage.LocalProvider* *method*), 115  
`_set_hub_creds_info()` (*deeplake.core.storage.S3Provider* *method*), 108  
`_state_keys()` (*deeplake.core.storage.S3Provider* *method*), 109

## A

`add()` (*deeplake.core.vectorstore.deeplake\_vectorstore.VectorStore* *method*), 89  
`add_creds_key()` (*deeplake.core.dataset.Dataset* *method*), 123  
`add_creds_key()` (*deeplake.core.dataset.DeepLakeCloudDataset* *method*), 149  
`AgreementError` (*class in deeplake.util.exceptions*), 176  
`AgreementNotAcceptedError` (*class in deeplake.util.exceptions*), 176  
`allow_delete` (*deeplake.core.dataset.Dataset* *property*), 123  
`append()` (*deeplake.core.dataset.Dataset* *method*), 123  
`append()` (*deeplake.core.tensor.Tensor* *method*), 156  
`apply()` (*deeplake.core.index.Index* *method*), 120  
`apply_squeeze()` (*deeplake.core.index.Index* *method*), 120  
`array` (*deeplake.core.sample.Sample* *property*), 99  
`AuthenticationException` (*class in deeplake.util.exceptions*), 171  
`AuthorizationException` (*class in deeplake.util.exceptions*), 171

AutoCompressionError (class in *deeplake.util.exceptions*), 170

## B

BadGatewayException (class in *deeplake.util.exceptions*), 172  
 BadRequestException (class in *deeplake.util.exceptions*), 172  
 base\_htype (*deeplake.core.tensor.Tensor* property), 156  
 batch() (*deeplake.enterprise.dataloader.DeepLakeDataLoader* method), 43  
 branch (*deeplake.core.dataset.Dataset* property), 124  
 branches (*deeplake.core.dataset.Dataset* property), 124  
 BufferError (class in *deeplake.util.exceptions*), 176

## C

cancel() (*deeplake.core.vectorstore.deep\_memory.deep\_memory.DeepMemory* method), 183  
 check\_readonly() (*deeplake.core.storage.StorageProvider* method), 102  
 checkout() (*deeplake.core.dataset.Dataset* method), 124  
 checkout() (*deeplake.core.vectorstore.deeplake\_vectorstore.VectorStore* method), 91  
 CheckoutError (class in *deeplake.util.exceptions*), 175  
 ChunkEngineError (class in *deeplake.util.exceptions*), 175  
 ChunkIdEncoderError (class in *deeplake.util.exceptions*), 175  
 ChunkSizeTooSmallError (class in *deeplake.util.exceptions*), 175  
 clear() (*deeplake.api.info.Info* method), 165  
 clear() (*deeplake.core.storage.GCSProvider* method), 110  
 clear() (*deeplake.core.storage.GDriveProvider* method), 112  
 clear() (*deeplake.core.storage.LocalProvider* method), 115  
 clear() (*deeplake.core.storage.LRUCache* method), 105  
 clear() (*deeplake.core.storage.MemoryProvider* method), 117  
 clear() (*deeplake.core.storage.S3Provider* method), 109  
 clear() (*deeplake.core.storage.StorageProvider* method), 102  
 clear() (*deeplake.core.tensor.Tensor* method), 156  
 clear\_cache() (*deeplake.core.dataset.Dataset* method), 125  
 clear\_cache() (*deeplake.core.storage.LRUCache* method), 105  
 clear\_deeplake\_objects() (*deeplake.core.storage.LRUCache* method), 105  
 client (*deeplake.core.dataset.Dataset* property), 125  
 client (*deeplake.core.dataset.DeepLakeCloudDataset* property), 149  
 close() (*deeplake.enterprise.dataloader.DeepLakeDataLoader* method), 43  
 commit() (*deeplake.core.dataset.Dataset* method), 125  
 commit() (*deeplake.core.vectorstore.deeplake\_vectorstore.VectorStore* method), 91  
 commit\_id (*deeplake.core.dataset.Dataset* property), 126  
 CommitError (class in *deeplake.util.exceptions*), 175  
 commits (*deeplake.core.dataset.Dataset* property), 126  
 compose() (in module *deeplake*), 86  
 compose\_at() (*deeplake.core.index.Index* method), 120  
 compressed\_bytes() (*deeplake.core.sample.Sample* method), 100  
 compute() (in module *deeplake*), 84  
 connect() (*deeplake.api.dataset.dataset* static method), 164  
 connect() (*deeplake.core.dataset.Dataset* method), 126  
 connect() (*deeplake.core.dataset.DeepLakeCloudDataset* method), 149  
 connect() (in module *deeplake*), 80  
 copy() (*deeplake.api.dataset.dataset* static method), 163  
 copy() (*deeplake.core.dataset.Dataset* method), 126  
 copy() (*deeplake.core.storage.StorageProvider* method), 102  
 copy() (in module *deeplake*), 79  
 CorruptedMetaError (class in *deeplake.util.exceptions*), 174  
 CorruptedSampleError (class in *deeplake.util.exceptions*), 175  
 CouldNotCreateNewDatasetException (class in *deeplake.util.exceptions*), 172  
 create\_group() (*deeplake.core.dataset.Dataset* method), 127  
 create\_read\_copy\_dataset() (in module *deeplake.util.remove\_cache*), 169  
 create\_tensor() (*deeplake.core.dataset.Dataset* method), 128  
 create\_tensor\_like() (*deeplake.core.dataset.Dataset* method), 129  
 creds\_key() (*deeplake.core.tensor.Tensor* method), 156

