

---

# **Xmodaler Documentation**

***Release 1.1.0***

**Yehao Li**

**Feb 27, 2023**



# CONTENTS

<b>1</b>	<b>Tutorials</b>	<b>3</b>
1.1	Installation . . . . .	3
1.2	Getting Started with X-modaler . . . . .	3
1.3	Use Builtin Datasets . . . . .	4
1.4	Extend X-modaler's Defaults . . . . .	5
1.5	Use Custom Datasets . . . . .	6
1.6	Dataloader . . . . .	7
1.7	Use Models . . . . .	8
1.8	Write Models . . . . .	9
1.9	Training . . . . .	10
1.10	Evaluation . . . . .	10
1.11	Configs . . . . .	11
<b>2</b>	<b>Notes</b>	<b>13</b>
2.1	Benchmarks . . . . .	13
<b>3</b>	<b>API Documentation</b>	<b>15</b>
3.1	xmodaler.checkpoint . . . . .	15
3.2	xmodaler.config . . . . .	18
3.3	xmodaler.datasets . . . . .	23
3.4	xmodaler.engine . . . . .	27
3.5	xmodaler.evaluation . . . . .	46
3.6	xmodaler.functional . . . . .	46
3.7	xmodaler.losses . . . . .	47
3.8	xmodaler.lr_scheduler . . . . .	50
3.9	xmodaler.modeling . . . . .	56
3.10	xmodaler.optim . . . . .	93
3.11	xmodaler.scorer . . . . .	95
3.12	xmodaler.tokenization . . . . .	95
3.13	xmodaler.utils . . . . .	97
	<b>Python Module Index</b>	<b>117</b>
	<b>Index</b>	<b>119</b>



X-modaler is a versatile and high-performance codebase for cross-modal analytics. This codebase unifies comprehensive high-quality modules in state-of-the-art vision-language techniques, which are organized in a standardized and user-friendly fashion.



## TUTORIALS

## 1.1 Installation

### 1.1.1 Requirements

- Linux or macOS with Python  $\geq 3.6$
- PyTorch  $\geq 1.8$  and torchvision that matches the PyTorch installation. Install them together at [pytorch.org](https://pytorch.org) to make sure of this
- fvcare
- pytorch\_transformers
- jsonlines
- pycocotools
- omegaconf
- cloudpickle
- tensorboard

### 1.1.2 Download X-modaler

After satisfying all the requirements, run:

```
git clone https://github.com/YehLi/xmodaler.git --recursive
```

Now, you can build your own frameworks using APIs supported by X-modaler !

## 1.2 Getting Started with X-modaler

This document provides a brief intro of the usage of builtin command-line tools in X-modaler, showing how to run inference with an existing model, and how to train a builtin model on a custom dataset (e.g., MSCOCO or MSVD).

### 1.2.1 Training & Evaluation in Command Line

We provide a script in “train\_net.py”, which is made to train all the configs provided in X-modaler. You may want to use it as a reference to write your own training script.

To train a model(e.g., UpDown) with “train\_net.py”, first setup the corresponding datasets following [datasets](#), then run:

```
# Teacher Force
python train_net.py --num-gpus 4 \
    --config-file configs/image_caption/updown.yaml

# Reinforcement Learning
python train_net.py --num-gpus 4 \
    --config-file configs/image_caption/updown_rl.yaml
```

## 1.3 Use Builtin Datasets

A dataset can be used by wrapping it into a torch Dataset. This document explains how to setup the builtin datasets so they can be used by X-modaler. The annotations for builtin datasets can be downloaded [here](#).

X-modaler has builtin supports for a few datasets (e.g., MSCOCO or MSVD). The corresponding dataset wrappers are provided in xmodaler/datasets:

```
xmodaler/datasets/
  images/
    mscoco.py
  videos/
    msvd.py
```

You can specify which dataset wrapper to use by DATASETS.TRAIN, DATASETS.VAL and DATASETS.TEST in the config file.

### 1.3.1 Expected dataset structure for COCO:

```
mscoco_dataset/
  mscoco_caption_anno_train.pkl
  mscoco_caption_anno_val.pkl
  mscoco_caption_anno_test.pkl
  vocabulary.txt
  captions_val5k.json
  captions_test5k.json
  # image files that are mentioned in the corresponding json
  features/
    up_down/
      *.npz
```



### 1.3.2 Expected dataset structure for MSVD:

```
msvd_dataset/
  msvd_caption_anno_train.pkl
  msvd_caption_anno_val.pkl
  msvd_caption_anno_test.pkl
  vocabulary.txt
  captions_val.json
  captions_test.json
  # videos files that are mentioned in the corresponding json
  features/
    resnet152/
      *.npz
```

### 1.3.3 Expected dataset structure for MSR-VTT:

```
msrvtt_dataset/
  msrvtt_caption_anno_train.pkl
  msrvtt_caption_anno_val.pkl
  msrvtt_caption_anno_test.pkl
  vocabulary.txt
  captions_val.json
  captions_test.json
  # videos files that are mentioned in the corresponding json
  msrvtt_torch/
    feature/
      resnet152/
        *.npz
```

When the dataset wrapper and data files are ready, you need to specify the corresponding paths to these data files in the config file. For example,

```
DATALOADER:
    FEATS_FOLDER: '../open_source_dataset/mscoco_dataset/features/up_down' #_
    ↪ feature folder
    ANNO_FOLDER: '../open_source_dataset/mscoco_dataset' # annotation folders
INFERENCE:
    VOCAB: '../open_source_dataset/mscoco_dataset/vocabulary.txt'
    VAL_ANNFILE: '../open_source_dataset/mscoco_dataset/captions_val5k.json'
    TEST_ANNFILE: '../open_source_dataset/mscoco_dataset/captions_test5k.json'
```

## 1.4 Extend X-modaler's Defaults

X-modaler provides very thin abstractions in APIs to allow for the possibility of doing everything in new ways, which means that it is easy to break existing abstractions and replace them with new ones. Also, high-level abstractions are available in X-modaler so that users can easily do things in standard ways, without worrying about the details that only certain researchers care about.

In X-modaler, the following interfaces are supported:

- Functions and classes that take a config (**cfg**) argument (sometimes with few extra arguments) as input. Such functions and classes implement the “**standard default**” behavior: it will read what it needs from the config. Users only need to load a given config and pass it around.
- Functions and classes that have well-defined explicit arguments, which requires users’ expertise to understand what each argument should be.
- A few functions and classes are implemented with the **@configurable** decorator - they can be called with either a config, or with explicit arguments.

As an example, a UpDown model can be built in the following ways:

```
cfg = ... # read the config
model = build_model(cfg) # users only need to pass a config
```

If you only need the standard behavior, the [Beginner’s Tutorial](#) should suffice. If you need to extend X-modaler to your own needs, see the following tutorials for more details:

- X-modaler includes a few standard datasets. To use custom ones, see [Use Custom Datasets](#).
- X-modaler contains the standard logic that creates a data loader for training/testing from a dataset, but you can write your own as well. See [DataLoader](#).
- X-modaler implements several state-of-the-art models for visoin-and-language tasks, and provides ways for you to overwrite their behaviors. See [Use Models](#) and [Write Models](#).
- X-modaler provides a default training loop that is good for common training tasks. You can customize it with hooks, or write your own loop instead. See [Training](#).

## 1.5 Use Custom Datasets

Datasets that have builtin support in X-modaler are listed in [builtin datasets](#). If you want to use a custom dataset while also reusing X-modaler’s data loaders, you will need to **register** your dataset (i.e., tell X-modaler how to obtain your dataset).

### 1.5.1 Register a Dataset

To let X-modaler know how to obtain a dataset named “MyDataset”, users need to implement a class either in the folder `xmodaler/datasets/images` or `xmodaler/datasets/videos` that returns the item in your dataset and then tell X-modaler about this class:

```
from ..build import DATASETS_REGISTRY

__all__ = ["MyDataset"]

@DATASETS_REGISTRY.register()
class MyDataset:
    @configurable
    def __init__(
        self,
        stage: str,
        anno_file: str,
        seq_per_img: int,
        max_feat_num: int,
```

(continues on next page)

(continued from previous page)

```

        max_seq_len: int,
        feats_folder: str
    ):
        # load dataset and set hyperparameters
        ...

    @classmethod
    def from_config(cls, cfg, stage: str = "train"):
        ...
        return arguments

    def __call__(self, dataset_dict):
        # process a data item for a specific task
        ...
        return item_dict

```

The function can do arbitrary things and should return the data dict in either of the following formats:

- X-modaler’s standard dataset dict using builtin keys defined by `xmodaler.config.kfg`. This will make it work with many other builtin features in X-modaler, so it’s recommended to use it when it’s sufficient.
- Any custom format. You can also return arbitrary dict in your own format by adding extra keys to `xmodaler.config.kfg` for new tasks. Then you will need to handle them properly downstream as well.

## 1.6 Dataloader

Dataloader is the component that provides data to models. A dataloader usually (but not necessarily) takes raw information from [datasets](#), and process them into a format needed by the model.

### 1.6.1 How the Existing Dataloader Works

X-modaler contains a builtin data loading pipeline. It’s good to understand how it works, in case you need to write a custom one.

X-modaler provides a function `xmodaler.datasets.build_xmodaler_{train,valtest}_loader` that creates a default dataloader from a given config. Here is how `build_xmodaler_{train,valtest}_loader` works:

- It takes a helper class (e.g., `xmodaler.datasets.common.DatasetFromList`) and loads a **list[dict]** representing the dataset items in a lightweight format. These dataset items are not yet ready to be used by the model (e.g., images are not loaded into memory).
- Each dict in this list is processed by the class `xmodaler.datasets.common.MapDataset`. Users can customize this data loading for specific datasets by implementing the `__call__` function in a wrapper class (e.g., `xmodaler.datasets.MSCoCoDataset`), which is one of the arguments to initialize `MapDataset`. The role of the wrapper class is to transform the lightweight representation of a dataset item into a format that is ready for the model to consume (including, e.g., read images, caption sampling or convert to torch Tensors).
- After gathering a list of items, batching schema is handled by defining the argument `collate_fn` of `torch.utils.data.DataLoader` in `xmodaler.datasets.build_xmodaler_{train,valtest}_loader` functions.

The batched data is the output of the data loader. Typically, it’s also the input of `model.forward()`.

## 1.6.2 Write a Custom Dataloader

Using a different “wrapper class” as the argument `dataset_mapper` with `build_xmodaler_{train, valtest}_loader` works for most cases of custom data loading. See [Use Custom Datasets](#) to custom the “wrapper class”.

## 1.6.3 Use a Custom Dataloader

If you use *DefaultTrainer*, you can overwrite its `build_xmodaler_{train, valtest}_loader` method to use your own dataloader.

# 1.7 Use Models

Models (and their sub-models) in X-modaler are built by the function `build_model`:

```
from xmodaler.modeling import build_model
cfg = ... # read the config
model = build_model(cfg)
```

`build_model` only builds the model structure and fills it with random parameters defined in the given config (`cfg`). See below for how to load an existing checkpoint to the model and how to use the model object.

## 1.7.1 Load/Save a Checkpoint

Users need to specify the path of the checkpoint to load by setting the argument `MODEL.WEIGHTS` in the given config (`cfg`):

```
from xmodaler.checkpoint import XmodalerCheckpoint
checkpointer = XmodalerCheckpoint(model, cfg.OUTPUT_DIR)
checkpointer.resume_or_load(cfg.MODEL.WEIGHTS, resume=True) # load a checkpoint
checkpointer.save("model_00") # save model
```

## 1.7.2 Use a Model

A model can be called by `outputs = model(inputs)`, where `inputs` is the batched output of `dataloader`. Each item in the dict `inputs` corresponds to a batch of images/videos and the required inputs depend on the type of model and task.

**Training:** When in training mode, all models are required to be used under an `EventStorage`. The training statistics will be put into the storage:

```
with EventStorage(start_iter) as self.storage:
    outputs_dict = model(inputs)
    losses = loss(outputs_dict)
```

You can run inference directly like this:

```
model.eval()
with torch.no_grad():
    outputs = model(inputs)
```

## 1.8 Write Models

If you are trying to do something completely new, you may wish to implement a model entirely from scratch. However, in many situations you may be interested in modifying or extending some components of an existing model. Therefore, we also provide mechanisms that let users override the behavior of certain internal components of standard models.

### 1.8.1 Register New Components

For common concepts that users often want to customize, such as “encoder/decoder”, “layers”, we provide a registration mechanism for users to inject custom implementations that will be immediately available to use in config files.

For example, to add a new encoder to `xmodaler/modeling/encoder`, import this code in your code:

```
from xmodaler.modeling.encoder.build import ENCODER_REGISTRY

__all__ = ["MyEncoder"]

@ENCODER_REGISTRY.register()
class MyEncoder(nn.Module):
    @configurable
    def __init__(self):
        super(MyEncoder, self).__init__()

    @classmethod
    def from_config(cls, cfg):
        ...
        return {} # return init arguments

    @classmethod
    def add_config(cls, cfg):
        ...

    def forward(self, batched_inputs, mode=None):
        ...
        return outputs
```

In this code, we implement a new encoder following the interface of the `torch.nn.Module`, and register it into the `ENCODER_REGISTRY`. After importing this code, X-modaler can link the name of the class to its implementation. Therefore you can write the following code:

```
cfg = ... # read a config
cfg.MODEL.ENCODER = 'MyEncoder' # or set it in the config file
model = build_model(cfg) # it will find `MyEncoder` defined above
```

## 1.9 Training

From the previous tutorials, you may now have a custom model and a dataloader. To run training, users typically have a preference in the following style:

### 1.9.1 Trainer Abstraction

X-modaler provides a standardized “trainer” abstraction with a hook system that helps simplify the standard training behavior. It includes the following instantiation:

- `DefaultTrainer` is a default trainer initialized from a config, used by many scripts. It includes more standard default behaviors that one might want to opt in, including default configurations for optimizer, learning rate scheduling, logging, evaluation, checkpointing etc.

Users can write their custom trainer by following the class `DefaultTrainer`.

## 1.10 Evaluation

Evaluation is a process that takes a number of inputs/outputs pairs and aggregate them. You can always [use the model](#) directly and just parse its inputs/outputs manually to perform evaluation. Alternatively, evaluation is implemented in `xmodaler.evaluation`.

### 1.10.1 Custom Evaluators

X-modaler includes a few evaluators that compute metrics using standard dataset-specific APIs (e.g., COCO Captions). You can also implement your own evaluator that performs some other jobs using the inputs/outputs pairs:

```
from xmodaler.evaluation.build import EVALUATION_REGISTRY
@EVALUATION_REGISTRY.register()
class MyEvaler(object):
    def __init__(self, cfg, annfile):
        super(MyEvaler, self).__init__()
        ...

    def eval(self, result):
        ...
```

### 1.10.2 Use Evaluators

To evaluate using the methods of evaluators manually:

```
evaluator = MyEvaler(cfg, reference_file)
model.eval()
with torch.no_grad():
    outputs = model(inputs)
eval_results = evaluator.eval(outputs)
```

## 1.11 Configs

X-modaler provides a key-value based config system that can be used to obtain standard, common behaviors.

X-modaler's config system uses YAML. In addition to the basic operations that access and update a config, we provide the following extra functionality:

- The config can have `_BASE_:` `base.yaml` field, which will load a base config first. Values in the base config will be overwritten in sub-configs, if there are any conflicts. We provided several base configs for standard model architectures in the folder `configs/`.

### 1.11.1 Basic Usage

Some basic usage of the `CfgNode` object is shown here:

```
from xmodaler.config import get_cfg
from xmodaler.modeling import add_config
cfg = get_cfg() # obtain X-modaler's default config
tmp_cfg = cfg.load_from_file_tmp(config_file) # load custom config
add_config(cfg, tmp_cfg) # combining default and custom configs
cfg.merge_from_file(config_file) # load values from a file
cfg.merge_from_list(opts) # # can also load values from a list of str
```

Many builtin tools in X-modaler accept command line config overwrite: key-value pairs provided in the command line will overwrite the existing values in the config file. For example,

```
python train_net.py --config-file config.yaml [--other-options] \
  --opts MODEL.WEIGHTS /path/to/weights
```





## 2.1 Benchmarks

### 2.1.1 Image Captioning on MSCOCO (Cross-Entropy Loss)

Model	BLEU@1	BLEU@2	BLEU@3	BLEU@4	ME-TEOR	ROUGE-L	CIDEr-D	SPICE
LSTM-A3	75.3	59.0	45.4	35.0	26.7	55.6	107.7	19.7
Attention	76.4	60.6	46.9	36.1	27.6	56.6	113.0	20.4
Up-Down	76.3	60.3	46.6	36.0	27.6	56.6	113.1	20.7
GCN-LSTM	76.8	61.1	47.6	36.9	28.2	57.2	116.3	21.2
Transformer	76.4	60.3	46.5	35.8	28.2	56.7	116.6	21.3
Meshed-Memory Transformer	76.3	60.2	46.4	35.6	28.1	56.5	116.0	21.2
X-LAN	77.5	61.9	48.3	37.5	28.6	57.6	120.7	21.9
TDEN	75.5	59.4	45.7	34.9	28.7	56.7	116.3	22.0

### 2.1.2 Image Captioning on MSCOCO (CIDEr Score Optimization)

Model	BLEU@1	BLEU@2	BLEU@3	BLEU@4	ME-TEOR	ROUGE-L	CIDEr-D	SPICE
LSTM-A3	77.9	61.5	46.7	35.0	27.1	56.3	117.0	20.5
Attention	79.4	63.5	48.9	37.1	27.9	57.6	123.1	21.3
Up-Down	80.1	64.3	49.7	37.7	28.0	58.0	124.7	21.5
GCN-LSTM	80.2	64.7	50.3	38.5	28.5	58.4	127.2	22.1
Transformer	80.5	65.4	51.1	39.2	29.1	58.7	130.0	23.0
Meshed-Memory Transformer	80.7	65.5	51.4	39.6	29.2	58.9	131.1	22.9
X-LAN	80.4	65.2	51.0	39.2	29.4	59.0	131.0	23.2
TDEN	81.3	66.3	52.0	40.1	29.6	59.8	132.6	23.4

### 2.1.3 Video Captioning on MSVD

Model	BLEU@1	BLEU@2	BLEU@3	BLEU@4	ME-TEOR	ROUGE-L	CIDEr-D	SPICE
MP-LSTM	77.0	65.6	56.9	48.1	32.4	68.1	73.1	4.8
TA	80.4	68.9	60.1	51.0	33.5	70.0	77.2	4.9
Trans-former	79.0	67.6	58.5	49.4	33.3	68.7	80.3	4.9
TD-ConvED	81.6	70.4	61.3	51.7	34.1	70.4	77.8	5.0

### 2.1.4 Video Captioning on MSR-VTT

Model	BLEU@1	BLEU@2	BLEU@3	BLEU@4	ME-TEOR	ROUGE-L	CIDEr-D	SPICE
MP-LSTM	73.6	60.8	49.0	38.6	26.0	58.3	41.1	5.6
TA	74.3	61.8	50.3	39.9	26.4	59.4	42.9	5.8
Trans-former	75.4	62.3	50.0	39.2	26.5	58.7	44.0	5.9
TD-ConvED	76.4	62.3	49.9	38.9	26.3	59.0	40.7	5.7

### 2.1.5 Visual Question Answering

Model	Overall	Yes/No	Number	Other
Uniter	70.1	86.8	53.7	59.6
TDEN	71.9	88.3	54.3	62.0

### 2.1.6 Caption-based image retrieval on Flickr30k

Model	R1	R5	R10
Uniter	61.6	87.7	92.8
TDEN	62.0	86.6	92.4

### 2.1.7 Visual commonsense reasoning

Model	Q -> A	QA -> R	Q -> AR
Uniter	73.0	75.3	55.4
TDEN	75.0	76.5	57.7

## API DOCUMENTATION

### 3.1 xmodaler.checkpoint

```
class xmodaler.checkpoint.PeriodicEpochCheckpoint(checkpoint: Checkpointer, period: int,  
                                                    max_iter: Optional[int] = None, max_to_keep:  
                                                    Optional[int] = None, file_prefix: str = 'model')
```

Bases: `PeriodicCheckpoint`

```
__init__(checkpoint: Checkpointer, period: int, max_iter: Optional[int] = None, max_to_keep:  
          Optional[int] = None, file_prefix: str = 'model') → None
```

#### Parameters

- **checkpointer** – the checkpointer object used to save checkpoints.
- **period** (*int*) – the period to save checkpoint.
- **max\_iter** (*int*) – maximum number of iterations. When it is reached, a checkpoint named “{file\_prefix}\_final” will be saved.
- **max\_to\_keep** (*int*) – maximum number of most current checkpoints to keep, previous checkpoints will be deleted
- **file\_prefix** (*str*) – the prefix of checkpoint’s filename

```
save(name: str, **kwargs: Any) → None
```

Same argument as `Checkpointer.save()`. Use this method to manually save checkpoints outside the schedule.

#### Parameters

- **name** (*str*) – file name.
- **kwargs** (*Any*) – extra data to save, same as in `Checkpointer.save()`.

```
step(iteration: int, epoch: int, **kwargs: Any) → None
```

Perform the appropriate action at the given iteration.

#### Parameters

- **iteration** (*int*) – the current iteration, ranged in `[0, max_iter-1]`.
- **kwargs** (*Any*) – extra data to save, same as in `Checkpointer.save()`.

```
class xmodaler.checkpoint.XmodalerCheckpoint(model, save_dir="", *, save_to_disk=None,  
                                              **checkpointables)
```

Bases: Checkpointer

Same as Checkpointer, but is able to handle models in xmodaler model zoo, and apply conversions for legacy models.

**\_\_init\_\_**(*model*, *save\_dir*="", \*, *save\_to\_disk*=None, *\*\*checkpointables*)

**Parameters**

- **model** (*nn.Module*) – model.
- **save\_dir** (*str*) – a directory to save and find checkpoints.
- **save\_to\_disk** (*bool*) – if True, save checkpoint to disk, otherwise disable saving for this checkpointer.
- **checkpointables** (*object*) – any checkpointable objects, i.e., objects that have the `state_dict()` and `load_state_dict()` method. For example, it can be used like `Checkpointer(model, "dir", optimizer=optimizer)`.

**\_convert\_ndarray\_to\_tensor**(*state\_dict: Dict[str, Any]*) → None

In-place convert all numpy arrays in the `state_dict` to torch tensor. :param `state_dict`: a state-dict to be loaded to the model.

Will be modified.

**\_load\_file**(*filename*)

Load a checkpoint file. Can be overwritten by subclasses to support different formats.

**Parameters**

**f** (*str*) – a locally mounted file path.

**Returns**

with keys “model” and optionally others that are saved by the checkpointer dict[“model”] must be a dict which maps strings to torch.Tensor or numpy arrays.

**Return type**

dict

**\_load\_model**(*checkpoint*)

Load weights from a checkpoint.

**Parameters**

**checkpoint** (*Any*) – checkpoint contains the weights.

**Returns**

**NamedTuple with missing\_keys, unexpected\_keys,**  
and `incorrect_shapes` fields: \* **missing\_keys** is a list of str containing the missing keys  
\* **unexpected\_keys** is a list of str containing the unexpected keys \* **incorrect\_shapes** is a list of (key, shape in checkpoint, shape in model)

This is just like the return value of `torch.nn.Module.load_state_dict()`, but with extra support for `incorrect_shapes`.

**\_log\_incompatible\_keys**(*incompatible: \_IncompatibleKeys*) → None

Log information about the incompatible keys returned by `_load_model`.

**add\_checkpointable**(*key: str, checkpointable: Any*) → None

Add checkpointable object for this checkpointer to track.

**Parameters**

- **key** (*str*) – the key used to save the object
- **checkpointable** – any object with `state_dict()` and `load_state_dict()` method

**get\_all\_checkpoint\_files**() → List[str]

**Returns**

All available checkpoint files (.pth files) in target directory.

**Return type**

list

**get\_checkpoint\_file**() → str

**Returns**

The latest checkpoint file in target directory.

**Return type**

str

**has\_checkpoint**() → bool

**Returns**

whether a checkpoint exists in the target directory.

**Return type**

bool

**load**(*path: str, checkpointables: Optional[List[str]] = None*) → Dict[str, Any]

Load from the given checkpoint.

**Parameters**

- **path** (*str*) – path or url to the checkpoint. If empty, will not load anything.
- **checkpointables** (*list*) – List of checkpointable names to load. If not specified (None), will load all the possible checkpointables.

**Returns**

extra data loaded from the checkpoint that has not been processed. For example, those saved with `save(**extra_data)()`.

**Return type**

dict

**resume\_or\_load**(*path: str, \*, resume: bool = True*) → Dict[str, Any]

If *resume* is True, this method attempts to resume from the last checkpoint, if exists. Otherwise, load checkpoint from the given path. This is useful when restarting an interrupted training job.

**Parameters**

- **path** (*str*) – path to the checkpoint.
- **resume** (*bool*) – if True, resume from the last checkpoint if it exists and load the model together with all the checkpointables. Otherwise only load the model without loading any checkpointables.

**Returns**

same as `load()`.

**save**(*name: str*, *\*\*kwargs: Any*) → None

Dump model and checkpointables to a file.

**Parameters**

- **name** (*str*) – name of the file.
- **kwargs** (*dict*) – extra arbitrary data to save.

**tag\_last\_checkpoint**(*last\_filename\_basename: str*) → None

Tag the last checkpoint.

**Parameters**

**last\_filename\_basename** (*str*) – the basename of the last filename.

## 3.2 xmodaler.config

**class** `xmodaler.config.CfgNode`(*init\_dict=None*, *key\_list=None*, *new\_allowed=False*)

Bases: `CfgNode`

The same as `fvcore.common.config.CfgNode`, but different in:

1. Use unsafe yaml loading by default. Note that this may lead to arbitrary code execution: you must not load a config file from untrusted sources before manually inspecting the content of the file.
2. Support config versioning. When attempting to merge an old config, it will convert the old config automatically.

**DEPRECATED\_KEYS** = `'__deprecated_keys__'`

**IMMUTABLE** = `'__immutable__'`

**NEW\_ALLOWED** = `'__new_allowed__'`

**RENAMED\_KEYS** = `'__renamed_keys__'`

**\_\_init\_\_**(*init\_dict=None*, *key\_list=None*, *new\_allowed=False*)

**Parameters**

- **init\_dict** (*dict*) – the possibly-nested dictionary to initialize the `CfgNode`.
- **key\_list** (*list[str]*) – a list of names which index this `CfgNode` from the root. Currently only used for logging purposes.
- **new\_allowed** (*bool*) – whether adding new key is allowed when merging with other configs.

**classmethod** `_create_config_tree_from_dict`(*dic*, *key\_list*)

Create a configuration tree using the given dict. Any dict-like objects inside dict will be treated as a new `CfgNode`.

**Parameters**

- **dic** (*dict*) –
- **key\_list** (*list[str]*) – a list of names which index this `CfgNode` from the root. Currently only used for logging purposes.

**classmethod** `_decode_cfg_value(value)`

Decodes a raw config value (e.g., from a yaml config files or command line argument) into a Python object.

If the value is a dict, it will be interpreted as a new CfgNode. If the value is a str, it will be evaluated as literals. Otherwise it is returned as-is.

**immutable**(*is\_immutable*)

Set immutability to *is\_immutable* and recursively apply the setting to all nested CfgNodes.

**classmethod** `_load_cfg_from_file(file_obj)`

Load a config from a YAML file or a Python source file.

**classmethod** `_load_cfg_from_yaml_str(str_obj)`

Load a config from a YAML string encoding.

**classmethod** `_load_cfg_py_source(filename)`

Load a config from a Python source file.

**classmethod** `_open_cfg(filename)`

Defines how a config file is opened. May be overridden to support different file schemas.

**clear()** → None. Remove all items from D.

**clone()**

Recursively copy this CfgNode.

**copy()** → a shallow copy of D

**defrost()**

Make this CfgNode and all of its children mutable.

**dump(\*args, \*\*kwargs)**

**Returns**

a yaml string representation of the config

**Return type**

str

**freeze()**

Make this CfgNode and all of its children immutable.

**fromkeys(value=None, /)**

Create a new dictionary with keys from iterable and values set to value.

**get(key, default=None, /)**

Return the value for key if key is in the dictionary, else default.

**is\_frozen()**

Return mutability.

**is\_new\_allowed()**

**items()** → a set-like object providing a view on D's items

**key\_is\_deprecated(full\_key)**

Test if a key is deprecated.

**key\_is\_renamed(full\_key)**

Test if a key is renamed.

**keys()** → a set-like object providing a view on D's keys

**classmethod load\_cfg**(*cfg\_file\_obj\_or\_str*)

Load a cfg. :param *cfg\_file\_obj\_or\_str*: Supports loading from:

- A file object backed by a YAML file
- A file object backed by a Python source file that exports an attribute “cfg” that is either a dict or a CfgNode
- A string that can be parsed as valid YAML

**load\_from\_file\_tmp**(*cfg\_filename: str, allow\_unsafe: bool = True*) → None

**classmethod load\_yaml\_with\_base**(*filename: str, allow\_unsafe: bool = False*) → Dict[str, Any]

Just like *yaml.load(open(filename))*, but inherit attributes from its *\_BASE\_*.

**Parameters**

- **filename** (*str or file-like object*) – the file name or file of the current config. Will be used to find the base config file.
- **allow\_unsafe** (*bool*) – whether to allow loading the config file with *yaml.unsafe\_load*.

**Returns**

the loaded yaml

**Return type**

(dict)

**merge\_from\_file**(*cfg\_filename: str, allow\_unsafe: bool = True*) → None

Merge configs from a given yaml file.

**Parameters**

- **cfg\_filename** – the file name of the yaml config.
- **allow\_unsafe** – whether to allow loading the config file with *yaml.unsafe\_load*.

**merge\_from\_list**(*cfg\_list: List[str]*) → Callable[[], None]

**Parameters**

**cfg\_list** (*list*) – list of configs to merge from.

**merge\_from\_other\_cfg**(*cfg\_other: CfgNode*) → Callable[[], None]

**Parameters**

**cfg\_other** (*CfgNode*) – configs to merge from.

**pop**(*k[, d]*) → *v*, remove specified key and return the corresponding value.

If key is not found, *d* is returned if given, otherwise *KeyError* is raised

**popitem**() → (*k, v*), remove and return some (key, value) pair as a

2-tuple; but raise *KeyError* if *D* is empty.

**raise\_key\_rename\_error**(*full\_key*)



**register\_deprecated\_key**(*key*)

Register key (e.g. *FOO.BAR*) a deprecated option. When merging deprecated keys a warning is generated and the key is ignored.

**register\_renamed\_key**(*old\_name*, *new\_name*, *message=None*)

Register a key as having been renamed from *old\_name* to *new\_name*. When merging a renamed key, an exception is thrown alerting to user to the fact that the key has been renamed.

**set\_new\_allowed**(*is\_new\_allowed*)

Set this config (and recursively its subconfigs) to allow merging new keys from other configs.

**setdefault**(*key*, *default=None*, /)

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**update**([*E*], *\*\*F*) → None. Update D from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values**() → an object providing a view on D's values**xmodaler.config.get\_cfg**() → *CfgNode*

Get a copy of the default config.

**Returns**

a X-modaler CfgNode instance.

**xmodaler.config.set\_global\_cfg**(*cfg: CfgNode*) → None

Let the global config point to the given cfg.

Assume that the given “cfg” has the key “KEY”, after calling *set\_global\_cfg(cfg)*, the key can be accessed by:

```
from xmodaler.config import global_cfg
print(global_cfg.KEY)
```

By using a hacky global config, you can access these configs anywhere, without having to pass the config object or the values deep into the code. This is a hacky feature introduced for quick prototyping / research exploration.

**xmodaler.config.downgrade\_config**(*cfg: CfgNode*, *to\_version: int*) → *CfgNode*

Downgrade a config from its current version to an older version.

**Parameters**

- **cfg** (*CfgNode*) –
- **to\_version** (*int*) –

**Note:** A general downgrade of arbitrary configs is not always possible due to the different functionalities in different versions. The purpose of downgrade is only to recover the defaults in old versions, allowing it to load an old partial yaml config. Therefore, the implementation only needs to fill in the default values in the old version when a general downgrade is not possible.

**xmodaler.config.upgrade\_config**(*cfg: CfgNode*, *to\_version: Optional[int] = None*) → *CfgNode*

Upgrade a config from its current version to a newer version.

**Parameters**

- **cfg** (*CfgNode*) –

- **to\_version** (*int*) – defaults to the latest version.

`xmodaler.config.configurable`(*init\_func=None*, \*, *from\_config=None*)

Decorate a function or a class's `__init__` method so that it can be called with a *CfgNode* object using a `from_config()` function that translates *CfgNode* to arguments.

Examples:

```
# Usage 1: Decorator on __init__:
class A:
    @configurable
    def __init__(self, a, b=2, c=3):
        pass

    @classmethod
    def from_config(cls, cfg): # 'cfg' must be the first argument
        # Returns kwargs to be passed to __init__
        return {"a": cfg.A, "b": cfg.B}

a1 = A(a=1, b=2) # regular construction
a2 = A(cfg)      # construct with a cfg
a3 = A(cfg, b=3, c=4) # construct with extra overwrite

# Usage 2: Decorator on any function. Needs an extra from_config argument:
@configurable(from_config=lambda cfg: {"a": cfg.A, "b": cfg.B})
def a_func(a, b=2, c=3):
    pass

a1 = a_func(a=1, b=2) # regular call
a2 = a_func(cfg)      # call with a cfg
a3 = a_func(cfg, b=3, c=4) # call with extra overwrite
```

### Parameters

- **init\_func** (*callable*) – a class's `__init__` method in usage 1. The class must have a `from_config` classmethod which takes *cfg* as the first argument.
- **from\_config** (*callable*) – the `from_config` function in usage 2. It must take *cfg* as its first argument.

`xmodaler.config.global_cfg`

The same as *fvcore.common.config.CfgNode*, but different in:

1. Use unsafe yaml loading by default. Note that this may lead to arbitrary code execution: you must not load a config file from untrusted sources before manually inspecting the content of the file.
2. Support config versioning. When attempting to merge an old config, it will convert the old config automatically.

`xmodaler.config.kfg`

The same as *fvcore.common.config.CfgNode*, but different in:

1. Use unsafe yaml loading by default. Note that this may lead to arbitrary code execution: you must not load a config file from untrusted sources before manually inspecting the content of the file.
2. Support config versioning. When attempting to merge an old config, it will convert the old config automatically.

### 3.3 xmodaler.datasets

**class** xmodaler.datasets.DatasetFromList(*lst: list, copy: bool = True, serialize: bool = True*)

Bases: Dataset

Wrap a list to a torch Dataset. It produces elements of the list as data.

**\_\_getitem\_\_**(*idx*)

**\_\_init\_\_**(*lst: list, copy: bool = True, serialize: bool = True*)

#### Parameters

- **lst** (*list*) – a list which contains elements to produce.
- **copy** (*bool*) – whether to deepcopy the element when producing it, so that the result can be modified in place without affecting the source in the list.
- **serialize** (*bool*) – whether to hold memory using serialized objects, when enabled, data loader workers can use shared RAM from master process instead of making a copy.

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

**class** xmodaler.datasets.MapDataset(*dataset, map\_func*)

Bases: Dataset

Map a function over the elements in a dataset.

#### Parameters

- **dataset** – a dataset where map function is applied.
- **map\_func** – a callable which maps the element in dataset. map\_func is responsible for error handling, when error happens, it needs to return None so the MapDataset will randomly use other elements from the dataset.

**\_\_getitem\_\_**(*idx*)

**\_\_init\_\_**(*dataset, map\_func*)

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

**class** xmodaler.datasets.MSCoCoDataset(*stage: str, anno\_file: str, seq\_per\_img: int, max\_feat\_num: int, max\_seq\_len: int, feats\_folder: str, relation\_file: str, gv\_feat\_file: str, attribute\_file: str*)

Bases: object

**\_\_call\_\_**(*dataset\_dict*)

Call self as a function.

