class **Data** (**x**: Optional[Tensor] = **None**, **edge_index**: Optional[Tensor] = **None**, **edge_attr**: Optional[Tensor] = **None**, **y**: Optional[Union[Tensor, int, float]] = **None**, **pos**: Optional[Tensor] = **None**, **time**: Optional[Tensor] = **None**, ****kwargs**) [source]

Bases: `BaseData` , `FeatureStore` , `GraphStore`

A data object describing a homogeneous graph. The data object can hold node-level, link-level and graph-level attributes. In general, `Data` tries to mimic the behavior of a regular Python dictionary. In addition, it provides useful functionality for analyzing graph structures, and provides basic PyTorch tensor functionalities. See here for the accompanying tutorial.

```
from torch_geometric.data import Data

data = Data(x=x, edge_index=edge_index, ...)

# Add additional arguments to `data`:
data.train_idx = torch.tensor([...], dtype=torch.long)
data.test_mask = torch.tensor([...], dtype=torch.bool)

# Analyzing the graph structure:
data.num_nodes
>>> 23

data.is_directed()
>>> False

# PyTorch tensor functionality:
data = data.pin_memory()
data = data.to('cuda:0', non_blocking=True)
```

PARAMETERS:

- **x** (*torch.Tensor*, *optional*) – Node feature matrix with shape `[num_nodes, num_node_features]` . (default: `None`)
- **edge_index** (*LongTensor*, *optional*) – Graph connectivity in COO format with shape `[2, num_edges]` . (default: `None`)
- **edge_attr** (*torch.Tensor*, *optional*) – Edge feature matrix with shape `[num_edges, num_edge_features]` . (default: `None`)

- **y** (*torch.Tensor*, *optional*) – Graph-level or node-level ground-truth labels with arbitrary shape. (default: `None`)
- **pos** (*torch.Tensor*, *optional*) – Node position matrix with shape `[num_nodes, num_dimensions]`. (default: `None`)
- **time** (*torch.Tensor*, *optional*) – The timestamps for each event with shape `[num_edges]` or `[num_nodes]`. (default: `None`)
- ****kwargs** (*optional*) – Additional attributes.

property **num_nodes**: *Optional[int]*

Returns the number of nodes in the graph.

> **Note**
>
> The number of nodes in the data object is automatically inferred in case node-level attributes are present, *e.g.*, `data.x`. In some cases, however, a graph may only be given without any node-level attributes. PyG then *guesses* the number of nodes according to `edge_index.max().item() + 1`. However, in case there exists isolated nodes, this number does not have to be correct which can result in unexpected behavior. Thus, we recommend to set the number of nodes in your data object explicitly via `data.num_nodes = ...`. You will be given a warning that requests you to do so.

RETURN TYPE:

`Optional [ int ]`

to_dict ( ) → Dict[str, Any] [source]

Returns a dictionary of stored key/value pairs.

RETURN TYPE:

`Dict [ str , Any ]`

to_namedtuple ( ) → NamedTuple [source]

Returns a `NamedTuple` of stored key/value pairs.

RETURN TYPE:

`NamedTuple`

update (**data**: Union[Self, Dict[str, Any]]) → Self [source]

Updates the data object with the elements from another data object. Add override existing ones (in case of duplicates).

RETURN TYPE:

`Self`

__cat_dim__ (**key**: str, **value**: Any, *__args__, **__kwargs__) → Any [source]

Returns the dimension for which the value `value` of the attribute `key` will get concatenated when creating mini-batches using `torch_geometric.loader.DataLoader`.

> **Note**
>
> This method is for internal use only, and should only be overridden in case the mini-batch creation process is corrupted for a specific attribute.

RETURN TYPE:

`Any`

__inc__ (**key**: str, **value**: Any, *__args__, **__kwargs__) → Any [source]

Returns the incremental count to cumulatively increase the value `value` of the attribute `key` when creating mini-batches using `torch_geometric.loader.DataLoader`.

> **Note**
>
> This method is for internal use only, and should only be overridden in case the mini-batch creation process is corrupted for a specific attribute.