## D

data() (*deeplake.core.tensor.Tensor* method), 157  
 DataFrame (class in *deeplake.auto.structured.dataframe*), 167  
 dataloader() (*deeplake.core.dataset.Dataset* method), 130  
 dataset (class in *deeplake.api.dataset*), 163  
 Dataset (class in *deeplake.core.dataset*), 123  
 dataset (*deeplake.core.vectorstore.deeplake\_vectorstore.VectorStore* property), 91

- dataset() (in module deeplake), 61
- DatasetHandlerError (class in deeplake.util.exceptions), 175
- DatasetUnsupportedPytorch (class in deeplake.util.exceptions), 174
- DatasetViewSavingError (class in deeplake.util.exceptions), 176
- deepcopy() (deeplake.api.dataset.dataset static method), 163
- deepcopy() (in module deeplake), 79
- deeplake module, 61
- deeplake.api.info module, 165
- deeplake.integrations.wandb.wandb module, 39
- DeepLakeCloudDataset (class in deeplake.core.dataset), 149
- DeepLakeDataLoader (class in deeplake.enterprise.data\_loader), 43
- DeepLakeRandom (class in deeplake.core.seed), 189
- DeepMemory (class in deeplake.core.vectorstore.deep\_memory.deep\_memory), 183
- delete() (deeplake.api.dataset.dataset static method), 163
- delete() (deeplake.core.dataset.Dataset method), 131
- delete() (deeplake.core.dataset.DeepLakeCloudDataset method), 150
- delete() (deeplake.core.dataset.ViewEntry method), 152
- delete() (deeplake.core.vectorstore.deep\_memory.deep\_memory.DeepMemory method), 183
- delete() (deeplake.core.vectorstore.deeplake\_vectorstore.VectorStore method), 91
- delete() (in module deeplake), 77
- delete\_branch() (deeplake.core.dataset.Dataset method), 131
- delete\_by\_path() (deeplake.core.vectorstore.deeplake\_vectorstore.VectorStore static method), 92
- delete\_group() (deeplake.core.dataset.Dataset method), 132
- delete\_tensor() (deeplake.core.dataset.Dataset method), 132
- delete\_view() (deeplake.core.dataset.Dataset method), 133
- dict() (deeplake.core.tensor.Tensor method), 157
- diff() (deeplake.core.dataset.Dataset method), 133
- DirectoryAtPathException (class in deeplake.util.exceptions), 171
- disable\_readonly() (deeplake.core.storage.StorageProvider method), 102
- download\_kaggle\_dataset() (in module deeplake.auto.unstructured.kaggle), 168
- downsample() (deeplake.core.index.Index method), 121
- downsample() (deeplake.core.index.IndexEntry method), 118
- dtype (deeplake.core.tensor.Tensor property), 157
- DynamicTensorNumpyError (class in deeplake.util.exceptions), 171
- ## E
- empty() (deeplake.api.dataset.dataset static method), 163
- empty() (in module deeplake), 64
- EmptyCommitError (class in deeplake.util.exceptions), 175
- EmptyTensorError (class in deeplake.util.exceptions), 176
- EmptyTokenException (class in deeplake.util.exceptions), 172
- enable\_readonly() (deeplake.core.storage.StorageProvider method), 102
- eval() (deeplake.core.transform.Pipeline method), 179
- evaluate() (deeplake.core.vectorstore.deep\_memory.deep\_memory.DeepMemory method), 184
- exists() (deeplake.api.dataset.dataset static method), 163
- exists() (in module deeplake), 81
- extend() (deeplake.core.dataset.Dataset method), 133
- extend() (deeplake.core.tensor.Tensor method), 157
- ExternalCommandError (class in deeplake.util.exceptions), 170
- ## F
- FileAtPathException (class in deeplake.util.exceptions), 171
- find\_dataset() (deeplake.auto.structured.dataframe.DataFrame method), 167
- filter() (deeplake.core.dataset.Dataset method), 134
- FilterError (class in deeplake.util.exceptions), 174
- find\_axis() (deeplake.core.index.Index method), 121
- fix\_vcc() (deeplake.core.dataset.Dataset method), 135
- flush() (deeplake.core.dataset.Dataset method), 135
- flush() (deeplake.core.storage.LRUCache method), 105
- flush() (deeplake.core.storage.StorageProvider method), 102
- FullChunkError (class in deeplake.util.exceptions), 175
- ## G
- GatewayTimeoutException (class in deeplake.util.exceptions), 172
- GCSDefaultCredsNotFoundError (class in deeplake.util.exceptions), 175
- GCSProvider (class in deeplake.core.storage), 109
- GDriveProvider (class in deeplake.core.storage), 111
- get() (deeplake.api.info.Info method), 165
- get\_base\_storage() (in module deeplake.util.remove\_cache), 169