**\_\_init\_\_**(*stage: str, anno\_file: str, seq\_per\_img: int, max\_feat\_num: int, max\_seq\_len: int, feats\_folder: str, relation\_file: str, gv\_feat\_file: str, attribute\_file: str*)

**classmethod** **from\_config**(*cfg, stage: str = 'train'*)

**load\_data**(*cfg*)

```
class xmodaler.datasets.MSCoCoSampleByTxtDataset(stage: str, anno_file: str, seq_per_img: int,
                                                max_feat_num: int, max_seq_len: int, feats_folder:
                                                str, relation_file: str, gv_feat_file: str, attribute_file:
                                                str)
```

Bases: [MSCoCoDataset](#)

```
__call__(dataset_dict)
```

Call self as a function.

```
__init__(stage: str, anno_file: str, seq_per_img: int, max_feat_num: int, max_seq_len: int, feats_folder: str,
         relation_file: str, gv_feat_file: str, attribute_file: str)
```

```
classmethod from_config(cfg, stage: str = 'train')
```

```
load_data(cfg)
```

```
class xmodaler.datasets.MSCoCoBertDataset(stage: str, anno_file: str, seq_per_img: int, max_seq_length:
int, max_feat_num: int, feats_folder: str, images_ids_file: str,
tokenizer)
```

Bases: object

```
__call__(dataset_dict)
```

Call self as a function.

```
__init__(stage: str, anno_file: str, seq_per_img: int, max_seq_length: int, max_feat_num: int, feats_folder:
str, images_ids_file: str, tokenizer)
```

```
classmethod from_config(cfg, stage: str = 'train')
```

```
load_data(cfg)
```

```
class xmodaler.datasets.ConceptualCaptionsDataset(stage: str, anno_file: str, max_seq_length: int,
max_feat_num: int, feats_folder: str,
images_ids_file: str, tokenizer)
```

Bases: object

```
__call__(dataset_dict)
```

Call self as a function.

```
__init__(stage: str, anno_file: str, max_seq_length: int, max_feat_num: int, feats_folder: str,
         images_ids_file: str, tokenizer)
```

```
classmethod from_config(cfg, stage: str = 'train')
```

```
load_data(cfg)
```

```
class xmodaler.datasets.ConceptualCaptionsDatasetForSingleStream(stage: str, anno_file: str,
max_seq_length: int,
max_feat_num: int,
feats_folder: str,
images_ids_file: str, tokenizer,
itm_neg_prob: float)
```

Bases: [ConceptualCaptionsDataset](#)

```
__call__(dataset_dict)
```

Call self as a function.

```

__init__(stage: str, anno_file: str, max_seq_length: int, max_feat_num: int, feats_folder: str,
         images_ids_file: str, tokenizer, itm_neg_prob: float)

classmethod from_config(cfg, stage: str = 'train')

load_data(cfg)

class xmodaler.datasets.VQADataset(stage: str, anno_folder: str, ans2label_path: str, label2ans_path: str,
                                   feats_folder: str, max_feat_num: int, max_seq_len: int, use_global_v:
                                   bool, tokenizer)

Bases: object

__call__(dataset_dict)
    Call self as a function.

__init__(stage: str, anno_folder: str, ans2label_path: str, label2ans_path: str, feats_folder: str,
         max_feat_num: int, max_seq_len: int, use_global_v: bool, tokenizer)

classmethod from_config(cfg, stage: str = 'train')

load_data(cfg)

class xmodaler.datasets.VCRDataset(stage: str, task_name: str, anno_folder: str, feats_folder: str,
                                   max_feat_num: int, max_seq_len: int, seq_per_img: int, use_global_v:
                                   bool, tokenizer)

Bases: object

__call__(dataset_dict)
    Call self as a function.

__init__(stage: str, task_name: str, anno_folder: str, feats_folder: str, max_feat_num: int, max_seq_len:
         int, seq_per_img: int, use_global_v: bool, tokenizer)

classmethod from_config(cfg, stage: str = 'train;VCR_Q-A')

load_data(cfg)

class xmodaler.datasets.Flickr30kDataset(stage: str, anno_folder: str, anno_file: str, feats_folder: str,
                                         max_feat_num: int, max_seq_len: int, use_global_v: bool,
                                         tokenizer)

Bases: object

__call__(dataset_dict)
    Call self as a function.

__init__(stage: str, anno_folder: str, anno_file: str, feats_folder: str, max_feat_num: int, max_seq_len: int,
         use_global_v: bool, tokenizer)

classmethod from_config(cfg, stage: str = 'train')

load_data(cfg)

class xmodaler.datasets.Flickr30kDatasetForSingleStream(stage: str, anno_folder: str, anno_file: str,
                                                         feats_folder: str, max_feat_num: int,
                                                         max_seq_len: int, use_global_v: bool,
                                                         negative_size: int, tokenizer, cfg)

Bases: Flickr30kDataset

```

```
__call__(dataset_dict)
```

Call self as a function.

```
__init__(stage: str, anno_folder: str, anno_file: str, feats_folder: str, max_feat_num: int, max_seq_len: int,
         use_global_v: bool, negative_size: int, tokenizer, cfg)
```

```
classmethod from_config(cfg, stage: str = 'train')
```

```
load_data(cfg)
```

```
class xmodaler.datasets.Flickr30kDatasetForSingleStreamVal(stage: str, anno_folder: str, anno_file:
                                                           str, feats_folder: str, max_feat_num:
                                                           int, max_seq_len: int, use_global_v:
                                                           bool, inf_batch_size: int, tokenizer,
                                                           cfg)
```

Bases: [Flickr30kDataset](#)

```
__call__(dataset_dict)
```

Call self as a function.

```
__init__(stage: str, anno_folder: str, anno_file: str, feats_folder: str, max_feat_num: int, max_seq_len: int,
         use_global_v: bool, inf_batch_size: int, tokenizer, cfg)
```

```
classmethod from_config(cfg, stage: str = 'train')
```

```
load_data(cfg)
```

```
class xmodaler.datasets.MSVDDataset(stage: str, anno_file: str, seq_per_img: int, max_feat_num: int,
                                     max_seq_len: int, feats_folder: str)
```

Bases: [object](#)

```
__call__(dataset_dict)
```

Call self as a function.

```
__init__(stage: str, anno_file: str, seq_per_img: int, max_feat_num: int, max_seq_len: int, feats_folder:
         str)
```

```
classmethod from_config(cfg, stage: str = 'train')
```

```
load_data(cfg)
```

```
class xmodaler.datasets.MSRVTTDataset(stage: str, anno_file: str, seq_per_img: int, max_feat_num: int,
                                       max_seq_len: int, feats_folder: str)
```

Bases: [object](#)

```
__call__(dataset_dict)
```

Call self as a function.

```
__init__(stage: str, anno_file: str, seq_per_img: int, max_feat_num: int, max_seq_len: int, feats_folder:
         str)
```

```
classmethod from_config(cfg, stage: str = 'train')
```

```
load_data(cfg)
```

```
xmodaler.datasets.build_xmodaler_train_loader(datalist, *, dataset_mapper, batch_size, num_workers)
```

```
xmodaler.datasets.build_xmodaler_valtest_loader(datalist, *, dataset_mapper, batch_size, num_workers,
                                                multi_gpu_eval)
```

```
xmodaler.datasets.build_dataset_mapper(cfg, name, stage)
```

## 3.4 xmodaler.engine

```
xmodaler.engine.launch(main_func, num_gpus_per_machine, num_machines=1, machine_rank=0,
                       dist_url=None, args=(), timeout=datetime.timedelta(seconds=1800))
```

Launch multi-gpu or distributed training. This function must be called on all machines involved in the training. It will spawn child processes (defined by `num_gpus_per_machine`) on each machine.

### Parameters

- **main\_func** – a function that will be called by `main_func(*args)`
- **num\_gpus\_per\_machine** (*int*) – number of GPUs per machine
- **num\_machines** (*int*) – the total number of machines
- **machine\_rank** (*int*) – the rank of this machine
- **dist\_url** (*str*) – url to connect to for distributed jobs, including protocol e.g. “`tcp://127.0.0.1:8686`”. Can be set to “auto” to automatically select a free port on localhost
- **timeout** (*timedelta*) – timeout of the distributed workers
- **args** (*tuple*) – arguments passed to `main_func`

```
xmodaler.engine.build_engine(cfg)
```

```
xmodaler.engine.default_argument_parser(epilog=None)
```

Create a parser with some common arguments used by X-modaler users.

### Parameters

**epilog** (*str*) – epilog passed to `ArgumentParser` describing the usage.

### Return type

`argparse.ArgumentParser`

```
xmodaler.engine.default_setup(cfg, args)
```

Perform some basic common setups at the beginning of a job, including:

1. Set up the X-modaler logger
2. Log basic information about environment, cmdline arguments, and config
3. Backup the config to the output directory

### Parameters

- **cfg** (*CfgNode*) – the full config to be used
- **args** (*argparse.Namespace*) – the command line arguments to be logged

```
xmodaler.engine.default_writers(output_dir: str, max_iter: Optional[int] = None)
```

Build a list of `EventWriter` to be used. It now consists of a `CommonMetricPrinter`, `TensorboardXWriter` and `JSONWriter`.

### Parameters

- **output\_dir** – directory to store JSON metrics and tensorboard events
- **max\_iter** – the total number of iterations

**Returns**

a list of `EventWriter` objects.

**Return type**

`list[EventWriter]`

**class** `xmodaler.engine.DefaultTrainer(cfg)`

Bases: `TrainerBase`

A trainer with default training logic. It does the following:

1. Create a `DefaultTrainer` using model, optimizer, dataloader defined by the given config. Create a LR scheduler defined by the config.
2. Load the last checkpoint or `cfg.MODEL.WEIGHTS`, if exists, when `resume_or_load` is called.
3. Register a few common hooks defined by the config.

It is created to simplify the **standard model training workflow** and reduce code boilerplate for users who only need the standard training workflow, with standard features. It means this class makes *many assumptions* about your training logic that may easily become invalid in a new research. In fact, any assumptions beyond those made in the `DefaultTrainer` are too much for research.

The code of this class has been annotated about restrictive assumptions it makes. When they do not work for you, you're encouraged to:

1. Overwrite methods of this class, OR:
2. Use `DefaultTrainer`, which only does minimal SGD training and nothing else. You can then add your own hooks if needed. OR:
3. Write your own training loop similar to `train_net.py`.

See the [Training](#) tutorials for more details.

Note that the behavior of this class, like other functions/classes in this file, is not stable, since it is meant to represent the “common default behavior”. It is only guaranteed to work well with the standard models and training workflow in X-modaler. To obtain more stable behavior, write your own training logic with other public APIs.

Examples:

```
trainer = DefaultTrainer(cfg)
trainer.resume_or_load() # load last checkpoint or MODEL.WEIGHTS
trainer.train()
```

**scheduler**

**checkpointer**

**Type**

`XmodalerCheckpointer`

**cfg**

**Type**

`CfgNode`



`__init__(cfg)`

**Parameters**

`cfg` (`CfgNode`) –

`_write_metrics(loss_dict: Dict[str, Tensor], data_time: float, prefix: str = "")`

**Parameters**

- `loss_dict` (`dict`) – dict of scalar losses
- `data_time` (`float`) – time taken by the dataloader iteration

`after_step()`

`after_train()`

`static auto_scale_workers(cfg, num_workers: int)`

`before_step()`

`before_train()`

`build_hooks()`

`classmethod build_losses(cfg)`

`classmethod build_lr_scheduler(cfg, optimizer, iters_per_epoch)`

`classmethod build_model(cfg)`

`classmethod build_optimizer(cfg, model)`

`classmethod build_test_loader(cfg)`

`classmethod build_train_loader(cfg)`

`classmethod build_val_loader(cfg)`

`build_writers()`

`load_state_dict(state_dict)`

`register_hooks(hooks: List[Optional[HookBase]]) → None`

Register hooks to the trainer. The hooks are executed in the order they are registered.

**Parameters**

`hooks` (`list[Optional[HookBase]]`) – list of hooks

`resume_or_load(resume=True)`

`run_step()`

Implement the standard training logic described above.

`state_dict()`

`classmethod test(cfg, model, test_data_loader, evaluator, epoch)`

**train()**

**Parameters**

- **start\_iter** (*int*) – See docs above
- **max\_iter** (*int*) – See docs above

**class** xmodaler.engine.HookBase

Bases: object

Base class for hooks that can be registered with [TrainerBase](#).

Each hook can implement 4 methods. The way they are called is demonstrated in the following snippet:

```
hook.before_train()
for iter in range(start_iter, max_iter):
    hook.before_step()
    trainer.run_step()
    hook.after_step()
iter += 1
hook.after_train()
```

**Notes**

1. In the hook method, users can access `self.trainer` to access more properties about the context (e.g., model, current iteration, or config if using [DefaultTrainer](#)).
2. A hook that does something in [before\\_step\(\)](#) can often be implemented equivalently in [after\\_step\(\)](#). If the hook takes non-trivial time, it is strongly recommended to implement the hook in [after\\_step\(\)](#) instead of [before\\_step\(\)](#). The convention is that [before\\_step\(\)](#) should only take negligible time.

Following this convention will allow hooks that do care about the difference between [before\\_step\(\)](#) and [after\\_step\(\)](#) (e.g., timer) to function properly.

**after\_step()**

Called after each iteration.

**after\_train()**

Called after the last iteration.

**before\_step()**

Called before each iteration.

**before\_train()**

Called before the first iteration.

**state\_dict()**

Hooks are stateless by default, but can be made checkpointable by implementing `state_dict` and `load_state_dict`.

**trainer:** [TrainerBase](#) = None

A weak reference to the trainer object. Set by the trainer when the hook is registered.

**class xmodaler.engine.TrainerBase**

Bases: object

Base class for iterative trainer with hooks.

The only assumption we made here is: the training runs in a loop. A subclass can implement what the loop is. We made no assumptions about the existence of dataloader, optimizer, model, etc.

**iter**

the current iteration.

**Type**

int

**start\_iter**

The iteration to start with. By convention the minimum possible value is 0.

**Type**

int

**max\_iter**

The iteration to end training.

**Type**

int

**storage**

An EventStorage that's opened during the course of training.

**Type**

*EventStorage*

**\_\_init\_\_**() → None

**after\_step**()

**after\_train**()

**before\_step**()

**before\_train**()

**load\_state\_dict**(*state\_dict*)

**register\_hooks**(*hooks*: List[Optional[HookBase]]) → None

Register hooks to the trainer. The hooks are executed in the order they are registered.

**Parameters**

**hooks** (list[Optional[HookBase]]) – list of hooks

**run\_step**()

**state\_dict**()

**train**(*start\_iter*: int, *max\_iter*: int)

**Parameters**

- **start\_iter** (*int*) – See docs above
- **max\_iter** (*int*) – See docs above

```
class xmodaler.engine.CallbackHook(*, before_train=None, after_train=None, before_step=None,
                                   after_step=None)
```

Bases: [HookBase](#)

Create a hook using callback functions provided by the user.

```
__init__(*, before_train=None, after_train=None, before_step=None, after_step=None)
```

Each argument is a function that takes one argument: the trainer.

**after\_step()**

Called after each iteration.

**after\_train()**

Called after the last iteration.

**before\_step()**

Called before each iteration.

**before\_train()**

Called before the first iteration.

**state\_dict()**

Hooks are stateless by default, but can be made checkpointable by implementing *state\_dict* and *load\_state\_dict*.

**trainer:** [TrainerBase](#) = None

A weak reference to the trainer object. Set by the trainer when the hook is registered.

```
class xmodaler.engine.IterationTimer(warmup_iter=3)
```

Bases: [HookBase](#)

Track the time spent for each iteration (each *run\_step* call in the trainer). Print a summary in the end of training.

This hook uses the time between the call to its *before\_step()* and *after\_step()* methods. Under the convention that *before\_step()* of all hooks should only take negligible amount of time, the *IterationTimer* hook should be placed at the beginning of the list of hooks to obtain accurate timing.

```
__init__(warmup_iter=3)
```

**Parameters**

**warmup\_iter** (*int*) – the number of iterations at the beginning to exclude from timing.

**after\_step()**

Called after each iteration.

**after\_train()**

Called after the last iteration.

**before\_step()**

Called before each iteration.

**before\_train()**

Called before the first iteration.

**state\_dict()**

Hooks are stateless by default, but can be made checkpointable by implementing *state\_dict* and *load\_state\_dict*.

**trainer:** *TrainerBase* = None

A weak reference to the trainer object. Set by the trainer when the hook is registered.

**class** xmodaler.engine.PeriodicWriter(*writers*, *period*=20)

Bases: *HookBase*

Write events to EventStorage (by calling `writer.write()`) periodically.

It is executed every *period* iterations and after the last iteration. Note that *period* does not affect how data is smoothed by each writer.

**\_\_init\_\_**(*writers*, *period*=20)

#### Parameters

- **writers** (*list*[*EventWriter*]) – a list of EventWriter objects
- **period** (*int*) –

**after\_step**()

Called after each iteration.

**after\_train**()

Called after the last iteration.

**before\_step**()

Called before each iteration.

**before\_train**()

Called before the first iteration.

**state\_dict**()

Hooks are stateless by default, but can be made checkpointable by implementing *state\_dict* and *load\_state\_dict*.

**trainer:** *TrainerBase* = None

A weak reference to the trainer object. Set by the trainer when the hook is registered.

**class** xmodaler.engine.PeriodicCheckpoint(*checkpointer: Checkpointer*, *period: int*, *max\_iter: Optional[int] = None*, *max\_to\_keep: Optional[int] = None*, *file\_prefix: str = 'model'*)

Bases: *PeriodicEpochCheckpoint*, *HookBase*

Same as `detectron2.checkpoint.PeriodicCheckpoint`, but as a hook.

Note that when used as a hook, it is unable to save additional data other than what's defined by the given *checkpointer*.

It is executed every *period* iterations and after the last iteration.

**\_\_init\_\_**(*checkpointer: Checkpointer*, *period: int*, *max\_iter: Optional[int] = None*, *max\_to\_keep: Optional[int] = None*, *file\_prefix: str = 'model'*) → None

#### Parameters

- **checkpointer** – the checkpointer object used to save checkpoints.
- **period** (*int*) – the period to save checkpoint.
- **max\_iter** (*int*) – maximum number of iterations. When it is reached, a checkpoint named “{file\_prefix}\_final” will be saved.

- **max\_to\_keep** (*int*) – maximum number of most current checkpoints to keep, previous checkpoints will be deleted
- **file\_prefix** (*str*) – the prefix of checkpoint's filename

**after\_step()**

Called after each iteration.

**after\_train()**

Called after the last iteration.

**before\_step()**

Called before each iteration.

**before\_train()**

Called before the first iteration.

**save**(*name: str, \*\*kwargs: Any*) → None

Same argument as `Checkpointter.save()`. Use this method to manually save checkpoints outside the schedule.

**Parameters**

- **name** (*str*) – file name.
- **kwargs** (*Any*) – extra data to save, same as in `Checkpointter.save()`.

**state\_dict()**

Hooks are stateless by default, but can be made checkpointable by implementing *state\_dict* and *load\_state\_dict*.

**step**(*iteration: int, epoch: int, \*\*kwargs: Any*) → None

Perform the appropriate action at the given iteration.

**Parameters**

- **iteration** (*int*) – the current iteration, ranged in `[0, max_iter-1]`.
- **kwargs** (*Any*) – extra data to save, same as in `Checkpointter.save()`.

**trainer:** *TrainerBase* = None

A weak reference to the trainer object. Set by the trainer when the hook is registered.

**class** `xmodaler.engine.LRScheduler`(*optimizer=None, scheduler=None*)

Bases: *HookBase*

A hook which executes a torch builtin LR scheduler and summarizes the LR. It is executed after every iteration.

**\_\_init\_\_**(*optimizer=None, scheduler=None*)

**Parameters**

- **optimizer** (*torch.optim.Optimizer*) –
- **scheduler** (*torch.optim.LRScheduler*) –

If any argument is not given, will try to obtain it from the trainer.

**after\_step()**

Called after each iteration.

**after\_train()**

Called after the last iteration.

**before\_step()**

Called before each iteration.

**before\_train()**

Called before the first iteration.

**state\_dict()**

Hooks are stateless by default, but can be made checkpointable by implementing *state\_dict* and *load\_state\_dict*.

**trainer:** *TrainerBase* = None

A weak reference to the trainer object. Set by the trainer when the hook is registered.

**class** xmodaler.engine.AutogradProfiler(*enable\_predicate*, *output\_dir*, \*, *use\_cuda=True*)

Bases: *HookBase*

A hook which runs *torch.autograd.profiler.profile*.

Examples:

```
hooks.AutogradProfiler(
    lambda trainer: trainer.iter > 10 and trainer.iter < 20, self.cfg.OUTPUT_DIR
)
```

The above example will run the profiler for iteration 10~20 and dump results to OUTPUT\_DIR. We did not profile the first few iterations because they are typically slower than the rest. The result files can be loaded in the `chrome://tracing` page in chrome browser.

---

**Note:** When used together with NCCL on older version of GPUs, autograd profiler may cause deadlock because it unnecessarily allocates memory on every device it sees. The memory management calls, if interleaved with NCCL calls, lead to deadlock on GPUs that do not support `cudaLaunchCooperativeKernelMultiDevice`.

---

**\_\_init\_\_**(*enable\_predicate*, *output\_dir*, \*, *use\_cuda=True*)

**Parameters**

- **enable\_predicate** (*callable*[*trainer* -> *bool*]) – a function which takes a trainer, and returns whether to enable the profiler. It will be called once every step, and can be used to select which steps to profile.
- **output\_dir** (*str*) – the output directory to dump tracing files.
- **use\_cuda** (*bool*) – same as in *torch.autograd.profiler.profile*.

**after\_step()**

Called after each iteration.

**after\_train()**

Called after the last iteration.

**before\_step()**

Called before each iteration.

**before\_train()**

Called before the first iteration.

**state\_dict()**

Hooks are stateless by default, but can be made checkpointable by implementing *state\_dict* and *load\_state\_dict*.

**trainer:** *TrainerBase* = None

A weak reference to the trainer object. Set by the trainer when the hook is registered.

```
class xmodaler.engine.EvalHook(eval_period, eval_start, eval_function, iters_per_epoch, stage,
                               multi_gpu_eval)
```

Bases: *HookBase*

Run an evaluation function periodically, and at the end of training.

It is executed every *eval\_period* iterations and after the last iteration.

```
__init__(eval_period, eval_start, eval_function, iters_per_epoch, stage, multi_gpu_eval)
```

**Parameters**

- **eval\_period** (*int*) – the period to run *eval\_function*. Set to 0 to not evaluate periodically (but still after the last iteration).
- **eval\_function** (*callable*) – a function which takes no arguments, and returns a nested dict of evaluation metrics.

---

**Note:** This hook must be enabled in all or none workers. If you would like only certain workers to perform evaluation, give other workers a no-op function (*eval\_function=lambda: None*).

---

**\_do\_eval**(*epoch*)**after\_step**()

Called after each iteration.

**after\_train**()

Called after the last iteration.

**before\_step**()

Called before each iteration.

**before\_train**()

Called before the first iteration.

**state\_dict**()

Hooks are stateless by default, but can be made checkpointable by implementing *state\_dict* and *load\_state\_dict*.

**trainer:** *TrainerBase* = None

A weak reference to the trainer object. Set by the trainer when the hook is registered.

```
class xmodaler.engine.PreciseBN(period, model, data_loader, num_iter)
```

Bases: *HookBase*

The standard implementation of BatchNorm uses EMA in inference, which is sometimes suboptimal. This class computes the true average of statistics rather than the moving average, and put true averages to every BN layer in the given model.

It is executed every *period* iterations and after the last iteration.



**\_\_init\_\_**(*period, model, data\_loader, num\_iter*)

#### Parameters

- **period** (*int*) – the period this hook is run, or 0 to not run during training. The hook will always run in the end of training.
- **model** (*nn.Module*) – a module whose all BN layers in training mode will be updated by precise BN. Note that user is responsible for ensuring the BN layers to be updated are in training mode when this hook is triggered.
- **data\_loader** (*iterable*) – it will produce data to be run by *model(data)*.
- **num\_iter** (*int*) – number of iterations used to compute the precise statistics.

**after\_step**()

Called after each iteration.

**after\_train**()

Called after the last iteration.

**before\_step**()

Called before each iteration.

**before\_train**()

Called before the first iteration.

**state\_dict**()

Hooks are stateless by default, but can be made checkpointable by implementing *state\_dict* and *load\_state\_dict*.

**trainer:** *TrainerBase* = None

A weak reference to the trainer object. Set by the trainer when the hook is registered.

**update\_stats**()

Update the model with precise statistics. Users can manually call this method.

**class** xmodaler.engine.**ModelWeightsManipulating**

Bases: *HookBase*

Init or bind weights after loading a model

**\_\_init\_\_**()

**after\_step**()

Called after each iteration.

**after\_train**()

Called after the last iteration.

**before\_step**()

Called before each iteration.

**before\_train**()

Called before the first iteration.

**state\_dict**()

Hooks are stateless by default, but can be made checkpointable by implementing *state\_dict* and *load\_state\_dict*.

**trainer:** `TrainerBase` = `None`

A weak reference to the trainer object. Set by the trainer when the hook is registered.

**class** `xmodaler.engine.RetrievalTrainer`(`cfg`)

Bases: `DefaultTrainer`

`__init__`(`cfg`)

Parameters

`cfg` (`CfgNode`) –

`_write_metrics`(`loss_dict`: `Dict[str, Tensor]`, `data_time`: `float`, `prefix`: `str` = `"`)

Parameters

- `loss_dict` (`dict`) – dict of scalar losses
- `data_time` (`float`) – time taken by the dataloader iteration

`after_step`()

`after_train`()

`static auto_scale_workers`(`cfg`, `num_workers`: `int`)

`before_step`()

`before_train`()

`build_hooks`()

`classmethod build_losses`(`cfg`)

`classmethod build_lr_scheduler`(`cfg`, `optimizer`, `iters_per_epoch`)

`classmethod build_model`(`cfg`)

`classmethod build_optimizer`(`cfg`, `model`)

`classmethod build_test_loader`(`cfg`)

`classmethod build_train_loader`(`cfg`)

`classmethod build_val_loader`(`cfg`)

`build_writers`()

`load_state_dict`(`state_dict`)

`register_hooks`(`hooks`: `List[Optional[HookBase]]`) → `None`

Register hooks to the trainer. The hooks are executed in the order they are registered.

Parameters

`hooks` (`list[Optional[HookBase]]`) – list of hooks

`resume_or_load`(`resume`=`True`)

`run_step`()

Implement the standard training logic described above.

`state_dict`()

```
classmethod test(cfg, model, test_data_loader, evaluator, epoch)
```

```
train()
```

#### Parameters

- **start\_iter** (*int*) – See docs above
- **max\_iter** (*int*) – See docs above

```
class xmodaler.engine.RLTrainer(cfg)
```

Bases: [DefaultTrainer](#)

```
__init__(cfg)
```

#### Parameters

**cfg** ([CfgNode](#)) –

```
_write_metrics(loss_dict: Dict[str, Tensor], data_time: float, prefix: str = "")
```

#### Parameters

- **loss\_dict** (*dict*) – dict of scalar losses
- **data\_time** (*float*) – time taken by the dataloader iteration

```
after_step()
```

```
after_train()
```

```
static auto_scale_workers(cfg, num_workers: int)
```

```
before_step()
```

```
before_train()
```

```
build_hooks()
```

```
classmethod build_losses(cfg)
```

```
classmethod build_lr_scheduler(cfg, optimizer, iters_per_epoch)
```

```
classmethod build_model(cfg)
```

```
classmethod build_optimizer(cfg, model)
```

```
classmethod build_scorer(cfg)
```

```
classmethod build_test_loader(cfg)
```

```
classmethod build_train_loader(cfg)
```

```
classmethod build_val_loader(cfg)
```

```
build_writers()
```

```
load_state_dict(state_dict)
```

```
register_hooks(hooks: List[Optional[HookBase]]) → None
```

Register hooks to the trainer. The hooks are executed in the order they are registered.

#### Parameters

**hooks** (*list[Optional[HookBase]]*) – list of hooks

```
resume_or_load(resume=True)
```

```
run_step()
```

Implement the standard training logic described above.

```
state_dict()
```

```
classmethod test(cfg, model, test_data_loader, evaluator, epoch)
```

```
train()
```

#### Parameters

- **start\_iter** (*int*) – See docs above
- **max\_iter** (*int*) – See docs above

```
class xmodaler.engine.RLBeamTrainer(cfg)
```

Bases: [\*DefaultTrainer\*](#)

```
__init__(cfg)
```

#### Parameters

**cfg** ([\*CfgNode\*](#)) –

```
_write_metrics(loss_dict: Dict[str, Tensor], data_time: float, prefix: str = "")
```

#### Parameters

- **loss\_dict** (*dict*) – dict of scalar losses
- **data\_time** (*float*) – time taken by the dataloader iteration

```
after_step()
```

```
after_train()
```

```
static auto_scale_workers(cfg, num_workers: int)
```

```
before_step()
```

```
before_train()
```

```
build_hooks()
```

```
classmethod build_losses(cfg)
```

```
classmethod build_lr_scheduler(cfg, optimizer, iters_per_epoch)
```

```
classmethod build_model(cfg)
```

```
classmethod build_optimizer(cfg, model)
```

```
classmethod build_scorer(cfg)
```

```
classmethod build_test_loader(cfg)
```

```
classmethod build_train_loader(cfg)
```

```
classmethod build_val_loader(cfg)
```

```
build_writers()
```

`load_state_dict(state_dict)`

`register_hooks(hooks: List[Optional[HookBase]]) → None`

Register hooks to the trainer. The hooks are executed in the order they are registered.

**Parameters**

`hooks` (`List[Optional[HookBase]]`) – list of hooks

`resume_or_load(resume=True)`

`run_step()`

Implement the standard training logic described above.

`state_dict()`

`classmethod test(cfg, model, test_data_loader, evaluator, epoch)`

`train()`

**Parameters**

- `start_iter` (`int`) – See docs above
- `max_iter` (`int`) – See docs above

`class xmodaler.engine.SingleStreamRetrievalTrainer(cfg)`

Bases: `DefaultTrainer`

`__init__(cfg)`

**Parameters**

`cfg` (`CfgNode`) –

`_write_metrics(loss_dict: Dict[str, Tensor], data_time: float, prefix: str = "")`

**Parameters**

- `loss_dict` (`dict`) – dict of scalar losses
- `data_time` (`float`) – time taken by the dataloader iteration

`after_step()`

`after_train()`

`static auto_scale_workers(cfg, num_workers: int)`

`before_step()`

`before_train()`

`build_hooks()`

`classmethod build_losses(cfg)`

`classmethod build_lr_scheduler(cfg, optimizer, iters_per_epoch)`

`classmethod build_model(cfg)`

`classmethod build_optimizer(cfg, model)`

`classmethod build_test_loader(cfg)`

**classmethod** `build_train_loader(cfg)`

**classmethod** `build_val_loader(cfg)`

**build\_writers()**

**load\_state\_dict**(*state\_dict*)

**register\_hooks**(*hooks*: *List[Optional[HookBase]]*) → None

Register hooks to the trainer. The hooks are executed in the order they are registered.

**Parameters**

**hooks** (*list[Optional[HookBase]]*) – list of hooks

**resume\_or\_load**(*resume=True*)

**run\_step()**

Implement the standard training logic described above.

**state\_dict()**

**classmethod** `test(cfg, model, test_data_loader, evaluator, epoch)`

**train()**

**Parameters**

- **start\_iter** (*int*) – See docs above
- **max\_iter** (*int*) – See docs above

**class** `xmodaler.engine.SingleStreamRetrievalTrainerHardNegatives(cfg)`

Bases: [SingleStreamRetrievalTrainer](#)

**\_\_init\_\_**(*cfg*)

**Parameters**

**cfg** (*CfgNode*) –

**\_write\_metrics**(*loss\_dict*: *Dict[str, Tensor]*, *data\_time*: *float*, *prefix*: *str* = "")

**Parameters**

- **loss\_dict** (*dict*) – dict of scalar losses
- **data\_time** (*float*) – time taken by the dataloader iteration

**after\_step()**

**after\_train()**

**static** `auto_scale_workers(cfg, num_workers: int)`

**before\_step()**

**before\_train()**

**build\_hooks()**

**classmethod** `build_losses(cfg)`

**classmethod** `build_lr_scheduler(cfg, optimizer, iters_per_epoch)`

```

classmethod build_model(cfg)

classmethod build_optimizer(cfg, model)

classmethod build_test_loader(cfg)

classmethod build_train_loader(cfg)

classmethod build_val_loader(cfg)

build_writers()

clip_inputs(data)

hard_negative_mining(data)

load_state_dict(state_dict)

register_hooks(hooks: List[Optional[HookBase]]) → None
    Register hooks to the trainer. The hooks are executed in the order they are registered.

    Parameters
        hooks (list[Optional[HookBase]]) – list of hooks

resume_or_load(resume=True)

run_step()
    Implement the standard training logic described above.

state_dict()

classmethod test(cfg, model, test_data_loader, evaluator, epoch)

train()

    Parameters
        • start_iter (int) – See docs above
        • max_iter (int) – See docs above
class xmodaler.engine.TDENPretrainer(cfg)
    Bases: DefaultTrainer
    __init__(cfg)

    Parameters
        cfg (CfgNode) –

    _write_metrics(loss_dict: Dict[str, Tensor], data_time: float, prefix: str = "")

    Parameters
        • loss_dict (dict) – dict of scalar losses
        • data_time (float) – time taken by the dataloader iteration

after_step()

after_train()

static auto_scale_workers(cfg, num_workers: int)

```

`before_step()`

`before_train()`

`build_hooks()`

`classmethod build_losses(cfg)`

`classmethod build_lr_scheduler(cfg, optimizer, iters_per_epoch)`

`classmethod build_model(cfg)`

`classmethod build_optimizer(cfg, model)`

`classmethod build_test_loader(cfg)`

`classmethod build_train_loader(cfg)`

`classmethod build_val_loader(cfg)`

`build_writers()`

`load_state_dict(state_dict)`

`register_hooks(hooks: List[Optional[HookBase]]) → None`

Register hooks to the trainer. The hooks are executed in the order they are registered.

**Parameters**

**hooks** (`list[Optional[HookBase]]`) – list of hooks

`resume_or_load(resume=True)`

`run_step()`

Implement the standard training logic described above.

`state_dict()`

`classmethod test(cfg, model, test_data_loader, evaluator, epoch)`

`train()`

**Parameters**

- **start\_iter** (`int`) – See docs above
- **max\_iter** (`int`) – See docs above

`class xmodaler.engine.VCRTrainer(cfg)`

Bases: `DefaultTrainer`

`__init__(cfg)`

**Parameters**

**cfg** (`CfgNode`) –

`_write_metrics(loss_dict: Dict[str, Tensor], data_time: float, prefix: str = "")`

**Parameters**

- **loss\_dict** (`dict`) – dict of scalar losses
- **data\_time** (`float`) – time taken by the dataloader iteration



```

after_step()
after_train()
static auto_scale_workers(cfg, num_workers: int)
before_step()
before_train()
build_hooks()
classmethod build_losses(cfg)
classmethod build_lr_scheduler(cfg, optimizer, iters_per_epoch)
classmethod build_model(cfg)
classmethod build_optimizer(cfg, model)
classmethod build_test_loader(cfg)
classmethod build_train_loader(cfg)
classmethod build_val_loader(cfg)
build_writers()
load_state_dict(state_dict)
register_hooks(hooks: List[Optional[HookBase]]) → None
    Register hooks to the trainer. The hooks are executed in the order they are registered.
    Parameters
        hooks (list[Optional[HookBase]]) – list of hooks
resume_or_load(resume=True)
run_step()
    Implement the standard training logic described above.
state_dict()
classmethod test(cfg, model, test_data_loader, evaluator, epoch)
train()
    Parameters
        • start_iter (int) – See docs above
        • max_iter (int) – See docs above

```

## 3.5 xmodaler.evaluation

`xmodaler.evaluation.build_evaluation(cfg, annfile, output_dir)`

**class** `xmodaler.evaluation.COCOEvaler(cfg, annfile, output_dir)`

Bases: `object`

**\_\_init\_\_**(*cfg, annfile, output\_dir*)

**eval**(*results, epoch*)

**class** `xmodaler.evaluation.VQAEvaler(cfg, annfile, output_dir)`

Bases: `object`

**\_\_init\_\_**(*cfg, annfile, output\_dir*)

**eval**(*results, epoch*)

**class** `xmodaler.evaluation.VCREvaler(cfg, annfile, output_dir)`

Bases: `object`

**\_\_init\_\_**(*cfg, annfile, output\_dir*)

**eval**(*results\_list, epoch*)

**class** `xmodaler.evaluation.RetrievalEvaler(cfg, annfile, output_dir)`

Bases: `object`

**\_\_init\_\_**(*cfg, annfile, output\_dir*)

**eval**(*vfeats, tfeats, labels*)

## 3.6 xmodaler.functional

`xmodaler.functional.bboxes_to_locfeats(bboxes, image_w, image_h)`

`xmodaler.functional.caption_to_mask_tokens(caption, max_seq_length, tokenizer, need_g_tokens=False, need_no_mask_tokens=False, must_mask=False)`

`xmodaler.functional.decode_sequence(vocab, seq)`

`xmodaler.functional.decode_sequence_bert(tokenizer, seq, sep_token_id)`

`xmodaler.functional.dict_as_tensor(input_dict)`

`xmodaler.functional.dict_to_cuda(input_dict)`

`xmodaler.functional.expand_tensor(tensor, size, dim=1)`

`xmodaler.functional.clip_v_inputs(v_feats, spatial, image_mask)`

`xmodaler.functional.clip_t_inputs(input_txt, segment_ids, input_mask)`

`xmodaler.functional.iou(anchors, gt_boxes)`

`anchors`: (N, 4) ndarray of float `gt_boxes`: (K, 4) ndarray of float `overlaps`: (N, K) ndarray of overlap between `boxes` and `query_boxes`

```

xmodaler.functional.load_vocab(path)
xmodaler.functional.pad_tensor(tensor, padding_value, use_mask)
xmodaler.functional.random_region(image_feats, overlaps)
xmodaler.functional.random_word(tokens, tokenizer, must_mask=False)
xmodaler.functional.read_lines(path)
xmodaler.functional.read_lines_set(path)
xmodaler.functional.read_np(path)
xmodaler.functional.read_np_bbox(path, max_feat_num, use_global_v=True)
xmodaler.functional.flat_list_of_lists(l)
    flatten a list of lists [[1,2], [3,4]] to [1,2,3,4]

```

### 3.7 xmodaler.losses

**class** xmodaler.losses.BCEWithLogits

Bases: Module

**\_\_init\_\_**()

Initializes internal Module state, shared by both nn.Module and ScriptModule.

**classmethod** add\_config(cfg)

**forward**(outputs\_dict)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod** from\_config(cfg)

**training:** bool

**class** xmodaler.losses.BatchTriplet(margin, max\_violation)

Bases: Module

**\_\_init\_\_**(margin, max\_violation)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

**forward**(outputs\_dict)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod** `from_config(cfg)`

**training:** `bool`

**class** `xmodaler.losses.Triplet(margin)`

Bases: `Module`

**\_\_init\_\_**(*margin*)

Initializes internal Module state, shared by both `nn.Module` and `ScriptModule`.

**classmethod** `add_config(cfg)`

**forward**(*outputs\_dict*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod** `from_config(cfg)`

**training:** `bool`

**class** `xmodaler.losses.CrossEntropy`

Bases: `Module`

**\_\_init\_\_**()

Initializes internal Module state, shared by both `nn.Module` and `ScriptModule`.

**classmethod** `add_config(cfg)`

**forward**(*outputs\_dict*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod** `from_config(cfg)`

**training:** `bool`

**class** `xmodaler.losses.LabelSmoothing(*, label_smoothing)`

Bases: `Module`

**Forward**(*logits, targets*)

---

```
__init__(*, label_smoothing)
```

Initializes internal Module state, shared by both nn.Module and ScriptModule.

```
classmethod add_config(cfg)
```

```
forward(outputs_dict)
```

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```
classmethod from_config(cfg)
```

```
training: bool
```

```
class xmodaler.losses.PretrainLosses(margin, max_violation)
```

Bases: Module

```
__init__(margin, max_violation)
```

Initializes internal Module state, shared by both nn.Module and ScriptModule.

```
classmethod add_config(cfg)
```

```
forward(batched_inputs)
```

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```
classmethod from_config(cfg)
```

```
select_logits_targets_by_mask(tensor, target, mask)
```

```
training: bool
```

```
class xmodaler.losses.RewardCriterion(eos_id)
```

Bases: Module

```
__init__(eos_id)
```

Initializes internal Module state, shared by both nn.Module and ScriptModule.

```
classmethod add_config(cfg)
```

```
forward(outputs_dict)
```

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```
classmethod from_config(cfg)
```

```
training: bool
```

```
xmodaler.losses.build_losses(cfg)
```

```
xmodaler.losses.build_rl_losses(cfg)
```

## 3.8 xmodaler.lr\_scheduler

```
xmodaler.lr_scheduler.build_lr_scheduler(cfg, optimizer, data_size)
```

```
class xmodaler.lr_scheduler.StepLR(*, optimizer, step_size, gamma=0.1)
```

```
Bases: StepLR
```

```
__init__(*, optimizer, step_size, gamma=0.1)
```

```
_get_closed_form_lr()
```

```
_initial_step()
```

Initialize step counts and performs a step

```
classmethod from_config(cfg, optimizer, data_size)
```

```
get_last_lr()
```

Return last computed learning rate by current scheduler.

```
get_lr()
```

```
load_state_dict(state_dict)
```

Loads the schedulers state.

### Parameters

**state\_dict** (*dict*) – scheduler state. Should be an object returned from a call to `state_dict()`.

```
print_lr(is_verbose, group, lr, epoch=None)
```

Display the current learning rate.

```
state_dict()
```

Returns the state of the scheduler as a `dict`.

It contains an entry for every variable in `self.__dict__` which is not the optimizer.