RETURN TYPE:

`Any`

validate (**raise_on_error**: bool = `True`) → bool [source]

Validates the correctness of the data.

RETURN TYPE:

`bool`

is_node_attr (**key**: str) → bool [source]

Returns `True` if the object at key `key` denotes a node-level tensor attribute.

RETURN TYPE:

`bool`

is_edge_attr (**key**: str) → bool [source]

Returns `True` if the object at key `key` denotes an edge-level tensor attribute.

RETURN TYPE:

`bool`

subgraph (**subset**: Tensor) → Self [source]

Returns the induced subgraph given by the node indices `subset` .

PARAMETERS:

subset (*LongTensor or BoolTensor*) – The nodes to keep.

RETURN TYPE:

`Self`

edge_subgraph (**subset**: Tensor) → Self [source]

Returns the induced subgraph given by the edge indices `subset` . Will currently preserve all the nodes in the graph, even if they are isolated after subgraph computation.

PARAMETERS:

subset (*LongTensor or BoolTensor*) – The edges to keep.

RETURN TYPE:

`Self`

to_heterogeneous (**node_type**: Optional[Tensor] = None, **edge_type**: Optional[Tensor] = None, **node_type_names**: Optional[List[str]] = None, **edge_type_names**: Optional[List[Tuple[str, str, str]]] = None) [source]

Converts a `Data` object to a heterogeneous `HeteroData` object. For this, node and edge attributes are splitted according to the node-level and edge-level vectors `node_type` and `edge_type` , respectively. `node_type_names` and `edge_type_names` can be used to give meaningful node and edge type names, respectively. That is, the node_type `0` is given by `node_type_names[0]` . If the `Data` object was constructed via `to_homogeneous()` , the object can be reconstructed without any need to pass in additional arguments.

PARAMETERS:

- **node_type** (*torch.Tensor*, *optional*) – A node-level vector denoting the type of each node. (default: `None`)
- **edge_type** (*torch.Tensor*, *optional*) – An edge-level vector denoting the type of each edge. (default: `None`)
- **node_type_names** (*List[str]*, *optional*) – The names of node types. (default: `None`)
- **edge_type_names** (*List[Tuple[str, str, str]]*, *optional*) – The names of edge types. (default: `None`)

classmethod **from_dict** (**mapping**: Dict[str, Any]) → Self [source]

Creates a `Data` object from a dictionary.

RETURN TYPE:

`Self`

property **num_node_features**: *int*

Returns the number of features per node in the graph.

RETURN TYPE:

`int`

property **num_features**: *int*

Returns the number of features per node in the graph. Alias for `num_node_features`.

RETURN TYPE:

`int`

property **num_edge_features**: *int*

Returns the number of features per edge in the graph.

RETURN TYPE:

`int`

property **num_node_types**: *int*

Returns the number of node types in the graph.

RETURN TYPE:

`int`

property **num_edge_types**: *int*

Returns the number of edge types in the graph.

RETURN TYPE:

`int`

apply (**func**: Callable, ***args**: *str*)

Applies the function `func`, either to all attributes or only the ones given in `*args`.

apply_ (**func**: Callable, ***args**: *str*)

Applies the in-place function `func`, either to all attributes or only the ones given in `*args`.

clone (***args**: *str*)

Performs cloning of tensors, either for all attributes or only the ones given in `*args`.

coalesce ( ) → Self

Sorts and removes duplicated entries from edge indices `edge_index`.

RETURN TYPE:

`Self`

concat (**data**: Self) → Self

Concatenates `self` with another `data` object. All values needs to have matching shapes at non-concat dimensions.

RETURN TYPE:

`Self`

contiguous (***args**: *str*)

Ensures a contiguous memory layout, either for all attributes or only the ones given in `*args`.

coo (**edge_types**: Optional[List[Any]] = `None`, **store**: bool = `False`) → Tuple[Dict[Tuple[str, str, str], Tensor], Dict[Tuple[str, str, str], Tensor], Dict[Tuple[str, str, str], Optional[Tensor]]]

Returns the edge indices in the `GraphStore` in COO format.