`get_bytes()` (*deeplake.core.storage.GCSProvider* method), 111  
`get_bytes()` (*deeplake.core.storage.LocalProvider* method), 115  
`get_bytes()` (*deeplake.core.storage.LRUCache* method), 105  
`get_bytes()` (*deeplake.core.storage.S3Provider* method), 109  
`get_bytes()` (*deeplake.core.storage.StorageProvider* method), 102  
`get_commit_details()` (*deeplake.core.dataset.Dataset* method), 135  
`get_creds_keys()` (*deeplake.core.dataset.Dataset* method), 135  
`get_dataset_with_zero_size_cache()` (in module *deeplake.util.remove\_cache*), 169  
`get_deeplake_object()` (*deeplake.core.storage.LRUCache* method), 106  
`get_items()` (*deeplake.core.storage.LRUCache* method), 106  
`get_managed_creds_keys()` (*deeplake.core.dataset.Dataset* method), 135  
`get_managed_creds_keys()` (*deeplake.core.dataset.DeepLakeCloudDataset* method), 150  
`get_model()` (*deeplake.core.vectorstore.deep\_memory.deeplake\_vectorstore* method), 185  
`get_seed()` (*deeplake.core.seed.DeeplakeRandom* method), 189  
`get_view()` (*deeplake.core.dataset.Dataset* method), 135  
`get_views()` (*deeplake.core.dataset.Dataset* method), 136  
`groups` (*deeplake.core.dataset.Dataset* property), 136  
**H**  
`has_head_changes` (*deeplake.core.dataset.Dataset* property), 136  
`hidden` (*deeplake.core.tensor.Tensor* property), 158  
`htype` (*deeplake.core.tensor.Tensor* property), 158  
`HubComposeEmptyListError` (class in *deeplake.util.exceptions*), 174  
`HubComposeIncompatibleFunction` (class in *deeplake.util.exceptions*), 174  
**I**  
`id` (*deeplake.core.dataset.ViewEntry* property), 152  
`ImageClassification` (class in *deeplake.auto.unstructured.image\_classification*), 168  
`Index` (class in *deeplake.core.index*), 119  
`IndexEntry` (class in *deeplake.core.index*), 118  
`indices()` (*deeplake.core.index.IndexEntry* method), 118  
`Info` (class in *deeplake.api.info*), 165  
`info` (*deeplake.core.dataset.Dataset* property), 136  
`info` (*deeplake.core.tensor.Tensor* property), 158  
`InfoError` (class in *deeplake.util.exceptions*), 176  
`ingest_classification()` (*deeplake.api.dataset.dataset* static method), 164  
`ingest_classification()` (in module *deeplake*), 66  
`ingest_coco()` (*deeplake.api.dataset.dataset* static method), 164  
`ingest_coco()` (in module *deeplake*), 68  
`ingest_dataframe()` (*deeplake.api.dataset.dataset* static method), 164  
`ingest_dataframe()` (in module *deeplake*), 72  
`ingest_huggingface()` (in module *deeplake*), 74  
`ingest_kaggle()` (*deeplake.api.dataset.dataset* static method), 164  
`ingest_kaggle()` (in module *deeplake*), 71  
`ingest_yolo()` (in module *deeplake*), 69  
`invalidate_libdeeplake_dataset()` (*deeplake.core.tensor.Tensor* method), 158  
`InvalidBytesRequestedError` (class in *deeplake.util.exceptions*), 171  
`InvalidFileExtension` (class in *deeplake.util.exceptions*), 170  
`InvalidHubPathException` (class in *deeplake.util.exceptions*), 171  
`InvalidImageDimensions` (class in *deeplake.util.exceptions*), 173  
`InvalidInputDataError` (class in *deeplake.util.exceptions*), 174  
`InvalidKeyTypeError` (class in *deeplake.util.exceptions*), 171  
`InvalidOperationError` (class in *deeplake.util.exceptions*), 175  
`InvalidOutputDatasetError` (class in *deeplake.util.exceptions*), 174  
`InvalidPasswordException` (class in *deeplake.util.exceptions*), 172  
`InvalidPathException` (class in *deeplake.util.exceptions*), 170  
`InvalidShapeIntervalError` (class in *deeplake.util.exceptions*), 171  
`InvalidTensorGroupNameError` (class in *deeplake.util.exceptions*), 171  
`InvalidTensorLinkError` (class in *deeplake.util.exceptions*), 174  
`InvalidTensorNameError` (class in *deeplake.util.exceptions*), 171  
`InvalidTokenException` (class in *deeplake.util.exceptions*), 176