```
step(epoch=None)
```

```
class xmodaler.lr_scheduler.NoamLR(*, optimizer, model_size, factor, warmup, last_epoch=-1)
```

```
Bases: _LRScheduler
```

```
__init__(*, optimizer, model_size, factor, warmup, last_epoch=-1)
```

**\_initial\_step()**

Initialize step counts and performs a step

**classmethod** **from\_config**(*cfg, optimizer, data\_size*)

**get\_last\_lr()**

Return last computed learning rate by current scheduler.

**get\_lr()**

**load\_state\_dict**(*state\_dict*)

Loads the schedulers state.

**Parameters**

**state\_dict** (*dict*) – scheduler state. Should be an object returned from a call to [state\\_dict\(\)](#).

**print\_lr**(*is\_verbose, group, lr, epoch=None*)

Display the current learning rate.

**state\_dict()**

Returns the state of the scheduler as a **dict**.

It contains an entry for every variable in self.\_\_dict\_\_ which is not the optimizer.

**step**(*epoch=None*)

**class** **xmodaler.lr\_scheduler.WarmupConstant**(*\*, optimizer, warmup\_steps, last\_epoch=-1*)

Bases: **LambdaLR**

Linear warmup and then constant. Linearly increases learning rate schedule from 0 to 1 over *warmup\_steps* training steps. Keeps learning rate schedule equal to 1. after *warmup\_steps*.

**\_\_init\_\_**(*\*, optimizer, warmup\_steps, last\_epoch=-1*)

**\_initial\_step()**

Initialize step counts and performs a step

**classmethod** **from\_config**(*cfg, optimizer, data\_size*)

**get\_last\_lr()**

Return last computed learning rate by current scheduler.

**get\_lr()**

**load\_state\_dict**(*state\_dict*)

Loads the schedulers state.

When saving or loading the scheduler, please make sure to also save or load the state of the optimizer.

**Parameters**

**state\_dict** (*dict*) – scheduler state. Should be an object returned from a call to [state\\_dict\(\)](#).

**lr\_lambda**(*step*)

**print\_lr**(*is\_verbose, group, lr, epoch=None*)

Display the current learning rate.

**state\_dict()**

Returns the state of the scheduler as a dict.

It contains an entry for every variable in self.\_\_dict\_\_ which is not the optimizer. The learning rate lambda functions will only be saved if they are callable objects and not if they are functions or lambdas.

When saving or loading the scheduler, please make sure to also save or load the state of the optimizer.

**step(epoch=None)**

**class xmodaler.lr\_scheduler.WarmupLinear**(\*, optimizer, min\_lr, warmup\_steps, t\_total, last\_epoch=-1)

Bases: LambdaLR

Linear warmup and then linear decay. Linearly increases learning rate from 0 to 1 over *warmup\_steps* training steps. Linearly decreases learning rate from 1. to 0. over remaining *t\_total* - *warmup\_steps* steps.

**\_\_init\_\_**(\*, optimizer, min\_lr, warmup\_steps, t\_total, last\_epoch=-1)

**\_initial\_step()**

Initialize step counts and performs a step

**classmethod from\_config**(cfg, optimizer, data\_size)

**get\_last\_lr()**

Return last computed learning rate by current scheduler.

**get\_lr()****load\_state\_dict(state\_dict)**

Loads the schedulers state.

When saving or loading the scheduler, please make sure to also save or load the state of the optimizer.

**Parameters**

**state\_dict** (dict) – scheduler state. Should be an object returned from a call to [state\\_dict\(\)](#).

**lr\_lambda(step)**

**print\_lr**(is\_verbose, group, lr, epoch=None)

Display the current learning rate.

**state\_dict()**

Returns the state of the scheduler as a dict.

It contains an entry for every variable in self.\_\_dict\_\_ which is not the optimizer. The learning rate lambda functions will only be saved if they are callable objects and not if they are functions or lambdas.

When saving or loading the scheduler, please make sure to also save or load the state of the optimizer.

**step(epoch=None)**

**class xmodaler.lr\_scheduler.WarmupCosine**(\*, optimizer, min\_lr, warmup\_steps, t\_total, cycles=0.5, last\_epoch=-1)

Bases: LambdaLR

Linear warmup and then cosine decay. Linearly increases learning rate from 0 to 1 over *warmup\_steps* training steps. Decreases learning rate from 1. to 0. over remaining *t\_total* - *warmup\_steps* steps following a cosine curve. If *cycles* (default=0.5) is different from default, learning rate follows cosine function after warmup.

**\_\_init\_\_**(\*, optimizer, min\_lr, warmup\_steps, t\_total, cycles=0.5, last\_epoch=-1)



**\_initial\_step()**

Initialize step counts and performs a step

**classmethod** **from\_config**(*cfg, optimizer, data\_size*)

**get\_last\_lr()**

Return last computed learning rate by current scheduler.

**get\_lr()****load\_state\_dict**(*state\_dict*)

Loads the schedulers state.

When saving or loading the scheduler, please make sure to also save or load the state of the optimizer.

**Parameters**

**state\_dict** (*dict*) – scheduler state. Should be an object returned from a call to [state\\_dict\(\)](#).

**lr\_lambda**(*step*)**print\_lr**(*is\_verbose, group, lr, epoch=None*)

Display the current learning rate.

**state\_dict**()

Returns the state of the scheduler as a dict.

It contains an entry for every variable in self.\_\_dict\_\_ which is not the optimizer. The learning rate lambda functions will only be saved if they are callable objects and not if they are functions or lambdas.

When saving or loading the scheduler, please make sure to also save or load the state of the optimizer.

**step**(*epoch=None*)

**class** `xmodaler.lr_scheduler.WarmupCosineWithHardRestarts`(*\*, optimizer, warmup\_steps, t\_total, cycles=1.0, last\_epoch=-1*)

Bases: `LambdaLR`

Linear warmup and then cosine cycles with hard restarts. Linearly increases learning rate from 0 to 1 over *warmup\_steps* training steps. If *cycles* (default=1.) is different from default, learning rate follows *cycles* times a cosine decaying learning rate (with hard restarts).

**\_\_init\_\_**(*\*, optimizer, warmup\_steps, t\_total, cycles=1.0, last\_epoch=-1*)

**\_initial\_step()**

Initialize step counts and performs a step

**classmethod** **from\_config**(*cfg, optimizer, data\_size*)

**get\_last\_lr()**

Return last computed learning rate by current scheduler.

**get\_lr()****load\_state\_dict**(*state\_dict*)

Loads the schedulers state.

When saving or loading the scheduler, please make sure to also save or load the state of the optimizer.

**Parameters**

**state\_dict** (*dict*) – scheduler state. Should be an object returned from a call to [state\\_dict\(\)](#).

**lr\_lambda**(step)

**print\_lr**(is\_verbose, group, lr, epoch=None)

Display the current learning rate.

**state\_dict**()

Returns the state of the scheduler as a dict.

It contains an entry for every variable in self.\_\_dict\_\_ which is not the optimizer. The learning rate lambda functions will only be saved if they are callable objects and not if they are functions or lambdas.

When saving or loading the scheduler, please make sure to also save or load the state of the optimizer.

**step**(epoch=None)

```
class xmodaler.lr_scheduler.WarmupMultiStepLR(*, optimizer, milestones, gamma=0.1,
                                              warmup_factor=0.3333333333333333,
                                              warmup_iters=500, warmup_method='linear',
                                              last_epoch=-1)
```

Bases: `_LRScheduler`

```
__init__(*, optimizer, milestones, gamma=0.1, warmup_factor=0.3333333333333333, warmup_iters=500,
        warmup_method='linear', last_epoch=-1)
```

**\_initial\_step**()

Initialize step counts and performs a step

**classmethod from\_config**(cfg, optimizer, data\_size)

**get\_last\_lr**()

Return last computed learning rate by current scheduler.

**get\_lr**()

**load\_state\_dict**(state\_dict)

Loads the schedulers state.

#### Parameters

**state\_dict** (dict) – scheduler state. Should be an object returned from a call to `state_dict()`.

**print\_lr**(is\_verbose, group, lr, epoch=None)

Display the current learning rate.

**state\_dict**()

Returns the state of the scheduler as a dict.

It contains an entry for every variable in self.\_\_dict\_\_ which is not the optimizer.

**step**(epoch=None)

```
class xmodaler.lr_scheduler.MultiStepLR(*, optimizer, milestones, gamma=0.1, last_epoch=-1)
```

Bases: `MultiStepLR`

```
__init__(*, optimizer, milestones, gamma=0.1, last_epoch=-1)
```

**\_get\_closed\_form\_lr**()

**\_initial\_step**()

Initialize step counts and performs a step

**classmethod** `from_config(cfg, optimizer, data_size)`

**get\_last\_lr()**

Return last computed learning rate by current scheduler.

**get\_lr()**

**load\_state\_dict**(*state\_dict*)

Loads the schedulers state.

**Parameters**

**state\_dict** (*dict*) – scheduler state. Should be an object returned from a call to `state_dict()`.

**print\_lr**(*is\_verbose, group, lr, epoch=None*)

Display the current learning rate.

**state\_dict()**

Returns the state of the scheduler as a `dict`.

It contains an entry for every variable in `self.__dict__` which is not the optimizer.

**step**(*epoch=None*)

**class** `xmodaler.lr_scheduler.FixLR(*, optimizer, last_epoch=-1)`

Bases: `LambdaLR`

Fix LR

**\_\_init\_\_**(*\*, optimizer, last\_epoch=-1*)

**\_initial\_step()**

Initialize step counts and performs a step

**classmethod** `from_config(cfg, optimizer, data_size)`

**get\_last\_lr()**

Return last computed learning rate by current scheduler.

**get\_lr()**

**load\_state\_dict**(*state\_dict*)

Loads the schedulers state.

When saving or loading the scheduler, please make sure to also save or load the state of the optimizer.

**Parameters**

**state\_dict** (*dict*) – scheduler state. Should be an object returned from a call to `state_dict()`.

**lr\_lambda**(*step*)

**print\_lr**(*is\_verbose, group, lr, epoch=None*)

Display the current learning rate.

**state\_dict()**

Returns the state of the scheduler as a `dict`.

It contains an entry for every variable in `self.__dict__` which is not the optimizer. The learning rate lambda functions will only be saved if they are callable objects and not if they are functions or lambdas.

When saving or loading the scheduler, please make sure to also save or load the state of the optimizer.

```
step(epoch=None)
```

## 3.9 xmodaler.modeling

### 3.9.1 xmodaler.modeling.decode\_strategy

```
xmodaler.modeling.decode_strategy.build_beam_searcher(cfg)
```

```
xmodaler.modeling.decode_strategy.build_greedy_decoder(cfg)
```

```
class xmodaler.modeling.decode_strategy.GreedyDecoder(*, vocab_path, beam_size, max_seq_len,
                                                       bert_tokenizer, bos_token_id, eos_token_id)
```

Bases: [DecodeStrategy](#)

```
_abc_impl = <_abc_data object>
_backward_hooks: Dict[int, Callable]
_buffers: Dict[str, Optional[Tensor]]
_forward(batched_inputs, model)
_forward_hooks: Dict[int, Callable]
_forward_pre_hooks: Dict[int, Callable]
_is_full_backward_hook: Optional[bool]
_load_state_dict_post_hooks: Dict[int, Callable]
_load_state_dict_pre_hooks: Dict[int, Callable]
_modules: Dict[str, Optional[Module]]
_non_persistent_buffers_set: Set[str]
_parameters: Dict[str, Optional[Parameter]]
_state_dict_hooks: Dict[int, Callable]
training: bool
```

```
class xmodaler.modeling.decode_strategy.BeamSearcher(*, vocab_path, beam_size, max_seq_len,
                                                       bert_tokenizer, bos_token_id, eos_token_id)
```

Bases: [DecodeStrategy](#)

```
_abc_impl = <_abc_data object>
_backward_hooks: Dict[int, Callable]
_buffers: Dict[str, Optional[Tensor]]
_expand_state(states, selected_beam, batch_size, beam_size, cur_beam_size)
_forward(batched_inputs, model)
_forward_hooks: Dict[int, Callable]
```

```

    _forward_pre_hooks: Dict[int, Callable]
    _is_full_backward_hook: Optional[bool]
    _load_state_dict_post_hooks: Dict[int, Callable]
    _load_state_dict_pre_hooks: Dict[int, Callable]
    _modules: Dict[str, Optional[Module]]
    _non_persistent_buffers_set: Set[str]
    _parameters: Dict[str, Optional[Parameter]]
    _select(batch_size, beam_size, t, candidate_logprob)
    _state_dict_hooks: Dict[int, Callable]

    training: bool

class xmodaler.modeling.decode_strategy.decode_strategy.DecodeStrategy(*, vocab_path,
                                                                    beam_size,
                                                                    max_seq_len,
                                                                    bert_tokenizer,
                                                                    bos_token_id,
                                                                    eos_token_id)

Bases: Module

__init__(*, vocab_path, beam_size, max_seq_len, bert_tokenizer, bos_token_id, eos_token_id)
    Initializes internal Module state, shared by both nn.Module and ScriptModule.

_abc_impl = <_abc_data object>

_backward_hooks: Dict[int, Callable]
_buffers: Dict[str, Optional[Tensor]]
abstract _forward(batched_inputs, model)
_forward_hooks: Dict[int, Callable]
_forward_pre_hooks: Dict[int, Callable]
_is_full_backward_hook: Optional[bool]
_load_state_dict_post_hooks: Dict[int, Callable]
_load_state_dict_pre_hooks: Dict[int, Callable]
_modules: Dict[str, Optional[Module]]
_non_persistent_buffers_set: Set[str]
_parameters: Dict[str, Optional[Parameter]]
_state_dict_hooks: Dict[int, Callable]

```

**forward**(*batched\_inputs, output\_sents, model*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod** `from_config(cfg)`

**training:** `bool`

### 3.9.2 xmodaler.modeling.decoder

`xmodaler.modeling.decoder.build_decoder(cfg)`

`xmodaler.modeling.decoder.add_decoder_config(cfg, tmp_cfg)`

**class** `xmodaler.modeling.decoder.UpDownDecoder`(\**hidden\_size: int, token\_embed\_dim: int,*  
*visual\_embed\_dim: int, att\_embed\_size: int, dropout1:*  
*float, dropout2: float, att\_embed\_dropout: float*)

Bases: `Decoder`

**\_\_init\_\_**(\**hidden\_size: int, token\_embed\_dim: int, visual\_embed\_dim: int, att\_embed\_size: int, dropout1:*  
*float, dropout2: float, att\_embed\_dropout: float*)

Initializes internal Module state, shared by both `nn.Module` and `ScriptModule`.

**\_abc\_impl** = `<_abc_data object>`

**\_backward\_hooks**: `Dict[int, Callable]`

**\_buffers**: `Dict[str, Optional[Tensor]]`

**\_forward\_hooks**: `Dict[int, Callable]`

**\_forward\_pre\_hooks**: `Dict[int, Callable]`

**\_is\_full\_backward\_hook**: `Optional[bool]`

**\_load\_state\_dict\_post\_hooks**: `Dict[int, Callable]`

**\_load\_state\_dict\_pre\_hooks**: `Dict[int, Callable]`

**\_modules**: `Dict[str, Optional[Module]]`

**\_non\_persistent\_buffers\_set**: `Set[str]`

**\_parameters**: `Dict[str, Optional[Parameter]]`

**\_state\_dict\_hooks**: `Dict[int, Callable]`

**classmethod** `add_config(cfg)`

**forward**(*batched\_inputs*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod** **from\_config**(*cfg*)

**preprocess**(*batched\_inputs*)

**training:** `bool`

**class** `xmodaler.modeling.decoder.SALSTMDecoder`(\**hidden\_size: int, token\_embed\_dim: int, visual\_embed\_dim: int, att\_embed\_size: int, att\_embed\_dropout: float*)

Bases: `Decoder`

**\_\_init\_\_**(\**hidden\_size: int, token\_embed\_dim: int, visual\_embed\_dim: int, att\_embed\_size: int, att\_embed\_dropout: float*)

Initializes internal Module state, shared by both `nn.Module` and `ScriptModule`.

**\_abc\_impl** = `<_abc_data object>`

**\_backward\_hooks**: `Dict[int, Callable]`

**\_buffers**: `Dict[str, Optional[Tensor]]`

**\_forward\_hooks**: `Dict[int, Callable]`

**\_forward\_pre\_hooks**: `Dict[int, Callable]`

**\_is\_full\_backward\_hook**: `Optional[bool]`

**\_load\_state\_dict\_post\_hooks**: `Dict[int, Callable]`

**\_load\_state\_dict\_pre\_hooks**: `Dict[int, Callable]`

**\_modules**: `Dict[str, Optional[Module]]`

**\_non\_persistent\_buffers\_set**: `Set[str]`

**\_parameters**: `Dict[str, Optional[Parameter]]`

**\_state\_dict\_hooks**: `Dict[int, Callable]`

**classmethod** **add\_config**(*cfg*)

**forward**(*batched\_inputs*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```
classmethod from_config(cfg)

preprocess(batched_inputs)

training: bool

class xmodaler.modeling.decoder.MPLSTMDecoder(*, hidden_size: int, token_embed_dim: int)
    Bases: Decoder
    __init__(*, hidden_size: int, token_embed_dim: int)
        Initializes internal Module state, shared by both nn.Module and ScriptModule.
    _abc_impl = <_abc_data object>
    _backward_hooks: Dict[int, Callable]
    _buffers: Dict[str, Optional[Tensor]]
    _forward_hooks: Dict[int, Callable]
    _forward_pre_hooks: Dict[int, Callable]
    _is_full_backward_hook: Optional[bool]
    _load_state_dict_post_hooks: Dict[int, Callable]
    _load_state_dict_pre_hooks: Dict[int, Callable]
    _modules: Dict[str, Optional[Module]]
    _non_persistent_buffers_set: Set[str]
    _parameters: Dict[str, Optional[Parameter]]
    _state_dict_hooks: Dict[int, Callable]
    classmethod add_config(cfg)
    forward(batched_inputs)
        Defines the computation performed at every call.
        Should be overridden by all subclasses.



---


    Note: Although the recipe for forward pass needs to be defined within this function, one should call the
    Module instance afterwards instead of this since the former takes care of running the registered hooks while
    the latter silently ignores them.



---



classmethod from_config(cfg)

preprocess(batched_inputs)

training: bool

class xmodaler.modeling.decoder.TransformerDecoder(*, num_generation_layers: int,
                                                    bert_generation_layers)
    Bases: Decoder
```



```
__init__(*, num_generation_layers: int, bert_generation_layers)
    Initializes internal Module state, shared by both nn.Module and ScriptModule.
```

```
_abc_impl = <_abc_data object>
```

```
_backward_hooks: Dict[int, Callable]
```

```
_buffers: Dict[str, Optional[Tensor]]
```

```
_forward_hooks: Dict[int, Callable]
```

```
_forward_pre_hooks: Dict[int, Callable]
```

```
_is_full_backward_hook: Optional[bool]
```

```
_load_state_dict_post_hooks: Dict[int, Callable]
```

```
_load_state_dict_pre_hooks: Dict[int, Callable]
```

```
_modules: Dict[str, Optional[Module]]
```

```
_non_persistent_buffers_set: Set[str]
```

```
_parameters: Dict[str, Optional[Parameter]]
```

```
_state_dict_hooks: Dict[int, Callable]
```

```
classmethod add_config(cfg)
```

```
forward(batched_inputs)
```

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```
classmethod from_config(cfg)
```

```
training: bool
```

```
class xmodaler.modeling.decoder.DecoupleBertDecoder(*, num_understanding_layers: int,
                                                    num_generation_layers: int, u_layer_drop: float,
                                                    g_layer_drop: float, bert_understanding_layers,
                                                    bert_generation_layers)
```

Bases: Decoder

```
__init__(*, num_understanding_layers: int, num_generation_layers: int, u_layer_drop: float, g_layer_drop:
float, bert_understanding_layers, bert_generation_layers)
```

Initializes internal Module state, shared by both nn.Module and ScriptModule.

```
_abc_impl = <_abc_data object>
```

```
_backward_hooks: Dict[int, Callable]
```

```
_buffers: Dict[str, Optional[Tensor]]
```

```
_forward_hooks: Dict[int, Callable]
_forward_pre_hooks: Dict[int, Callable]
_is_full_backward_hook: Optional[bool]
_load_state_dict_post_hooks: Dict[int, Callable]
_load_state_dict_pre_hooks: Dict[int, Callable]
_modules: Dict[str, Optional[Module]]
_non_persistent_buffers_set: Set[str]
_parameters: Dict[str, Optional[Parameter]]
_state_dict_hooks: Dict[int, Callable]
```

```
classmethod add_config(cfg)
```

```
forward(batched_inputs)
```

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```
classmethod from_config(cfg)
```

```
training: bool
```

```
class xmodaler.modeling.decoder.MeshedDecoder(*, d_model: int, num_layer: int, num_att_head: int,
                                              d_ff: int, dropout: float, padding_idx: int,
                                              enc_layer_num: int)
```

Bases: `Decoder`

```
__init__(*, d_model: int, num_layer: int, num_att_head: int, d_ff: int, dropout: float, padding_idx: int,
         enc_layer_num: int)
```

Initializes internal Module state, shared by both `nn.Module` and `ScriptModule`.

```
_abc_impl = <_abc_data object>
```

```
_backward_hooks: Dict[int, Callable]
_buffers: Dict[str, Optional[Tensor]]
_forward_hooks: Dict[int, Callable]
_forward_pre_hooks: Dict[int, Callable]
_is_full_backward_hook: Optional[bool]
_load_state_dict_post_hooks: Dict[int, Callable]
_load_state_dict_pre_hooks: Dict[int, Callable]
```

```

_modules: Dict[str, Optional[Module]]
_non_persistent_buffers_set: Set[str]
_parameters: Dict[str, Optional[Parameter]]
_state_dict_hooks: Dict[int, Callable]
classmethod add_config(cfg)

```

```
forward(batched_inputs)
```

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```
classmethod from_config(cfg)
```

```
training: bool
```

```

class xmodaler.modeling.decoder.XLANDecoder(*, hidden_size: int, ctx_drop: float, bilinear_dim: int,
                                             att_heads: int, att_mid_dim: int, att_mid_drop: float,
                                             att_embed_dropout: float, layer_num: int, act_type: str,
                                             elu_alpha: float)

```

Bases: Decoder

```

__init__(*, hidden_size: int, ctx_drop: float, bilinear_dim: int, att_heads: int, att_mid_dim: int,
          att_mid_drop: float, att_embed_dropout: float, layer_num: int, act_type: str, elu_alpha: float)

```

Initializes internal Module state, shared by both nn.Module and ScriptModule.

```

_abc_impl = <_abc_data object>
_backward_hooks: Dict[int, Callable]
_buffers: Dict[str, Optional[Tensor]]
_forward_hooks: Dict[int, Callable]
_forward_pre_hooks: Dict[int, Callable]
_is_full_backward_hook: Optional[bool]
_load_state_dict_post_hooks: Dict[int, Callable]
_load_state_dict_pre_hooks: Dict[int, Callable]
_modules: Dict[str, Optional[Module]]
_non_persistent_buffers_set: Set[str]
_parameters: Dict[str, Optional[Parameter]]
_state_dict_hooks: Dict[int, Callable]
classmethod add_config(cfg)

```

**forward**(*batched\_inputs*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod** **from\_config**(*cfg*)

**preprocess**(*batched\_inputs*)

**training:** `bool`

**class** `xmodaler.modeling.decoder.TDConvEDDecoder`(\**num\_hidden\_layers: int, hidden\_size: int, kernel\_sizes: list, conv\_dropout: float, att\_embed\_size: int, att\_embed\_dropout: float, use\_norm: bool*)

Bases: `Module`

**\_\_init\_\_**(\**num\_hidden\_layers: int, hidden\_size: int, kernel\_sizes: list, conv\_dropout: float, att\_embed\_size: int, att\_embed\_dropout: float, use\_norm: bool*)

Initializes internal Module state, shared by both `nn.Module` and `ScriptModule`.

**\_backward\_hooks:** `Dict[int, Callable]`

**\_buffers:** `Dict[str, Optional[Tensor]]`

**\_clear\_decoding\_buffer**()

**\_forward\_hooks:** `Dict[int, Callable]`

**\_forward\_pre\_hooks:** `Dict[int, Callable]`

**\_init\_decoding\_buffer**(*batch\_size*)

**\_is\_full\_backward\_hook:** `Optional[bool]`

**\_load\_state\_dict\_post\_hooks:** `Dict[int, Callable]`

**\_load\_state\_dict\_pre\_hooks:** `Dict[int, Callable]`

**\_modules:** `Dict[str, Optional[Module]]`

**\_non\_persistent\_buffers\_set:** `Set[str]`

**\_parameters:** `Dict[str, Optional[Parameter]]`

**\_state\_dict\_hooks:** `Dict[int, Callable]`

**classmethod** **add\_config**(*cfg*)

**forward**(*batched\_inputs*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod** **from\_config**(*cfg*)

**preprocess**(*batched\_inputs*)

**training:** `bool`

**class** `xmodaler.modeling.decoder.AttributeDecoder`(\**hidden\_size: int, token\_embed\_dim: int, visual\_feat\_dim: int, attribute\_dim: int, dropout: float*)

Bases: `Decoder`

**\_\_init\_\_**(\**hidden\_size: int, token\_embed\_dim: int, visual\_feat\_dim: int, attribute\_dim: int, dropout: float*)

Initializes internal Module state, shared by both `nn.Module` and `ScriptModule`.

**\_abc\_impl** = `<_abc_data object>`

**\_backward\_hooks:** `Dict[int, Callable]`

**\_buffers:** `Dict[str, Optional[Tensor]]`

**\_forward\_hooks:** `Dict[int, Callable]`

**\_forward\_pre\_hooks:** `Dict[int, Callable]`

**\_is\_full\_backward\_hook:** `Optional[bool]`

**\_load\_state\_dict\_post\_hooks:** `Dict[int, Callable]`

**\_load\_state\_dict\_pre\_hooks:** `Dict[int, Callable]`

**\_modules:** `Dict[str, Optional[Module]]`

**\_non\_persistent\_buffers\_set:** `Set[str]`

**\_parameters:** `Dict[str, Optional[Parameter]]`

**\_state\_dict\_hooks:** `Dict[int, Callable]`

**classmethod** **add\_config**(*cfg*)

**forward**(*batched\_inputs*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```
classmethod from_config(cfg)

preprocess(batched_inputs)

training: bool
```

### 3.9.3 xmodaler.modeling.embedding

```
xmodaler.modeling.embedding.build_embeddings(cfg, name)
```

```
class xmodaler.modeling.embedding.TokenBaseEmbedding(*, dim: int, vocab_size: int, **kwargs)
```

Bases: Module

```
__init__(*, dim: int, vocab_size: int, **kwargs)
```

Initializes internal Module state, shared by both nn.Module and ScriptModule.

```
_backward_hooks: Dict[int, Callable]
```

```
_buffers: Dict[str, Optional[Tensor]]
```

```
_forward(input_ids, token_type_ids=None, time_step=None)
```

```
_forward_hooks: Dict[int, Callable]
```

```
_forward_pre_hooks: Dict[int, Callable]
```

```
_is_full_backward_hook: Optional[bool]
```

```
_load_state_dict_post_hooks: Dict[int, Callable]
```

```
_load_state_dict_pre_hooks: Dict[int, Callable]
```

```
_modules: Dict[str, Optional[Module]]
```

```
_non_persistent_buffers_set: Set[str]
```

```
_parameters: Dict[str, Optional[Parameter]]
```

```
_state_dict_hooks: Dict[int, Callable]
```

```
forward(batched_inputs)
```

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```
classmethod from_config(cfg)
```

```
training: bool
```

```
class xmodaler.modeling.embedding.VisualBaseEmbedding(*, in_dim: int, out_dim: int, **kwargs)
```

Bases: Module

---

```
__init__(*, in_dim: int, out_dim: int, **kwargs)
```

Initializes internal Module state, shared by both nn.Module and ScriptModule.

```
_backward_hooks: Dict[int, Callable]
```

```
_buffers: Dict[str, Optional[Tensor]]
```

```
_forward_hooks: Dict[int, Callable]
```

```
_forward_pre_hooks: Dict[int, Callable]
```

```
_is_full_backward_hook: Optional[bool]
```

```
_load_state_dict_post_hooks: Dict[int, Callable]
```

```
_load_state_dict_pre_hooks: Dict[int, Callable]
```

```
_modules: Dict[str, Optional[Module]]
```

```
_non_persistent_buffers_set: Set[str]
```

```
_parameters: Dict[str, Optional[Parameter]]
```

```
_state_dict_hooks: Dict[int, Callable]
```

```
forward(batched_inputs)
```

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```
classmethod from_config(cfg)
```

```
training: bool
```

```
class xmodaler.modeling.embedding.VisualIdentityEmbedding
```

Bases: Module

```
__init__()
```

Initializes internal Module state, shared by both nn.Module and ScriptModule.

```
_backward_hooks: Dict[int, Callable]
```

```
_buffers: Dict[str, Optional[Tensor]]
```

```
_forward_hooks: Dict[int, Callable]
```

```
_forward_pre_hooks: Dict[int, Callable]
```

```
_is_full_backward_hook: Optional[bool]
```

```
_load_state_dict_post_hooks: Dict[int, Callable]
```

```
_load_state_dict_pre_hooks: Dict[int, Callable]
```

```
_modules: Dict[str, Optional[Module]]
_non_persistent_buffers_set: Set[str]
_parameters: Dict[str, Optional[Parameter]]
_state_dict_hooks: Dict[int, Callable]
```

**forward**(*batched\_inputs*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod** **from\_config**(*cfg*)

**training:** `bool`

```
class xmodaler.modeling.embedding.TDConvEDVisualBaseEmbedding(*, in_dim: int, out_dim: int,
                                                             **kwargs)
```

Bases: `Module`

**\_\_init\_\_**(*\*, in\_dim: int, out\_dim: int, \*\*kwargs*)

Initializes internal `Module` state, shared by both `nn.Module` and `ScriptModule`.

```
_backward_hooks: Dict[int, Callable]
_buffers: Dict[str, Optional[Tensor]]
_forward_hooks: Dict[int, Callable]
_forward_pre_hooks: Dict[int, Callable]
_is_full_backward_hook: Optional[bool]
_load_state_dict_post_hooks: Dict[int, Callable]
_load_state_dict_pre_hooks: Dict[int, Callable]
_modules: Dict[str, Optional[Module]]
_non_persistent_buffers_set: Set[str]
_parameters: Dict[str, Optional[Parameter]]
_state_dict_hooks: Dict[int, Callable]
```

**forward**(*batched\_inputs*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---



```
classmethod from_config(cfg)
```

```
training: bool
```

```
class xmodaler.modeling.embedding.position_embedding.SinusoidEncoding(dim, max_len)
```

```
Bases: Module
```

```
__init__(dim, max_len)
```

Initializes internal Module state, shared by both nn.Module and ScriptModule.

```
_backward_hooks: Dict[int, Callable]
```

```
_buffers: Dict[str, Optional[Tensor]]
```

```
_forward_hooks: Dict[int, Callable]
```

```
_forward_pre_hooks: Dict[int, Callable]
```

```
_is_full_backward_hook: Optional[bool]
```

```
_load_state_dict_post_hooks: Dict[int, Callable]
```

```
_load_state_dict_pre_hooks: Dict[int, Callable]
```

```
_modules: Dict[str, Optional[Module]]
```

```
_non_persistent_buffers_set: Set[str]
```

```
_parameters: Dict[str, Optional[Parameter]]
```

```
_state_dict_hooks: Dict[int, Callable]
```

```
forward(x)
```

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```
training: bool
```

```
class xmodaler.modeling.embedding.position_embedding.NNEmbeddingEncoding(dim, max_len)
```

```
Bases: Module
```

```
__init__(dim, max_len)
```

Initializes internal Module state, shared by both nn.Module and ScriptModule.

```
_backward_hooks: Dict[int, Callable]
```

```
_buffers: Dict[str, Optional[Tensor]]
```

```
_forward_hooks: Dict[int, Callable]
```

```
_forward_pre_hooks: Dict[int, Callable]
```

```
_is_full_backward_hook: Optional[bool]
```

```
_load_state_dict_post_hooks: Dict[int, Callable]
_load_state_dict_pre_hooks: Dict[int, Callable]
_modules: Dict[str, Optional[Module]]
_non_persistent_buffers_set: Set[str]
_parameters: Dict[str, Optional[Parameter]]
_state_dict_hooks: Dict[int, Callable]

forward(x)
```

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```
training: bool
```

### 3.9.4 xmodaler.modeling.encoder

```
xmodaler.modeling.encoder.build_encoder(cfg)
```

```
xmodaler.modeling.encoder.add_encoder_config(cfg, tmp_cfg)
```

```
class xmodaler.modeling.encoder.Encoder
```

```
    Bases: Module
```

```
    __init__()
```

Initializes internal Module state, shared by both `nn.Module` and `ScriptModule`.

```
_backward_hooks: Dict[int, Callable]
_buffers: Dict[str, Optional[Tensor]]
_forward_hooks: Dict[int, Callable]
_forward_pre_hooks: Dict[int, Callable]
_is_full_backward_hook: Optional[bool]
_load_state_dict_post_hooks: Dict[int, Callable]
_load_state_dict_pre_hooks: Dict[int, Callable]
_modules: Dict[str, Optional[Module]]
_non_persistent_buffers_set: Set[str]
_parameters: Dict[str, Optional[Parameter]]
_state_dict_hooks: Dict[int, Callable]
```

---

```
classmethod add_config(cfg)
```

```
forward(batched_inputs, mode=None)
```

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```
classmethod from_config(cfg)
```

```
training: bool
```

```
class xmodaler.modeling.encoder.UpDownEncoder
```

Bases: `Module`

```
__init__()
```

Initializes internal Module state, shared by both `nn.Module` and `ScriptModule`.

```
_backward_hooks: Dict[int, Callable]
```

```
_buffers: Dict[str, Optional[Tensor]]
```

```
_forward_hooks: Dict[int, Callable]
```

```
_forward_pre_hooks: Dict[int, Callable]
```

```
_is_full_backward_hook: Optional[bool]
```

```
_load_state_dict_post_hooks: Dict[int, Callable]
```

```
_load_state_dict_pre_hooks: Dict[int, Callable]
```

```
_modules: Dict[str, Optional[Module]]
```

```
_non_persistent_buffers_set: Set[str]
```

```
_parameters: Dict[str, Optional[Parameter]]
```

```
_state_dict_hooks: Dict[int, Callable]
```

```
classmethod add_config(cfg)
```

```
forward(batched_inputs, mode=None)
```

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```
classmethod from_config(cfg)
```

**training:** bool

**class** xmodaler.modeling.encoder.**TransformerEncoder**(\**, num\_hidden\_layers: int, bert\_layers*)

Bases: Module

**\_\_init\_\_**(\**, num\_hidden\_layers: int, bert\_layers*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

**\_backward\_hooks:** Dict[int, Callable]

**\_buffers:** Dict[str, Optional[Tensor]]

**\_forward\_hooks:** Dict[int, Callable]

**\_forward\_pre\_hooks:** Dict[int, Callable]

**\_is\_full\_backward\_hook:** Optional[bool]

**\_load\_state\_dict\_post\_hooks:** Dict[int, Callable]

**\_load\_state\_dict\_pre\_hooks:** Dict[int, Callable]

**\_modules:** Dict[str, Optional[Module]]

**\_non\_persistent\_buffers\_set:** Set[str]

**\_parameters:** Dict[str, Optional[Parameter]]

**\_state\_dict\_hooks:** Dict[int, Callable]

**classmethod** add\_config(*cfg*)

**forward**(*batched\_inputs, mode=None*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod** from\_config(*cfg*)

**training:** bool

**class** xmodaler.modeling.encoder.**SingleStreamBertEncoder**(\**, num\_hidden\_layers: int, bert\_layers*)

Bases: Module

**\_\_init\_\_**(\**, num\_hidden\_layers: int, bert\_layers*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

**\_backward\_hooks:** Dict[int, Callable]

**\_buffers:** Dict[str, Optional[Tensor]]

**\_forward\_hooks:** Dict[int, Callable]

**\_forward\_pre\_hooks:** Dict[int, Callable]

```

_is_full_backward_hook: Optional[bool]
_load_state_dict_post_hooks: Dict[int, Callable]
_load_state_dict_pre_hooks: Dict[int, Callable]
_modules: Dict[str, Optional[Module]]
_non_persistent_buffers_set: Set[str]
_parameters: Dict[str, Optional[Parameter]]
_state_dict_hooks: Dict[int, Callable]
classmethod add_config(cfg)

```

**forward**(*batched\_inputs*, *mode=None*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```

classmethod from_config(cfg)

```

```

training: bool

```

```

class xmodaler.modeling.encoder.TwoStreamBertEncoder(*, num_hidden_layers: int,
                                                    v_num_hidden_layers: int, layer_drop: float,
                                                    v_layer_drop: float, bert_layers,
                                                    v_bert_layers)

```

Bases: `Module`

```

__init__(*, num_hidden_layers: int, v_num_hidden_layers: int, layer_drop: float, v_layer_drop: float,
         bert_layers, v_bert_layers)

```

Initializes internal Module state, shared by both `nn.Module` and `ScriptModule`.

```

_backward_hooks: Dict[int, Callable]
_buffers: Dict[str, Optional[Tensor]]
_forward_hooks: Dict[int, Callable]
_forward_pre_hooks: Dict[int, Callable]
_is_full_backward_hook: Optional[bool]
_load_state_dict_post_hooks: Dict[int, Callable]
_load_state_dict_pre_hooks: Dict[int, Callable]
_modules: Dict[str, Optional[Module]]
_non_persistent_buffers_set: Set[str]
_parameters: Dict[str, Optional[Parameter]]

```

**\_state\_dict\_hooks:** Dict[int, Callable]

**classmethod add\_config**(*cfg*)

**forward**(*batched\_inputs*, *mode=None*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod from\_config**(*cfg*)

**training:** bool

**class** xmodaler.modeling.encoder.GCNEncoder(\*, *in\_dim: int, out\_dim: int, relation\_num: int, dropout: float*)

Bases: Module

**\_\_init\_\_**(\*, *in\_dim: int, out\_dim: int, relation\_num: int, dropout: float*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

**\_backward\_hooks:** Dict[int, Callable]

**\_buffers:** Dict[str, Optional[Tensor]]

**\_forward\_hooks:** Dict[int, Callable]

**\_forward\_pre\_hooks:** Dict[int, Callable]

**\_is\_full\_backward\_hook:** Optional[bool]

**\_load\_state\_dict\_post\_hooks:** Dict[int, Callable]

**\_load\_state\_dict\_pre\_hooks:** Dict[int, Callable]

**\_modules:** Dict[str, Optional[Module]]

**\_non\_persistent\_buffers\_set:** Set[str]

**\_parameters:** Dict[str, Optional[Parameter]]

**\_state\_dict\_hooks:** Dict[int, Callable]

**classmethod add\_config**(*cfg*)

**forward**(*batched\_inputs*, *mode=None*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```
classmethod from_config(cfg)
```

```
training: bool
```

```
class xmodaler.modeling.encoder.MemoryAugmentedEncoder(*, input_dim: int, d_model: int, num_layer:
                                                    int, num_att_head: int, num_att_memory:
                                                    int, d_ff: int, dropout: float, padding_idx:
                                                    int)
```

Bases: Module

```
__init__(*, input_dim: int, d_model: int, num_layer: int, num_att_head: int, num_att_memory: int, d_ff:
        int, dropout: float, padding_idx: int)
```

Initializes internal Module state, shared by both nn.Module and ScriptModule.

```
_backward_hooks: Dict[int, Callable]
```

```
_buffers: Dict[str, Optional[Tensor]]
```

```
_forward_hooks: Dict[int, Callable]
```

```
_forward_pre_hooks: Dict[int, Callable]
```

```
_get_global_feat(feats, masks)
```

```
_is_full_backward_hook: Optional[bool]
```

```
_load_state_dict_post_hooks: Dict[int, Callable]
```

```
_load_state_dict_pre_hooks: Dict[int, Callable]
```

```
_modules: Dict[str, Optional[Module]]
```

```
_non_persistent_buffers_set: Set[str]
```

```
_parameters: Dict[str, Optional[Parameter]]
```

```
_state_dict_hooks: Dict[int, Callable]
```

```
classmethod add_config(cfg)
```

```
forward(batched_inputs, mode=None)
```

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```
classmethod from_config(cfg)
```

```
training: bool
```

```
class xmodaler.modeling.encoder.LowRankBilinearEncoder(*, embed_dim: int, att_heads: int,
                                                    att_mid_dim: int, att_mid_drop: float,
                                                    dropout: float, bifeat_emb_dropout: float,
                                                    layer_num: int, emb_act_type: str, act_type:
                                                    str, elu_alpha: float)
```

Bases: Module

```
__init__(*, embed_dim: int, att_heads: int, att_mid_dim: int, att_mid_drop: float, dropout: float,  
         bifeat_emb_dropout: float, layer_num: int, emb_act_type: str, act_type: str, elu_alpha: float)
```

Initializes internal Module state, shared by both nn.Module and ScriptModule.