PARAMETERS:

- **edge_types** (*List[Any]*, *optional*) – The edge types of edge indices to obtain. If set to `None`, will return the edge indices of all existing edge types. (default: `None`)
- **store** (*bool*, *optional*) – Whether to store converted edge indices in the `GraphStore`. (default: `False`)

RETURN TYPE:

`Tuple [ Dict [ Tuple [ str , str , str ], Tensor ], Dict [ Tuple [ str , str , str ], Tensor ], Dict [ Tuple [ str , str , str ], Optional [ Tensor ]]]`

cpu (***args**: *str*)

Copies attributes to CPU memory, either for all attributes or only the one

csc (**edge_types**: Optional[List[Any]] = None, **store**: bool = False) → Tuple[Dict[Tuple[str, str, str], Tensor], Dict[Tuple[str, str, str], Tensor], Dict[Tuple[str, str, str], Optional[Tensor]]]

Returns the edge indices in the `GraphStore` in CSC format.

PARAMETERS:

- **edge_types** (*List[Any]*, *optional*) – The edge types of edge indices to obtain. If set to `None`, will return the edge indices of all existing edge types. (default: `None`)
- **store** (*bool*, *optional*) – Whether to store converted edge indices in the `GraphStore`. (default: `False`)

RETURN TYPE:

`Tuple [ Dict [ Tuple [ str , str , str ], Tensor ], Dict [ Tuple [ str , str , str ], Tensor ], Dict [ Tuple [ str , str , str ], Optional [ Tensor ]]]`

csr (**edge_types**: Optional[List[Any]] = None, **store**: bool = False) → Tuple[Dict[Tuple[str, str, str], Tensor], Dict[Tuple[str, str, str], Tensor], Dict[Tuple[str, str, str], Optional[Tensor]]]

Returns the edge indices in the `GraphStore` in CSR format.

PARAMETERS:

- **edge_types** (*List[Any]*, *optional*) – The edge types of edge indices to obtain. If set to `None`, will return the edge indices of all existing edge types. (default: `None`)
- **store** (*bool*, *optional*) – Whether to store converted edge indices in the `GraphStore`. (default: `False`)

RETURN TYPE:

`Tuple [ Dict [ Tuple [ str , str , str ], Tensor ], Dict [ Tuple [ str , str , str ], Tensor ], Dict [ Tuple [ str , str , str ], Optional [ Tensor ]]]`

cuda (**device**: Optional[Union[int, str]] = None, ***args**: *str*, **non_blocking**: bool = False)

Copies attributes to CUDA memory, either for all attributes or only the ones given in `*args`.

detach (***args**: *str*)

Detaches attributes from the computation graph by creating a new tensor, either for all attributes or only the ones given in `*args`.

detach_ (***args**: *str*)

Detaches attributes from the computation graph, either for all attributes or only the ones given in `*args`.

edge_attrs ( ) → List[str]

Returns all edge-level tensor attribute names.

RETURN TYPE:

`List [ str ]`

`generate_ids ( )`

Generates and sets `n_id` and `e_id` attributes to assign each node and edge to a continuously ascending and unique ID.

`get_edge_index ( *args, **kwargs ) → Tuple[Tensor, Tensor]`

Synchronously obtains an `edge_index` tuple from the `GraphStore` .

PARAMETERS:

- ***args** – Arguments passed to `EdgeAttr` .
- ****kwargs** – Keyword arguments passed to `EdgeAttr` .

RAISES:

KeyError – If the `edge_index` corresponding to the input `EdgeAttr` was not found.

RETURN TYPE:

`Tuple [ Tensor , Tensor ]`

`get_tensor ( *args, convert_type: bool = False, **kwargs ) → Union[Tensor, ndarray]`

Synchronously obtains a `tensor` from the `FeatureStore` .

PARAMETERS:

- ***args** – Arguments passed to `TensorAttr` .
- **convert_type** (*bool*, *optional*) – Whether to convert the type of the output tensor to the type of the attribute index. (default: `False`)
- ****kwargs** – Keyword arguments passed to `TensorAttr` .