- InvalidTransformDataset (class in *LRUCache* (class in *deeplake.core.storage*), 103  
*deeplake.util.exceptions*), 174
- is\_actually\_cloud(*deeplake.core.dataset.DeepLakeCloudDataset* property), 150
- is\_colab() (in module *deeplake.util.notebook*), 170
- is\_dynamic(*deeplake.core.tensor.Tensor* property), 158
- is\_head\_node(*deeplake.core.dataset.Dataset* property), 136
- is\_jupyter() (in module *deeplake.util.notebook*), 170
- is\_link(*deeplake.core.tensor.Tensor* property), 158
- is\_notebook() (in module *deeplake.util.notebook*), 170
- is\_sequence(*deeplake.core.tensor.Tensor* property), 158
- is\_trivial()(*deeplake.core.index.Index* method), 121
- is\_trivial()(*deeplake.core.index.IndexEntry* method), 119
- is\_view(*deeplake.core.dataset.Dataset* property), 136
- items()(*deeplake.api.info.Info* method), 165
- ## K
- KaggleDatasetAlreadyDownloadedError (class in *deeplake.util.exceptions*), 170
- KaggleError (class in *deeplake.util.exceptions*), 170
- KaggleMissingCredentialsError (class in *deeplake.util.exceptions*), 170
- keys()(*deeplake.api.info.Info* method), 165
- ## L
- length()(*deeplake.core.index.Index* method), 121
- length()(*deeplake.core.index.IndexEntry* method), 119
- like()(*deeplake.api.dataset.dataset* static method), 163
- like() (in module *deeplake*), 65
- link() (in module *deeplake*), 83
- link() (in module *deeplake.api.link*), 165
- link\_tiled() (in module *deeplake*), 83
- link\_tiled() (in module *deeplake.api.link\_tiled*), 166
- LinkedSample (class in *deeplake.core.linked\_sample*), 100
- LinkedTiledSample (class in *deeplake.core.linked\_tiled\_sample*), 101
- list()(*deeplake.api.dataset.dataset* static method), 164
- list()(*deeplake.core.tensor.Tensor* method), 158
- list\_jobs()(*deeplake.core.vectorstore.deep\_memory.deep\_memory.DeepMemory* method), 185
- load()(*deeplake.api.dataset.dataset* static method), 163
- load()(*deeplake.core.dataset.ViewEntry* method), 152
- load() (in module *deeplake*), 75
- load\_view()(*deeplake.core.dataset.Dataset* method), 136
- LocalProvider (class in *deeplake.core.storage*), 113
- LockedException (class in *deeplake.util.exceptions*), 172
- log()(*deeplake.core.dataset.Dataset* method), 137
- LoginException (class in *deeplake.util.exceptions*), 171
- ## M
- ManagedCredentialsNotFoundError (class in *deeplake.util.exceptions*), 176
- max\_len(*deeplake.core.dataset.Dataset* property), 137
- max\_view(*deeplake.core.dataset.Dataset* property), 137
- maybe\_flush()(*deeplake.core.storage.StorageProvider* method), 103
- MemoryDatasetCanNotBePickledError (class in *deeplake.util.exceptions*), 175
- MemoryProvider (class in *deeplake.core.storage*), 116
- merge()(*deeplake.core.dataset.Dataset* method), 137
- merge\_slices() (in module *deeplake.core.index*), 121
- MergeConflictError (class in *deeplake.util.exceptions*), 175
- MergeError (class in *deeplake.util.exceptions*), 175
- MergeMismatchError (class in *deeplake.util.exceptions*), 175
- MergeNotSupportedError (class in *deeplake.util.exceptions*), 175
- message(*deeplake.core.dataset.ViewEntry* property), 152
- meta(*deeplake.core.dataset.Dataset* property), 138
- meta(*deeplake.core.tensor.Tensor* property), 159
- MetaAlreadyExistsError (class in *deeplake.util.exceptions*), 173
- MetaDoesNotExistError (class in *deeplake.util.exceptions*), 173
- MetaError (class in *deeplake.util.exceptions*), 173
- MetaInvalidKey (class in *deeplake.util.exceptions*), 173
- MetaInvalidRequiredMetaKey (class in *deeplake.util.exceptions*), 173
- min\_len(*deeplake.core.dataset.Dataset* property), 138
- min\_view(*deeplake.core.dataset.Dataset* property), 138
- modified\_samples()(*deeplake.core.tensor.Tensor* method), 159
- module
- deeplake*, 61
  - deeplake.api.info*, 165
  - deeplake.integrations.wandb.wandb*, 39
- ModuleNotInstalledException (class in *deeplake.util.exceptions*), 171
- ## N
- nbytes(*deeplake.api.info.Info* property), 165
- ndim(*deeplake.core.tensor.Tensor* property), 159
- need\_to\_reload\_creds()(*deeplake.core.storage.S3Provider* method), 109
- no\_view\_dataset(*deeplake.core.dataset.Dataset* property), 139
- num\_samples(*deeplake.core.dataset.Dataset* property), 139