```
_backward_hooks: Dict[int, Callable]
```

```
_buffers: Dict[str, Optional[Tensor]]
```

```
_forward_hooks: Dict[int, Callable]
```

```
_forward_pre_hooks: Dict[int, Callable]
```

```
_is_full_backward_hook: Optional[bool]
```

```
_load_state_dict_post_hooks: Dict[int, Callable]
```

```
_load_state_dict_pre_hooks: Dict[int, Callable]
```

```
_modules: Dict[str, Optional[Module]]
```

```
_non_persistent_buffers_set: Set[str]
```

```
_parameters: Dict[str, Optional[Parameter]]
```

```
_state_dict_hooks: Dict[int, Callable]
```

```
classmethod add_config(cfg)
```

```
forward(batched_inputs, mode=None)
```

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```
classmethod from_config(cfg)
```

```
training: bool
```

```
class xmodaler.modeling.encoder.TDConvEDEncoder(*, num_hidden_layers: int, hidden_size: int,  
        kernel_sizes: list, padding_mode: str, offset_act: str,  
        min_idx: int, max_idx: int, clamp_idx: bool, dropout:  
        float, use_norm: bool)
```

Bases: Module

```
__init__(*, num_hidden_layers: int, hidden_size: int, kernel_sizes: list, padding_mode: str, offset_act: str,  
         min_idx: int, max_idx: int, clamp_idx: bool, dropout: float, use_norm: bool)
```

Initializes internal Module state, shared by both nn.Module and ScriptModule.

```
_backward_hooks: Dict[int, Callable]
```

```
_buffers: Dict[str, Optional[Tensor]]
```



```

_forward_hooks: Dict[int, Callable]
_forward_pre_hooks: Dict[int, Callable]
_is_full_backward_hook: Optional[bool]
_load_state_dict_post_hooks: Dict[int, Callable]
_load_state_dict_pre_hooks: Dict[int, Callable]
_modules: Dict[str, Optional[Module]]
_non_persistent_buffers_set: Set[str]
_parameters: Dict[str, Optional[Parameter]]
_state_dict_hooks: Dict[int, Callable]

classmethod add_config(cfg)

```

**forward**(*batched\_inputs*, *mode=None*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```

classmethod from_config(cfg)

```

```

training: bool

```

### 3.9.5 xmodaler.modeling.layers

```

xmodaler.modeling.layers.create_act.get_act_layer(name='none')

```

```

xmodaler.modeling.layers.create_act.get_activation(activation_string)

```

```

class xmodaler.modeling.layers.attention_pooler.AttentionPooler(*, hidden_size: int, output_size:
                                                                int, dropout: float, use_bn: bool)

```

Bases: `Module`

```

__init__(*, hidden_size: int, output_size: int, dropout: float, use_bn: bool)

```

Initializes internal Module state, shared by both `nn.Module` and `ScriptModule`.

**forward**(*hidden\_states*, *masks=None*, *\*\*kwargs*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**training:** bool

**class** xmodaler.modeling.layers.base\_attention.**BaseAttention**(\**hidden\_size: int, att\_embed\_size: int, att\_embed\_dropout: float*)

Bases: Module

**\_\_init\_\_**(\**hidden\_size: int, att\_embed\_size: int, att\_embed\_dropout: float*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

**forward**(*hidden\_states, att\_feats, p\_att\_feats, att\_masks=None, \*\*kwargs*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**training:** bool

**class** xmodaler.modeling.layers.bert.**BertSelfAttention**(\**hidden\_size, num\_attention\_heads, attention\_probs\_dropout\_prob*)

Bases: Module

**\_\_init\_\_**(\**hidden\_size, num\_attention\_heads, attention\_probs\_dropout\_prob*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

**forward**(*hidden\_states, attention\_mask, history\_states=None*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod** **from\_config**(*cfg*)

**training:** bool

**transpose\_for\_scores**(*x*)

**class** xmodaler.modeling.layers.bert.**BertSelfOutput**(\**hidden\_size: int, layer\_norm\_eps: float, hidden\_dropout\_prob: float*)

Bases: Module

**\_\_init\_\_**(\**hidden\_size: int, layer\_norm\_eps: float, hidden\_dropout\_prob: float*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

**forward**(*hidden\_states, input\_tensor*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod** `from_config(cfg)`

**training:** `bool`

**class** `xmodaler.modeling.layers.bert.BertAttention(*, bert_self_attention, bert_self_output)`

Bases: `Module`

**\_\_init\_\_**(*\*, bert\_self\_attention, bert\_self\_output*)

Initializes internal Module state, shared by both `nn.Module` and `ScriptModule`.

**forward**(*input\_tensor, attention\_mask, history\_states=None*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod** `from_config(cfg)`

**training:** `bool`

**class** `xmodaler.modeling.layers.bert.BertIntermediate(*, hidden_size: int, hidden_act: str, intermediate_size: int, intermediate_drop: float)`

Bases: `Module`

**\_\_init\_\_**(*\*, hidden\_size: int, hidden\_act: str, intermediate\_size: int, intermediate\_drop: float*)

Initializes internal Module state, shared by both `nn.Module` and `ScriptModule`.

**forward**(*hidden\_states*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod** `from_config(cfg)`

**training:** `bool`

**class** `xmodaler.modeling.layers.bert.BertOutput(*, hidden_size: int, intermediate_size: int, layer_norm_eps: float, ffn_dropout_prob: float)`

Bases: `Module`

**\_\_init\_\_**(\* , *hidden\_size: int, intermediate\_size: int, layer\_norm\_eps: float, ffn\_dropout\_prob: float*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

**forward**(*hidden\_states, input\_tensor*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod from\_config**(*cfg*)

**training:** `bool`

**class** xmodaler.modeling.layers.bert.**BertXAttention**(\* , *hidden\_size, num\_attention\_heads, attention\_probs\_dropout\_prob*)

Bases: `Module`

**\_\_init\_\_**(\* , *hidden\_size, num\_attention\_heads, attention\_probs\_dropout\_prob*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

**forward**(*query, key, value, attention\_mask*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod from\_config**(*cfg*)

**training:** `bool`

**transpose\_for\_scores**(*x*)

**class** xmodaler.modeling.layers.bert.**BertCrossAttention**(\* , *bert\_cross\_attention, bert\_self\_output*)

Bases: `Module`

**\_\_init\_\_**(\* , *bert\_cross\_attention, bert\_self\_output*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

**forward**(*query, key, value, attention\_mask, q\_attention\_mask*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

---

```
classmethod from_config(cfg)
```

```
training: bool
```

```
class xmodaler.modeling.layers.bert.BertLayer(*, bert_attention, bert_intermediate, bert_output)
```

```
Bases: Module
```

```
__init__(*, bert_attention, bert_intermediate, bert_output)
```

```
    Initializes internal Module state, shared by both nn.Module and ScriptModule.
```

```
forward(hidden_states, attention_mask, history_states=None)
```

```
    Defines the computation performed at every call.
```

```
    Should be overridden by all subclasses.
```

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```
classmethod from_config(cfg)
```

```
training: bool
```

```
class xmodaler.modeling.layers.bert.BertUnderstandingLayer(*, bert_attention, v_bert_intermediate,  
                                                             v_bert_output, t_bert_intermediate,  
                                                             t_bert_output)
```

```
Bases: Module
```

```
__init__(*, bert_attention, v_bert_intermediate, v_bert_output, t_bert_intermediate, t_bert_output)
```

```
    Initializes internal Module state, shared by both nn.Module and ScriptModule.
```

```
forward(input_tensor1, attention_mask1, input_tensor2, attention_mask2)
```

```
    Defines the computation performed at every call.
```

```
    Should be overridden by all subclasses.
```

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```
classmethod from_config(cfg)
```

```
training: bool
```

```
class xmodaler.modeling.layers.bert.BertGenerationLayer(*, bert_attention, bert_cross_attention,  
                                                             bert_intermediate, bert_output)
```

```
Bases: Module
```

```
__init__(*, bert_attention, bert_cross_attention, bert_intermediate, bert_output)
```

```
    Initializes internal Module state, shared by both nn.Module and ScriptModule.
```

```
forward(lang_feats, v_feats, lang_attention_mask=None, v_attention_mask=None, t_history_states=None)
```

```
    Defines the computation performed at every call.
```

```
    Should be overridden by all subclasses.
```

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod** `from_config(cfg)`

**training:** `bool`

**class** `xmodaler.modeling.layers.bert.BertPooler(*, hidden_size: int)`

Bases: `Module`

**\_\_init\_\_**(`*, hidden_size: int`)

Initializes internal Module state, shared by both `nn.Module` and `ScriptModule`.

**forward**(`hidden_states`)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod** `from_config(cfg)`

**training:** `bool`

**class** `xmodaler.modeling.layers.bert.BertPredictionHeadTransform(*, hidden_size: int, hidden_act: str, layer_norm_eps: float)`

Bases: `Module`

**\_\_init\_\_**(`*, hidden_size: int, hidden_act: str, layer_norm_eps: float`)

Initializes internal Module state, shared by both `nn.Module` and `ScriptModule`.

**forward**(`hidden_states`)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod** `from_config(cfg)`

**training:** `bool`

**class** `xmodaler.modeling.layers.multihead_attention.MultiHeadAttentionMemory(*, d_model: int, d_k: int, d_v: int, num_head: int, dropout: float, num_memory: int)`

Bases: Module

Multi-head attention layer with Dropout and Layer Normalization.

**\_\_init\_\_**(\*, *d\_model*: int, *d\_k*: int, *d\_v*: int, *num\_head*: int, *dropout*: float, *num\_memory*: int)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

**forward**(*queries*, *keys*, *values*, *attention\_mask*=None, *attention\_weights*=None)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**training:** bool

**class** xmodaler.modeling.layers.multihead\_attention.**MultiHeadAttention**(\*, *d\_model*: int, *d\_k*: int, *d\_v*: int, *num\_head*: int, *dropout*: float)

Bases: Module

Multi-head attention layer with Dropout and Layer Normalization.

**\_\_init\_\_**(\*, *d\_model*: int, *d\_k*: int, *d\_v*: int, *num\_head*: int, *dropout*: float)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

**forward**(*queries*, *keys*, *values*, *attention\_mask*=None, *attention\_weights*=None, *history\_states*=None)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**training:** bool

**class** xmodaler.modeling.layers.positionwise\_feedforward.**PositionWiseFeedForward**(\*, *d\_model*: int, *d\_ff*: int, *dropout*: float)

Bases: Module

Position-wise feed forward layer

**\_\_init\_\_**(\*, *d\_model*: int, *d\_ff*: int, *dropout*: float)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

**forward**(*inputs*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**training:** `bool`

```
class xmodaler.modeling.layers.lowrank_bilinear_layers.LowRankBilinearAttention(*,
                                                                              embed_dim:
                                                                              int,
                                                                              att_heads:
                                                                              int,
                                                                              att_mid_dim:
                                                                              list,
                                                                              att_mid_drop:
                                                                              float,
                                                                              dropout:
                                                                              float,
                                                                              layer_num:
                                                                              int,
                                                                              act_type:
                                                                              str,
                                                                              elu_alpha:
                                                                              float)
```

Bases: `Module`

```
__init__(*, embed_dim: int, att_heads: int, att_mid_dim: list, att_mid_drop: float, dropout: float,
          layer_num: int, act_type: str, elu_alpha: float)
```

Initializes internal `Module` state, shared by both `nn.Module` and `ScriptModule`.

```
forward(gv_feat, att_feats, att_mask, p_att_feats=None, precompute=False)
```

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```
precompute(key, value2)
```

**training:** `bool`

```
class xmodaler.modeling.layers.lowrank_bilinear_layers.LowRankBilinearLayer(*, embed_dim:
                                                                              int, att_heads:
                                                                              int, att_mid_dim:
                                                                              list,
                                                                              att_mid_drop:
                                                                              float, dropout:
                                                                              float, act_type:
                                                                              str, elu_alpha:
                                                                              float)
```

Bases: `Module`



---

```
__init__(* , embed_dim: int, att_heads: int, att_mid_dim: list, att_mid_drop: float, dropout: float, act_type: str, elu_alpha: float)
```

Initializes internal Module state, shared by both nn.Module and ScriptModule.

```
forward(x, key=None, mask=None, value1=None, value2=None, precompute=False)
```

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```
precompute(key, value2)
```

```
training: bool
```

```
class xmodaler.modeling.layers.scattention.SCAttention(mid_dims: list, mid_dropout: float)
```

Bases: BasicAtt

```
__init__(mid_dims: list, mid_dropout: float)
```

Initializes internal Module state, shared by both nn.Module and ScriptModule.

```
forward(att_map, att_mask, value1, value2)
```

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```
training: bool
```

```
class xmodaler.modeling.layers.tdconvd_layers.TemporalDeformableLayer(in_channels: int,
                                                                    out_channels: int,
                                                                    kernel_size: int, stride:
                                                                    int, padding_mode: str,
                                                                    offset_act: str, min_idx:
                                                                    int, max_idx: int,
                                                                    clamp_idx: bool,
                                                                    dropout: float,
                                                                    use_norm: bool)
```

Bases: Module

```
__init__(in_channels: int, out_channels: int, kernel_size: int, stride: int, padding_mode: str, offset_act: str,
          min_idx: int, max_idx: int, clamp_idx: bool, dropout: float, use_norm: bool)
```

Initializes internal Module state, shared by both nn.Module and ScriptModule.

```
forward(inputs)
```

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**training:** `bool`

```
class xmodaler.modeling.layers.tdconvd_layers.TemporalDeformableBlock(in_channels: int,
                                                                       out_channels: int,
                                                                       kernel_size: int, stride:
                                                                       int, padding_mode: str,
                                                                       offset_act: str, min_idx:
                                                                       int, max_idx: int,
                                                                       clamp_idx: bool,
                                                                       use_norm: bool)
```

Bases: `Module`

```
__init__(in_channels: int, out_channels: int, kernel_size: int, stride: int, padding_mode: str, offset_act: str,
          min_idx: int, max_idx: int, clamp_idx: bool, use_norm: bool)
```

Initializes internal Module state, shared by both `nn.Module` and `ScriptModule`.

**forward**(*inputs*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**training:** `bool`

```
class xmodaler.modeling.layers.tdconvd_layers.ShiftedConvLayer(in_channels: int, out_channels:
                                                                int, kernel_size: list, stride: int,
                                                                padding_mode: str, dropout:
                                                                float, use_norm: bool)
```

Bases: `Module`

```
__init__(in_channels: int, out_channels: int, kernel_size: list, stride: int, padding_mode: str, dropout:
          float, use_norm: bool)
```

Initializes internal Module state, shared by both `nn.Module` and `ScriptModule`.

**forward**(*inputs*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**training:** `bool`

---

```
class xmodaler.modeling.layers.tdconvd_layers.SoftAttention(*, hidden_size: int, att_embed_size:
                                                         int, att_embed_dropout: float,
                                                         use_norm: bool)
```

Bases: Module

```
__init__(*, hidden_size: int, att_embed_size: int, att_embed_dropout: float, use_norm: bool)
```

Initializes internal Module state, shared by both nn.Module and ScriptModule.

```
forward(hidden_states, att_feats, p_att_feats, att_masks=None, **kwargs)
```

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**training:** bool

### 3.9.6 xmodaler.modeling.meta\_arch

```
xmodaler.modeling.meta_arch.build_model(cfg)
```

Build the whole model architecture, defined by `cfg.MODEL.META_ARCHITECTURE`. Note that it does not load any weights from `cfg`.

```
xmodaler.modeling.meta_arch.add_config(cfg, tmp_cfg)
```

```
class xmodaler.modeling.meta_arch.base_enc_dec.BaseEncoderDecoder(*, vocab_size, max_seq_len,
                                                                    token_embed, visual_embed,
                                                                    encoder, decoder, predictor,
                                                                    greedy_decoder,
                                                                    beam_searcher)
```

Bases: Module

```
__init__(*, vocab_size, max_seq_len, token_embed, visual_embed, encoder, decoder, predictor,
          greedy_decoder, beam_searcher)
```

Initializes internal Module state, shared by both nn.Module and ScriptModule.

```
classmethod add_config(cfg, tmp_cfg)
```

```
bind_or_init_weights()
```

```
decode_beam_search(batched_inputs, output_sents=False)
```

```
forward(batched_inputs, use_beam_search=None, output_sents=False)
```

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```
classmethod from_config(cfg)

abstract get_extended_attention_mask(batched_inputs)

greedy_decode(batched_inputs, output_sents=False)

preprocess_batch(batched_inputs)

training: bool

class xmodaler.modeling.meta_arch.RnnAttEncoderDecoder(*, vocab_size, max_seq_len, token_embed,
                                                       visual_embed, encoder, decoder, predictor,
                                                       greedy_decoder, beam_searcher)

Bases: BaseEncoderDecoder

get_extended_attention_mask(batched_inputs)

training: bool

class xmodaler.modeling.meta_arch.TransformerEncoderDecoder(*, vocab_size, max_seq_len,
                                                            token_embed, visual_embed, encoder,
                                                            decoder, predictor, greedy_decoder,
                                                            beam_searcher, v_predictor)

Bases: BaseEncoderDecoder

__init__(*, vocab_size, max_seq_len, token_embed, visual_embed, encoder, decoder, predictor,
         greedy_decoder, beam_searcher, v_predictor)

    Initializes internal Module state, shared by both nn.Module and ScriptModule.

classmethod from_config(cfg)

get_extended_attention_mask(batched_inputs)

training: bool

class xmodaler.modeling.meta_arch.TDENBiTransformer(*, vocab_size, max_seq_len, token_embed,
                                                     visual_embed, encoder, decoder, predictor,
                                                     greedy_decoder, beam_searcher, v_predictor)

Bases: TransformerEncoderDecoder

__init__(*, vocab_size, max_seq_len, token_embed, visual_embed, encoder, decoder, predictor,
         greedy_decoder, beam_searcher, v_predictor)

    Initializes internal Module state, shared by both nn.Module and ScriptModule.

classmethod from_config(cfg)

get_extended_attention_mask(batched_inputs)

training: bool

class xmodaler.modeling.meta_arch.TDENPretrain(*, vocab_size, max_seq_len, token_embed,
                                                visual_embed, encoder, decoder, predictor,
                                                greedy_decoder, beam_searcher, v_predictor,
                                                similarity_predictor)

Bases: TransformerEncoderDecoder
```

```
__init__(* , vocab_size, max_seq_len, token_embed, visual_embed, encoder, decoder, predictor,
          greedy_decoder, beam_searcher, v_predictor, similarity_predictor)
```

Initializes internal Module state, shared by both nn.Module and ScriptModule.

```
classmethod add_config(cfg, tmp_cfg)
```

```
classmethod from_config(cfg)
```

```
training: bool
```

```
class xmodaler.modeling.meta_arch.TDENCaptioner(* , vocab_size, max_seq_len, token_embed,
          visual_embed, encoder, decoder, predictor,
          greedy_decoder, beam_searcher, v_predictor)
```

Bases: [TransformerEncoderDecoder](#)

```
__init__(* , vocab_size, max_seq_len, token_embed, visual_embed, encoder, decoder, predictor,
          greedy_decoder, beam_searcher, v_predictor)
```

Initializes internal Module state, shared by both nn.Module and ScriptModule.

```
classmethod from_config(cfg)
```

```
training: bool
```

```
class xmodaler.modeling.meta_arch.UniterPretrain(* , vocab_size, max_seq_len, token_embed,
          visual_embed, encoder, decoder, predictor,
          greedy_decoder, beam_searcher, v_predictor,
          v_regressor, itm_predictor, tasks, mix_ratio)
```

Bases: [TransformerEncoderDecoder](#)

```
__init__(* , vocab_size, max_seq_len, token_embed, visual_embed, encoder, decoder, predictor,
          greedy_decoder, beam_searcher, v_predictor, v_regressor, itm_predictor, tasks, mix_ratio)
```

Initializes internal Module state, shared by both nn.Module and ScriptModule.

```
classmethod add_config(cfg, tmp_cfg)
```

```
classmethod from_config(cfg)
```

```
preprocess_inputs(inputs, task_name)
```

```
training: bool
```

```
class xmodaler.modeling.meta_arch.UniterForMMUnderstanding(* , vocab_size, max_seq_len,
          token_embed, visual_embed, encoder,
          decoder, predictor, greedy_decoder,
          beam_searcher, v_predictor,
          itm_predictor)
```

Bases: [TransformerEncoderDecoder](#)

```
__init__(* , vocab_size, max_seq_len, token_embed, visual_embed, encoder, decoder, predictor,
          greedy_decoder, beam_searcher, v_predictor, itm_predictor)
```

Initializes internal Module state, shared by both nn.Module and ScriptModule.

```
bind_or_init_weights()
```

```
classmethod from_config(cfg)
```

```
get_extended_attention_mask(batched_inputs)
```

```
training: bool
```

### 3.9.7 xmodaler.modeling.predictor

`xmodaler.modeling.predictor.build_predictor(cfg)`

`xmodaler.modeling.predictor.build_v_predictor(cfg)`

`xmodaler.modeling.predictor.add_predictor_config(cfg, tmp_cfg)`

`xmodaler.modeling.predictor.build_predictor_with_name(cfg, name)`

**class** `xmodaler.modeling.predictor.BasePredictor(*, hidden_size: int, vocab_size: int, dropout: float)`

Bases: Module

**\_\_init\_\_**(\*, hidden\_size: int, vocab\_size: int, dropout: float)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

**classmethod** `add_config(cfg)`

**forward**(batched\_inputs)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod** `from_config(cfg)`

**training:** bool

**class** `xmodaler.modeling.predictor.BertPredictionHead(*, hidden_size, vocab_size, transform)`

Bases: Module

**\_\_init\_\_**(\*, hidden\_size, vocab\_size, transform)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

**classmethod** `add_config(cfg)`

**forward**(batched\_inputs)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod** `from_config(cfg)`

**training:** bool

**class** `xmodaler.modeling.predictor.BertVisualPredictionHead(*, hidden_size, v_target_size, transform)`

Bases: Module

---

```
__init__(*, hidden_size, v_target_size, transform)
```

Initializes internal Module state, shared by both nn.Module and ScriptModule.

```
classmethod add_config(cfg)
```

```
forward(batched_inputs)
```

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```
classmethod from_config(cfg)
```

```
training: bool
```

```
class xmodaler.modeling.predictor.BertVisualFeatureRegressionHead(*, hidden_size, v_feat_dim,  
                                                                    transform)
```

Bases: Module

```
__init__(*, hidden_size, v_feat_dim, transform)
```

Initializes internal Module state, shared by both nn.Module and ScriptModule.

```
classmethod add_config(cfg)
```

```
forward(batched_inputs)
```

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```
classmethod from_config(cfg)
```

```
training: bool
```

```
class xmodaler.modeling.predictor.BertIsMatchedPredictor(*, hidden_size, pooler)
```

Bases: Module

```
__init__(*, hidden_size, pooler)
```

Initializes internal Module state, shared by both nn.Module and ScriptModule.

```
classmethod add_config(cfg)
```

```
forward(batched_inputs)
```

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod** `from_config(cfg)`

**training:** `bool`

**class** `xmodaler.modeling.predictor.MultiModalPredictor`(*\*, labels\_num: int, pooler\_input\_size: int, pooler\_output\_size: int, pooler\_bn: bool, pooler\_dropout: float, num\_understanding\_layers: int, num\_generation\_layers: int*)

Bases: `Module`

**\_\_init\_\_**(*\*, labels\_num: int, pooler\_input\_size: int, pooler\_output\_size: int, pooler\_bn: bool, pooler\_dropout: float, num\_understanding\_layers: int, num\_generation\_layers: int*)

Initializes internal Module state, shared by both `nn.Module` and `ScriptModule`.

**classmethod** `add_config(cfg)`

**forward**(*batched\_inputs*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod** `from_config(cfg)`

**training:** `bool`

**class** `xmodaler.modeling.predictor.SingleStreamMultiModalPredictor`(*\*, hidden\_size: int, labels\_num: int, pooler*)

Bases: `Module`

**\_\_init\_\_**(*\*, hidden\_size: int, labels\_num: int, pooler*)

Initializes internal Module state, shared by both `nn.Module` and `ScriptModule`.

**classmethod** `add_config(cfg)`

**forward**(*batched\_inputs*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**classmethod** `from_config(cfg)`



```
test_forward(u_logits)
```

```
training: bool
```

```
class xmodaler.modeling.predictor.MultiModalSimilarity(*, pooler_input_size: int,
                                                         pooler_output_size: int, pooler_bn: bool,
                                                         pooler_dropout: float, num_hidden_layers:
                                                         int, v_num_hidden_layers: int)
```

Bases: Module

```
__init__(*, pooler_input_size: int, pooler_output_size: int, pooler_bn: bool, pooler_dropout: float,
          num_hidden_layers: int, v_num_hidden_layers: int)
```

Initializes internal Module state, shared by both nn.Module and ScriptModule.

```
classmethod add_config(cfg)
```

```
forward(batched_inputs)
```

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```
classmethod from_config(cfg)
```

```
training: bool
```

## 3.10 xmodaler.optim

```
xmodaler.optim.build_optimizer(cfg: CfgNode, model: Module) → Optimizer
```

Build an optimizer from config.

```
class xmodaler.optim.Adagrad(*, params, lr=0.01, lr_decay=0, weight_decay=0,
                              initial_accumulator_value=0, eps=1e-10)
```

Bases: Adagrad

```
__init__(*, params, lr=0.01, lr_decay=0, weight_decay=0, initial_accumulator_value=0, eps=1e-10)
```

```
classmethod from_config(cfg, params)
```

```
class xmodaler.optim.Adam(*, params, lr=0.001, betas=(0.9, 0.999), eps=1e-08, weight_decay=0,
                           amsgrad=False)
```

Bases: Adam

```
__init__(*, params, lr=0.001, betas=(0.9, 0.999), eps=1e-08, weight_decay=0, amsgrad=False)
```

```
classmethod from_config(cfg, params)
```

```
class xmodaler.optim.AdamW(*, params, lr=0.001, betas=(0.9, 0.999), eps=1e-08, weight_decay=0.01,
                            amsgrad=False)
```

Bases: AdamW

```
__init__(* , params, lr=0.001, betas=(0.9, 0.999), eps=1e-08, weight_decay=0.01, amsgrad=False)
```

```
classmethod from_config(cfg, params)
```

```
class xmodaler.optim.Adamax(* , params, lr=0.002, betas=(0.9, 0.999), eps=1e-08, weight_decay=0)
```

Bases: Adamax

```
__init__(* , params, lr=0.002, betas=(0.9, 0.999), eps=1e-08, weight_decay=0)
```

```
classmethod from_config(cfg, params)
```

```
class xmodaler.optim.RAdam(* , params, lr=0.001, betas=(0.9, 0.999), eps=1e-08, weight_decay=0)
```

Bases: Optimizer

```
__init__(* , params, lr=0.001, betas=(0.9, 0.999), eps=1e-08, weight_decay=0)
```

```
classmethod from_config(cfg, params)
```

```
step(closure=None)
```

Performs a single optimization step (parameter update).

**Parameters**

**closure** (*Callable*) – A closure that reevaluates the model and returns the loss. Optional for most optimizers.

---

**Note:** Unless otherwise specified, this function should not modify the `.grad` field of the parameters.

---

```
class xmodaler.optim.RMSprop(* , params, lr=0.01, alpha=0.99, eps=1e-08, weight_decay=0, momentum=0, centered=False)
```

Bases: RMSprop

```
__init__(* , params, lr=0.01, alpha=0.99, eps=1e-08, weight_decay=0, momentum=0, centered=False)
```

```
classmethod from_config(cfg, params)
```

```
class xmodaler.optim.SGD(* , params, lr=0.1, momentum=0, dampening=0, weight_decay=0, nesterov=False)
```

Bases: SGD

```
__init__(* , params, lr=0.1, momentum=0, dampening=0, weight_decay=0, nesterov=False)
```

```
classmethod from_config(cfg, params)
```

```
class xmodaler.optim.BertAdam(* , params, lr=0.001, b1=0.9, b2=0.999, eps=1e-06, weight_decay=0.01, max_grad_norm=1.0)
```

Bases: Optimizer

```
__init__(* , params, lr=0.001, b1=0.9, b2=0.999, eps=1e-06, weight_decay=0.01, max_grad_norm=1.0)
```

```
classmethod from_config(cfg, params)
```

```
step(closure=None)
```

Performs a single optimization step. :param closure: A closure that reevaluates the model and returns the loss.

### 3.11 xmodaler.scorer

`xmodaler.scorer.build_scorer(cfg)`

**class** `xmodaler.scorer.BaseScorer(*, types, scorers, weights, gt_path, eos_id)`

Bases: object

`__init__(*, types, scorers, weights, gt_path, eos_id)`

**classmethod** `from_config(cfg)`

`get_sents(sent)`

**class** `xmodaler.scorer.Cider(*, n: int, sigma: float, cider_cached: str)`

Bases: object

Main Class to compute the CIDEr metric

`__init__(*, n: int, sigma: float, cider_cached: str)`

**compute\_score**(gts, res)

Main function to compute CIDEr score :param hypo\_for\_image (dict) : dictionary with key <image> and value <tokenized hypothesis / candidate sentence>

ref\_for\_image (dict) : dictionary with key <image> and value <tokenized reference sentence>

**Returns**

cider (float) : computed CIDEr score for the corpus

**classmethod** `from_config(cfg)`

**method**()

**class** `xmodaler.scorer.BertTokenizedScorer(*, types, scorers, weights, gt_path, bert_tokenizer)`

Bases: object

`__init__(*, types, scorers, weights, gt_path, bert_tokenizer)`

**classmethod** `from_config(cfg)`

`get_sents(sent)`

### 3.12 xmodaler.tokenization

**class** `xmodaler.tokenization.BertTokenizer(vocab_file, do_lower_case=True, do_basic_tokenize=True, never_split=None, unk_token='[UNK]', sep_token='[SEP]', pad_token='[PAD]', cls_token='[CLS]', mask_token='[MASK]', tokenize_chinese_chars=True, **kwargs)`

Bases: PreTrainedTokenizer

Constructs a BertTokenizer. BertTokenizer runs end-to-end tokenization: punctuation splitting + wordpiece

**Parameters**

- **vocab\_file** – Path to a one-wordpiece-per-line vocabulary file

- **do\_lower\_case** – Whether to lower case the input. Only has an effect when `do_wordpiece_only=False`
- **do\_basic\_tokenize** – Whether to do basic tokenization before wordpiece.
- **max\_len** – An artificial maximum length to truncate tokenized sequences to; Effective maximum length is always the minimum of this value (if specified) and the underlying BERT model's sequence length.
- **never\_split** – List of tokens which will never be split during tokenization. Only has an effect when `do_wordpiece_only=False`

**add\_special\_tokens\_sentences\_pair**(*token\_ids\_0, token\_ids\_1*)

Adds special tokens to a sequence pair for sequence classification tasks. A BERT sequence pair has the following format: [CLS] A [SEP] B [SEP]

**add\_special\_tokens\_single\_sentence**(*token\_ids*)

Adds special tokens to the a sequence for sequence classification tasks. A BERT sequence has the following format: [CLS] X [SEP]

**convert\_tokens\_to\_string**(*tokens*)

Converts a sequence of tokens (string) in a single string.

```
max_model_input_sizes = {'bert-base-cased': 512, 'bert-base-cased-finetuned-mrpc': 512, 'bert-base-chinese': 512, 'bert-base-german-cased': 512, 'bert-base-multilingual-cased': 512, 'bert-base-multilingual-uncased': 512, 'bert-base-uncased': 512, 'bert-large-cased': 512, 'bert-large-cased-whole-word-masking': 512, 'bert-large-cased-whole-word-masking-finetuned-squad': 512, 'bert-large-uncased': 512, 'bert-large-uncased-whole-word-masking': 512, 'bert-large-uncased-whole-word-masking-finetuned-squad': 512}
```

```
pretrained_init_configuration = {'bert-base-cased': {'do_lower_case': False}, 'bert-base-cased-finetuned-mrpc': {'do_lower_case': False}, 'bert-base-chinese': {'do_lower_case': False}, 'bert-base-german-cased': {'do_lower_case': False}, 'bert-base-multilingual-cased': {'do_lower_case': False}, 'bert-base-multilingual-uncased': {'do_lower_case': True}, 'bert-base-uncased': {'do_lower_case': True}, 'bert-large-cased': {'do_lower_case': False}, 'bert-large-cased-whole-word-masking': {'do_lower_case': False}, 'bert-large-cased-whole-word-masking-finetuned-squad': {'do_lower_case': False}, 'bert-large-uncased': {'do_lower_case': True}, 'bert-large-uncased-whole-word-masking': {'do_lower_case': True}, 'bert-large-uncased-whole-word-masking-finetuned-squad': {'do_lower_case': True}}
```

```

pretrained_vocab_files_map = {'vocab_file': {'bert-base-cased':
'https://s3.amazonaws.com/models.huggingface.co/bert/bert-base-cased-vocab.txt',
'bert-base-cased-finetuned-mrpc': 'https://s3.amazonaws.com/models.huggingface.co/
bert/bert-base-cased-finetuned-mrpc-vocab.txt', 'bert-base-chinese':
'https://s3.amazonaws.com/models.huggingface.co/bert/bert-base-chinese-vocab.txt',
'bert-base-german-cased': 'https://int-deepset-models-bert.s3.eu-central-1.
amazonaws.com/pytorch/bert-base-german-cased-vocab.txt',
'bert-base-multilingual-cased': 'https://s3.amazonaws.com/models.huggingface.co/
bert/bert-base-multilingual-cased-vocab.txt', 'bert-base-multilingual-uncased':
'https://s3.amazonaws.com/models.huggingface.co/bert/
bert-base-multilingual-uncased-vocab.txt', 'bert-base-uncased':
'../pretrain/BERT/bert-base-uncased-vocab.txt', 'bert-large-cased':
'https://s3.amazonaws.com/models.huggingface.co/bert/bert-large-cased-vocab.txt',
'bert-large-cased-whole-word-masking': 'https://s3.amazonaws.com/models.
huggingface.co/bert/bert-large-cased-whole-word-masking-vocab.txt',
'bert-large-cased-whole-word-masking-finetuned-squad':
'https://s3.amazonaws.com/models.huggingface.co/bert/
bert-large-cased-whole-word-masking-finetuned-squad-vocab.txt',
'bert-large-uncased':
'https://s3.amazonaws.com/models.huggingface.co/bert/bert-large-uncased-vocab.txt',
'bert-large-uncased-whole-word-masking': 'https://s3.amazonaws.com/models.
huggingface.co/bert/bert-large-uncased-whole-word-masking-vocab.txt',
'bert-large-uncased-whole-word-masking-finetuned-squad':
'https://s3.amazonaws.com/models.huggingface.co/bert/
bert-large-uncased-whole-word-masking-finetuned-squad-vocab.txt'}}

```

**save\_vocabulary**(*vocab\_path*)

Save the tokenizer vocabulary to a directory or file.

**vocab\_files\_names** = {'vocab\_file': 'vocab.txt'}

**property vocab\_size**

Size of the base vocabulary (without the added tokens)

### 3.13 xmodaler.utils

**xmodaler.utils.collect\_env.collect\_env\_info()**

**xmodaler.utils.colormap.colormap**(*rgb=False, maximum=255*)

#### Parameters

- **rgb** (*bool*) – whether to return RGB colors or BGR colors.
- **maximum** (*int*) – either 255 or 1

#### Returns

a float32 array of Nx3 colors, in range [0, 255] or [0, 1]

#### Return type

ndarray

**xmodaler.utils.colormap.random\_color**(*rgb=False, maximum=255*)

#### Parameters

- **rgb** (*bool*) – whether to return RGB colors or BGR colors.
- **maximum** (*int*) – either 255 or 1

**Returns**

a vector of 3 numbers

**Return type**

ndarray

`xmodaler.utils.comm.all_gather(data, group=None)`

Run `all_gather` on arbitrary picklable data (not necessarily tensors).

**Parameters**

- **data** – any picklable object
- **group** – a torch process group. By default, will use a group which contains all ranks on gloo backend.

**Returns**

list of data gathered from each rank

**Return type**

list[data]

`xmodaler.utils.comm.gather(data, dst=0, group=None)`

Run `gather` on arbitrary picklable data (not necessarily tensors).

**Parameters**

- **data** – any picklable object
- **dst** (*int*) – destination rank
- **group** – a torch process group. By default, will use a group which contains all ranks on gloo backend.

**Returns**

on **dst**, a list of data gathered from each rank. Otherwise, an empty list.

**Return type**

list[data]

`xmodaler.utils.distributed.all_gather_list(data)`

Gathers arbitrary data from all nodes into a list.

. autofunction:: `xmodaler.utils.comm.any_broadcast`

`xmodaler.utils.comm.get_local_rank()` → int

**Returns**

The rank of the current process within the local (per-machine) process group.

`xmodaler.utils.comm.get_local_size()` → int

**Returns**

The size of the per-machine process group, i.e. the number of processes per machine.

`xmodaler.utils.comm.get_rank()` → int

`xmodaler.utils.comm.get_world_size()` → int

`xmodaler.utils.comm.is_main_process()` → bool

`xmodaler.utils.comm.reduce_dict(input_dict, average=True)`

Reduce the values in the dictionary from all processes so that process with rank 0 has the reduced results.

#### Parameters

- **input\_dict** (*dict*) – inputs to be reduced. All the values must be scalar CUDA Tensor.
- **average** (*bool*) – whether to do average or sum

#### Returns

a dict with the same keys as `input_dict`, after reduction.

`xmodaler.utils.comm.shared_random_seed()`

#### Returns

**a random number that is the same across all workers.**

If workers need a shared RNG, they can use this shared seed to create one.

#### Return type

int

All workers must call this function, otherwise it will deadlock.

`xmodaler.utils.comm.synchronize()`

Helper function to synchronize (barrier) among all processes when using distributed training

`xmodaler.utils.comm.unwrap_model(model)`

`xmodaler.utils.env.seed_all_rng(seed=None)`

Set the random seed for the RNG in torch, numpy and python.

#### Parameters

**seed** (*int*) – if None, will use a strong random seed.

`xmodaler.utils.env.setup_environment()`

Perform environment setup work. The default setup is a no-op, but this function allows the user to specify a Python source file or a module in the `$DETECTRON2_ENV_MODULE` environment variable, that performs custom setup work that may be necessary to their computing environment.

`xmodaler.utils.env.setup_custom_environment(custom_module)`

Load custom environment setup by importing a Python source file or a module, and run the setup function.

`xmodaler.utils.events.get_event_storage()`

#### Returns

The [EventStorage](#) object that's currently being used. Throws an error if no [EventStorage](#) is currently enabled.

**class** `xmodaler.utils.events.JSONWriter(json_file, window_size=20)`

Bases: `EventWriter`

Write scalars to a json file.

It saves scalars as one json per line (instead of a big json) for easy parsing.

