**XrayGPT Training Code Tracing**

Step 1: Entry Point & Imports

**train.py**

**task = tasks.setup\_task(cfg)**

Calls : xraygpt/tasks/\_\_init\_\_.py :: setup\_task()

Step 2 : Task Setup

**xraygpt/tasks/\_\_init\_\_.py**

**task = registry.get\_task\_class(task\_name).setup\_task(cfg=cfg)**

Calls : xraygpt/common/registry.py :: get\_task\_class()

Returns : ImageTextPretrainTask (registered in image\_text\_pretrain.py)

Step 3 : Task Registration

**xraygpt/tasks/image\_text\_pretrain.py**

**@registry.register\_task("image\_text\_pretrain")**

Registers: ImageTextPretrainTask class in registry

Step 4 : Dataset Building

**train.py**

**datasets = task.build\_datasets(cfg)**

Calls : xraygpt/tasks/base\_task.py :: build\_datasets()

**xraygpt/tasks/base\_task.py**

**builder = registry.get\_builder\_class(name)(dataset\_config)**

Calls : xraygpt/common/registry.py :: get\_builder\_class()

Returns : MIMICBuilder or OpenIBuilder

Step 5 : Dataset Builder Registration

**xraygpt/datasets/builders/image\_text\_pair\_builder.py**

@registry.register\_builder("mimic")

Registers : MIMICBuilder class

@registry.register\_builder("openi")

Registers : OpenIBuilder class

Step 6 : Dataset Creation

**xraygpt/datasets/builders/image\_text\_pair\_builder.py**

datasets['train'] = dataset\_cls(...)

Calls : xraygpt/datasets/datasets/mimic\_dataset.py :: MIMICDataset.\_\_init\_\_()

**xraygpt/datasets/datasets/mimic\_dataset.py**

class MIMICDataset(CaptionDataset)

**def \_\_getitem\_\_(self, index)**

Called by: DataLoader during training

Step 7 : Model Building

**train.py**

**model = task.build\_model(cfg)**

Calls : xraygpt/tasks/base\_task.py :: build\_model()

**xraygpt/tasks/base\_task.py**

model\_cls = registry.get\_model\_class(model\_config.arch)

Calls : xraygpt/common/registry.py::get\_model\_class()

Returns: MiniGPT4 class

Step 8 : Model Registration

**xraygpt/models/mini\_gpt4.py**

@registry.register\_model("mini\_gpt4")

Registers : MiniGPT4 class in registry

Step 9 : Runner Creation

**train.py**

runner\_cls = registry.get\_runner\_class(cfg.run\_cfg.get("runner","runner\_base")) Calls: **xraygpt/common/registry.py** :: get\_runner\_class()

Returns : RunnerBase class

runner = get\_runner\_class(cfg)(...)

Calls : xraygpt/runners/runner\_base.py :: RunnerBase.\_\_init\_\_()

Step 10 : Runner Registration

**train.py**

**runner.train()**

Calls : xraygpt/runners/runner\_base.py :: train()

**xraygpt/runners/runner\_base.py**

**train\_stats = self.train\_epoch(cur\_epoch)**

Calls : xraygpt/runners/runner\_base.py :: train\_epoch()

**return self.task.train\_epoch(...)**

Calls: xraygpt/tasks/base\_task.py :: train\_epoch()

**samples = next(data\_loader)**

Calls : xraygpt/datasets/datasets/mimic\_dataset.py :: \_\_getitem\_\_()

Step 12 : Inner Training Loop

**xraygpt/tasks/base\_task.py**

return self.\_train\_inner\_loop(...)

Calls : xraygpt/tasks/base\_task.py :: \_train\_inner\_loop()

**image = self.vis\_processor(image)**

Calls : xraygpt/processors/blip\_processors.py :: Blip2ImageTrainProcessor.\_\_call\_\_()

Calls : xraygpt/processors/blip\_processors.py :: Blip2CaptionProcessor()

Step 13 : Data Preparation

**xraygpt/tasks/base\_task.py**

**samples = prepare\_sample(samples, cuda\_enabled=cuda\_enabled)**

Calls: xraygpt/datasets/data\_utils.py : : prepare\_sample()

samples = move\_to\_cuda(samples)

Calls : xraygpt/datasets/data\_utils.py :: move\_to\_cuda()

Step 14 : Forward Pass

**xraygpt/tasks/base\_task.py**

**loss = self.train\_step(model=model, samples=samples)**

Calls : xraygpt/tasks/base\_task.py :: train\_step()

**loss = model(samples)["loss"]**

Calls : xraygpt/models/mini\_gpt4.py :: forward()

**xraygpt/models/mini\_gpt4.py**

**img\_embeds, atts\_img = self.encode\_img(image)**

Calls: xraygpt/models/mini\_gpt4.py :: encode\_img()

**prompt = random.choice(self.prompt\_list)**

Uses : prompts/alignment.txt (loaded in \_\_init\_\_)

**img\_embeds, atts\_img = self.prompt\_wrap(img\_embeds, atts\_img, prompt)**

Calls : xraygpt/models/mini\_gpt4.py :: prompt\_wrap()

**to\_regress\_tokens = self.llama\_tokenizer(...)**

Calls : LlamaTokenizer.from\_pretrained() (HuggingFace)

**outputs = self.llama\_model(...)**

Calls : LlamaForCausalLM.forward() (HuggingFace)