[num\\_samples](#) (*deeplake.core.tensor.Tensor* property), [159](#)  
[numpy\(\)](#) (*deeplake.core.tensor.Tensor* method), [159](#)  
[numpy\(\)](#) (*deeplake.enterprise.dataloader.DeepLakeDataLoader* method), [43](#)  
**O**  
[offset\(\)](#) (*deeplake.enterprise.dataloader.DeepLakeDataLoader* method), [44](#)  
[optimize\(\)](#) (*deeplake.core.dataset.ViewEntry* method), [152](#)  
[OutOfChunkCountError](#) (class in *deeplake.util.exceptions*), [176](#)  
[OutOfSampleCountError](#) (class in *deeplake.util.exceptions*), [176](#)  
[OverLimitException](#) (class in *deeplake.util.exceptions*), [172](#)  
**P**  
[parent](#) (*deeplake.core.dataset.Dataset* property), [139](#)  
[PartialSample](#) (class in *deeplake.core.partial\_sample*), [101](#)  
[path\(\)](#) (*deeplake.core.tensor.Tensor* method), [160](#)  
[PathNotEmptyException](#) (class in *deeplake.util.exceptions*), [171](#)  
[pending\\_commit\\_id](#) (*deeplake.core.dataset.Dataset* property), [139](#)  
[pil](#) (*deeplake.core.sample.Sample* property), [100](#)  
[Pipeline](#) (class in *deeplake.core.transform*), [179](#)  
[play\(\)](#) (*deeplake.core.tensor.Tensor* method), [160](#)  
[pop\(\)](#) (*deeplake.api.info.Info* method), [165](#)  
[pop\(\)](#) (*deeplake.core.dataset.Dataset* method), [139](#)  
[pop\(\)](#) (*deeplake.core.tensor.Tensor* method), [160](#)  
[popitem\(\)](#) (*deeplake.api.info.Info* method), [165](#)  
[populate\\_creds\(\)](#) (*deeplake.core.dataset.Dataset* method), [139](#)  
[ProviderListEmptyError](#) (class in *deeplake.util.exceptions*), [171](#)  
[ProviderSizeListMismatch](#) (class in *deeplake.util.exceptions*), [171](#)  
[pytorch\(\)](#) (*deeplake.core.dataset.Dataset* method), [140](#)  
[pytorch\(\)](#) (*deeplake.enterprise.dataloader.DeepLakeDataLoader* method), [44](#)  
**Q**  
[query\(\)](#) (*deeplake.core.dataset.Dataset* method), [141](#)  
[query\(\)](#) (*deeplake.enterprise.dataloader.DeepLakeDataLoader* method), [46](#)  
**R**  
[random\\_split\(\)](#) (*deeplake.core.dataset.Dataset* method), [142](#)  
[read\(\)](#) (in module *deeplake*), [81](#)  
[read\(\)](#) (in module *deeplake.api.read*), [165](#)  
[read\\_only](#) (*deeplake.core.dataset.Dataset* property), [143](#)  