Examples parsing such a json file:

```
$ cat metrics.json | jq -s '.[0:2]'
```

```
[
  {
    "data_time": 0.008433341979980469,
    "iteration": 19,
    "loss": 1.9228371381759644,
    "loss_box_reg": 0.050025828182697296,
    "loss_classifier": 0.5316952466964722,
    "loss_mask": 0.7236229181289673,
    "loss_rpn_box": 0.0856662318110466,
    "loss_rpn_cls": 0.48198649287223816,
    "lr": 0.007173333333333333,
    "time": 0.25401854515075684
  },
  {
    "data_time": 0.007216215133666992,
    "iteration": 39,
    "loss": 1.282649278640747,
    "loss_box_reg": 0.06222952902317047,
    "loss_classifier": 0.30682939291000366,
    "loss_mask": 0.6970193982124329,
    "loss_rpn_box": 0.038663312792778015,
    "loss_rpn_cls": 0.1471673548221588,
    "lr": 0.007706666666666667,
    "time": 0.2490077018737793
  }
]
```

```
$ cat metrics.json | jq '.loss_mask'
```

```
0.7126231789588928
0.689423680305481
0.6776131987571716
...
```

```
__init__(json_file, window_size=20)
```

#### Parameters

- **json\_file** (*str*) – path to the json file. New data will be appended if the file exists.
- **window\_size** (*int*) – the window size of median smoothing for the scalars whose *smoothing\_hint* are True.

```
close()
```

```
write()
```

```
class xmodaler.utils.events.TensorboardWriter(log_dir: str, window_size: int = 20, **kwargs)
```

Bases: `EventWriter`

Write all scalars to a tensorboard file.

```
__init__(log_dir: str, window_size: int = 20, **kwargs)
```

#### Parameters

- **log\_dir** (*str*) – the directory to save the output events



- **window\_size** (*int*) – the scalars will be median-smoothed by this window size
- **kwargs** – other arguments passed to `torch.utils.tensorboard.SummaryWriter(...)`

**close()**

**write()**

**class** `xmodaler.utils.events.CommonMetricPrinter`(*max\_iter: Optional[int] = None*)

Bases: `EventWriter`

Print **common** metrics to the terminal, including iteration time, ETA, memory, all losses, and the learning rate. It also applies smoothing using a window of 20 elements.

It's meant to print common metrics in common ways. To print something in more customized ways, please implement a similar printer by yourself.

**\_\_init\_\_**(*max\_iter: Optional[int] = None*)

**Parameters**

**max\_iter** – the maximum number of iterations to train. Used to compute ETA. If not given, ETA will not be printed.

**write()**

**class** `xmodaler.utils.events.EventStorage`(*start\_iter=0*)

Bases: `object`

The user-facing class that provides metric storage functionalities.

In the future we may add support for storing / logging other types of data if needed.

**\_\_init\_\_**(*start\_iter=0*)

**Parameters**

**start\_iter** (*int*) – the iteration number to start with

**clear\_histograms()**

Delete all the stored histograms for visualization. This should be called after histograms are written to tensorboard.

**clear\_images()**

Delete all the stored images for visualization. This should be called after images are written to tensorboard.

**histories()**

**Returns**

the HistoryBuffer for all scalars

**Return type**

dict[name -> HistoryBuffer]

**history**(*name*)

**Returns**

the scalar history for name

**Return type**

HistoryBuffer

**property iter**

Returns: int: The current iteration number. When used together with a trainer, this is ensured to be the same as `trainer.iter`.

**property iteration****latest()****Returns**

**mapping from the name of each scalar to the most**  
recent value and the iteration number its added.

**Return type**

dict[str -> (float, int)]

**latest\_with\_smoothing\_hint(window\_size=20)**

Similar to `latest()`, but the returned values are either the un-smoothed original latest value, or a median of the given `window_size`, depend on whether the `smoothing_hint` is `True`.

This provides a default behavior that other writers can use.

**name\_scope(name)****Yields**

A context within which all the events added to this storage will be prefixed by the name scope.

**put\_histogram(hist\_name, hist\_tensor, bins=1000)**

Create a histogram from a tensor.

**Parameters**

- **hist\_name** (*str*) – The name of the histogram to put into tensorboard.
- **hist\_tensor** (*torch.Tensor*) – A Tensor of arbitrary shape to be converted into a histogram.
- **bins** (*int*) – Number of histogram bins.

**put\_image(img\_name, img\_tensor)**

Add an *img\_tensor* associated with *img\_name*, to be shown on tensorboard.

**Parameters**

- **img\_name** (*str*) – The name of the image to put into tensorboard.
- **img\_tensor** (*torch.Tensor or numpy.array*) – An *uint8* or *float* Tensor of shape [*channel, height, width*] where *channel* is 3. The image format should be RGB. The elements in *img\_tensor* can either have values in [0, 1] (*float32*) or [0, 255] (*uint8*). The *img\_tensor* will be visualized in tensorboard.

**put\_scalar(name, value, smoothing\_hint=True)**

Add a scalar *value* to the *HistoryBuffer* associated with *name*.

**Parameters**

**smoothing\_hint** (*bool*) – a ‘hint’ on whether this scalar is noisy and should be smoothed when logged. The hint will be accessible through `EventStorage.smoothing_hints()`. A writer may ignore the hint and apply custom smoothing rule.

It defaults to `True` because most scalars we save need to be smoothed to provide any useful signal.

**put\_scalars**(\*, *smoothing\_hint=True*, \*\*kwargs)

Put multiple scalars from keyword arguments.

### Examples

```
storage.put_scalars(loss=my_loss, accuracy=my_accuracy, smoothing_hint=True)
```

**smoothing\_hints**()

#### Returns

**the user-provided hint on whether the scalar**  
is noisy and needs smoothing.

#### Return type

dict[name -> bool]

**step**()

User should either: (1) Call this function to increment `storage.iter` when needed. Or (2) Set `storage.iter` to the correct iteration number before each iteration.

The storage will then be able to associate the new data with an iteration number.

**class** `xmodaler.utils.file_io.PathHandler`(*async\_executor: Optional[Executor] = None*)

Bases: `EventLogger`

`PathHandler` is a base class that defines common I/O functionality for a URI protocol. It routes I/O for a generic URI which may look like “protocol://\*” or a canonical filepath “/foo/bar/baz”.

**\_\_init\_\_**(*async\_executor: Optional[Executor] = None*) → None

When registering a `PathHandler`, the user can optionally pass in a `Executor` to run the asynchronous file operations. NOTE: For regular non-async operations of `PathManager`, there is no need to pass `async_executor`.

#### Parameters

**async\_executor** (optional `Executor`) – Used for async file operations. Usage: `'''`

```
path_handler          =          NativePathHandler(async_executor=exe)
path_manager.register_handler(path_handler)
```

`'''`

`xmodaler.utils.file_io.PathManager`

This is a detectron2 project-specific `PathManager`. We try to stay away from global `PathManager` in `fvcore` as it introduces potential conflicts among other libraries.

**class** `xmodaler.utils.logger.Counter`(\*\*kws)

Bases: `dict`

Dict subclass for counting hashable items. Sometimes called a bag or multiset. Elements are stored as dictionary keys and their counts are stored as dictionary values.

```
>>> c = Counter('abcdeababcdabcaba') # count elements from a string
```

```
>>> c.most_common(3)                  # three most common elements
```

```
[('a', 5), ('b', 4), ('c', 3)]
```

```
>>> sorted(c)                         # list all unique elements
```

```
['a', 'b', 'c', 'd', 'e']
```

(continues on next page)

(continued from previous page)

```
>>> ''.join(sorted(c.elements())) # list elements with repetitions
'aaaaabbbbcccdde'
>>> sum(c.values())               # total of all counts
15
```

```
>>> c['a']                        # count of letter 'a'
5
>>> for elem in 'shazam':         # update counts from an iterable
...     c[elem] += 1             # by adding 1 to each element's count
>>> c['a']                        # now there are seven 'a'
7
>>> del c['b']                    # remove all 'b'
>>> c['b']                        # now there are zero 'b'
0
```

```
>>> d = Counter('simsalabim')     # make another counter
>>> c.update(d)                   # add in the second counter
>>> c['a']                         # now there are nine 'a'
9
```

```
>>> c.clear()                     # empty the counter
>>> c
Counter()
```

Note: If a count is set to zero or reduced to zero, it will remain in the counter until the entry is deleted or the counter is cleared:

```
>>> c = Counter('aaabbc')
>>> c['b'] -= 2                   # reduce the count of 'b' by two
>>> c.most_common()              # 'b' is still in, but its count is zero
[('a', 3), ('c', 1), ('b', 0)]
```

### **\_\_init\_\_**(\*\*kws)

Create a new, empty Counter object. And if given, count elements from an input iterable. Or, initialize the count from another mapping of elements to their counts.

```
>>> c = Counter()                 # a new, empty counter
>>> c = Counter('gallahad')      # a new counter from an iterable
>>> c = Counter({'a': 4, 'b': 2}) # a new counter from a mapping
>>> c = Counter(a=4, b=2)         # a new counter from keyword args
```

### **copy()**

Return a shallow copy.

### **elements()**

Iterator over elements repeating each as many times as its count.

```
>>> c = Counter('ABCABC')
>>> sorted(c.elements())
['A', 'A', 'B', 'B', 'C', 'C']
```

# Knuth's example for prime factors of 1836:  $2^2 \cdot 3^3 \cdot 17$

```
>>> prime_factors = Counter({2: 2, 3:
```

```
3, 17: 1}) >>> product = 1 >>> for factor in prime_factors.elements(): # loop over factors ... product *=
factor # and multiply them >>> product 1836
```

Note, if an element's count has been set to zero or is a negative number, elements() will ignore it.

**classmethod fromkeys**(iterable, v=None)

Create a new dictionary with keys from iterable and values set to value.

**most\_common**(n=None)

List the n most common elements and their counts from the most common to the least. If n is None, then list all element counts.

```
>>> Counter('abcdeabcdabcaba').most_common(3)
[('a', 5), ('b', 4), ('c', 3)]
```

**subtract**(\*\*kws)

Like dict.update() but subtracts counts instead of replacing them. Counts can be reduced below zero. Both the inputs and outputs are allowed to contain zero and negative counts.

Source can be an iterable, a dictionary, or another Counter instance.

```
>>> c = Counter('which')
>>> c.subtract('witch')           # subtract elements from another iterable
>>> c.subtract(Counter('watch')) # subtract elements from another counter
>>> c['h']                       # 2 in which, minus 1 in witch, minus 1 in_
↪watch
0
>>> c['w']                       # 1 in which, minus 1 in witch, minus 1 in_
↪watch
-1
```

**update**(\*\*kws)

Like dict.update() but add counts instead of replacing them.

Source can be an iterable, a dictionary, or another Counter instance.

```
>>> c = Counter('which')
>>> c.update('witch')           # add elements from another iterable
>>> d = Counter('watch')
>>> c.update(d)                 # add elements from another counter
>>> c['h']                     # four 'h' in which, witch, and watch
4
```

xmodaler.utils.logger.colored(text: str, color: str | None = None, on\_color: str | None = None, attrs: Iterable[str] | None = None) → str

Colorize text.

**Available text colors:**

black, red, green, yellow, blue, magenta, cyan, white, light\_grey, dark\_grey, light\_red, light\_green, light\_yellow, light\_blue, light\_magenta, light\_cyan.

**Available text highlights:**

on\_black, on\_red, on\_green, on\_yellow, on\_blue, on\_magenta, on\_cyan, on\_white, on\_light\_grey, on\_dark\_grey, on\_light\_red, on\_light\_green, on\_light\_yellow, on\_light\_blue, on\_light\_magenta, on\_light\_cyan.

**Available attributes:**

bold, dark, underline, blink, reverse, concealed.

### Example

```
colored('Hello, World!', 'red', 'on_black', ['bold', 'blink']) colored('Hello, World!', 'green')
```

```
xmodaler.utils.logger.create_small_table(small_dict)
```

Create a small table using the keys of `small_dict` as headers. This is only suitable for small dictionaries.

#### Parameters

**small\_dict** (*dict*) – a result dictionary of only a few items.

#### Returns

the table as a string.

#### Return type

str

```
xmodaler.utils.logger.log_every_n(lvl, msg, n=1, *, name=None)
```

Log once per *n* times.

#### Parameters

- **lvl** (*int*) – the logging level
- **msg** (*str*) –
- **n** (*int*) –
- **name** (*str*) – name of the logger to use. Will use the caller's module by default.

```
xmodaler.utils.logger.log_every_n_seconds(lvl, msg, n=1, *, name=None)
```

Log no more than once per *n* seconds.

#### Parameters

- **lvl** (*int*) – the logging level
- **msg** (*str*) –
- **n** (*int*) –
- **name** (*str*) – name of the logger to use. Will use the caller's module by default.

```
xmodaler.utils.logger.log_first_n(lvl, msg, n=1, *, name=None, key='caller')
```

Log only for the first *n* times.

#### Parameters

- **lvl** (*int*) – the logging level
- **msg** (*str*) –
- **n** (*int*) –
- **name** (*str*) – name of the logger to use. Will use the caller's module by default.
- **key** (*str* or *tuple[str]*) – the string(s) can be one of “caller” or “message”, which defines how to identify duplicated logs. For example, if called with *n=1*, *key=“caller”*, this function will only log the first call from the same caller, regardless of the message content. If called with *n=1*, *key=“message”*, this function will log the same content only once, even if they are called from different places. If called with *n=1*, *key=(“caller”, “message”)*, this function will not log only if the same caller has logged the same message before.

```
xmodaler.utils.logger.setup_logger(output=None, distributed_rank=0, *, color=True, name='xmodaler',
                                   abbrev_name=None)
```

Initialize the xmodaler logger and set its verbosity level to “DEBUG”.

#### Parameters

- **output** (*str*) – a file name or a directory to save log. If None, will not save log file. If ends with “.txt” or “.log”, assumed to be a file name. Otherwise, logs will be saved to *output/log.txt*.
- **name** (*str*) – the root module name of this logger
- **abbrev\_name** (*str*) – an abbreviation of the module, to avoid long names in logs. Set to “” to not log the root module in logs. By default, will abbreviate “xmodaler” to “d2” and leave other modules unchanged.

#### Returns

a logger

#### Return type

logging.Logger

```
xmodaler.utils.logger.tabulate(tabular_data, headers=(), tablefmt='simple', floatfmt='g', intfmt="",
                               numalign='default', stralign='default', missingval="", showindex='default',
                               disable_numparse=False, colalign=None, maxcolwidths=None,
                               rowalign=None, maxheadercolwidths=None)
```

Format a fixed width table for pretty printing.

```
>>> print(tabulate([[1, 2.34], [-56, "8.999"], ["2", "10001"]]))
-----
 1         2.34
-56         8.999
 2    10001
-----
```

The first required argument (*tabular\_data*) can be a list-of-lists (or another iterable of iterables), a list of named tuples, a dictionary of iterables, an iterable of dictionaries, an iterable of dataclasses (Python 3.7+), a two-dimensional NumPy array, NumPy record array, or a Pandas’ dataframe.

### 3.13.1 Table headers

To print nice column headers, supply the second argument (*headers*):

- *headers* can be an explicit list of column headers
- if *headers* = “firstrow”, then the first row of data is used
- if *headers* = “keys”, then dictionary keys or column indices are used

Otherwise a headerless table is produced.

If the number of headers is less than the number of columns, they are supposed to be names of the last columns. This is consistent with the plain-text format of R and Pandas’ dataframes.

```
>>> print(tabulate([["sex", "age"], ["Alice", "F", 24], ["Bob", "M", 19]],
...               headers="firstrow"))
      sex    age
-----
```

(continues on next page)

(continued from previous page)

Alice	F	24
Bob	M	19

By default, pandas.DataFrame data have an additional column called row index. To add a similar column to all other types of data, use `showindex="always"` or `showindex=True`. To suppress row indices for all types of data, pass `showindex="never"` or `showindex=False`. To add a custom row index column, pass `showindex=some_iterable`.

```
>>> print(tabulate([["F",24],["M",19]], showindex="always"))
- - -
0  F  24
1  M  19
- - -
```

### 3.13.2 Column alignment

`tabulate` tries to detect column types automatically, and aligns the values properly. By default it aligns decimal points of the numbers (or flushes integer numbers to the right), and flushes everything else to the left. Possible column alignments (`numalign`, `stralign`) are: "right", "center", "left", "decimal" (only for `numalign`), and None (to disable alignment).

### 3.13.3 Table formats

`intfmt` is a format specification used for columns which contain numeric data without a decimal point. This can also be a list or tuple of format strings, one per column.

`floatfmt` is a format specification used for columns which contain numeric data with a decimal point. This can also be a list or tuple of format strings, one per column.

`None` values are replaced with a `missingval` string (like `floatfmt`, this can also be a list of values for different columns):

```
>>> print(tabulate([["spam", 1, None],
...                 ["eggs", 42, 3.14],
...                 ["other", None, 2.7]], missingval="?"))
-----
spam    1   ?
eggs    42  3.14
other   ?   2.7
-----
```

Various plain-text table formats (`tablefmt`) are supported: 'plain', 'simple', 'grid', 'pipe', 'orgtbl', 'rst', 'mediawiki', 'latex', 'latex\_raw', 'latex\_booktabs', 'latex\_longtable' and tsv. Variable `tabulate_formats` contains the list of currently supported formats.

"plain" format doesn't use any pseudographics to draw tables, it separates columns with a double space:

```
>>> print(tabulate([["spam", 41.9999], ["eggs", "451.0"]],
...                 ["strings", "numbers", "plain"]))
strings  numbers
spam     41.9999
eggs     451
```



```
>>> print(tabulate([["spam", 41.9999], ["eggs", "451.0"]], tablefmt="plain"))
spam  41.9999
eggs  451
```

“simple” format is like Pandoc `simple_tables`:

```
>>> print(tabulate([["spam", 41.9999], ["eggs", "451.0"]],
...                ["strings", "numbers"], "simple"))
strings      numbers
-----
spam         41.9999
eggs         451
```

```
>>> print(tabulate([["spam", 41.9999], ["eggs", "451.0"]], tablefmt="simple"))
----  -----
spam  41.9999
eggs  451
----  -----
```

“grid” is similar to tables produced by Emacs `table.el` package or Pandoc `grid_tables`:

```
>>> print(tabulate([["spam", 41.9999], ["eggs", "451.0"]],
...                ["strings", "numbers"], "grid"))
+-----+-----+
| strings | numbers |
+-----+-----+
| spam   | 41.9999 |
+-----+-----+
| eggs   | 451     |
+-----+-----+
```

```
>>> print(tabulate([["spam", 41.9999], ["eggs", "451.0"]], tablefmt="grid"))
+-----+-----+
| spam | 41.9999 |
+-----+-----+
| eggs | 451     |
+-----+-----+
```

“simple\_grid” draws a grid using single-line box-drawing characters:

```
>>> print(tabulate([["spam", 41.9999], ["eggs", "451.0"]],
...                ["strings", "numbers"], "simple_grid"))
┌── strings ─┐┌── numbers ─┐
│ spam      ││ 41.9999 │
├───┴───┘├───┴───┘
│ eggs      ││ 451     │
├───┴───┘├───┴───┘
```

“rounded\_grid” draws a grid using single-line box-drawing characters with rounded corners:

```
>>> print(tabulate([["spam", 41.9999], ["eggs", "451.0"]],
...                 ["strings", "numbers"], "rounded_grid"))
```

strings	numbers
spam	41.9999
eggs	451

“heavy\_grid” draws a grid using bold (thick) single-line box-drawing characters:

```
>>> print(tabulate([["spam", 41.9999], ["eggs", "451.0"]],
...                 ["strings", "numbers"], "heavy_grid"))
```

strings	numbers
spam	41.9999
eggs	451

“mixed\_grid” draws a grid using a mix of light (thin) and heavy (thick) lines box-drawing characters:

```
>>> print(tabulate([["spam", 41.9999], ["eggs", "451.0"]],
...                 ["strings", "numbers"], "mixed_grid"))
```

strings	numbers
spam	41.9999
eggs	451

“double\_grid” draws a grid using double-line box-drawing characters:

```
>>> print(tabulate([["spam", 41.9999], ["eggs", "451.0"]],
...                 ["strings", "numbers"], "double_grid"))
```

strings	numbers
spam	41.9999
eggs	451

“fancy\_grid” draws a grid using a mix of single and double-line box-drawing characters:

```
>>> print(tabulate([["spam", 41.9999], ["eggs", "451.0"]],
...                 ["strings", "numbers"], "fancy_grid"))
```

strings	numbers
---------	---------

(continues on next page)

(continued from previous page)

spam	41.9999
eggs	451

“outline” is the same as the “grid” format but doesn’t draw lines between rows:

```
>>> print(tabulate([["spam", 41.9999], ["eggs", "451.0"]],
...                 ["strings", "numbers"], "outline"))
```

strings	numbers
spam	41.9999
eggs	451

```
>>> print(tabulate([["spam", 41.9999], ["eggs", "451.0"]], tablefmt="outline"))
```

spam	41.9999
eggs	451

“simple\_outline” is the same as the “simple\_grid” format but doesn’t draw lines between rows:

```
>>> print(tabulate([["spam", 41.9999], ["eggs", "451.0"]],
...                 ["strings", "numbers"], "simple_outline"))
```

strings	numbers
spam	41.9999
eggs	451

“rounded\_outline” is the same as the “rounded\_grid” format but doesn’t draw lines between rows:

```
>>> print(tabulate([["spam", 41.9999], ["eggs", "451.0"]],
...                 ["strings", "numbers"], "rounded_outline"))
```

strings	numbers
spam	41.9999
eggs	451

“heavy\_outline” is the same as the “heavy\_grid” format but doesn’t draw lines between rows:

```
>>> print(tabulate([["spam", 41.9999], ["eggs", "451.0"]],
...                 ["strings", "numbers"], "heavy_outline"))
```

strings	numbers
spam	41.9999

(continues on next page)

(continued from previous page)

eggs	451	
------	-----	--

“mixed\_outline” is the same as the “mixed\_grid” format but doesn’t draw lines between rows:

```
>>> print(tabulate([["spam", 41.9999], ["eggs", "451.0"]],
...                 ["strings", "numbers"], "mixed_outline"))
```

strings	numbers	
spam	41.9999	
eggs	451	

“double\_outline” is the same as the “double\_grid” format but doesn’t draw lines between rows:

```
>>> print(tabulate([["spam", 41.9999], ["eggs", "451.0"]],
...                 ["strings", "numbers"], "double_outline"))
```

strings	numbers	
spam	41.9999	
eggs	451	

“fancy\_outline” is the same as the “fancy\_grid” format but doesn’t draw lines between rows:

```
>>> print(tabulate([["spam", 41.9999], ["eggs", "451.0"]],
...                 ["strings", "numbers"], "fancy_outline"))
```

strings	numbers	
spam	41.9999	
eggs	451	

“pipe” is like tables in PHP Markdown Extra extension or Pandoc pipe\_tables:

```
>>> print(tabulate([["spam", 41.9999], ["eggs", "451.0"]],
...                 ["strings", "numbers"], "pipe"))
```

strings	numbers	
:-----:-----:		
spam	41.9999	
eggs	451	

“presto” is like tables produce by the Presto CLI:

```
>>> print(tabulate([["spam", 41.9999], ["eggs", "451.0"]],
...                 ["strings", "numbers"], "presto"))
```

strings	numbers	
-----+-----		
spam	41.9999	
eggs	451	

```
>>> print(tabulate(["spam", 41.9999], ["eggs", "451.0"], tablefmt="pipe"))
|:-----|-----:|
| spam | 41.9999 |
| eggs | 451      |
```

“orgtbl” is like tables in Emacs org-mode and orgtbl-mode. They are slightly different from “pipe” format by not using colons to define column alignment, and using a “+” sign to indicate line intersections:

```
>>> print(tabulate(["spam", 41.9999], ["eggs", "451.0"],
...               ["strings", "numbers", "orgtbl"]))
| strings | numbers |
|-----+-----|
| spam    | 41.9999 |
| eggs    | 451     |
```

```
>>> print(tabulate(["spam", 41.9999], ["eggs", "451.0"], tablefmt="orgtbl"))
| spam | 41.9999 |
| eggs | 451     |
```

“rst” is like a simple table format from reStructuredText; please note that reStructuredText accepts also “grid” tables:

```
>>> print(tabulate(["spam", 41.9999], ["eggs", "451.0"],
...               ["strings", "numbers", "rst"]))
=====
strings      numbers
=====
spam         41.9999
eggs         451
=====
```

```
>>> print(tabulate(["spam", 41.9999], ["eggs", "451.0"], tablefmt="rst"))
=====
spam  41.9999
eggs  451
=====
```

“mediawiki” produces a table markup used in Wikipedia and on other MediaWiki-based sites:

```
>>> print(tabulate(["strings", "numbers"], ["spam", 41.9999], ["eggs", "451.0"],
...               headers="firstrow", tablefmt="mediawiki"))
{| class="wikitable" style="text-align: left;"
|+ <!-- caption -->
|-
! strings    !! align="right"|    numbers
|-
| spam       || align="right"|    41.9999
|-
| eggs       || align="right"|    451
|}
```

“html” produces HTML markup as an `html.escape`’d str with a `._repr_html_` method so that Jupyter Lab and Notebook display the HTML and a `.str` property so that the raw HTML remains accessible the `unsafehtml` table format can be used if an unescaped HTML format is required:

```
>>> print(tabulate(["strings", "numbers"], ["spam", 41.9999], ["eggs", "451.0"],
...               headers="firstrow", tablefmt="html"))
<table>
<thead>
<tr><th>strings </th><th style="text-align: right;"> numbers</th></tr>
</thead>
<tbody>
<tr><td>spam      </td><td style="text-align: right;"> 41.9999</td></tr>
<tr><td>eggs      </td><td style="text-align: right;"> 451      </td></tr>
</tbody>
</table>
```

“latex” produces a tabular environment of LaTeX document markup:

```
>>> print(tabulate(["spam", 41.9999], ["eggs", "451.0"], tablefmt="latex"))
\begin{tabular}{lr}
\hline
spam & 41.9999 \\
eggs & 451      \\
\hline
\end{tabular}
```

“latex\_raw” is similar to “latex”, but doesn’t escape special characters, such as backslash and underscore, so LaTeX commands may be embedded into cells’ values:

```
>>> print(tabulate(["spam$_9$", 41.9999], ["\\emph{eggs}", "451.0"], tablefmt=
↪ "latex_raw"))
\begin{tabular}{lr}
\hline
spam$_9$    & 41.9999 \\
\\emph{eggs} & 451      \\
\hline
\end{tabular}
```

“latex\_booktabs” produces a tabular environment of LaTeX document markup using the booktabs.sty package:

```
>>> print(tabulate(["spam", 41.9999], ["eggs", "451.0"], tablefmt="latex_booktabs
↪"))
\begin{tabular}{lr}
\toprule
spam & 41.9999 \\
eggs & 451      \\
\bottomrule
\end{tabular}
```

“latex\_longtable” produces a tabular environment that can stretch along multiple pages, using the longtable package for LaTeX.

```
>>> print(tabulate(["spam", 41.9999], ["eggs", "451.0"], tablefmt="latex_longtable
↪"))
\begin{longtable}{lr}
\hline
spam & 41.9999 \\
```

(continues on next page)

(continued from previous page)

```
eggs & 451      \\
\hline
\end{longtable}
```

### 3.13.4 Number parsing

By default, anything which can be parsed as a number is a number. This ensures numbers represented as strings are aligned properly. This can lead to weird results for particular strings such as specific git SHAs e.g. “42992e1” will be parsed into the number 429920 and aligned as such.

To completely disable number parsing (and alignment), use `disable_numparse=True`. For more fine grained control, a list column indices is used to disable number parsing only on those columns e.g. `disable_numparse=[0, 2]` would disable number parsing only on the first and third columns.

### 3.13.5 Column Widths and Auto Line Wrapping

Tabulate will, by default, set the width of each column to the length of the longest element in that column. However, in situations where fields are expected to reasonably be too long to look good as a single line, tabulate can help automate word wrapping long fields for you. Use the parameter `maxcolwidth` to provide a list of maximal column widths

```
>>> print(tabulate(
    [(['1', 'John Smith',
      'This is a rather long
      description that might look better if it is wrapped a bit'])],
    headers=("Issue Id", "Author", "Description"),
    maxcolwidths=[None, None, 30],
    tablefmt="grid"
))
```

Issue Id	Author	Description
1	John Smith	This is a rather long description that might look better if it is wrapped a bit

Header column width can be specified in a similar way using `maxheadercolwidth`

`xmodaler.utils.memory.retry_if_cuda_oom(func)`

Makes a function retry itself after encountering pytorch’s CUDA OOM error. It will first retry after calling `torch.cuda.empty_cache()`.

If that still fails, it will then retry by trying to convert inputs to CPUs. In this case, it expects the function to dispatch to CPU implementation. The return values may become CPU tensors as well and it’s user’s responsibility to convert it back to CUDA tensor if needed.

#### Parameters

**func** – a stateless callable that takes tensor-like objects as arguments

#### Returns

a callable which retries `func` if OOM is encountered.

Examples:

```
output = retry_if_cuda_oom(some_torch_function)(input1, input2)
# output may be on CPU even if inputs are on GPU
```

**Note:**

1. When converting inputs to CPU, it will only look at each argument and check if it has `.device` and `.to` for conversion. Nested structures of tensors are not supported.
  2. Since the function might be called more than once, it has to be stateless.
- 

**class** xmodaler.utils.registry.**Registry**(*name: str*)

Bases: Iterable[Tuple[str, Any]]

The registry that provides name -> object mapping, to support third-party users' custom modules.

To create a registry (e.g. a backbone registry):

```
BACKBONE_REGISTRY = Registry('BACKBONE')
```

To register an object:

```
@BACKBONE_REGISTRY.register()
class MyBackbone():
    ...
```

Or:

```
BACKBONE_REGISTRY.register(MyBackbone)
```

**\_\_init\_\_**(*name: str*) → None

**Parameters**

**name** (*str*) – the name of this registry

**get**(*name: str*) → Any

**register**(*obj: Optional[Any] = None*) → Any

Register the given object under the the name *obj.\_\_name\_\_*. Can be used as either a decorator or not. See docstring of this class for usage.

xmodaler.utils.registry.**locate**(*name: str*) → Any

Locate and return an object *x* using an input string *{x.\_\_module\_\_}.{x.\_\_qualname\_\_}*, such as “module.submodule.class\_name”.

Raise Exception if it cannot be found.

**class** xmodaler.utils.serialize.**PicklableWrapper**(*obj*)

Bases: object

Wrap an object to make it more picklable, note that it uses heavy weight serialization libraries that are slower than pickle. It's best to use it only on closures (which are usually not picklable).

This is a simplified version of [https://github.com/joblib/joblib/blob/master/joblib/externals/loky/cloudpickle\\_wrapper.py](https://github.com/joblib/joblib/blob/master/joblib/externals/loky/cloudpickle_wrapper.py)

**\_\_init\_\_**(*obj*)



## PYTHON MODULE INDEX

### X

`xmodaler.tokenization`, [95](#)



## Symbols

<code>__call__()</code> ( <i>xmodaler.datasets.ConceptualCaptionsDataset</i> method), 24	<code>__init__()</code> ( <i>xmodaler.datasets.Flickr30kDatasetForSingleStream</i> method), 26
<code>__call__()</code> ( <i>xmodaler.datasets.ConceptualCaptionsDatasetForSingleStream</i> method), 24	<code>__init__()</code> ( <i>xmodaler.datasets.Flickr30kDatasetForSingleStreamVal</i> method), 26
<code>__call__()</code> ( <i>xmodaler.datasets.Flickr30kDataset</i> method), 25	<code>__init__()</code> ( <i>xmodaler.datasets.MSCoCoBertDataset</i> method), 24
<code>__call__()</code> ( <i>xmodaler.datasets.Flickr30kDatasetForSingleStream</i> method), 25	<code>__init__()</code> ( <i>xmodaler.datasets.MSCoCoDataset</i> method), 23
<code>__call__()</code> ( <i>xmodaler.datasets.Flickr30kDatasetForSingleStreamVal</i> method), 26	<code>__init__()</code> ( <i>xmodaler.datasets.MSCoCoSampleByTxtDataset</i> method), 24
<code>__call__()</code> ( <i>xmodaler.datasets.MSCoCoBertDataset</i> method), 24	<code>__init__()</code> ( <i>xmodaler.datasets.MSRVTTDataset</i> method), 26
<code>__call__()</code> ( <i>xmodaler.datasets.MSCoCoDataset</i> method), 23	<code>__init__()</code> ( <i>xmodaler.datasets.MSVDDataset</i> method), 26
<code>__call__()</code> ( <i>xmodaler.datasets.MSCoCoSampleByTxtDataset</i> method), 24	<code>__init__()</code> ( <i>xmodaler.datasets.MapDataset</i> method), 23
<code>__call__()</code> ( <i>xmodaler.datasets.MSRVTTDataset</i> method), 26	<code>__init__()</code> ( <i>xmodaler.datasets.VCRDataset</i> method), 25
<code>__call__()</code> ( <i>xmodaler.datasets.MSVDDataset</i> method), 26	<code>__init__()</code> ( <i>xmodaler.datasets.VQADataset</i> method), 25
<code>__call__()</code> ( <i>xmodaler.datasets.VCRDataset</i> method), 25	<code>__init__()</code> ( <i>xmodaler.engine.AutogradProfiler</i> method), 35
<code>__call__()</code> ( <i>xmodaler.datasets.VQADataset</i> method), 25	<code>__init__()</code> ( <i>xmodaler.engine.CallbackHook</i> method), 32
<code>__getitem__()</code> ( <i>xmodaler.datasets.DatasetFromList</i> method), 23	<code>__init__()</code> ( <i>xmodaler.engine.DefaultTrainer</i> method), 28
<code>__getitem__()</code> ( <i>xmodaler.datasets.MapDataset</i> method), 23	<code>__init__()</code> ( <i>xmodaler.engine.EvalHook</i> method), 36
<code>__init__()</code> ( <i>xmodaler.checkpoint.PeriodicEpochCheckpoint</i> method), 15	<code>__init__()</code> ( <i>xmodaler.engine.IterationTimer</i> method), 32
<code>__init__()</code> ( <i>xmodaler.checkpoint.XmodalerCheckpoint</i> method), 16	<code>__init__()</code> ( <i>xmodaler.engine.LRScheduler</i> method), 34
<code>__init__()</code> ( <i>xmodaler.config.CfgNode</i> method), 18	<code>__init__()</code> ( <i>xmodaler.engine.ModelWeightsManipulating</i> method), 37
<code>__init__()</code> ( <i>xmodaler.datasets.ConceptualCaptionsDataset</i> method), 24	<code>__init__()</code> ( <i>xmodaler.engine.PeriodicCheckpoint</i> method), 33
<code>__init__()</code> ( <i>xmodaler.datasets.ConceptualCaptionsDatasetForSingleStream</i> method), 24	<code>__init__()</code> ( <i>xmodaler.engine.PeriodicWriter</i> method), 33
<code>__init__()</code> ( <i>xmodaler.datasets.DatasetFromList</i> method), 23	<code>__init__()</code> ( <i>xmodaler.engine.PreciseBN</i> method), 36
<code>__init__()</code> ( <i>xmodaler.datasets.Flickr30kDataset</i> method), 25	<code>__init__()</code> ( <i>xmodaler.engine.RLBeamTrainer</i> method), 40
	<code>__init__()</code> ( <i>xmodaler.engine.RLTrainer</i> method), 39
	<code>__init__()</code> ( <i>xmodaler.engine.RetrievalTrainer</i> method), 38

`__init__()` (`xmodaler.engine.SingleStreamRetrievalTrainer` method), 41  
`__init__()` (`xmodaler.engine.SingleStreamRetrievalTrainerHardNegatives` method), 42  
`__init__()` (`xmodaler.engine.TDENPretrainer` method), 43  
`__init__()` (`xmodaler.engine.TrainerBase` method), 31  
`__init__()` (`xmodaler.engine.VCRTrainer` method), 44  
`__init__()` (`xmodaler.evaluation.COCOEvaler` method), 46  
`__init__()` (`xmodaler.evaluation.RetrievalEvaler` method), 46  
`__init__()` (`xmodaler.evaluation.VCREvaler` method), 46  
`__init__()` (`xmodaler.evaluation.VQAEvaler` method), 46  
`__init__()` (`xmodaler.losses.BCEWithLogits` method), 47  
`__init__()` (`xmodaler.losses.BatchTriplet` method), 47  
`__init__()` (`xmodaler.losses.CrossEntropy` method), 48  
`__init__()` (`xmodaler.losses.LabelSmoothing` method), 48  
`__init__()` (`xmodaler.losses.PretrainLosses` method), 49  
`__init__()` (`xmodaler.losses.RewardCriterion` method), 49  
`__init__()` (`xmodaler.losses.Triplet` method), 48  
`__init__()` (`xmodaler.lr_scheduler.FixLR` method), 55  
`__init__()` (`xmodaler.lr_scheduler.MultiStepLR` method), 54  
`__init__()` (`xmodaler.lr_scheduler.NoamLR` method), 50  
`__init__()` (`xmodaler.lr_scheduler.StepLR` method), 50  
`__init__()` (`xmodaler.lr_scheduler.WarmupConstant` method), 51  
`__init__()` (`xmodaler.lr_scheduler.WarmupCosine` method), 52  
`__init__()` (`xmodaler.lr_scheduler.WarmupCosineWithHardRestarts` method), 53  
`__init__()` (`xmodaler.lr_scheduler.WarmupLinear` method), 52  
`__init__()` (`xmodaler.lr_scheduler.WarmupMultiStepLR` method), 54  
`__init__()` (`xmodaler.modeling.decode_strategy.decode_strategy` method), 57  
`__init__()` (`xmodaler.modeling.decoder.AttributeDecoder` method), 65  
`__init__()` (`xmodaler.modeling.decoder.DecoupleBertDecoder` method), 61  
`__init__()` (`xmodaler.modeling.decoder.MPLSTMDecoder` method), 60  
`__init__()` (`xmodaler.modeling.decoder.MeshedDecoder` method), 62  
`__init__()` (`xmodaler.modeling.decoder.SALSTMDecoder` method), 59  
`__init__()` (`xmodaler.modeling.decoder.TDConvEDDecoder` method), 64  
`__init__()` (`xmodaler.modeling.decoder.TransformerDecoder` method), 60  
`__init__()` (`xmodaler.modeling.decoder.UpDownDecoder` method), 58  
`__init__()` (`xmodaler.modeling.decoder.XLANDecoder` method), 63  
`__init__()` (`xmodaler.modeling.embedding.TDConvEDVisualBaseEmbedder` method), 68  
`__init__()` (`xmodaler.modeling.embedding.TokenBaseEmbedding` method), 66  
`__init__()` (`xmodaler.modeling.embedding.VisualBaseEmbedding` method), 66  
`__init__()` (`xmodaler.modeling.embedding.VisualIdentityEmbedding` method), 67  
`__init__()` (`xmodaler.modeling.embedding.position_embedding.NNEmbedder` method), 69  
`__init__()` (`xmodaler.modeling.embedding.position_embedding.SinusoidalEmbedder` method), 69  
`__init__()` (`xmodaler.modeling.encoder.Encoder` method), 70  
`__init__()` (`xmodaler.modeling.encoder.GCNEncoder` method), 74  
`__init__()` (`xmodaler.modeling.encoder.LowRankBilinearEncoder` method), 76  
`__init__()` (`xmodaler.modeling.encoder.MemoryAugmentedEncoder` method), 75  
`__init__()` (`xmodaler.modeling.encoder.SingleStreamBertEncoder` method), 72  
`__init__()` (`xmodaler.modeling.encoder.TDConvEDEncoder` method), 76  
`__init__()` (`xmodaler.modeling.encoder.TransformerEncoder` method), 72  
`__init__()` (`xmodaler.modeling.encoder.TwoStreamBertEncoder` method), 73  
`__init__()` (`xmodaler.modeling.encoder.UpDownEncoder` method), 71  
`__init__()` (`xmodaler.modeling.layers.attention_pooler.AttentionPooler` method), 77  
`__init__()` (`xmodaler.modeling.layers.base_attention.BaseAttention` method), 78  
`__init__()` (`xmodaler.modeling.layers.bert.BertAttention` method), 79  
`__init__()` (`xmodaler.modeling.layers.bert.BertCrossAttention` method), 80  
`__init__()` (`xmodaler.modeling.layers.bert.BertGenerationLayer` method), 81  
`__init__()` (`xmodaler.modeling.layers.bert.BertIntermediate` method), 79  
`__init__()` (`xmodaler.modeling.layers.bert.BertLayer` method), 81  
`__init__()` (`xmodaler.modeling.layers.bert.BertOutput` method), 81

method), 79

`__init__` () (xmodaler.modeling.layers.bert.BertPooler method), 82

`__init__` () (xmodaler.modeling.layers.bert.BertPredictionHead method), 82

`__init__` () (xmodaler.modeling.layers.bert.BertSelfAttention method), 78

`__init__` () (xmodaler.modeling.layers.bert.BertSelfOutput method), 78

`__init__` () (xmodaler.modeling.layers.bert.BertUnderstandingLayer method), 81

`__init__` () (xmodaler.modeling.layers.bert.BertXAttention method), 80

`__init__` () (xmodaler.modeling.layers.lowrank\_bilinear\_layer method), 84

`__init__` () (xmodaler.modeling.layers.lowrank\_bilinear\_layer method), 84

`__init__` () (xmodaler.modeling.layers.multihead\_attention.MultiHeadAttention method), 83