RAISES:

ValueError – If the input `TensorAttr` is not fully specified.

RETURN TYPE:

`Union [ Tensor , ndarray ]`

`get_tensor_size ( *args, **kwargs ) → Optional[Tuple[int, ...]]`

Obtains the size of a tensor given its `TensorAttr` , or `None` if the tensor does not exist.

RETURN TYPE:

`Optional [ Tuple [ int , ... ]]`

`has_isolated_nodes ( ) → bool`

Returns `True` if the graph contains isolated nodes.

RETURN TYPE:

bool

has_self_loops ( ) → bool

Returns `True` if the graph contains self-loops.

RETURN TYPE:

bool

is_coalesced ( ) → bool

Returns `True` if edge indices `edge_index` are sorted and do not contain duplicate entries.

RETURN TYPE:

bool

property **is_cuda**: *bool*

Returns `True` if any `torch.Tensor` attribute is stored on the GPU, `False` otherwise.

RETURN TYPE:

bool

is_directed ( ) → bool

Returns `True` if graph edges are directed.

RETURN TYPE:

bool

is_sorted (**sort_by_row**: bool = `True`) → bool

Returns `True` if edge indices `edge_index` are sorted.

PARAMETERS:

sort_by_row (*bool*, *optional*) – If set to `False` , will require column-wise order/by destination node order of `edge_index` . (default: `True`)

RETURN TYPE:

bool

is_sorted_by_time ( ) → bool

Returns `True` if `time` is sorted.

RETURN TYPE:

bool

is_undirected ( ) → bool

Returns `True` if graph edges are undirected.

RETURN TYPE:

`bool`

keys ( ) → List[str]

Returns a list of all graph attribute names.

RETURN TYPE:

`List [ str ]`

multi_get_tensor (**attrs**: List[TensorAttr], **convert_type**: bool = `False`) → List[Union[Tensor, ndarray]]

Synchronously obtains a list of tensors from the `FeatureStore` for each tensor associated with the attributes in `attrs` .

> **Note**
>
> The default implementation simply iterates over all calls to `get_tensor()` . Implementor classes that can provide additional, more performant functionality are recommended to to override this method.

PARAMETERS:

- **attrs** (*List[TensorAttr]*) – A list of input `TensorAttr` objects that identify the tensors to obtain.
- **convert_type** (*bool*, *optional*) – Whether to convert the type of the output tensor to the type of the attribute index. (default: `False`)

RAISES:

ValueError – If any input `TensorAttr` is not fully specified.

RETURN TYPE:

`List [ Union [ Tensor , ndarray ]]`

node_attrs ( ) → List[str]

Returns all node-level tensor attribute names.

RETURN TYPE:

`List [ str ]`

property **num_edges**: *int*

Returns the number of edges in the graph. For undirected graphs, this will return the number of bi-directional edges, which is double the amount of unique edges.

RETURN TYPE:

`int`

pin_memory (***args**: *str*)

Copies attributes to pinned memory, either for all attributes or only the ones given in `*args` .

put_edge_index (**edge_index**: Tuple[Tensor, Tensor], ***args**, ****kwargs**) → bool

Synchronously adds an `edge_index` tuple to the `GraphStore` . Returns whether insertion was successful.

PARAMETERS:

- **edge_index** (*Tuple[torch.Tensor, torch.Tensor]*) – The `edge_index` tuple in a format specified in `EdgeAttr` .
- ***args** – Arguments passed to `EdgeAttr` .
- ****kwargs** – Keyword arguments passed to `EdgeAttr` .

RETURN TYPE:

`bool`

put_tensor (**tensor**: Union[Tensor, ndarray], ***args**, ****kwargs**) → bool

Synchronously adds a `tensor` to the `FeatureStore` . Returns whether insertion was successful.

PARAMETERS:

- **tensor** (*torch.Tensor or np.ndarray*) – The feature tensor to be added.
- ***args** – Arguments passed to `TensorAttr` .
- ****kwargs** – Keyword arguments passed to `TensorAttr` .