[ReadOnlyModeError](#) (class in *deeplake.util.exceptions*), [174](#)  
[rechunk\(\)](#) (*deeplake.core.dataset.Dataset* method), [143](#)  
[register\\_deeplake\\_object\(\)](#) (*deeplake.core.storage.LRUCache* method), [106](#)  
[remove\\_deeplake\\_object\(\)](#) (*deeplake.core.storage.LRUCache* method), [106](#)  
[remove\\_memory\\_cache\(\)](#) (in module *deeplake.util.remove\_cache*), [169](#)  
[rename\(\)](#) (*deeplake.api.dataset.dataset* static method), [163](#)  
[rename\(\)](#) (*deeplake.core.dataset.Dataset* method), [143](#)  
[rename\(\)](#) (*deeplake.core.dataset.DeepLakeCloudDataset* method), [150](#)  
[rename\(\)](#) (*deeplake.core.storage.GCSProvider* method), [111](#)  
[rename\(\)](#) (*deeplake.core.storage.LocalProvider* method), [116](#)  
[rename\(\)](#) (*deeplake.core.storage.S3Provider* method), [109](#)  
[rename\(\)](#) (in module *deeplake*), [78](#)  
[rename\\_group\(\)](#) (*deeplake.core.dataset.Dataset* method), [144](#)  
[rename\\_tensor\(\)](#) (*deeplake.core.dataset.Dataset* method), [144](#)  
[RenameError](#) (class in *deeplake.util.exceptions*), [176](#)  
[replace\\_with\(\)](#) (*deeplake.api.info.Info* method), [165](#)  
[reset\(\)](#) (*deeplake.core.dataset.Dataset* method), [144](#)  
[ResourceNotFoundException](#) (class in *deeplake.util.exceptions*), [172](#)  
[root](#) (*deeplake.core.dataset.Dataset* property), [144](#)  
**S**  
[S3DeletionError](#) (class in *deeplake.util.exceptions*), [173](#)  
[S3Error](#) (class in *deeplake.util.exceptions*), [172](#)  
[S3GetError](#) (class in *deeplake.util.exceptions*), [172](#)  
[S3ListError](#) (class in *deeplake.util.exceptions*), [173](#)  
[S3Provider](#) (class in *deeplake.core.storage*), [106](#)  
[S3SetError](#) (class in *deeplake.util.exceptions*), [172](#)  
[SamePathException](#) (class in *deeplake.util.exceptions*), [170](#)  
[Sample](#) (class in *deeplake.core.sample*), [99](#)  
[sample\\_by\(\)](#) (*deeplake.core.dataset.Dataset* method), [145](#)  
[sample\\_by\(\)](#) (*deeplake.enterprise.dataloader.DeepLakeDataLoader* method), [46](#)  
[sample\\_indices](#) (*deeplake.core.dataset.Dataset* property), [145](#)