`__init__` () (xmodaler.modeling.layers.multihead\_attention.MultiHeadAttention method), 83

`__init__` () (xmodaler.modeling.layers.positionwise\_feedforward method), 83

`__init__` () (xmodaler.modeling.layers.scattention.SCAAttention method), 85

`__init__` () (xmodaler.modeling.layers.tdconvd\_layers.ShiftedConvLayer method), 86

`__init__` () (xmodaler.modeling.layers.tdconvd\_layers.SoftAttention method), 87

`__init__` () (xmodaler.modeling.layers.tdconvd\_layers.TemporalDeforableBlock method), 86

`__init__` () (xmodaler.modeling.layers.tdconvd\_layers.TemporalDeforableBlock method), 85

`__init__` () (xmodaler.modeling.meta\_arch.TDENBiTransformer method), 88

`__init__` () (xmodaler.modeling.meta\_arch.TDENCaptioner method), 89

`__init__` () (xmodaler.modeling.meta\_arch.TDENPretrain method), 88

`__init__` () (xmodaler.modeling.meta\_arch.TransformerEncoder method), 88

`__init__` () (xmodaler.modeling.meta\_arch.UniterForMMUnderstanding method), 89

`__init__` () (xmodaler.modeling.meta\_arch.UniterPretrain method), 89

`__init__` () (xmodaler.modeling.meta\_arch.base\_enc\_dec.BaseEncoder method), 87

`__init__` () (xmodaler.modeling.predictor.BasePredictor method), 90

`__init__` () (xmodaler.modeling.predictor.BertIsMatchedPredictor method), 91

`__init__` () (xmodaler.modeling.predictor.BertPredictionHead method), 90

`__init__` () (xmodaler.modeling.predictor.BertVisualFeatureRegression method), 91

`__init__` () (xmodaler.modeling.predictor.BertVisualPredictionHead method), 90

`__init__` () (xmodaler.modeling.predictor.MultiModalPredictor method), 92

`__init__` () (xmodaler.modeling.predictor.MultiModalSimilarity method), 93

`__init__` () (xmodaler.modeling.predictor.SingleStreamMultiModalPredictor method), 92

`__init__` () (xmodaler.optim.Adagrad method), 93

`__init__` () (xmodaler.optim.Adam method), 93

`__init__` () (xmodaler.optim.AdamW method), 93

`__init__` () (xmodaler.optim.Adamax method), 94

`__init__` () (xmodaler.optim.BertAdam method), 94

`__init__` () (xmodaler.optim.RAdam method), 94

`__init__` () (xmodaler.optim.RMSprop method), 94

`__init__` () (xmodaler.optim.SGD method), 94

`__init__` () (xmodaler.scorer.BaseScorer method), 95

`__init__` () (xmodaler.scorer.BertTokenizedScorer method), 95

`__init__` () (xmodaler.scorer.Cider method), 95

`__init__` () (xmodaler.scorer.CIDEr method), 95

`__init__` () (xmodaler.scorer.CommonMetricPrinter method), 101

`__init__` () (xmodaler.scorer.EventStorage method), 101

`__init__` () (xmodaler.scorer.JSONWriter method), 100

`__init__` () (xmodaler.scorer.TensorboardXWriter method), 100

`__init__` () (xmodaler.scorer.PathHandler method), 103

`__init__` () (xmodaler.scorer.Counter method), 104

`__init__` () (xmodaler.scorer.Registry method), 116

`__init__` () (xmodaler.scorer.PicklableWrapper method), 116

`__init__` () (xmodaler.datasets.DatasetFromList method), 23

`__init__` () (xmodaler.datasets.MapDataset method), 23

`__abc_impl` (xmodaler.modeling.decode\_strategy.BeamSearcher attribute), 56

`__abc_impl` (xmodaler.modeling.decode\_strategy.GreedyDecoder attribute), 56

`__abc_impl` (xmodaler.modeling.decode\_strategy.decode\_strategy.DecodeStrategy attribute), 57

`__abc_impl` (xmodaler.modeling.decoder.AttributeDecoder attribute), 65

`__abc_impl` (xmodaler.modeling.decoder.DecoupleBertDecoder attribute), 61

`__abc_impl` (xmodaler.modeling.decoder.MPLSTMDecoder attribute), 60

`__abc_impl` (xmodaler.modeling.decoder.MeshedDecoder attribute), 62

`_abc_impl (xmodaler.modeling.decoder.SALSTMDecoder attribute), 59`  
`_abc_impl (xmodaler.modeling.decoder.TransformerDecoder attribute), 61`  
`_abc_impl (xmodaler.modeling.decoder.UpDownDecoder attribute), 58`  
`_abc_impl (xmodaler.modeling.decoder.XLANDecoder attribute), 63`  
`_backward_hooks (xmodaler.modeling.decode_strategy.BeamSearcher attribute), 56`  
`_backward_hooks (xmodaler.modeling.decode_strategy.GreedyDecoder attribute), 56`  
`_backward_hooks (xmodaler.modeling.decode_strategy.decode_strategy.DecodeStrategy attribute), 57`  
`_backward_hooks (xmodaler.modeling.decoder.AttributeDecoder attribute), 65`  
`_backward_hooks (xmodaler.modeling.decoder.DecoupleBertDecoder attribute), 61`  
`_backward_hooks (xmodaler.modeling.decoder.MPLSTMDecoder attribute), 60`  
`_backward_hooks (xmodaler.modeling.decoder.MeshedDecoder attribute), 62`  
`_backward_hooks (xmodaler.modeling.decoder.SALSTMDecoder attribute), 59`  
`_backward_hooks (xmodaler.modeling.decoder.TDConvEDDecoder attribute), 64`  
`_backward_hooks (xmodaler.modeling.decoder.TransformerDecoder attribute), 61`  
`_backward_hooks (xmodaler.modeling.decoder.UpDownDecoder attribute), 58`  
`_backward_hooks (xmodaler.modeling.decoder.XLANDecoder attribute), 63`  
`_backward_hooks (xmodaler.modeling.embedding.TDConvEDVisualBaseEmbedding attribute), 68`  
`_backward_hooks (xmodaler.modeling.embedding.TokenBaseEmbedding attribute), 66`  
`_backward_hooks (xmodaler.modeling.embedding.VisualBaseEmbedding attribute), 67`  
`_backward_hooks (xmodaler.modeling.embedding.VisualIdentityEmbedding attribute), 67`  
`_backward_hooks (xmodaler.modeling.embedding.position_embedding.NNEmbedding attribute), 69`  
`_backward_hooks (xmodaler.modeling.embedding.position_embedding.SinusoidEmbedding attribute), 69`  
`_backward_hooks (xmodaler.modeling.encoder.Encoder attribute), 70`  
`_backward_hooks (xmodaler.modeling.encoder.GCNEncoder attribute), 74`  
`_backward_hooks (xmodaler.modeling.encoder.LowRankBilinearEncoder attribute), 76`  
`_backward_hooks (xmodaler.modeling.encoder.MemoryAugmentedEncoder attribute), 75`  
`_backward_hooks (xmodaler.modeling.encoder.SingleStreamBertEncoder attribute), 72`  
`_backward_hooks (xmodaler.modeling.encoder.TDConvEDEncoder attribute), 76`  
`_backward_hooks (xmodaler.modeling.encoder.TransformerEncoder attribute), 72`  
`_backward_hooks (xmodaler.modeling.encoder.TwoStreamBertEncoder attribute), 73`  
`_backward_hooks (xmodaler.modeling.encoder.UpDownEncoder attribute), 71`  
`_buffers (xmodaler.modeling.decode_strategy.BeamSearcher attribute), 56`  
`_buffers (xmodaler.modeling.decode_strategy.GreedyDecoder attribute), 56`  
`_buffers (xmodaler.modeling.decode_strategy.decode_strategy.DecodeStrategy attribute), 57`  
`_buffers (xmodaler.modeling.decoder.AttributeDecoder attribute), 65`  
`_buffers (xmodaler.modeling.decoder.DecoupleBertDecoder attribute), 61`  
`_buffers (xmodaler.modeling.decoder.MPLSTMDecoder attribute), 60`  
`_buffers (xmodaler.modeling.decoder.MeshedDecoder attribute), 62`  
`_buffers (xmodaler.modeling.decoder.SALSTMDecoder attribute), 59`  
`_buffers (xmodaler.modeling.decoder.TDConvEDDecoder attribute), 64`  
`_buffers (xmodaler.modeling.decoder.TransformerDecoder attribute), 61`  
`_buffers (xmodaler.modeling.decoder.UpDownDecoder attribute), 58`  
`_buffers (xmodaler.modeling.decoder.XLANDecoder attribute), 63`  
`_buffers (xmodaler.modeling.embedding.TDConvEDVisualBaseEmbedding attribute), 68`  
`_buffers (xmodaler.modeling.embedding.TokenBaseEmbedding attribute), 66`  
`_buffers (xmodaler.modeling.embedding.VisualBaseEmbedding attribute), 67`  
`_buffers (xmodaler.modeling.embedding.VisualIdentityEmbedding attribute), 67`  
`_buffers (xmodaler.modeling.embedding.position_embedding.NNEmbedding attribute), 69`  
`_buffers (xmodaler.modeling.embedding.position_embedding.SinusoidEmbedding attribute), 69`  
`_buffers (xmodaler.modeling.encoder.Encoder attribute), 70`  
`_buffers (xmodaler.modeling.encoder.GCNEncoder attribute), 74`  
`_buffers (xmodaler.modeling.encoder.LowRankBilinearEncoder attribute), 76`  
`_buffers (xmodaler.modeling.encoder.MemoryAugmentedEncoder attribute), 75`  
`_buffers (xmodaler.modeling.encoder.SingleStreamBertEncoder attribute), 72`



---

`_buffers` (`xmodaler.modeling.encoder.TDConvEDEncoder.forward_hooks` (`xmodaler.modeling.embedding.TDConvEDVisualBaseE`  
`attribute`), 76 `attribute`), 68  
`_buffers` (`xmodaler.modeling.encoder.TransformerEncoder.forward_hooks` (`xmodaler.modeling.embedding.TokenBaseEmbedding`  
`attribute`), 72 `attribute`), 66  
`_buffers` (`xmodaler.modeling.encoder.TwoStreamBertEncoder.forward_hooks` (`xmodaler.modeling.embedding.VisualBaseEmbedding`  
`attribute`), 73 `attribute`), 67  
`_buffers` (`xmodaler.modeling.encoder.UpDownEncoder.forward_hooks` (`xmodaler.modeling.embedding.VisualIdentityEmbedding`  
`attribute`), 71 `attribute`), 67  
`_clear_decoding_buffer()` `_forward_hooks` (`xmodaler.modeling.embedding.position_embedding.NN`  
(`xmodaler.modeling.decoder.TDConvEDDecoder` `attribute`), 69  
`method`), 64 `_forward_hooks` (`xmodaler.modeling.embedding.position_embedding.Sinu`  
`attribute`), 69  
`_convert_ndarray_to_tensor()` `_forward_hooks` (`xmodaler.modeling.encoder.Encoder`  
(`xmodaler.checkpoint.XmodalerCheckpoint` `attribute`), 70  
`method`), 16 `_forward_hooks` (`xmodaler.modeling.encoder.GCNEncoder`  
`attribute`), 74  
`_create_config_tree_from_dict()` `_forward_hooks` (`xmodaler.modeling.encoder.LowRankBilinearEncoder`  
(`xmodaler.config.CfgNode` `class` `method`), `attribute`), 76  
18 `_forward_hooks` (`xmodaler.modeling.encoder.MemoryAugmentedEncoder`  
`attribute`), 75  
`_decode_cfg_value()` (`xmodaler.config.CfgNode` `class` `method`), 18 `_forward_hooks` (`xmodaler.modeling.encoder.SingleStreamBertEncoder`  
`attribute`), 72  
`_do_eval()` (`xmodaler.engine.EvalHook` `method`), 36 `_forward_hooks` (`xmodaler.modeling.encoder.TDConvEDEncoder`  
`attribute`), 76  
`_expand_state()` (`xmodaler.modeling.decode_strategy.BeamSearch` `attribute`), 72  
`method`), 56 `_forward_hooks` (`xmodaler.modeling.encoder.TransformerEncoder`  
`attribute`), 72  
`_forward()` (`xmodaler.modeling.decode_strategy.BeamSearch` `attribute`), 73  
`method`), 56 `_forward_hooks` (`xmodaler.modeling.encoder.TwoStreamBertEncoder`  
`attribute`), 73  
`_forward()` (`xmodaler.modeling.decode_strategy.GreedyDecoder` `attribute`), 71  
`method`), 56 `_forward_hooks` (`xmodaler.modeling.encoder.UpDownEncoder`  
`attribute`), 71  
`_forward()` (`xmodaler.modeling.decode_strategy.decode_strategy` `attribute`), 62  
`method`), 57 `_forward_pre_hooks` (`xmodaler.modeling.decode_strategy.BeamSearcher`  
`attribute`), 56  
`_forward()` (`xmodaler.modeling.embedding.TokenBaseEmbedding` `attribute`), 56  
`method`), 66 `_forward_pre_hooks` (`xmodaler.modeling.decode_strategy.GreedyDecoder`  
`attribute`), 56  
`_forward_hooks` (`xmodaler.modeling.decode_strategy.BeamSearch` `attribute`), 56  
`attribute`), 56 `_forward_pre_hooks` (`xmodaler.modeling.decode_strategy.GreedyDecoder`  
`attribute`), 56  
`_forward_hooks` (`xmodaler.modeling.decode_strategy.decode_strategy` `attribute`), 57  
`attribute`), 57 `_forward_pre_hooks` (`xmodaler.modeling.decode_strategy.decode_strategy`  
`attribute`), 57  
`_forward_hooks` (`xmodaler.modeling.decoder.AttributeDecoder` `attribute`), 65  
`attribute`), 65 `_forward_pre_hooks` (`xmodaler.modeling.decoder.AttributeDecoder`  
`attribute`), 65  
`_forward_hooks` (`xmodaler.modeling.decoder.DecoupleBertDecoder` `attribute`), 61  
`attribute`), 62 `_forward_pre_hooks` (`xmodaler.modeling.decoder.DecoupleBertDecoder`  
`attribute`), 62  
`_forward_hooks` (`xmodaler.modeling.decoder.MPLSTMDecoder` `attribute`), 60  
`attribute`), 60 `_forward_pre_hooks` (`xmodaler.modeling.decoder.MPLSTMDecoder`  
`attribute`), 60  
`_forward_hooks` (`xmodaler.modeling.decoder.MeshedDecoder` `attribute`), 62  
`attribute`), 62 `_forward_pre_hooks` (`xmodaler.modeling.decoder.MeshedDecoder`  
`attribute`), 62  
`_forward_hooks` (`xmodaler.modeling.decoder.SALSTMDecoder` `attribute`), 59  
`attribute`), 59 `_forward_pre_hooks` (`xmodaler.modeling.decoder.SALSTMDecoder`  
`attribute`), 59  
`_forward_hooks` (`xmodaler.modeling.decoder.TDConvEDDecoder` `attribute`), 64  
`attribute`), 64 `_forward_pre_hooks` (`xmodaler.modeling.decoder.TDConvEDDecoder`  
`attribute`), 64  
`_forward_hooks` (`xmodaler.modeling.decoder.TransformerDecoder` `attribute`), 61  
`attribute`), 61 `_forward_pre_hooks` (`xmodaler.modeling.decoder.TransformerDecoder`  
`attribute`), 61  
`_forward_hooks` (`xmodaler.modeling.decoder.UpDownDecoder` `attribute`), 58  
`attribute`), 58 `_forward_pre_hooks` (`xmodaler.modeling.decoder.UpDownDecoder`  
`attribute`), 58  
`_forward_hooks` (`xmodaler.modeling.decoder.XLANDecoder` `attribute`), 63  
`attribute`), 63 `_forward_pre_hooks` (`xmodaler.modeling.decoder.XLANDecoder`  
`attribute`), 63

`_forward_pre_hooks(xmodaler.modeling.embedding.TDConvEDVisualBaseEmbedding attribute), 68`  
`_forward_pre_hooks(xmodaler.modeling.embedding.TokenBaseEmbedding attribute), 66`  
`_forward_pre_hooks(xmodaler.modeling.embedding.VisualBaseEmbedding attribute), 67`  
`_forward_pre_hooks(xmodaler.modeling.embedding.VisualIdentityEmbedding attribute), 67`  
`_forward_pre_hooks(xmodaler.modeling.embedding.position_embedding.Embedding attribute), 69`  
`_forward_pre_hooks(xmodaler.modeling.embedding.position_embedding.SinusoidalEmbedding attribute), 69`  
`_forward_pre_hooks(xmodaler.modeling.encoder.Encoder attribute), 70`  
`_forward_pre_hooks(xmodaler.modeling.encoder.GCNEncoder attribute), 74`  
`_forward_pre_hooks(xmodaler.modeling.encoder.LowRankBilinearEncoder attribute), 76`  
`_forward_pre_hooks(xmodaler.modeling.encoder.MemoryAugmentedEncoder attribute), 75`  
`_forward_pre_hooks(xmodaler.modeling.encoder.SingleStreamBertEncoder attribute), 72`  
`_forward_pre_hooks(xmodaler.modeling.encoder.TDConvEDEncoder attribute), 77`  
`_forward_pre_hooks(xmodaler.modeling.encoder.TransformerEncoder attribute), 72`  
`_forward_pre_hooks(xmodaler.modeling.encoder.TwoStreamBertEncoder attribute), 73`  
`_forward_pre_hooks(xmodaler.modeling.encoder.UpDownEncoder attribute), 71`  
`_get_closed_form_lr(xmodaler.lr_scheduler.MultiStepLR method), 54`  
`_get_closed_form_lr(xmodaler.lr_scheduler.StepLR method), 50`  
`_get_global_feat(xmodaler.modeling.encoder.MemoryAugmentedEncoder method), 75`  
`_immutable(xmodaler.config.CfgNode method), 19`  
`_init_decoding_buffer(xmodaler.modeling.decoder.TDConvEDDecoder method), 64`  
`_initial_step(xmodaler.lr_scheduler.FixLR method), 55`  
`_initial_step(xmodaler.lr_scheduler.MultiStepLR method), 54`  
`_initial_step(xmodaler.lr_scheduler.NoamLR method), 50`  
`_initial_step(xmodaler.lr_scheduler.StepLR method), 50`  
`_initial_step(xmodaler.lr_scheduler.WarmupConstant method), 51`  
`_initial_step(xmodaler.lr_scheduler.WarmupCosine method), 52`  
`_is_full_backward_hook(xmodaler.modeling.decode_strategy.BeamSearcher attribute), 57`  
`_is_full_backward_hook(xmodaler.modeling.decode_strategy.GreedyDecoder attribute), 56`  
`_is_full_backward_hook(xmodaler.modeling.decode_strategy.decode_strategy.DecodeStrategy attribute), 57`  
`_is_full_backward_hook(xmodaler.modeling.decoder.AttributeDecoder attribute), 65`  
`_is_full_backward_hook(xmodaler.modeling.decoder.DecoupleBertDecoder attribute), 62`  
`_is_full_backward_hook(xmodaler.modeling.decoder.MPLSTMDDecoder attribute), 60`  
`_is_full_backward_hook(xmodaler.modeling.decoder.MeshedDecoder attribute), 62`  
`_is_full_backward_hook(xmodaler.modeling.decoder.SALSTMDDecoder attribute), 59`  
`_is_full_backward_hook(xmodaler.modeling.decoder.TDConvEDDecoder attribute), 64`  
`_is_full_backward_hook(xmodaler.modeling.decoder.TransformerDecoder attribute), 61`  
`_is_full_backward_hook(xmodaler.modeling.decoder.UpDownDecoder attribute), 58`  
`_is_full_backward_hook(xmodaler.modeling.decoder.XLANDecoder attribute), 63`  
`_is_full_backward_hook(xmodaler.modeling.embedding.TDConvEDVisualBaseEmbedding attribute), 68`  
`_is_full_backward_hook(xmodaler.modeling.embedding.TokenBaseEmbedding attribute), 66`  
`_is_full_backward_hook(xmodaler.modeling.embedding.VisualBaseEmbedding attribute), 67`  
`_is_full_backward_hook(xmodaler.modeling.embedding.VisualIdentityEmbedding attribute), 67`



<code>_is_full_backward_hook</code>	( <code>xmodaler.modeling.decoder.AttributeDecoder</code> ( <code>xmodaler.modeling.embedding.position_embedding.NNEmbedding</code> <code>attribute</code> ), 69)	<code>_load_state_dict_post_hooks</code>
<code>_is_full_backward_hook</code>	( <code>xmodaler.modeling.decoder.DecoupleBertDecoder</code> ( <code>xmodaler.modeling.embedding.position_embedding.SinusoidEmbedding</code> <code>attribute</code> ), 69)	<code>_load_state_dict_post_hooks</code>
<code>_is_full_backward_hook</code>	( <code>xmodaler.modeling.decoder.MPLSTMDecoder</code> ( <code>xmodaler.modeling.encoder.Encoder</code> <code>at-</code> <code>tribute</code> ), 70)	<code>_load_state_dict_post_hooks</code>
<code>_is_full_backward_hook</code>	( <code>xmodaler.modeling.decoder.MeshedDecoder</code> ( <code>xmodaler.modeling.encoder.GCNEncoder</code> <code>attribute</code> ), 74)	<code>_load_state_dict_post_hooks</code>
<code>_is_full_backward_hook</code>	( <code>xmodaler.modeling.decoder.SALSTMDecoder</code> ( <code>xmodaler.modeling.encoder.LowRankBilinearEncoder</code> <code>attribute</code> ), 76)	<code>_load_state_dict_post_hooks</code>
<code>_is_full_backward_hook</code>	( <code>xmodaler.modeling.decoder.TDConvEDDecoder</code> ( <code>xmodaler.modeling.encoder.MemoryAugmentedEncoder</code> <code>attribute</code> ), 75)	<code>_load_state_dict_post_hooks</code>
<code>_is_full_backward_hook</code>	( <code>xmodaler.modeling.decoder.TransformerDecoder</code> ( <code>xmodaler.modeling.encoder.SingleStreamBertEncoder</code> <code>attribute</code> ), 72)	<code>_load_state_dict_post_hooks</code>
<code>_is_full_backward_hook</code>	( <code>xmodaler.modeling.decoder.UpDownDecoder</code> ( <code>xmodaler.modeling.encoder.TDConvEDEncoder</code> <code>attribute</code> ), 77)	<code>_load_state_dict_post_hooks</code>
<code>_is_full_backward_hook</code>	( <code>xmodaler.modeling.decoder.XLANDecoder</code> ( <code>xmodaler.modeling.encoder.TransformerEncoder</code> <code>attribute</code> ), 72)	<code>_load_state_dict_post_hooks</code>
<code>_is_full_backward_hook</code>	( <code>xmodaler.modeling.embedding.TDConvEDVisualBaseEmbedding</code> ( <code>xmodaler.modeling.encoder.TwoStreamBertEncoder</code> <code>attribute</code> ), 73)	<code>_load_state_dict_post_hooks</code>
<code>_is_full_backward_hook</code>	( <code>xmodaler.modeling.embedding.TokenBaseEmbedding</code> ( <code>xmodaler.modeling.encoder.UpDownEncoder</code> <code>attribute</code> ), 71)	<code>_load_state_dict_post_hooks</code>
<code>_load_cfg_from_file()</code>	( <code>xmodaler.config.CfgNode</code> <code>class method</code> ), 19)	<code>_load_state_dict_post_hooks</code>
<code>_load_cfg_from_yaml_str()</code>	( <code>xmodaler.config.CfgNode</code> <code>class</code> <code>method</code> ), 19)	<code>_load_state_dict_post_hooks</code>
<code>_load_cfg_py_source()</code>	( <code>xmodaler.config.CfgNode</code> <code>class</code> <code>method</code> ), 19)	<code>_load_state_dict_post_hooks</code>
<code>_load_file()</code>	( <code>xmodaler.checkpoint.XmodalerCheckpoint</code> <code>method</code> ), 16)	<code>_load_state_dict_post_hooks</code>
<code>_load_model()</code>	( <code>xmodaler.checkpoint.XmodalerCheckpoint</code> <code>method</code> ), 16)	<code>_load_state_dict_post_hooks</code>
<code>_load_state_dict_post_hooks</code>	( <code>xmodaler.modeling.decode_strategy.BeamSearcher</code> <code>attribute</code> ), 57)	<code>_load_state_dict_post_hooks</code>
<code>_load_state_dict_post_hooks</code>	( <code>xmodaler.modeling.decode_strategy.GreedyDecoder</code> <code>attribute</code> ), 56)	<code>_load_state_dict_post_hooks</code>
<code>_load_state_dict_post_hooks</code>	( <code>xmodaler.modeling.decode_strategy.decode_strategy.DecodeStrategy</code> <code>attribute</code> ), 57)	<code>_load_state_dict_post_hooks</code>
<code>_load_state_dict_post_hooks</code>	( <code>xmodaler.modeling.encoder.LowRankBilinearEncoder</code> <code>attribute</code> ), 76)	<code>_load_state_dict_post_hooks</code>

<code>(xmodaler.modeling.encoder.MemoryAugmentedEncoder attribute), 75</code>	<code>(xmodaler.modeling.embedding.TDConvEDVisualBaseEmbedding attribute), 68</code>
<code>_load_state_dict_post_hooks</code>	<code>_load_state_dict_pre_hooks</code>
<code>(xmodaler.modeling.encoder.SingleStreamBertEncoder attribute), 73</code>	<code>(xmodaler.modeling.embedding.TokenBaseEmbedding attribute), 66</code>
<code>_load_state_dict_post_hooks</code>	<code>_load_state_dict_pre_hooks</code>
<code>(xmodaler.modeling.encoder.TDConvEDEncoder attribute), 77</code>	<code>(xmodaler.modeling.embedding.VisualBaseEmbedding attribute), 67</code>
<code>_load_state_dict_post_hooks</code>	<code>_load_state_dict_pre_hooks</code>
<code>(xmodaler.modeling.encoder.TransformerEncoder attribute), 72</code>	<code>(xmodaler.modeling.embedding.VisualIdentityEmbedding attribute), 67</code>
<code>_load_state_dict_post_hooks</code>	<code>_load_state_dict_pre_hooks</code>
<code>(xmodaler.modeling.encoder.TwoStreamBertEncoder attribute), 73</code>	<code>(xmodaler.modeling.embedding.position_embedding.NNEmbedding attribute), 70</code>
<code>_load_state_dict_post_hooks</code>	<code>_load_state_dict_pre_hooks</code>
<code>(xmodaler.modeling.encoder.UpDownEncoder attribute), 71</code>	<code>(xmodaler.modeling.embedding.position_embedding.SinusoidEncoder attribute), 69</code>
<code>_load_state_dict_pre_hooks</code>	<code>_load_state_dict_pre_hooks</code>
<code>(xmodaler.modeling.decode_strategy.BeamSearcher attribute), 57</code>	<code>(xmodaler.modeling.encoder.Encoder attribute), 70</code>
<code>_load_state_dict_pre_hooks</code>	<code>_load_state_dict_pre_hooks</code>
<code>(xmodaler.modeling.decode_strategy.GreedyDecoder attribute), 56</code>	<code>(xmodaler.modeling.encoder.GCNEncoder attribute), 74</code>
<code>_load_state_dict_pre_hooks</code>	<code>_load_state_dict_pre_hooks</code>
<code>(xmodaler.modeling.decode_strategy.decode_strategy.DecodeStrategy attribute), 57</code>	<code>(xmodaler.modeling.encoder.LowRankBilinearEncoder attribute), 76</code>
<code>_load_state_dict_pre_hooks</code>	<code>_load_state_dict_pre_hooks</code>
<code>(xmodaler.modeling.decoder.AttributeDecoder attribute), 65</code>	<code>(xmodaler.modeling.encoder.MemoryAugmentedEncoder attribute), 75</code>
<code>_load_state_dict_pre_hooks</code>	<code>_load_state_dict_pre_hooks</code>
<code>(xmodaler.modeling.decoder.DecoupleBertDecoder attribute), 62</code>	<code>(xmodaler.modeling.encoder.SingleStreamBertEncoder attribute), 73</code>
<code>_load_state_dict_pre_hooks</code>	<code>_load_state_dict_pre_hooks</code>
<code>(xmodaler.modeling.decoder.MPLSTMDecoder attribute), 60</code>	<code>(xmodaler.modeling.encoder.TDConvEDEncoder attribute), 77</code>
<code>_load_state_dict_pre_hooks</code>	<code>_load_state_dict_pre_hooks</code>
<code>(xmodaler.modeling.decoder.MeshedDecoder attribute), 62</code>	<code>(xmodaler.modeling.encoder.TransformerEncoder attribute), 72</code>
<code>_load_state_dict_pre_hooks</code>	<code>_load_state_dict_pre_hooks</code>
<code>(xmodaler.modeling.decoder.SALSTMDecoder attribute), 59</code>	<code>(xmodaler.modeling.encoder.TwoStreamBertEncoder attribute), 73</code>
<code>_load_state_dict_pre_hooks</code>	<code>_load_state_dict_pre_hooks</code>
<code>(xmodaler.modeling.decoder.TDConvEDDecoder attribute), 64</code>	<code>(xmodaler.modeling.encoder.UpDownEncoder attribute), 71</code>
<code>_load_state_dict_pre_hooks</code>	<code>_log_incompatible_keys()</code>
<code>(xmodaler.modeling.decoder.TransformerDecoder attribute), 61</code>	<code>(xmodaler.checkpoint.XmodalerCheckpointInterface method), 16</code>
<code>_load_state_dict_pre_hooks</code>	<code>_modules (xmodaler.modeling.decode_strategy.BeamSearcher attribute), 57</code>
<code>(xmodaler.modeling.decoder.UpDownDecoder attribute), 58</code>	<code>_modules (xmodaler.modeling.decode_strategy.GreedyDecoder attribute), 56</code>
<code>_load_state_dict_pre_hooks</code>	<code>_modules (xmodaler.modeling.decode_strategy.decode_strategy.DecodeStrategy attribute), 57</code>
<code>(xmodaler.modeling.decoder.XLANDecoder attribute), 63</code>	
<code>_load_state_dict_pre_hooks</code>	<code>_modules (xmodaler.modeling.decoder.AttributeDecoder attribute), 65</code>

attribute), 65  
 \_modules (xmodaler.modeling.decoder.DecoupleBertDecoder attribute), 62  
 \_modules (xmodaler.modeling.decoder.MPLSTMDecoder attribute), 60  
 \_modules (xmodaler.modeling.decoder.MeshedDecoder attribute), 62  
 \_modules (xmodaler.modeling.decoder.SALSTMDecoder attribute), 59  
 \_modules (xmodaler.modeling.decoder.TDConvEDDecoder attribute), 64  
 \_modules (xmodaler.modeling.decoder.TransformerDecoder attribute), 61  
 \_modules (xmodaler.modeling.decoder.UpDownDecoder attribute), 58  
 \_modules (xmodaler.modeling.decoder.XLANDecoder attribute), 63  
 \_modules (xmodaler.modeling.embedding.TDConvEDVisualBaseEmbedding attribute), 68  
 \_modules (xmodaler.modeling.embedding.TokenBaseEmbedding attribute), 66  
 \_modules (xmodaler.modeling.embedding.VisualBaseEmbedding attribute), 67  
 \_modules (xmodaler.modeling.embedding.VisualIdentityEmbedding attribute), 67  
 \_modules (xmodaler.modeling.embedding.position\_embedding.NNEmbedding attribute), 70  
 \_modules (xmodaler.modeling.embedding.position\_embedding.SinusoidEncoding attribute), 69  
 \_modules (xmodaler.modeling.encoder.Encoder attribute), 70  
 \_modules (xmodaler.modeling.encoder.GCNEncoder attribute), 74  
 \_modules (xmodaler.modeling.encoder.LowRankBilinearEncoder attribute), 76  
 \_modules (xmodaler.modeling.encoder.MemoryAugmentedEncoder attribute), 75  
 \_modules (xmodaler.modeling.encoder.SingleStreamBertEncoder attribute), 73  
 \_modules (xmodaler.modeling.encoder.TDConvEDEncoder attribute), 77  
 \_modules (xmodaler.modeling.encoder.TransformerEncoder attribute), 72  
 \_modules (xmodaler.modeling.encoder.TwoStreamBertEncoder attribute), 73  
 \_modules (xmodaler.modeling.encoder.UpDownEncoder attribute), 71  
 \_non\_persistent\_buffers\_set  
 (xmodaler.modeling.decode\_strategy.BeamSearcher attribute), 57  
 \_non\_persistent\_buffers\_set  
 (xmodaler.modeling.decode\_strategy.GreedyDecoder attribute), 56  
 \_non\_persistent\_buffers\_set  
 (xmodaler.modeling.decode\_strategy.decode\_strategy.DecodeStrategy attribute), 57  
 \_non\_persistent\_buffers\_set  
 (xmodaler.modeling.decoder.AttributeDecoder attribute), 65  
 \_non\_persistent\_buffers\_set  
 (xmodaler.modeling.decoder.DecoupleBertDecoder attribute), 62  
 \_non\_persistent\_buffers\_set  
 (xmodaler.modeling.decoder.MPLSTMDecoder attribute), 60  
 \_non\_persistent\_buffers\_set  
 (xmodaler.modeling.decoder.MeshedDecoder attribute), 63  
 \_non\_persistent\_buffers\_set  
 (xmodaler.modeling.decoder.SALSTMDecoder attribute), 59  
 \_non\_persistent\_buffers\_set  
 (xmodaler.modeling.decoder.TDConvEDDecoder attribute), 64  
 \_non\_persistent\_buffers\_set  
 (xmodaler.modeling.decoder.TransformerDecoder attribute), 61  
 \_non\_persistent\_buffers\_set  
 (xmodaler.modeling.decoder.UpDownDecoder attribute), 58  
 \_non\_persistent\_buffers\_set  
 (xmodaler.modeling.decoder.XLANDecoder attribute), 63  
 \_non\_persistent\_buffers\_set  
 (xmodaler.modeling.embedding.TDConvEDVisualBaseEmbedding attribute), 68  
 \_non\_persistent\_buffers\_set  
 (xmodaler.modeling.embedding.TokenBaseEmbedding attribute), 66  
 \_non\_persistent\_buffers\_set  
 (xmodaler.modeling.embedding.VisualBaseEmbedding attribute), 67  
 \_non\_persistent\_buffers\_set  
 (xmodaler.modeling.embedding.VisualIdentityEmbedding attribute), 67  
 \_non\_persistent\_buffers\_set  
 (xmodaler.modeling.embedding.position\_embedding.NNEmbedding attribute), 70  
 \_non\_persistent\_buffers\_set  
 (xmodaler.modeling.embedding.position\_embedding.SinusoidEncoding attribute), 69  
 \_non\_persistent\_buffers\_set  
 (xmodaler.modeling.encoder.Encoder attribute), 70  
 \_non\_persistent\_buffers\_set  
 (xmodaler.modeling.encoder.GCNEncoder attribute), 74  
 \_non\_persistent\_buffers\_set  
 (xmodaler.modeling.encoder.LowRankBilinearEncoder attribute), 76  
 \_non\_persistent\_buffers\_set  
 (xmodaler.modeling.encoder.MemoryAugmentedEncoder attribute), 75  
 \_non\_persistent\_buffers\_set  
 (xmodaler.modeling.encoder.SingleStreamBertEncoder attribute), 73  
 \_non\_persistent\_buffers\_set  
 (xmodaler.modeling.encoder.TDConvEDEncoder attribute), 77  
 \_non\_persistent\_buffers\_set  
 (xmodaler.modeling.encoder.TransformerEncoder attribute), 72  
 \_non\_persistent\_buffers\_set  
 (xmodaler.modeling.encoder.TwoStreamBertEncoder attribute), 73  
 \_non\_persistent\_buffers\_set  
 (xmodaler.modeling.encoder.UpDownEncoder attribute), 71  
 \_non\_persistent\_buffers\_set  
 (xmodaler.modeling.encoder.Encoder attribute), 70  
 \_non\_persistent\_buffers\_set  
 (xmodaler.modeling.encoder.GCNEncoder attribute), 74  
 \_non\_persistent\_buffers\_set



`_state_dict_hooks(xmodaler.modeling.embedding.VisualizerConfig)` (xmodaler.losses.CrossEntropy class attribute), 68  
`_state_dict_hooks(xmodaler.modeling.embedding.positional_embedding)` (xmodaler.losses.CrossEntropy class attribute), 70  
`_state_dict_hooks(xmodaler.modeling.embedding.positional_embedding)` (xmodaler.losses.CrossEntropy class attribute), 69  
`_state_dict_hooks(xmodaler.modeling.encoder.Encoder)` (xmodaler.losses.RewardCriterion class attribute), 70  
`_state_dict_hooks(xmodaler.modeling.encoder.GCNEncoder)` (xmodaler.losses.Triplet class method), 48  
`_state_dict_hooks(xmodaler.modeling.encoder.LowRankBilinearEncoder)` (xmodaler.modeling.decoder.AttributeDecoder class method), 65  
`_state_dict_hooks(xmodaler.modeling.encoder.MemoryAugmentedEncoder)` (xmodaler.modeling.decoder.DecoupleBertDecoder class method), 62  
`_state_dict_hooks(xmodaler.modeling.encoder.SingleStreamBertEncoder)` (xmodaler.modeling.decoder.MeshedDecoder class method), 63  
`_state_dict_hooks(xmodaler.modeling.encoder.TDConvEDEncoder)` (xmodaler.modeling.decoder.MPLSTMDecoder class method), 60  
`_state_dict_hooks(xmodaler.modeling.encoder.TransformerEncoder)` (xmodaler.modeling.decoder.SALSTMDecoder class method), 59  
`_state_dict_hooks(xmodaler.modeling.encoder.TwoStreamBertEncoder)` (xmodaler.modeling.decoder.TDConvEDDecoder class method), 64  
`_state_dict_hooks(xmodaler.modeling.encoder.UpDownEncoder)` (xmodaler.modeling.decoder.TransformerDecoder class method), 61  
`_write_metrics()` (xmodaler.engine.DefaultTrainer) (xmodaler.modeling.decoder.UpDownDecoder class method), 29  
`_write_metrics()` (xmodaler.engine.RLBeamTrainer) (xmodaler.modeling.decoder.XLANDecoder class method), 40  
`_write_metrics()` (xmodaler.engine.RLTrainer) (xmodaler.modeling.encoder.Encoder class method), 39  
`_write_metrics()` (xmodaler.engine.RetrievalTrainer) (xmodaler.modeling.encoder.GCNEncoder class method), 38  
`_write_metrics()` (xmodaler.engine.SingleStreamRetrievalTrainer) (xmodaler.modeling.encoder.LowRankBilinearEncoder class method), 41  
`_write_metrics()` (xmodaler.engine.SingleStreamRetrievalTrainer) (xmodaler.modeling.encoder.MemoryAugmentedEncoder class method), 42  
`_write_metrics()` (xmodaler.engine.TDENPretrainer) (xmodaler.modeling.encoder.SingleStreamBertEncoder class method), 43  
`_write_metrics()` (xmodaler.engine.VCRTrainer) (xmodaler.modeling.encoder.TDConvEDEncoder class method), 44  
`add_config()` (xmodaler.modeling.encoder.TransformerEncoder class method), 72  
`add_config()` (xmodaler.modeling.encoder.TwoStreamBertEncoder class method), 74  
`add_config()` (xmodaler.modeling.encoder.UpDownEncoder class method), 71  
`add_config()` (xmodaler.modeling.meta\_arch.base\_enc\_dec.BaseEncoder class method), 87  
`add_config()` (xmodaler.modeling.meta\_arch.TDENPretrain class method), 89  
`add_config()` (xmodaler.modeling.meta\_arch.UniterPretrain class method), 89  
`add_config()` (xmodaler.modeling.predictor.BasePredictor class method), 90