RAISES:

ValueError – If the input `TensorAttr` is not fully specified.

RETURN TYPE:

`bool`

record_stream (**stream**: Stream, ***args**: *str*)

Ensures that the tensor memory is not reused for another tensor until all current work queued on `stream` has been completed, either for all attributes or only the ones given in `*args` .

remove_edge_index (***args**, ****kwargs**) → bool

Synchronously deletes an `edge_index` tuple from the `GraphStore`. Returns whether deletion was successful.

PARAMETERS:

- ***args** – Arguments passed to `EdgeAttr`.
- ****kwargs** – Keyword arguments passed to `EdgeAttr`.

RETURN TYPE:

`bool`

remove_tensor (***args**, ****kwargs**) → bool

Removes a tensor from the `FeatureStore`. Returns whether deletion was successful.

PARAMETERS:

- ***args** – Arguments passed to `TensorAttr`.
- ****kwargs** – Keyword arguments passed to `TensorAttr`.

RAISES:

ValueError – If the input `TensorAttr` is not fully specified.

RETURN TYPE:

`bool`

requires_grad_ (***args**: *str*, **requires_grad**: bool = `True`)

Tracks gradient computation, either for all attributes or only the ones given in `*args`.

share_memory_ (***args**: *str*)

Moves attributes to shared memory, either for all attributes or only the ones given in `*args`.

size (**dim**: Optional[int] = `None`) → Optional[Union[Tuple[Optional[int], Optional[int]], int]]

Returns the size of the adjacency matrix induced by the graph.

RETURN TYPE:

`Union [ Tuple [ Optional [ int ], Optional [ int ]], int , None ]`

snapshot (**start_time**: Union[float, int], **end_time**: Union[float, int], **attr**: str = `'time'`) → Self

Returns a snapshot of `data` to only hold events that occurred in period `[start_time, end_time]`.

RETURN TYPE:

`Self`

latest

sort (**sort_by_row**: bool = `True`) → Self

Sorts edge indices `edge_index` and their corresponding edge features.

PARAMETERS:

sort_by_row (*bool*, *optional*) – If set to `False`, will sort `edge_index` in column-wise order/by destination node. (default: `True`)

RETURN TYPE:

`Self`

sort_by_time ( ) → Self

Sorts data associated with `time` according to `time`.

RETURN TYPE:

`Self`

to (**device**: Union[int, str], ***args**: *str*, **non_blocking**: bool = `False`)

Performs tensor device conversion, either for all attributes or only the ones given in `*args`.

up_to (**end_time**: Union[float, int]) → Self

Returns a snapshot of `data` to only hold events that occurred up to `end_time` (inclusive of `edge_time`).

RETURN TYPE:

`Self`

update_tensor (**tensor**: Union[Tensor, ndarray], ***args**, ****kwargs**) → bool

Updates a `tensor` in the `FeatureStore` with a new value. Returns whether the update was succesful.

> **Note**
>
> Implementor classes can choose to define more efficient update methods; the default performs a removal and insertion.

PARAMETERS:

- **tensor** (*torch.Tensor or np.ndarray*) – The feature tensor to be updated.
- ***args** – Arguments passed to `TensorAttr`.
- ****kwargs** – Keyword arguments passed to `TensorAttr`.

RETURN TYPE:

`bool`

view (*__args__, **__kwargs__) → AttrView

Returns a view of the `FeatureStore` given a not yet fully-specified `TensorAttr`.

RETURN TYPE:

`AttrView`

property **num_faces**: *Optional[int]*

Returns the number of faces in the mesh.

RETURN TYPE:

`Optional [ int ]`

get_all_tensor_attrs ( ) → List[TensorAttr] [source]

Obtains all feature attributes stored in *Data*.

RETURN TYPE:

`List [ TensorAttr ]`

get_all_edge_attrs ( ) → List[EdgeAttr] [source]

Returns all registered edge attributes.

RETURN TYPE:

`List [ EdgeAttr ]`