- sample\_indices (*deeplake.core.tensor.Tensor* property), 160
- sample\_info (*deeplake.core.tensor.Tensor* property), 160
- SampleCompressionError (class in *deeplake.util.exceptions*), 173
- SampleDecompressionError (class in *deeplake.util.exceptions*), 173
- SampleHtypeMismatchError (class in *deeplake.util.exceptions*), 176
- save\_view() (*deeplake.core.dataset.Dataset* method), 145
- search() (*deeplake.core.vectorstore.deeplake\_vectorstore.VectorStore* method), 93
- seed() (*deeplake.core.seed.DeeplakeRandom* method), 189
- ServerException (class in *deeplake.util.exceptions*), 172
- set\_bytes() (*deeplake.core.storage.StorageProvider* method), 103
- set\_model() (*deeplake.core.vectorstore.deep\_memory.deep\_memory.DeepMemory* method), 185
- set\_token() (*deeplake.core.dataset.Dataset* method), 147
- setdefault() (*deeplake.api.info.Info* method), 165
- shape (*deeplake.core.tensor.Tensor* property), 160
- shape\_interval (*deeplake.core.tensor.Tensor* property), 161
- ShapeInterval (class in *deeplake.util.shape\_interval*), 169
- shapes() (*deeplake.core.tensor.Tensor* method), 161
- shuffle() (*deeplake.enterprise.dataloader.DeepLakeDataLoader* method), 47
- size\_approx() (*deeplake.core.dataset.Dataset* method), 147
- slice\_at\_int() (in module *deeplake.core.index*), 121
- slice\_length() (in module *deeplake.core.index*), 122
- status() (*deeplake.core.vectorstore.deep\_memory.deep\_memory.DeepMemory* method), 185
- StorageProvider (class in *deeplake.core.storage*), 101
- structure() (*deeplake.auto.unstructured.image\_classification.ImageClassification* method), 168
- StructuredDataset (class in *deeplake.auto.structured.base*), 167
- subscriptable() (*deeplake.core.index.IndexEntry* method), 119
- summary() (*deeplake.core.dataset.Dataset* method), 147
- summary() (*deeplake.core.tensor.Tensor* method), 161
- summary() (*deeplake.core.vectorstore.deeplake\_vectorstore.VectorStore* method), 95
- sync() (*deeplake.core.storage.GDriveProvider* method), 112
- ## T
- Tensor (class in *deeplake.core.tensor*), 155
- TensorAlreadyExistsError (class in *deeplake.util.exceptions*), 170
- TensorDoesNotExistError (class in *deeplake.util.exceptions*), 170
- TensorDtypeMismatchError (class in *deeplake.util.exceptions*), 174
- tensorflow() (*deeplake.core.dataset.Dataset* method), 147
- tensorflow() (*deeplake.enterprise.dataloader.DeepLakeDataLoader* method), 47
- TensorGroupAlreadyExistsError (class in *deeplake.util.exceptions*), 170
- TensorGroupDoesNotExistError (class in *deeplake.util.exceptions*), 170
- TensorInvalidSampleShapeError (class in *deeplake.util.exceptions*), 170
- TensorMetaInvalidHtype (class in *deeplake.util.exceptions*), 173
- TensorMetaInvalidHtypeOverwriteKey (class in *deeplake.util.exceptions*), 173
- TensorMetaInvalidHtypeOverwriteValue (class in *deeplake.util.exceptions*), 173
- TensorMetaMissingKey (class in *deeplake.util.exceptions*), 170
- TensorMetaMissingRequiredValue (class in *deeplake.util.exceptions*), 173
- TensorMetaMutuallyExclusiveKeysError (class in *deeplake.util.exceptions*), 174
- TensorMismatchError (class in *deeplake.util.exceptions*), 174
- TensorModifiedError (class in *deeplake.util.exceptions*), 175
- tensors (*deeplake.core.dataset.Dataset* property), 147
- tensors() (*deeplake.core.vectorstore.deeplake\_vectorstore.VectorStore* method), 95
- TensorUnsupportedSampleType (class in *deeplake.util.exceptions*), 173
- text() (*deeplake.core.tensor.Tensor* method), 161
- to\_bytes() (in module *deeplake*), 84
- tiled() (in module *deeplake.api.tiled*), 166
- timestamps (*deeplake.core.tensor.Tensor* property), 162
- to\_bytes() (*deeplake.core.tensor.Tensor* method), 162
- token (*deeplake.core.dataset.Dataset* property), 147
- token (*deeplake.core.dataset.DeepLakeCloudDataset* property), 151
- TokenPermissionError (class in *deeplake.util.exceptions*), 176
- train() (*deeplake.core.vectorstore.deep\_memory.deep\_memory.DeepMemory* method), 186
- transform() (*deeplake.enterprise.dataloader.DeepLakeDataLoader* method), 48
- TransformError (class in *deeplake.util.exceptions*), 174