## A

Adagrad (class in xmodaler.optim), 93  
 Adam (class in xmodaler.optim), 93  
 Adamax (class in xmodaler.optim), 94  
 AdamW (class in xmodaler.optim), 93  
 add\_checkpointable()  
     (xmodaler.checkpoint.XmodalerCheckpointTrainer method), 16  
 add\_config() (in module xmodaler.modeling.meta\_arch), 87  
 add\_config() (xmodaler.losses.BCEWithLogits class method), 47



`add_config()` (`xmodaler.modeling.predictor.BertIsMatchedPredictor` class method), 91  
`add_config()` (`xmodaler.modeling.predictor.BertPredictionHead` class method), 90  
`add_config()` (`xmodaler.modeling.predictor.BertVisualFeaturePredictor` class method), 91  
`add_config()` (`xmodaler.modeling.predictor.BertVisualPractitioner` class method), 91  
`add_config()` (`xmodaler.modeling.predictor.MultiModalPairwisePredictor` class method), 92  
`add_config()` (`xmodaler.modeling.predictor.MultiModalSimilarityPredictor` class method), 93  
`add_config()` (`xmodaler.modeling.predictor.SingleStreamRetrievalPredictor` class method), 92  
`add_decoder_config()` (in module `xmodaler.modeling.decoder`), 58  
`add_encoder_config()` (in module `xmodaler.modeling.encoder`), 70  
`add_predictor_config()` (in module `xmodaler.modeling.predictor`), 90  
`add_special_tokens_sentences_pair()` (`xmodaler.tokenization.BertTokenizer` method), 96  
`add_special_tokens_single_sentence()` (`xmodaler.tokenization.BertTokenizer` method), 96  
`after_step()` (`xmodaler.engine.AutogradProfiler` method), 35  
`after_step()` (`xmodaler.engine.CallbackHook` method), 32  
`after_step()` (`xmodaler.engine.DefaultTrainer` method), 29  
`after_step()` (`xmodaler.engine.EvalHook` method), 36  
`after_step()` (`xmodaler.engine.HookBase` method), 30  
`after_step()` (`xmodaler.engine.IterationTimer` method), 32  
`after_step()` (`xmodaler.engine.LRScheduler` method), 34  
`after_step()` (`xmodaler.engine.ModelWeightsManipulatingHook` method), 37  
`after_step()` (`xmodaler.engine.PeriodicCheckpoint` method), 34  
`after_step()` (`xmodaler.engine.PeriodicWriter` method), 33  
`after_step()` (`xmodaler.engine.PreciseBN` method), 37  
`after_step()` (`xmodaler.engine.RetrievalTrainer` method), 38  
`after_step()` (`xmodaler.engine.RLBeamTrainer` method), 40  
`after_step()` (`xmodaler.engine.RLTrainer` method), 39  
`after_step()` (`xmodaler.engine.SingleStreamRetrievalTrainer` method), 41  
`after_step()` (`xmodaler.engine.SingleStreamRetrievalTrainerHardNegatives` method), 42  
`after_step()` (`xmodaler.engine.TDENPretrainer` method), 43  
`after_step()` (`xmodaler.engine.TrainerBase` method), 31  
`after_step()` (`xmodaler.engine.VCRTrainer` method), 45  
`all_gather()` (in module `xmodaler.utils.comm`), 98  
`all_gather_list()` (in module `xmodaler.utils.distributed`), 98  
`AttentionPooler` (class in `xmodaler.modeling.layers.attention_pooler`), 77  
`AttributeDecoder` (class in `xmodaler.modeling.decoder`), 65  
`auto_has_data_maskers()` (`xmodaler.engine.DefaultTrainer` static

method), 29

auto\_scale\_workers() (xmodaler.engine.RetrievalTrainer static method), 38

auto\_scale\_workers() (xmodaler.engine.RLBeamTrainer static method), 40

auto\_scale\_workers() (xmodaler.engine.RLTrainer static method), 39

auto\_scale\_workers() (xmodaler.engine.SingleStreamRetrievalTrainer static method), 41

auto\_scale\_workers() (xmodaler.engine.SingleStreamRetrievalTrainerHardNegative static method), 42

auto\_scale\_workers() (xmodaler.engine.TDENPretrainer static method), 43

auto\_scale\_workers() (xmodaler.engine.VCRTrainer static method), 45

AutogradProfiler (class in xmodaler.engine), 35

## B

BaseAttention (class in xmodaler.modeling.layers.base\_attention), 78

BaseEncoderDecoder (class in xmodaler.modeling.meta\_arch.base\_enc\_dec), 87

BasePredictor (class in xmodaler.modeling.predictor), 90

BaseScorer (class in xmodaler.scorer), 95

BatchTriplet (class in xmodaler.losses), 47

BCEWithLogits (class in xmodaler.losses), 47

BeamSearcher (class in xmodaler.modeling.decode\_strategy), 56

before\_step() (xmodaler.engine.AutogradProfiler method), 35

before\_step() (xmodaler.engine.CallbackHook method), 32

before\_step() (xmodaler.engine.DefaultTrainer method), 29

before\_step() (xmodaler.engine.EvalHook method), 36

before\_step() (xmodaler.engine.HookBase method), 30

before\_step() (xmodaler.engine.IterationTimer method), 32

before\_step() (xmodaler.engine.LRScheduler method), 34

before\_step() (xmodaler.engine.ModelWeightsManipulating method), 37

before\_step() (xmodaler.engine.PeriodicCheckpointing method), 34

before\_step() (xmodaler.engine.PeriodicWriter method), 33

before\_step() (xmodaler.engine.PreciseBN method), 37

before\_step() (xmodaler.engine.RetrievalTrainer method), 38

before\_step() (xmodaler.engine.RLBeamTrainer method), 40

before\_step() (xmodaler.engine.RLTrainer method), 39

before\_step() (xmodaler.engine.SingleStreamRetrievalTrainer method), 41

before\_step() (xmodaler.engine.SingleStreamRetrievalTrainerHardNegative method), 42

before\_step() (xmodaler.engine.TDENPretrainer method), 44

before\_train() (xmodaler.engine.AutogradProfiler method), 35

before\_train() (xmodaler.engine.CallbackHook method), 32

before\_train() (xmodaler.engine.DefaultTrainer method), 29

before\_train() (xmodaler.engine.EvalHook method), 36

before\_train() (xmodaler.engine.HookBase method), 30

before\_train() (xmodaler.engine.IterationTimer method), 32

before\_train() (xmodaler.engine.LRScheduler method), 35

before\_train() (xmodaler.engine.ModelWeightsManipulating method), 37

before\_train() (xmodaler.engine.PeriodicCheckpointing method), 34

before\_train() (xmodaler.engine.PeriodicWriter method), 33

before\_train() (xmodaler.engine.PreciseBN method), 37

before\_train() (xmodaler.engine.RetrievalTrainer method), 38

before\_train() (xmodaler.engine.RLBeamTrainer method), 40

before\_train() (xmodaler.engine.RLTrainer method), 39

before\_train() (xmodaler.engine.SingleStreamRetrievalTrainer method), 41

before\_train() (xmodaler.engine.SingleStreamRetrievalTrainerHardNegative method), 42

before\_train() (xmodaler.engine.TDENPretrainer method), 44

before_train()	(xmodaler.engine.TrainerBase method), 31	build_engine() (in module xmodaler.engine), 27
before_train()	(xmodaler.engine.VCRTrainer method), 45	build_evaluation() (in module xmodaler.evaluation), 46
BertAdam (class in xmodaler.optim), 94		build_greedy_decoder() (in module xmodaler.modeling.decode_strategy), 56
BertAttention (class in xmodaler.modeling.layers.bert), 79	in	build_hooks() (xmodaler.engine.DefaultTrainer method), 29
BertCrossAttention (class in xmodaler.modeling.layers.bert), 80	in	build_hooks() (xmodaler.engine.RetrievalTrainer method), 38
BertGenerationLayer (class in xmodaler.modeling.layers.bert), 81	in	build_hooks() (xmodaler.engine.RLBeamTrainer method), 40
BertIntermediate (class in xmodaler.modeling.layers.bert), 79	in	build_hooks() (xmodaler.engine.RLTrainer method), 39
BertIsMatchedPredictor (class in xmodaler.modeling.predictor), 91	in	build_hooks() (xmodaler.engine.SingleStreamRetrievalTrainer method), 41
BertLayer (class in xmodaler.modeling.layers.bert), 81		build_hooks() (xmodaler.engine.SingleStreamRetrievalTrainerHardNeg method), 42
BertOutput (class in xmodaler.modeling.layers.bert), 79		build_hooks() (xmodaler.engine.TDENPretrainer method), 44
BertPooler (class in xmodaler.modeling.layers.bert), 82	in	build_hooks() (xmodaler.engine.VCRTrainer method), 45
BertPredictionHead (class in xmodaler.modeling.predictor), 90	in	build_losses() (in module xmodaler.losses), 50
BertPredictionHeadTransform (class in xmodaler.modeling.layers.bert), 82	in	build_losses() (xmodaler.engine.DefaultTrainer class method), 29
BertSelfAttention (class in xmodaler.modeling.layers.bert), 78	in	build_losses() (xmodaler.engine.RetrievalTrainer class method), 38
BertSelfOutput (class in xmodaler.modeling.layers.bert), 78	in	build_losses() (xmodaler.engine.RLBeamTrainer class method), 40
BertTokenizedScorer (class in xmodaler.scorer), 95		build_losses() (xmodaler.engine.RLTrainer class method), 39
BertTokenizer (class in xmodaler.tokenization), 95	in	build_losses() (xmodaler.engine.SingleStreamRetrievalTrainer class method), 41
BertUnderstandingLayer (class in xmodaler.modeling.layers.bert), 81	in	build_losses() (xmodaler.engine.SingleStreamRetrievalTrainerHardNeg class method), 42
BertVisualFeatureRegressionHead (class in xmodaler.modeling.predictor), 91	in	build_losses() (xmodaler.engine.TDENPretrainer class method), 44
BertVisualPredictionHead (class in xmodaler.modeling.predictor), 90	in	build_losses() (xmodaler.engine.VCRTrainer class method), 45
BertXAttention (class in xmodaler.modeling.layers.bert), 80	in	build_lr_scheduler() (in module xmodaler.lr_scheduler), 50
bind_or_init_weights()	(xmodaler.modeling.meta_arch.base_enc_dec.BaseEncoderDecoder method), 87	build_lr_scheduler() (xmodaler.engine.DefaultTrainer class method), 29
bind_or_init_weights()	(xmodaler.modeling.meta_arch.UniterForMMUnderstanding method), 89	build_lr_scheduler() (xmodaler.engine.RetrievalTrainer class method), 38
boxes_to_locs_feats()	(in module xmodaler.functional), 46	build_lr_scheduler() (xmodaler.engine.RLBeamTrainer class method), 40
build_beam_searcher()	(in module xmodaler.modeling.decode_strategy), 56	build_lr_scheduler() (xmodaler.engine.RLTrainer class method), 39
build_dataset_mapper()	(in module xmodaler.datasets), 27	build_lr_scheduler() (xmodaler.engine.SingleStreamRetrievalTrainer class method), 41
build_decoder()	(in module xmodaler.modeling.decoder), 58	
build_embeddings()	(in module xmodaler.modeling.embedding), 66	
build_encoder()	(in module xmodaler.modeling.encoder), 70	



`build_lr_scheduler()` (`xmodaler.engine.DefaultTrainer` class method), 29  
`build_lr_scheduler()` (`xmodaler.engine.SingleStreamRetrievalTrainerHardNegatives` class method), 42  
`build_lr_scheduler()` (`xmodaler.engine.TDENPretrainer` class method), 44  
`build_lr_scheduler()` (`xmodaler.engine.VCRTrainer` class method), 45  
`build_model()` (in module `xmodaler.modeling.meta_arch`), 87  
`build_model()` (`xmodaler.engine.DefaultTrainer` class method), 29  
`build_model()` (`xmodaler.engine.RetrievalTrainer` class method), 38  
`build_model()` (`xmodaler.engine.RLBeamTrainer` class method), 40  
`build_model()` (`xmodaler.engine.RLTrainer` class method), 39  
`build_model()` (`xmodaler.engine.SingleStreamRetrievalTrainer` class method), 41  
`build_model()` (`xmodaler.engine.SingleStreamRetrievalTrainerHardNegatives` class method), 45  
`build_model()` (`xmodaler.engine.TDENPretrainer` class method), 44  
`build_model()` (`xmodaler.engine.VCRTrainer` class method), 45  
`build_optimizer()` (in module `xmodaler.optim`), 93  
`build_optimizer()` (`xmodaler.engine.DefaultTrainer` class method), 29  
`build_optimizer()` (`xmodaler.engine.RetrievalTrainer` class method), 38  
`build_optimizer()` (`xmodaler.engine.RLBeamTrainer` class method), 40  
`build_optimizer()` (`xmodaler.engine.RLTrainer` class method), 39  
`build_optimizer()` (`xmodaler.engine.SingleStreamRetrievalTrainer` class method), 41  
`build_optimizer()` (`xmodaler.engine.SingleStreamRetrievalTrainerHardNegatives` class method), 43  
`build_optimizer()` (`xmodaler.engine.TDENPretrainer` class method), 44  
`build_optimizer()` (`xmodaler.engine.VCRTrainer` class method), 45  
`build_predictor()` (in module `xmodaler.modeling.predictor`), 90  
`build_predictor_with_name()` (in module `xmodaler.modeling.predictor`), 90  
`build_rl_losses()` (in module `xmodaler.losses`), 50  
`build_scorer()` (in module `xmodaler.scorer`), 95  
`build_scorer()` (`xmodaler.engine.RLBeamTrainer` class method), 40  
`build_scorer()` (`xmodaler.engine.RLTrainer` class method), 39  
`build_test_loader()` (`xmodaler.engine.DefaultTrainer` class method), 29  
`build_test_loader()` (`xmodaler.engine.RetrievalTrainer` class method), 38  
`build_test_loader()` (`xmodaler.engine.RLBeamTrainer` class method), 40  
`build_test_loader()` (`xmodaler.engine.RLTrainer` class method), 39  
`build_test_loader()` (`xmodaler.engine.SingleStreamRetrievalTrainer` class method), 41  
`build_test_loader()` (`xmodaler.engine.SingleStreamRetrievalTrainerHardNegatives` class method), 43  
`build_test_loader()` (`xmodaler.engine.TDENPretrainer` class method), 44  
`build_test_loader()` (`xmodaler.engine.VCRTrainer` class method), 45  
`build_train_loader()` (`xmodaler.engine.DefaultTrainer` class method), 29  
`build_train_loader()` (`xmodaler.engine.RetrievalTrainer` class method), 38  
`build_train_loader()` (`xmodaler.engine.RLBeamTrainer` class method), 40  
`build_train_loader()` (`xmodaler.engine.RLTrainer` class method), 39  
`build_train_loader()` (`xmodaler.engine.SingleStreamRetrievalTrainer` class method), 41  
`build_train_loader()` (`xmodaler.engine.SingleStreamRetrievalTrainerHardNegatives` class method), 43  
`build_train_loader()` (`xmodaler.engine.TDENPretrainer` class method), 44  
`build_train_loader()` (`xmodaler.engine.VCRTrainer` class method), 45  
`build_v_predictor()` (in module `xmodaler.modeling.predictor`), 90  
`build_val_loader()` (`xmodaler.engine.DefaultTrainer` class method), 29  
`build_val_loader()` (`xmodaler.engine.RetrievalTrainer` class method), 38  
`build_val_loader()` (`xmodaler.engine.RLBeamTrainer` class method), 40  
`build_val_loader()` (`xmodaler.engine.RLTrainer` class method), 39  
`build_val_loader()` (`xmodaler.engine.SingleStreamRetrievalTrainer` class method), 41

- class method*), 42
  - `build_val_loader()` (*xmodaler.engine.SingleStreamRetrievalTrainer* *class method*), 43
  - `build_val_loader()` (*xmodaler.engine.TDENPretrainer* *class method*), 44
  - `build_val_loader()` (*xmodaler.engine.VCRTrainer* *class method*), 45
  - `build_writers()` (*xmodaler.engine.DefaultTrainer* *method*), 29
  - `build_writers()` (*xmodaler.engine.RetrievalTrainer* *method*), 38
  - `build_writers()` (*xmodaler.engine.RLBeamTrainer* *method*), 40
  - `build_writers()` (*xmodaler.engine.RLTrainer* *method*), 39
  - `build_writers()` (*xmodaler.engine.SingleStreamRetrievalTrainer* *method*), 42
  - `build_writers()` (*xmodaler.engine.SingleStreamRetrievalTrainerHandler* *method*), 43
  - `build_writers()` (*xmodaler.engine.TDENPretrainer* *method*), 44
  - `build_writers()` (*xmodaler.engine.VCRTrainer* *method*), 45
  - `build_xmodaler_train_loader()` (*in module xmodaler.datasets*), 26
  - `build_xmodaler_valtest_loader()` (*in module xmodaler.datasets*), 26
- ## C
- `CallbackHook` (*class in xmodaler.engine*), 31
  - `caption_to_mask_tokens()` (*in module xmodaler.functional*), 46
  - `cfg` (*xmodaler.engine.DefaultTrainer* *attribute*), 28
  - `CfgNode` (*class in xmodaler.config*), 18
  - `checkpointer` (*xmodaler.engine.DefaultTrainer* *attribute*), 28
  - `Cider` (*class in xmodaler.scorer*), 95
  - `clear()` (*xmodaler.config.CfgNode* *method*), 19
  - `clear_histograms()` (*xmodaler.utils.events.EventStorage* *method*), 101
  - `clear_images()` (*xmodaler.utils.events.EventStorage* *method*), 101
  - `clip_inputs()` (*xmodaler.engine.SingleStreamRetrievalTrainer* *method*), 43
  - `clip_t_inputs()` (*in module xmodaler.functional*), 46
  - `clip_v_inputs()` (*in module xmodaler.functional*), 46
  - `clone()` (*xmodaler.config.CfgNode* *method*), 19
  - `close()` (*xmodaler.utils.events.JSONWriter* *method*), 100
  - `close()` (*xmodaler.utils.events.TensorboardXWriter* *method*), 101
  - `COCOEvaler` (*class in xmodaler.evaluation*), 46
  - `collect_env_info()` (*in module xmodaler.utils.collect\_env*), 97
  - `colored()` (*in module xmodaler.utils.logger*), 105
  - `colormaps` (*in module xmodaler.utils.colormap*), 97
  - `CommonMetricPrinter` (*class in xmodaler.utils.events*), 101
  - `compute_score()` (*xmodaler.scorer.Cider* *method*), 95
  - `ConceptualCaptionsDataset` (*class in xmodaler.datasets*), 24
  - `ConceptualCaptionsDatasetForSingleStream` (*class in xmodaler.datasets*), 24
  - `configurable()` (*in module xmodaler.config*), 22
  - `convert_tokens_to_string()` (*xmodaler.tokenization.BertTokenizer* *method*), 96
  - `copy()` (*xmodaler.config.CfgNode* *method*), 19
  - `copy()` (*xmodaler.utils.logger.Counter* *method*), 104
  - `Counter` (*class in xmodaler.utils.logger*), 103
  - `create_small_table()` (*in module xmodaler.utils.logger*), 106
  - `CrossEntropy` (*class in xmodaler.losses*), 48
- ## D
- `DatasetFromList` (*class in xmodaler.datasets*), 23
  - `decode_beam_search()` (*xmodaler.modeling.meta\_arch.base\_enc\_dec.BaseEncoderDecoder* *method*), 87
  - `decode_sequence()` (*in module xmodaler.functional*), 46
  - `decode_sequence_bert()` (*in module xmodaler.functional*), 46
  - `DecodeStrategy` (*class in xmodaler.modeling.decode\_strategy.decode\_strategy*), 57
  - `DecoupleBertDecoder` (*class in xmodaler.modeling.decoder*), 61
  - `default_argument_parser()` (*in module xmodaler.engine*), 27
  - `default_setup()` (*in module xmodaler.engine*), 27
  - `default_writers()` (*in module xmodaler.engine*), 27
  - `DefaultTrainer` (*class in xmodaler.engine*), 28
  - `defrost()` (*xmodaler.config.CfgNode* *method*), 19
  - `DEPRECATED_KEYS` (*xmodaler.config.CfgNode* *attribute*), 18
  - `dict_has_nans()` (*in module xmodaler.functional*), 46
  - `dict_to_cuda()` (*in module xmodaler.functional*), 46
  - `downgrade_config()` (*in module xmodaler.config*), 21
  - `dump()` (*xmodaler.config.CfgNode* *method*), 19
- ## E
- `elements()` (*xmodaler.utils.logger.Counter* *method*), 104
  - `Encoder` (*class in xmodaler.modeling.encoder*), 70
  - `eval()` (*xmodaler.evaluation.COCOEvaler* *method*), 46
  - `eval()` (*xmodaler.evaluation.RetrievalEvaler* *method*), 46

`eval()` (*xmodaler.evaluation.VCREvaler* method), 46  
`eval()` (*xmodaler.evaluation.VQAEvaler* method), 46  
`EvalHook` (class in *xmodaler.engine*), 36  
`EventStorage` (class in *xmodaler.utils.events*), 101  
`expand_tensor()` (in module *xmodaler.functional*), 46

## F

`FixLR` (class in *xmodaler.lr\_scheduler*), 55  
`flat_list_of_lists()` (in module *xmodaler.functional*), 47  
`Flickr30kDataset` (class in *xmodaler.datasets*), 25  
`Flickr30kDatasetForSingleStream` (class in *xmodaler.datasets*), 25  
`Flickr30kDatasetForSingleStreamVal` (class in *xmodaler.datasets*), 26  
`forward()` (*xmodaler.losses.BatchTriplet* method), 47  
`forward()` (*xmodaler.losses.BCEWithLogits* method), 47  
`forward()` (*xmodaler.losses.CrossEntropy* method), 48  
`Forward()` (*xmodaler.losses.LabelSmoothing* method), 48  
`forward()` (*xmodaler.losses.LabelSmoothing* method), 49  
`forward()` (*xmodaler.losses.PretrainLosses* method), 49  
`forward()` (*xmodaler.losses.RewardCriterion* method), 49  
`forward()` (*xmodaler.losses.Triplet* method), 48  
`forward()` (*xmodaler.modeling.decode\_strategy.decode\_strategy.DecodeStrategy* method), 57  
`forward()` (*xmodaler.modeling.decoder.AttributeDecoder* method), 65  
`forward()` (*xmodaler.modeling.decoder.DecoupleBertDecoder* method), 62  
`forward()` (*xmodaler.modeling.decoder.MeshedDecoder* method), 63  
`forward()` (*xmodaler.modeling.decoder.MPLSTMDecoder* method), 60  
`forward()` (*xmodaler.modeling.decoder.SALSTMDecoder* method), 59  
`forward()` (*xmodaler.modeling.decoder.TDConvEDDecoder* method), 64  
`forward()` (*xmodaler.modeling.decoder.TransformerDecoder* method), 61  
`forward()` (*xmodaler.modeling.decoder.UpDownDecoder* method), 58  
`forward()` (*xmodaler.modeling.decoder.XLANDecoder* method), 63  
`forward()` (*xmodaler.modeling.embedding.position\_embedding.PositionEmbedding* method), 70  
`forward()` (*xmodaler.modeling.embedding.position\_embedding.PositionEmbedding* method), 69  
`forward()` (*xmodaler.modeling.embedding.TDConvEDVisualBaseEmbedding* method), 68  
`forward()` (*xmodaler.modeling.embedding.TokenBaseEmbedding* method), 66

`forward()` (*xmodaler.modeling.embedding.VisualBaseEmbedding* method), 67  
`forward()` (*xmodaler.modeling.embedding.VisualIdentityEmbedding* method), 68  
`forward()` (*xmodaler.modeling.encoder.Encoder* method), 71  
`forward()` (*xmodaler.modeling.encoder.GCNEncoder* method), 74  
`forward()` (*xmodaler.modeling.encoder.LowRankBilinearEncoder* method), 76  
`forward()` (*xmodaler.modeling.encoder.MemoryAugmentedEncoder* method), 75  
`forward()` (*xmodaler.modeling.encoder.SingleStreamBertEncoder* method), 73  
`forward()` (*xmodaler.modeling.encoder.TDConvEDEncoder* method), 77  
`forward()` (*xmodaler.modeling.encoder.TransformerEncoder* method), 72  
`forward()` (*xmodaler.modeling.encoder.TwoStreamBertEncoder* method), 74  
`forward()` (*xmodaler.modeling.encoder.UpDownEncoder* method), 71  
`forward()` (*xmodaler.modeling.layers.attention\_pooler.AttentionPooler* method), 77  
`forward()` (*xmodaler.modeling.layers.base\_attention.BaseAttention* method), 78  
`forward()` (*xmodaler.modeling.layers.bert.BertAttention* method), 79  
`forward()` (*xmodaler.modeling.layers.bert.BertCrossAttention* method), 80  
`forward()` (*xmodaler.modeling.layers.bert.BertGenerationLayer* method), 81  
`forward()` (*xmodaler.modeling.layers.bert.BertIntermediate* method), 79  
`forward()` (*xmodaler.modeling.layers.bert.BertLayer* method), 81  
`forward()` (*xmodaler.modeling.layers.bert.BertOutput* method), 80  
`forward()` (*xmodaler.modeling.layers.bert.BertPooler* method), 82  
`forward()` (*xmodaler.modeling.layers.bert.BertPredictionHeadTransform* method), 82  
`forward()` (*xmodaler.modeling.layers.bert.BertSelfAttention* method), 78  
`forward()` (*xmodaler.modeling.layers.bert.BertSelfOutput* method), 78  
`forward()` (*xmodaler.modeling.layers.bert.BertUnderstandingLayer* method), 81  
`forward()` (*xmodaler.modeling.layers.bert.BertXAttention* method), 80  
`forward()` (*xmodaler.modeling.layers.lowrank\_bilinear\_layers.LowRankBilinearEncoder* method), 84  
`forward()` (*xmodaler.modeling.layers.lowrank\_bilinear\_layers.LowRankBilinearEncoder* method), 85

`forward()` (`xmodaler.modeling.layers.multihead_attention.MultiHeadAttention` class method), 83  
`forward()` (`xmodaler.modeling.layers.multihead_attention.MultiHeadAttention` class method), 83  
`forward()` (`xmodaler.modeling.layers.positionwise_feedforward.PositionwiseFeedForward` class method), 83  
`forward()` (`xmodaler.modeling.layers.scattention.SCAAttention` class method), 48  
`forward()` (`xmodaler.modeling.layers.scattention.SCAAttention` class method), 85  
`forward()` (`xmodaler.modeling.layers.tdconvd_layers.ShiftedConvLayer` class method), 47  
`forward()` (`xmodaler.modeling.layers.tdconvd_layers.ShiftedConvLayer` class method), 86  
`forward()` (`xmodaler.modeling.layers.tdconvd_layers.SoftAttention` class method), 48  
`forward()` (`xmodaler.modeling.layers.tdconvd_layers.SoftAttention` class method), 87  
`forward()` (`xmodaler.modeling.layers.tdconvd_layers.TemporalDeformableBlock` class method), 86  
`forward()` (`xmodaler.modeling.layers.tdconvd_layers.TemporalDeformableBlock` class method), 85  
`forward()` (`xmodaler.modeling.meta_arch.base_enc_dec.BaseEncoderDecoder` class method), 87  
`forward()` (`xmodaler.modeling.predictor.BasePredictor` class method), 90  
`forward()` (`xmodaler.modeling.predictor.BertIsMatchedPredictor` class method), 91  
`forward()` (`xmodaler.modeling.predictor.BertPredictionHead` class method), 90  
`forward()` (`xmodaler.modeling.predictor.BertPredictionHead` class method), 54  
`forward()` (`xmodaler.modeling.predictor.BertVisualFeatureRegressionHead` class method), 91  
`forward()` (`xmodaler.modeling.predictor.BertVisualFeatureRegressionHead` class method), 51  
`forward()` (`xmodaler.modeling.predictor.BertVisualPredictionHead` class method), 91  
`forward()` (`xmodaler.modeling.predictor.BertVisualPredictionHead` class method), 50  
`forward()` (`xmodaler.modeling.predictor.MultiModalPredictor` class method), 92  
`forward()` (`xmodaler.modeling.predictor.MultiModalPredictor` class method), 51  
`forward()` (`xmodaler.modeling.predictor.MultiModalSimilarity` class method), 93  
`forward()` (`xmodaler.modeling.predictor.MultiModalSimilarity` class method), 53  
`forward()` (`xmodaler.modeling.predictor.SingleStreamMultiModalPredictor` class method), 92  
`forward()` (`xmodaler.modeling.predictor.SingleStreamMultiModalPredictor` class method), 53  
`freeze()` (`xmodaler.config.CfgNode` class method), 19  
`from_config()` (`xmodaler.datasets.ConceptualCaptionsDataset` class method), 24  
`from_config()` (`xmodaler.datasets.ConceptualCaptionsDataset` class method), 25  
`from_config()` (`xmodaler.datasets.Flickr30kDataset` class method), 25  
`from_config()` (`xmodaler.datasets.Flickr30kDataset` class method), 26  
`from_config()` (`xmodaler.datasets.Flickr30kDatasetForSingleStream` class method), 26  
`from_config()` (`xmodaler.datasets.Flickr30kDatasetForSingleStream` class method), 26  
`from_config()` (`xmodaler.datasets.MSCoCoBertDataset` class method), 24  
`from_config()` (`xmodaler.datasets.MSCoCoBertDataset` class method), 24  
`from_config()` (`xmodaler.datasets.MSCoCoDataset` class method), 23  
`from_config()` (`xmodaler.datasets.MSCoCoDataset` class method), 23  
`from_config()` (`xmodaler.datasets.MSCoCoSampleByText` class method), 24  
`from_config()` (`xmodaler.datasets.MSCoCoSampleByText` class method), 24  
`from_config()` (`xmodaler.datasets.MSRVTTDataset` class method), 26  
`from_config()` (`xmodaler.datasets.MSRVTTDataset` class method), 26  
`from_config()` (`xmodaler.datasets.MSVDDataset` class method), 26  
`from_config()` (`xmodaler.datasets.MSVDDataset` class method), 26  
`from_config()` (`xmodaler.datasets.VCRDataset` class method), 26  
`from_config()` (`xmodaler.datasets.VCRDataset` class method), 26  
`from_config()` (`xmodaler.datasets.VQADataset` class method), 26  
`from_config()` (`xmodaler.datasets.VQADataset` class method), 26  
`from_config()` (`xmodaler.losses.BatchTriplet` class method), 48  
`from_config()` (`xmodaler.losses.BatchTriplet` class method), 48  
`from_config()` (`xmodaler.losses.BCEWithLogits` class method), 48  
`from_config()` (`xmodaler.losses.BCEWithLogits` class method), 48  
`from_config()` (`xmodaler.losses.CrossEntropy` class method), 48  
`from_config()` (`xmodaler.losses.CrossEntropy` class method), 48  
`from_config()` (`xmodaler.losses.LabelSmoothing` class method), 48  
`from_config()` (`xmodaler.losses.LabelSmoothing` class method), 48  
`from_config()` (`xmodaler.losses.PretrainLosses` class method), 48  
`from_config()` (`xmodaler.losses.PretrainLosses` class method), 48  
`from_config()` (`xmodaler.losses.RewardCriterion` class method), 48  
`from_config()` (`xmodaler.losses.RewardCriterion` class method), 48  
`from_config()` (`xmodaler.losses.Triplet` class method), 48  
`from_config()` (`xmodaler.losses.Triplet` class method), 48  
`from_config()` (`xmodaler.lr_scheduler.FixLR` class method), 52  
`from_config()` (`xmodaler.lr_scheduler.FixLR` class method), 52  
`from_config()` (`xmodaler.lr_scheduler.MultiStepLR` class method), 52  
`from_config()` (`xmodaler.lr_scheduler.MultiStepLR` class method), 52  
`from_config()` (`xmodaler.lr_scheduler.NoamLR` class method), 52  
`from_config()` (`xmodaler.lr_scheduler.NoamLR` class method), 52  
`from_config()` (`xmodaler.lr_scheduler.StepLR` class method), 52  
`from_config()` (`xmodaler.lr_scheduler.StepLR` class method), 52  
`from_config()` (`xmodaler.lr_scheduler.WarmupConstant` class method), 52  
`from_config()` (`xmodaler.lr_scheduler.WarmupConstant` class method), 52  
`from_config()` (`xmodaler.lr_scheduler.WarmupCosine` class method), 52  
`from_config()` (`xmodaler.lr_scheduler.WarmupCosine` class method), 52  
`from_config()` (`xmodaler.lr_scheduler.WarmupCosineWithHardRestarts` class method), 52  
`from_config()` (`xmodaler.lr_scheduler.WarmupCosineWithHardRestarts` class method), 52  
`from_config()` (`xmodaler.lr_scheduler.WarmupLinear` class method), 52  
`from_config()` (`xmodaler.lr_scheduler.WarmupLinear` class method), 52  
`from_config()` (`xmodaler.lr_scheduler.WarmupMultiStepLR` class method), 52  
`from_config()` (`xmodaler.lr_scheduler.WarmupMultiStepLR` class method), 52  
`from_config()` (`xmodaler.modeling.decode_strategy.decode_strategy.Decoder` class method), 58  
`from_config()` (`xmodaler.modeling.decode_strategy.decode_strategy.Decoder` class method), 58  
`from_config()` (`xmodaler.modeling.decoder.AttributeDecoder` class method), 65  
`from_config()` (`xmodaler.modeling.decoder.AttributeDecoder` class method), 65  
`from_config()` (`xmodaler.modeling.decoder.DecoupleBertDecoder` class method), 62  
`from_config()` (`xmodaler.modeling.decoder.DecoupleBertDecoder` class method), 62  
`from_config()` (`xmodaler.modeling.decoder.MeshedDecoder` class method), 63  
`from_config()` (`xmodaler.modeling.decoder.MeshedDecoder` class method), 63  
`from_config()` (`xmodaler.modeling.decoder.MPLSTMDecoder` class method), 60  
`from_config()` (`xmodaler.modeling.decoder.MPLSTMDecoder` class method), 60  
`from_config()` (`xmodaler.modeling.decoder.SALSTMDecoder` class method), 59  
`from_config()` (`xmodaler.modeling.decoder.SALSTMDecoder` class method), 59  
`from_config()` (`xmodaler.modeling.decoder.TDConvEDDecoder` class method), 65  
`from_config()` (`xmodaler.modeling.decoder.TDConvEDDecoder` class method), 65  
`from_config()` (`xmodaler.modeling.decoder.TransformerDecoder` class method), 61  
`from_config()` (`xmodaler.modeling.decoder.TransformerDecoder` class method), 61  
`from_config()` (`xmodaler.modeling.decoder.UpDownDecoder` class method), 61  
`from_config()` (`xmodaler.modeling.decoder.UpDownDecoder` class method), 61



class method), 59  
 from\_config() (xmodaler.modeling.decoder.XLANDecoder class method), 64  
 from\_config() (xmodaler.modeling.embedding.TDConvEDEncoder class method), 68  
 from\_config() (xmodaler.modeling.embedding.TokenBaseEncoder class method), 66  
 from\_config() (xmodaler.modeling.embedding.VisualBaseEncoder class method), 67  
 from\_config() (xmodaler.modeling.embedding.VisualIdentityEncoder class method), 68  
 from\_config() (xmodaler.modeling.encoder.Encoder class method), 71  
 from\_config() (xmodaler.modeling.encoder.GCNEncoder class method), 74  
 from\_config() (xmodaler.modeling.encoder.LowRankBilinearEncoder class method), 76  
 from\_config() (xmodaler.modeling.encoder.MemoryAugmentedEncoder class method), 75  
 from\_config() (xmodaler.modeling.encoder.SingleStreamEncoder class method), 73  
 from\_config() (xmodaler.modeling.encoder.TDConvEDEncoder class method), 77  
 from\_config() (xmodaler.modeling.encoder.TransformerEncoder class method), 72  
 from\_config() (xmodaler.modeling.encoder.TwoStreamBaseEncoder class method), 74  
 from\_config() (xmodaler.modeling.encoder.UpDownEncoder class method), 71  
 from\_config() (xmodaler.modeling.layers.bert.BertAttention class method), 79  
 from\_config() (xmodaler.modeling.layers.bert.BertCrossAttention class method), 80  
 from\_config() (xmodaler.modeling.layers.bert.BertGenerationLayer class method), 82  
 from\_config() (xmodaler.modeling.layers.bert.BertIntermediateLayer class method), 79  
 from\_config() (xmodaler.modeling.layers.bert.BertLayer class method), 81  
 from\_config() (xmodaler.modeling.layers.bert.BertOutputLayer class method), 80  
 from\_config() (xmodaler.modeling.layers.bert.BertPooler class method), 82  
 from\_config() (xmodaler.modeling.layers.bert.BertPredictionHead class method), 82  
 from\_config() (xmodaler.modeling.layers.bert.BertSelfAttention class method), 78  
 from\_config() (xmodaler.modeling.layers.bert.BertSelfOutput class method), 79  
 from\_config() (xmodaler.modeling.layers.bert.BertUnderstandingLayer class method), 81  
 from\_config() (xmodaler.modeling.layers.bert.BertXAttention class method), 80  
 from\_config() (xmodaler.modeling.meta\_arch.base\_enc\_dec.BaseEncoderDecoder class method), 87  
 from\_config() (xmodaler.modeling.meta\_arch.TDENBiTransformer class method), 88  
 from\_config() (xmodaler.modeling.meta\_arch.TDENCaptioner class method), 89  
 from\_config() (xmodaler.modeling.meta\_arch.TDENPretrain class method), 89  
 from\_config() (xmodaler.modeling.meta\_arch.TransformerEncoderDecoder class method), 88  
 from\_config() (xmodaler.modeling.meta\_arch.UniterForMMUnderstanding class method), 89  
 from\_config() (xmodaler.modeling.meta\_arch.UniterPretrain class method), 89  
 from\_config() (xmodaler.modeling.predictor.BasePredictor class method), 90  
 from\_config() (xmodaler.modeling.predictor.BertIsMatchedPredictor class method), 92  
 from\_config() (xmodaler.modeling.predictor.BertPredictionHead class method), 90  
 from\_config() (xmodaler.modeling.predictor.BertVisualFeatureRegression class method), 91  
 from\_config() (xmodaler.modeling.predictor.BertVisualPredictionHead class method), 91  
 from\_config() (xmodaler.modeling.predictor.MultiModalPredictor class method), 92  
 from\_config() (xmodaler.modeling.predictor.MultiModalSimilarity class method), 93  
 from\_config() (xmodaler.modeling.predictor.SingleStreamMultiModalPredictor class method), 92  
 from\_config() (xmodaler.optim.Adagrad class method), 93  
 from\_config() (xmodaler.optim.Adam class method), 93  
 from\_config() (xmodaler.optim.Adamax class method), 94  
 from\_config() (xmodaler.optim.AdamW class method), 94  
 from\_config() (xmodaler.optim.BertAdam class method), 94  
 from\_config() (xmodaler.optim.RAdam class method), 94  
 from\_config() (xmodaler.optim.RMSprop class method), 94  
 from\_config() (xmodaler.optim.SGD class method), 94  
 from\_config() (xmodaler.scorer.BaseScorer class method), 95  
 from\_config() (xmodaler.scorer.BertTokenizedScorer class method), 95  
 from\_config() (xmodaler.scorer.Cider class method), 95  
 fromkeys() (xmodaler.config.CfgNode method), 19  
 fromkeys() (xmodaler.utils.logger.Counter class method), 105

## G

- `gather()` (in module `xmodaler.utils.comm`), 98
  - `GCNEncoder` (class in `xmodaler.modeling.encoder`), 74
  - `get()` (`xmodaler.config.CfgNode` method), 19
  - `get()` (`xmodaler.utils.registry.Registry` method), 116
  - `get_act_layer()` (in module `xmodaler.modeling.layers.create_act`), 77
  - `get_activation()` (in module `xmodaler.modeling.layers.create_act`), 77
  - `get_all_checkpoint_files()` (`xmodaler.checkpoint.XmodalerCheckpoint` method), 17
  - `get_cfg()` (in module `xmodaler.config`), 21
  - `get_checkpoint_file()` (`xmodaler.checkpoint.XmodalerCheckpoint` method), 17
  - `get_event_storage()` (in module `xmodaler.utils.events`), 99
  - `get_extended_attention_mask()` (`xmodaler.modeling.meta_arch.base_enc_dec.BaseEncoderDecoder` method), 88
  - `get_extended_attention_mask()` (`xmodaler.modeling.meta_arch.RnnAttEncoderDecoder` method), 88
  - `get_extended_attention_mask()` (`xmodaler.modeling.meta_arch.TDENBiTransformer` method), 88
  - `get_extended_attention_mask()` (`xmodaler.modeling.meta_arch.TransformerEncoderDecoder` method), 88
  - `get_extended_attention_mask()` (`xmodaler.modeling.meta_arch.UniterForMMUnderstanding` method), 89
  - `get_last_lr()` (`xmodaler.lr_scheduler.FixLR` method), 55
  - `get_last_lr()` (`xmodaler.lr_scheduler.MultiStepLR` method), 55
  - `get_last_lr()` (`xmodaler.lr_scheduler.NoamLR` method), 51
  - `get_last_lr()` (`xmodaler.lr_scheduler.StepLR` method), 50
  - `get_last_lr()` (`xmodaler.lr_scheduler.WarmupConstant` method), 51
  - `get_last_lr()` (`xmodaler.lr_scheduler.WarmupCosine` method), 53
  - `get_last_lr()` (`xmodaler.lr_scheduler.WarmupCosineWithHardRestarts` method), 53
  - `get_last_lr()` (`xmodaler.lr_scheduler.WarmupLinear` method), 52
  - `get_last_lr()` (`xmodaler.lr_scheduler.WarmupMultiStepLR` method), 54
  - `get_local_rank()` (in module `xmodaler.utils.comm`), 98
  - `get_local_size()` (in module `xmodaler.utils.comm`), 98
  - `get_lr()` (`xmodaler.lr_scheduler.FixLR` method), 55
  - `get_lr()` (`xmodaler.lr_scheduler.MultiStepLR` method), 55
  - `get_lr()` (`xmodaler.lr_scheduler.NoamLR` method), 51
  - `get_lr()` (`xmodaler.lr_scheduler.StepLR` method), 50
  - `get_lr()` (`xmodaler.lr_scheduler.WarmupConstant` method), 51
  - `get_lr()` (`xmodaler.lr_scheduler.WarmupCosine` method), 53
  - `get_lr()` (`xmodaler.lr_scheduler.WarmupCosineWithHardRestarts` method), 53
  - `get_lr()` (`xmodaler.lr_scheduler.WarmupLinear` method), 52
  - `get_lr()` (`xmodaler.lr_scheduler.WarmupMultiStepLR` method), 54
  - `get_rank()` (in module `xmodaler.utils.comm`), 98
  - `get_sents()` (`xmodaler.scorer.BaseScorer` method), 95
  - `get_sents()` (`xmodaler.scorer.BertTokenizedScorer` method), 95
  - `get_world_size()` (in module `xmodaler.utils.comm`), 98
  - `global_cfg` (in module `xmodaler.config`), 22
  - `greedy_decode()` (`xmodaler.modeling.meta_arch.base_enc_dec.BaseEncoderDecoder` method), 88
  - `GreedyDecoder` (class in `xmodaler.modeling.decode_strategy`), 56
- ## H
- `hard_negative_mining()` (`xmodaler.engine.SingleStreamRetrievalTrainerHardNegatives` method), 43
  - `has_checkpoint()` (`xmodaler.checkpoint.XmodalerCheckpoint` method), 17
  - `histories()` (`xmodaler.utils.events.EventStorage` method), 101
  - `history()` (`xmodaler.utils.events.EventStorage` method), 101
  - `HookBase` (class in `xmodaler.engine`), 30
- ## I
- `IMMUTABLE` (`xmodaler.config.CfgNode` attribute), 18
  - `iou()` (in module `xmodaler.functional`), 46
  - `is_frozen()` (`xmodaler.config.CfgNode` method), 19
  - `is_main_process()` (in module `xmodaler.utils.comm`), 19
  - `is_new_allowed()` (`xmodaler.config.CfgNode` method), 19
  - `items()` (`xmodaler.config.CfgNode` method), 19
  - `Iter` (`xmodaler.engine.TrainerBase` attribute), 31
  - `iter` (`xmodaler.utils.events.EventStorage` property), 101
  - `iteration` (`xmodaler.utils.events.EventStorage` property), 102
  - `IterationTimer` (class in `xmodaler.engine`), 32

## J

JSONWriter (class in *xmodaler.utils.events*), 99

## K

key\_is\_deprecated() (*xmodaler.config.CfgNode* method), 19

key\_is\_renamed() (*xmodaler.config.CfgNode* method), 19

keys() (*xmodaler.config.CfgNode* method), 19

kfg (in module *xmodaler.config*), 22

## L

LabelSmoothing (class in *xmodaler.losses*), 48

latest() (*xmodaler.utils.events.EventStorage* method), 102

latest\_with\_smoothing\_hint() (*xmodaler.utils.events.EventStorage* method), 102

launch() (in module *xmodaler.engine*), 27

load() (*xmodaler.checkpoint.XmodalerCheckpoint* method), 17

load\_cfg() (*xmodaler.config.CfgNode* class method), 20

load\_data() (*xmodaler.datasets.ConceptualCaptionsDataset* method), 24

load\_data() (*xmodaler.datasets.ConceptualCaptionsDatasetForSingleStream* method), 25

load\_data() (*xmodaler.datasets.Flickr30kDataset* method), 25

load\_data() (*xmodaler.datasets.Flickr30kDatasetForSingleStream* method), 26

load\_data() (*xmodaler.datasets.Flickr30kDatasetForSingleStreamVal* method), 26

load\_data() (*xmodaler.datasets.MSCoCoBertDataset* method), 24

load\_data() (*xmodaler.datasets.MSCoCoDataset* method), 23

load\_data() (*xmodaler.datasets.MSCoCoSampleByTxtDataset* method), 24

load\_data() (*xmodaler.datasets.MSRVTTDataset* method), 26

load\_data() (*xmodaler.datasets.MSVDDataset* method), 26

load\_data() (*xmodaler.datasets.VCRDataset* method), 25

load\_data() (*xmodaler.datasets.VQADataset* method), 25

load\_from\_file\_tmp() (*xmodaler.config.CfgNode* method), 20

load\_state\_dict() (*xmodaler.engine.DefaultTrainer* method), 29

load\_state\_dict() (*xmodaler.engine.RetrievalTrainer* method), 38

load\_state\_dict() (*xmodaler.engine.RLBeamTrainer* method), 40

load\_state\_dict() (*xmodaler.engine.RLTrainer* method), 39

load\_state\_dict() (*xmodaler.engine.SingleStreamRetrievalTrainer* method), 42

load\_state\_dict() (*xmodaler.engine.SingleStreamRetrievalTrainerHardRestarts* method), 43

load\_state\_dict() (*xmodaler.engine.TDENPretrainer* method), 44

load\_state\_dict() (*xmodaler.engine.TrainerBase* method), 31

load\_state\_dict() (*xmodaler.engine.VCRTrainer* method), 45

load\_state\_dict() (*xmodaler.lr\_scheduler.FixLR* method), 55

load\_state\_dict() (*xmodaler.lr\_scheduler.MultiStepLR* method), 55

load\_state\_dict() (*xmodaler.lr\_scheduler.NoamLR* method), 51

load\_state\_dict() (*xmodaler.lr\_scheduler.StepLR* method), 50

load\_state\_dict() (*xmodaler.lr\_scheduler.WarmupConstant* method), 51

load\_state\_dict() (*xmodaler.lr\_scheduler.WarmupCosine* method), 53

load\_state\_dict() (*xmodaler.lr\_scheduler.WarmupCosineWithHardRestarts* method), 53

load\_state\_dict() (*xmodaler.lr\_scheduler.WarmupLinear* method), 52

load\_state\_dict() (*xmodaler.lr\_scheduler.WarmupMultiStepLR* method), 54

load\_vocab() (in module *xmodaler.functional*), 46

load\_yaml\_with\_base() (*xmodaler.config.CfgNode* class method), 20

locate() (in module *xmodaler.utils.registry*), 116

log\_every\_n() (in module *xmodaler.utils.logger*), 106

log\_every\_n\_seconds() (in module *xmodaler.utils.logger*), 106

log\_first\_n() (in module *xmodaler.utils.logger*), 106

LowRankBilinearAttention (class in *xmodaler.modeling.layers.lowrank\_bilinear\_layers*), 84

LowRankBilinearEncoder (class in *xmodaler.modeling.encoder*), 75

LowRankBilinearLayer (class in *xmodaler.modeling.layers.lowrank\_bilinear\_layers*), 84

lr\_lambda() (*xmodaler.lr\_scheduler.FixLR* method), 55

lr\_lambda() (*xmodaler.lr\_scheduler.WarmupConstant* method), 51

lr\_lambda() (*xmodaler.lr\_scheduler.WarmupCosine* method), 53

lr\_lambda() (*xmodaler.lr\_scheduler.WarmupCosineWithHardRestarts* method), 53

method), 54  
 lr\_lambda() (xmodaler.lr\_scheduler.WarmupLinear  
 method), 52  
 LRScheduler (class in xmodaler.engine), 34

## M

MapDataset (class in xmodaler.datasets), 23  
 max\_iter (xmodaler.engine.TrainerBase attribute), 31  
 max\_model\_input\_sizes  
 (xmodaler.tokenization.BertTokenizer at-  
 tribute), 96  
 MemoryAugmentedEncoder (class in  
 xmodaler.modeling.encoder), 75  
 merge\_from\_file() (xmodaler.config.CfgNode  
 method), 20  
 merge\_from\_list() (xmodaler.config.CfgNode  
 method), 20  
 merge\_from\_other\_cfg() (xmodaler.config.CfgNode  
 method), 20  
 MeshedDecoder (class in xmodaler.modeling.decoder),  
 62  
 method() (xmodaler.scorer.Cider method), 95  
 ModelWeightsManipulating (class in  
 xmodaler.engine), 37  
 module  
 xmodaler.tokenization, 95  
 most\_common() (xmodaler.utils.logger.Counter  
 method), 105  
 MPLSTMDecoder (class in xmodaler.modeling.decoder),  
 60  
 MSCoCoBertDataset (class in xmodaler.datasets), 24  
 MSCoCoDataset (class in xmodaler.datasets), 23  
 MSCoCoSampleByTxtDataset (class in  
 xmodaler.datasets), 23  
 MSRVTTDataset (class in xmodaler.datasets), 26  
 MSVDDataset (class in xmodaler.datasets), 26  
 MultiHeadAttention (class in  
 xmodaler.modeling.layers.multihead\_attention),  
 83  
 MultiHeadAttentionMemory (class in  
 xmodaler.modeling.layers.multihead\_attention),  
 82  
 MultiModalPredictor (class in  
 xmodaler.modeling.predictor), 92  
 MultiModalSimilarity (class in  
 xmodaler.modeling.predictor), 93  
 MultiStepLR (class in xmodaler.lr\_scheduler), 54

## N

name\_scope() (xmodaler.utils.events.EventStorage  
 method), 102  
 NEW\_ALLOWED (xmodaler.config.CfgNode attribute), 18  
 NNEmbeddingEncoding (class in  
 xmodaler.modeling.embedding.position\_embedding),

69

NoamLR (class in xmodaler.lr\_scheduler), 50

## P

pad\_tensor() (in module xmodaler.functional), 47  
 PathHandler (class in xmodaler.utils.file\_io), 103  
 PathManager (in module xmodaler.utils.file\_io), 103  
 PeriodicCheckpoint (class in xmodaler.engine), 33  
 PeriodicEpochCheckpoint (class in  
 xmodaler.checkpoint), 15  
 PeriodicWriter (class in xmodaler.engine), 33  
 PicklableWrapper (class in xmodaler.utils.serialize),  
 116  
 pop() (xmodaler.config.CfgNode method), 20  
 popitem() (xmodaler.config.CfgNode method), 20  
 PositionWiseFeedForward (class in  
 xmodaler.modeling.layers.positionwise\_feedforward),  
 83  
 PreciseBN (class in xmodaler.engine), 36  
 precompute() (xmodaler.modeling.layers.lowrank\_bilinear\_layers.LowRa  
 method), 84  
 precompute() (xmodaler.modeling.layers.lowrank\_bilinear\_layers.LowRa  
 method), 85  
 preprocess() (xmodaler.modeling.decoder.AttributeDecoder  
 method), 66  
 preprocess() (xmodaler.modeling.decoder.MPLSTMDecoder  
 method), 60  
 preprocess() (xmodaler.modeling.decoder.SALSTMDecoder  
 method), 60  
 preprocess() (xmodaler.modeling.decoder.TDConvEDDecoder  
 method), 65  
 preprocess() (xmodaler.modeling.decoder.UpDownDecoder  
 method), 59  
 preprocess() (xmodaler.modeling.decoder.XLANDecoder  
 method), 64  
 preprocess\_batch() (xmodaler.modeling.meta\_arch.base\_enc\_dec.Base  
 method), 88  
 preprocess\_inputs()  
 (xmodaler.modeling.meta\_arch.UniterPretrain  
 method), 89  
 pretrained\_init\_configuration  
 (xmodaler.tokenization.BertTokenizer at-  
 tribute), 96  
 pretrained\_vocab\_files\_map  
 (xmodaler.tokenization.BertTokenizer at-  
 tribute), 96  
 PretrainLosses (class in xmodaler.losses), 49  
 print\_lr() (xmodaler.lr\_scheduler.FixLR method), 55  
 print\_lr() (xmodaler.lr\_scheduler.MultiStepLR  
 method), 55  
 print\_lr() (xmodaler.lr\_scheduler.NoamLR method),  
 51  
 print\_lr() (xmodaler.lr\_scheduler.StepLR method), 50



[print\\_lr\(\)](#) (*xmodaler.lr\_scheduler.WarmupConstant method*), 51  
[print\\_lr\(\)](#) (*xmodaler.lr\_scheduler.WarmupCosine method*), 53  
[print\\_lr\(\)](#) (*xmodaler.lr\_scheduler.WarmupCosineWithHardRestart method*), 54  
[print\\_lr\(\)](#) (*xmodaler.lr\_scheduler.WarmupLinear method*), 52  
[print\\_lr\(\)](#) (*xmodaler.lr\_scheduler.WarmupMultiStepLR method*), 54  
[put\\_histogram\(\)](#) (*xmodaler.utils.events.EventStorage method*), 102  
[put\\_image\(\)](#) (*xmodaler.utils.events.EventStorage method*), 102  
[put\\_scalar\(\)](#) (*xmodaler.utils.events.EventStorage method*), 102  
[put\\_scalars\(\)](#) (*xmodaler.utils.events.EventStorage method*), 102

## R

[RAdam](#) (class in *xmodaler.optim*), 94  
[raise\\_key\\_rename\\_error\(\)](#) (*xmodaler.config.CfgNode method*), 20  
[random\\_color\(\)](#) (in module *xmodaler.utils.colormap*), 97  
[random\\_region\(\)](#) (in module *xmodaler.functional*), 47  
[random\\_word\(\)](#) (in module *xmodaler.functional*), 47  
[read\\_lines\(\)](#) (in module *xmodaler.functional*), 47  
[read\\_lines\\_set\(\)](#) (in module *xmodaler.functional*), 47  
[read\\_np\(\)](#) (in module *xmodaler.functional*), 47  
[read\\_np\\_bbox\(\)](#) (in module *xmodaler.functional*), 47  
[reduce\\_dict\(\)](#) (in module *xmodaler.utils.comm*), 99  
[register\(\)](#) (*xmodaler.utils.registry.Registry method*), 116  
[register\\_deprecated\\_key\(\)](#) (*xmodaler.config.CfgNode method*), 20  
[register\\_hooks\(\)](#) (*xmodaler.engine.DefaultTrainer method*), 29  
[register\\_hooks\(\)](#) (*xmodaler.engine.RetrievalTrainer method*), 38  
[register\\_hooks\(\)](#) (*xmodaler.engine.RLBeamTrainer method*), 41  
[register\\_hooks\(\)](#) (*xmodaler.engine.RLTrainer method*), 39  
[register\\_hooks\(\)](#) (*xmodaler.engine.SingleStreamRetrievalTrainer method*), 42  
[register\\_hooks\(\)](#) (*xmodaler.engine.SingleStreamRetrievalTrainerHardNegatives method*), 43  
[register\\_hooks\(\)](#) (*xmodaler.engine.TDENPretrainer method*), 44  
[register\\_hooks\(\)](#) (*xmodaler.engine.TrainerBase method*), 31  
[register\\_hooks\(\)](#) (*xmodaler.engine.VCRTrainer method*), 45  
[register\\_renamed\\_key\(\)](#) (*xmodaler.config.CfgNode method*), 21  
[Registry](#) (class in *xmodaler.utils.registry*), 116  
[RENAMED\\_KEYS](#) (*xmodaler.config.CfgNode attribute*), 18  
[resume\\_or\\_load\(\)](#) (*xmodaler.checkpoint.XmodalerCheckpointInterface method*), 17  
[resume\\_or\\_load\(\)](#) (*xmodaler.engine.DefaultTrainer method*), 29  
[resume\\_or\\_load\(\)](#) (*xmodaler.engine.RetrievalTrainer method*), 38  
[resume\\_or\\_load\(\)](#) (*xmodaler.engine.RLBeamTrainer method*), 41  
[resume\\_or\\_load\(\)](#) (*xmodaler.engine.RLTrainer method*), 39  
[resume\\_or\\_load\(\)](#) (*xmodaler.engine.SingleStreamRetrievalTrainer method*), 42  
[resume\\_or\\_load\(\)](#) (*xmodaler.engine.SingleStreamRetrievalTrainerHardNegatives method*), 43  
[resume\\_or\\_load\(\)](#) (*xmodaler.engine.TDENPretrainer method*), 44  
[resume\\_or\\_load\(\)](#) (*xmodaler.engine.VCRTrainer method*), 45  
[RetrievalEvaler](#) (class in *xmodaler.evaluation*), 46  
[RetrievalTrainer](#) (class in *xmodaler.engine*), 38  
[retry\\_if\\_cuda\\_oom\(\)](#) (in module *xmodaler.utils.memory*), 115  
[RewardCriterion](#) (class in *xmodaler.losses*), 49  
[RLBeamTrainer](#) (class in *xmodaler.engine*), 40  
[RLTrainer](#) (class in *xmodaler.engine*), 39  
[RMSprop](#) (class in *xmodaler.optim*), 94  
[RnnAttEncoderDecoder](#) (class in *xmodaler.modeling.meta\_arch*), 88  
[run\\_step\(\)](#) (*xmodaler.engine.DefaultTrainer method*), 29  
[run\\_step\(\)](#) (*xmodaler.engine.RetrievalTrainer method*), 38  
[run\\_step\(\)](#) (*xmodaler.engine.RLBeamTrainer method*), 41  
[run\\_step\(\)](#) (*xmodaler.engine.RLTrainer method*), 40  
[run\\_step\(\)](#) (*xmodaler.engine.SingleStreamRetrievalTrainer method*), 42  
[run\\_step\(\)](#) (*xmodaler.engine.SingleStreamRetrievalTrainerHardNegatives method*), 43  
[run\\_step\(\)](#) (*xmodaler.engine.TDENPretrainer method*), 44  
[run\\_step\(\)](#) (*xmodaler.engine.TrainerBase method*), 31  
[run\\_step\(\)](#) (*xmodaler.engine.VCRTrainer method*), 45

## S

[SALSTMDecoder](#) (class in *xmodaler.modeling.decoder*), 59  
[save\(\)](#) (*xmodaler.checkpoint.PeriodicEpochCheckpointInterface method*), 15

`save()` (`xmodaler.checkpoint.XmodalerCheckpoint` method), 18  
`save()` (`xmodaler.engine.PeriodicCheckpoint` method), 34  
`save_vocabulary()` (`xmodaler.tokenization.BertTokenizer` method), 97  
`SCAttention` (class in `xmodaler.modeling.layers.scattention`), 85  
`scheduler` (`xmodaler.engine.DefaultTrainer` attribute), 28  
`seed_all_rng()` (in module `xmodaler.utils.env`), 99  
`select_logits_targets_by_mask()` (`xmodaler.losses.PretrainLosses` method), 49  
`set_global_cfg()` (in module `xmodaler.config`), 21  
`set_new_allowed()` (`xmodaler.config.CfgNode` method), 21  
`setdefault()` (`xmodaler.config.CfgNode` method), 21  
`setup_custom_environment()` (in module `xmodaler.utils.env`), 99  
`setup_environment()` (in module `xmodaler.utils.env`), 99  
`setup_logger()` (in module `xmodaler.utils.logger`), 106  
`SGD` (class in `xmodaler.optim`), 94  
`shared_random_seed()` (in module `xmodaler.utils.comm`), 99  
`ShiftedConvLayer` (class in `xmodaler.modeling.layers.tdconvd_layers`), 86  
`SingleStreamBertEncoder` (class in `xmodaler.modeling.encoder`), 72  
`SingleStreamMultiModalPredictor` (class in `xmodaler.modeling.predictor`), 92  
`SingleStreamRetrievalTrainer` (class in `xmodaler.engine`), 41  
`SingleStreamRetrievalTrainerHardNegatives` (class in `xmodaler.engine`), 42  
`SinusoidEncoding` (class in `xmodaler.modeling.embedding.position_embedding`), 69  
`smoothing_hints()` (`xmodaler.utils.events.EventStorage` method), 103  
`SoftAttention` (class in `xmodaler.modeling.layers.tdconvd_layers`), 86  
`start_iter` (`xmodaler.engine.TrainerBase` attribute), 31  
`state_dict()` (`xmodaler.engine.AutogradProfiler` method), 35  
`state_dict()` (`xmodaler.engine.CallbackHook` method), 32  
`state_dict()` (`xmodaler.engine.DefaultTrainer` method), 29  
`state_dict()` (`xmodaler.engine.EvalHook` method), 36  
`state_dict()` (`xmodaler.engine.HookBase` method), 30  
`state_dict()` (`xmodaler.engine.IterationTimer` method), 32  
`state_dict()` (`xmodaler.engine.LRScheduler` method), 35  
`state_dict()` (`xmodaler.engine.ModelWeightsManipulating` method), 37  
`state_dict()` (`xmodaler.engine.PeriodicCheckpoint` method), 34  
`state_dict()` (`xmodaler.engine.PeriodicWriter` method), 33  
`state_dict()` (`xmodaler.engine.PreciseBN` method), 37  
`state_dict()` (`xmodaler.engine.RetrievalTrainer` method), 38  
`state_dict()` (`xmodaler.engine.RLBeamTrainer` method), 41  
`state_dict()` (`xmodaler.engine.RLTrainer` method), 40  
`state_dict()` (`xmodaler.engine.SingleStreamRetrievalTrainer` method), 42  
`state_dict()` (`xmodaler.engine.SingleStreamRetrievalTrainerHardNegatives` method), 43  
`state_dict()` (`xmodaler.engine.TDENPretrainer` method), 44  
`state_dict()` (`xmodaler.engine.TrainerBase` method), 31  
`state_dict()` (`xmodaler.engine.VCRTrainer` method), 45  
`state_dict()` (`xmodaler.lr_scheduler.FixLR` method), 55  
`state_dict()` (`xmodaler.lr_scheduler.MultiStepLR` method), 55  
`state_dict()` (`xmodaler.lr_scheduler.NoamLR` method), 51  
`state_dict()` (`xmodaler.lr_scheduler.StepLR` method), 50  
`state_dict()` (`xmodaler.lr_scheduler.WarmupConstant` method), 51  
`state_dict()` (`xmodaler.lr_scheduler.WarmupCosine` method), 53  
`state_dict()` (`xmodaler.lr_scheduler.WarmupCosineWithHardRestarts` method), 54  
`state_dict()` (`xmodaler.lr_scheduler.WarmupLinear` method), 52  
`state_dict()` (`xmodaler.lr_scheduler.WarmupMultiStepLR` method), 54  
`step()` (`xmodaler.checkpoint.PeriodicEpochCheckpoint` method), 15  
`step()` (`xmodaler.engine.PeriodicCheckpoint` method), 34  
`step()` (`xmodaler.lr_scheduler.FixLR` method), 55  
`step()` (`xmodaler.lr_scheduler.MultiStepLR` method), 55  
`step()` (`xmodaler.lr_scheduler.NoamLR` method), 51  
`step()` (`xmodaler.lr_scheduler.StepLR` method), 50  
`step()` (`xmodaler.lr_scheduler.WarmupConstant` method), 52  
`step()` (`xmodaler.lr_scheduler.WarmupCosine` method), 53

`step()` (*xmodaler.lr\_scheduler.WarmupCosineWithHardRestarts* method), 54  
`step()` (*xmodaler.lr\_scheduler.WarmupLinear* method), 52  
`step()` (*xmodaler.lr\_scheduler.WarmupMultiStepLR* method), 54  
`step()` (*xmodaler.optim.BertAdam* method), 94  
`step()` (*xmodaler.optim.RAdam* method), 94  
`step()` (*xmodaler.utils.events.EventStorage* method), 103  
`StepLR` (class in *xmodaler.lr\_scheduler*), 50  
`storage` (*xmodaler.engine.TrainerBase* attribute), 31  
`subtract()` (*xmodaler.utils.logger.Counter* method), 105  
`synchronize()` (in module *xmodaler.utils.comm*), 99

## T

`tabulate()` (in module *xmodaler.utils.logger*), 107  
`tag_last_checkpoint()` (*xmodaler.checkpoint.XmodalerCheckpoint* method), 18  
`TDConvEDDecoder` (class in *xmodaler.modeling.decoder*), 64  
`TDConvEDEncoder` (class in *xmodaler.modeling.encoder*), 76  
`TDConvEDVisualBaseEmbedding` (class in *xmodaler.modeling.embedding*), 68  
`TDENBiTransformer` (class in *xmodaler.modeling.meta\_arch*), 88  
`TDENCaptioner` (class in *xmodaler.modeling.meta\_arch*), 89  
`TDENPretrain` (class in *xmodaler.modeling.meta\_arch*), 88  
`TDENPretrainer` (class in *xmodaler.engine*), 43  
`TemporalDeformableBlock` (class in *xmodaler.modeling.layers.tdconvd\_layers*), 86  
`TemporalDeformableLayer` (class in *xmodaler.modeling.layers.tdconvd\_layers*), 85  
`TensorboardXWriter` (class in *xmodaler.utils.events*), 100  
`test()` (*xmodaler.engine.DefaultTrainer* class method), 29  
`test()` (*xmodaler.engine.RetrievalTrainer* class method), 38  
`test()` (*xmodaler.engine.RLBeamTrainer* class method), 41  
`test()` (*xmodaler.engine.RLTrainer* class method), 40  
`test()` (*xmodaler.engine.SingleStreamRetrievalTrainer* class method), 42  
`test()` (*xmodaler.engine.SingleStreamRetrievalTrainerHardNegatives* class method), 43  
`test()` (*xmodaler.engine.TDENPretrainer* class method), 44  
`test()` (*xmodaler.engine.VCRTrainer* class method), 45  
`test_forward()` (*xmodaler.modeling.predictor.SingleStreamMultiModalPredictor* method), 92  
`TokenBaseEmbedding` (class in *xmodaler.modeling.embedding*), 66  
`train()` (*xmodaler.engine.DefaultTrainer* method), 29  
`train()` (*xmodaler.engine.RetrievalTrainer* method), 39  
`train()` (*xmodaler.engine.RLBeamTrainer* method), 41  
`train()` (*xmodaler.engine.RLTrainer* method), 40  
`train()` (*xmodaler.engine.SingleStreamRetrievalTrainer* method), 42  
`train()` (*xmodaler.engine.SingleStreamRetrievalTrainerHardNegatives* method), 43  
`train()` (*xmodaler.engine.TDENPretrainer* method), 44  
`train()` (*xmodaler.engine.TrainerBase* method), 31  
`train()` (*xmodaler.engine.VCRTrainer* method), 45  
`trainer` (*xmodaler.engine.AutogradProfiler* attribute), 36  
`trainer` (*xmodaler.engine.CallbackHook* attribute), 32  
`trainer` (*xmodaler.engine.EvalHook* attribute), 36  
`trainer` (*xmodaler.engine.HookBase* attribute), 30  
`trainer` (*xmodaler.engine.IterationTimer* attribute), 32  
`trainer` (*xmodaler.engine.LRScheduler* attribute), 35  
`trainer` (*xmodaler.engine.ModelWeightsManipulating* attribute), 37  
`trainer` (*xmodaler.engine.PeriodicCheckpoint* attribute), 34  
`trainer` (*xmodaler.engine.PeriodicWriter* attribute), 33  
`trainer` (*xmodaler.engine.PreciseBN* attribute), 37  
`TrainerBase` (class in *xmodaler.engine*), 30  
`training` (*xmodaler.losses.BatchTriplet* attribute), 48  
`training` (*xmodaler.losses.BCEWithLogits* attribute), 47  
`training` (*xmodaler.losses.CrossEntropy* attribute), 48  
`training` (*xmodaler.losses.LabelSmoothing* attribute), 49  
`training` (*xmodaler.losses.PretrainLosses* attribute), 49  
`training` (*xmodaler.losses.RewardCriterion* attribute), 50  
`training` (*xmodaler.losses.Triplet* attribute), 48  
`training` (*xmodaler.modeling.decode\_strategy.BeamSearcher* attribute), 57  
`training` (*xmodaler.modeling.decode\_strategy.decode\_strategy.DecodeStrategy* attribute), 58  
`training` (*xmodaler.modeling.decode\_strategy.GreedyDecoder* attribute), 56  
`training` (*xmodaler.modeling.decoder.AttributeDecoder* attribute), 66  
`training` (*xmodaler.modeling.decoder.DecoupleBertDecoder* attribute), 62  
`training` (*xmodaler.modeling.decoder.MeshedDecoder* attribute), 63  
`training` (*xmodaler.modeling.decoder.MPLSTMDecoder* attribute), 60  
`training` (*xmodaler.modeling.decoder.SALSTMDecoder* attribute), 60

training (xmodaler.modeling.decoder.TDConvEDDecoder attribute), 65	training (xmodaler.modeling.layers.bert.BertPooler attribute), 82
training (xmodaler.modeling.decoder.TransformerDecoder attribute), 61	training (xmodaler.modeling.layers.bert.BertPredictionHeadTransform attribute), 82
training (xmodaler.modeling.decoder.UpDownDecoder attribute), 59	training (xmodaler.modeling.layers.bert.BertSelfAttention attribute), 78
training (xmodaler.modeling.decoder.XLANDecoder attribute), 64	training (xmodaler.modeling.layers.bert.BertSelfOutput attribute), 79
training (xmodaler.modeling.embedding.position_embedding attribute), 70	training (xmodaler.modeling.layers.bert.BertUnderstandingLayer attribute), 81
training (xmodaler.modeling.embedding.position_embedding attribute), 69	training (xmodaler.modeling.layers.bert.BertXAttention attribute), 80
training (xmodaler.modeling.embedding.TDConvEDVisualBaseEmbedding attribute), 69	training (xmodaler.modeling.layers.lowrank_bilinear_layers.LowRankBilinear attribute), 84
training (xmodaler.modeling.embedding.TokenBaseEmbedding attribute), 66	training (xmodaler.modeling.layers.lowrank_bilinear_layers.LowRankBilinear attribute), 85
training (xmodaler.modeling.embedding.VisualBaseEmbedding attribute), 67	training (xmodaler.modeling.layers.multihead_attention.MultiHeadAttention attribute), 83
training (xmodaler.modeling.embedding.VisualIdentityEmbedding attribute), 68	training (xmodaler.modeling.layers.multihead_attention.MultiHeadAttention attribute), 83
training (xmodaler.modeling.encoder.Encoder attribute), 71	training (xmodaler.modeling.layers.positionwise_feedforward.PositionwiseFeedForward attribute), 84
training (xmodaler.modeling.encoder.GCNEncoder attribute), 75	training (xmodaler.modeling.layers.scattention.SCAttention attribute), 85
training (xmodaler.modeling.encoder.LowRankBilinearEncoder attribute), 76	training (xmodaler.modeling.layers.tdconvded_layers.ShiftedConvLayer attribute), 86
training (xmodaler.modeling.encoder.MemoryAugmentedEncoder attribute), 75	training (xmodaler.modeling.layers.tdconvded_layers.SoftAttention attribute), 87
training (xmodaler.modeling.encoder.SingleStreamBertEncoder attribute), 73	training (xmodaler.modeling.layers.tdconvded_layers.TemporalDeformableConvLayer attribute), 86
training (xmodaler.modeling.encoder.TDConvEDEncoder attribute), 77	training (xmodaler.modeling.layers.tdconvded_layers.TemporalDeformableConvLayer attribute), 86
training (xmodaler.modeling.encoder.TransformerEncoder attribute), 72	training (xmodaler.modeling.meta_arch.base_enc_dec.BaseEncoderDecoder attribute), 88
training (xmodaler.modeling.encoder.TwoStreamBertEncoder attribute), 74	training (xmodaler.modeling.meta_arch.RnnAttEncoderDecoder attribute), 88
training (xmodaler.modeling.encoder.UpDownEncoder attribute), 71	training (xmodaler.modeling.meta_arch.TDENBiTransformer attribute), 88
training (xmodaler.modeling.layers.attention_pooler.AttentionPooler attribute), 77	training (xmodaler.modeling.meta_arch.TDENCaptioner attribute), 89
training (xmodaler.modeling.layers.base_attention.BaseAttention attribute), 78	training (xmodaler.modeling.meta_arch.TDENPretrain attribute), 89
training (xmodaler.modeling.layers.bert.BertAttention attribute), 79	training (xmodaler.modeling.meta_arch.TransformerEncoderDecoder attribute), 88
training (xmodaler.modeling.layers.bert.BertCrossAttention attribute), 81	training (xmodaler.modeling.meta_arch.UniterForMMUnderstanding attribute), 89
training (xmodaler.modeling.layers.bert.BertGenerationLayer attribute), 82	training (xmodaler.modeling.meta_arch.UniterPretrain attribute), 89
training (xmodaler.modeling.layers.bert.BertIntermediateLayer attribute), 79	training (xmodaler.modeling.predictor.BasePredictor attribute), 90
training (xmodaler.modeling.layers.bert.BertLayer attribute), 81	training (xmodaler.modeling.predictor.BertIsMatchedPredictor attribute), 92
training (xmodaler.modeling.layers.bert.BertOutput attribute), 80	training (xmodaler.modeling.predictor.BertPredictionHead attribute), 90



[training \(xmodaler.modeling.predictor.BertVisualFeatureExtractor attribute\), 91](#)  
[training \(xmodaler.modeling.predictor.BertVisualPredictor attribute\), 91](#)  
[training \(xmodaler.modeling.predictor.MultiModalPredictor attribute\), 92](#)  
[training \(xmodaler.modeling.predictor.MultiModalSimilarity attribute\), 93](#)  
[training \(xmodaler.modeling.predictor.SingleStreamMultiModalPredictor attribute\), 93](#)  
[TransformerDecoder \(class in xmodaler.modeling.decoder\), 60](#)  
[TransformerEncoder \(class in xmodaler.modeling.encoder\), 72](#)  
[TransformerEncoderDecoder \(class in xmodaler.modeling.meta\\_arch\), 88](#)  
[transpose\\_for\\_scores\(\) \(xmodaler.modeling.layers.bert.BertSelfAttention method\), 78](#)  
[transpose\\_for\\_scores\(\) \(xmodaler.modeling.layers.bert.BertXAttention method\), 80](#)  
[Triplet \(class in xmodaler.losses\), 48](#)  
[TwoStreamBertEncoder \(class in xmodaler.modeling.encoder\), 73](#)

**U**

[UniterForMMUnderstanding \(class in xmodaler.modeling.meta\\_arch\), 89](#)  
[UniterPretrain \(class in xmodaler.modeling.meta\\_arch\), 89](#)  
[unwrap\\_model\(\) \(in module xmodaler.utils.comm\), 99](#)  
[update\(\) \(xmodaler.config.CfgNode method\), 21](#)  
[update\(\) \(xmodaler.utils.logger.Counter method\), 105](#)  
[update\\_stats\(\) \(xmodaler.engine.PreciseBN method\), 37](#)  
[UpDownDecoder \(class in xmodaler.modeling.decoder\), 58](#)  
[UpDownEncoder \(class in xmodaler.modeling.encoder\), 71](#)  
[upgrade\\_config\(\) \(in module xmodaler.config\), 21](#)

**V**

[values\(\) \(xmodaler.config.CfgNode method\), 21](#)  
[VCRDataset \(class in xmodaler.datasets\), 25](#)  
[VCREvaler \(class in xmodaler.evaluation\), 46](#)  
[VCRTrainer \(class in xmodaler.engine\), 44](#)  
[VisualBaseEmbedding \(class in xmodaler.modeling.embedding\), 66](#)  
[VisualIdentityEmbedding \(class in xmodaler.modeling.embedding\), 67](#)  
[vocab\\_files\\_names \(xmodaler.tokenization.BertTokenizer attribute\), 97](#)

**W**

[WarmupConstant \(class in xmodaler.lr\\_scheduler\), 51](#)  
[WarmupCosine \(class in xmodaler.lr\\_scheduler\), 52](#)  
[WarmupCosineWithHardRestarts \(class in xmodaler.lr\\_scheduler\), 53](#)  
[WarmupLinear \(class in xmodaler.lr\\_scheduler\), 52](#)  
[WarmupMultiStepLR \(class in xmodaler.lr\\_scheduler\), 54](#)  
[write\(\) \(xmodaler.utils.events.CommonMetricPrinter method\), 101](#)  
[write\(\) \(xmodaler.utils.events.JSONWriter method\), 100](#)  
[write\(\) \(xmodaler.utils.events.TensorboardXWriter method\), 101](#)

**X**

[XLANDecoder \(class in xmodaler.modeling.decoder\), 63](#)  
[xmodaler.tokenization module, 95](#)  
[XmodalerCheckpoint \(class in xmodaler.checkpoint\), 15](#)