## U

`UnableToReadFromUrlError` (class in *deeplake.util.exceptions*), 176  
`uncompressed_bytes()` (*deeplake.core.sample.Sample* method), 100  
`UnexpectedStatusCodeException` (class in *deeplake.util.exceptions*), 172  
`UnstructuredDataset` (class in *deeplake.auto.unstructured.base*), 167  
`UnsupportedCompressionError` (class in *deeplake.util.exceptions*), 173  
`UnsupportedSchedulerError` (class in *deeplake.util.exceptions*), 174  
`UnsupportedTensorTypeError` (class in *deeplake.util.exceptions*), 171  
`update()` (*deeplake.api.info.Info* method), 165  
`update()` (*deeplake.core.dataset.Dataset* method), 148  
`update_creds_key()` (*deeplake.core.dataset.Dataset* method), 148  
`update_creds_key()` (*deeplake.core.dataset.DeepLakeCloudDataset* method), 151  
`update_embedding()` (*deeplake.core.vectorstore.deeplake\_vectorstore.VectorStore* method), 95  
`UserNotLoggedInException` (class in *deeplake.util.exceptions*), 171

## V

`validate()` (*deeplake.core.index.Index* method), 121  
`validate()` (*deeplake.core.index.IndexEntry* method), 119  
`values()` (*deeplake.api.info.Info* method), 165  
`VectorStore` (class in *deeplake.core.vectorstore.deeplake\_vectorstore*), 87  
`verify` (*deeplake.core.tensor.Tensor* property), 162  
`VersionControlError` (class in *deeplake.util.exceptions*), 175  
`ViewEntry` (class in *deeplake.core.dataset*), 152  
`visualize()` (*deeplake.core.dataset.Dataset* method), 149  
`visualize()` (*deeplake.core.dataset.DeepLakeCloudDataset* method), 151

## W

`WaitTimeoutException` (class in *deeplake.util.exceptions*), 172