

SUPERVISED FINE-TUNING

INF385T: Post-Training of LLMs | Week 5

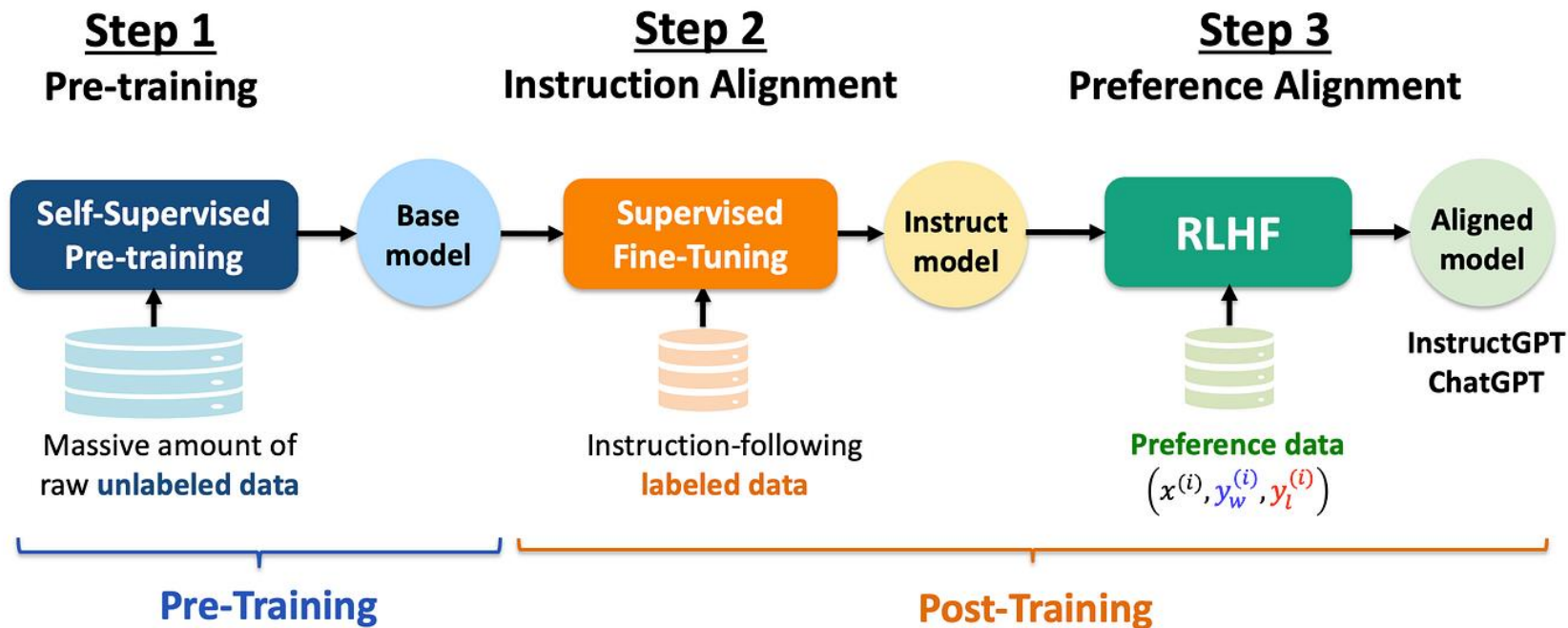
CHAN YOUNG PARK

Assistant Professor, iSchool, School of Computing, The University of Texas at Austin

Recap: what did Week 1-4 establish?

- Week 1: intro to post-training. define behavior
 - Week 2: build a strong baseline (prompting)
 - Week 3: design benchmark and metrics for claims
 - Week 4: validate the scorer/judges
- evaluation contract (how and which model to compare)

Week 5: SFT



SFT: teaching through demonstrations

USER Rewrite this in exactly two sentences.

ASSISTANT Here is a two-sentence rewrite ...

- Increase the likelihood of the demonstrated assistant tokens
 - Repeated examples can make direct answers, requested formats, and turn boundaries more probable

Central Q: What exactly happens when we fine tune an instruction model?

A short historical bridge



**The big change was not inventing a new loss
it was changing what behavior the LM is trained to model**

Five questions for today

Data

example
pairs

Render

model
interface

Labels

supervised
tokens

Update

loss and
gradients

Generate

held out
outputs

Q2. What token
sequence does
the model see?

Q3. Which
tokens contribute
to the loss?

Q1. What
objective are we
optimizing?

Q4. what
engineering
choices control
the run?

Q5. how do we
know the fine-
tuning worked?

Q1. What objective are we optimizing?

OBJECTIVE

Causal language modeling objective

Explain

gradient

descent

simply

Loss
 minimized during training

$$L_{LM}(\theta) = - \sum_t \log p_{\theta}(x_t \mid x_{<t})$$

Parameters
 model being trained

Update
 raise the target token probability

Context
 Every token before position t

Target token
 The observed token at position t

Conditional SFT objective

$$\mathcal{L}_{\text{SFT}} = - \sum_{t \in \mathcal{T}_{\text{target}}} \log p_{\theta}(y_t \mid x, y_{<t})$$

Target token positions
which positions receive supervision

Response
model response generated so far

Context
System message, instruction, user input, and prior turns

- Same next-token prediction
- Different supervised sequence + target positions

Teacher forcing

TRAINING

Ground-truth prefix

The

answer

is



Paris

Each prediction conditions on gold previous tokens.

INFERENCE

Model-generated prefix

The

answer

might



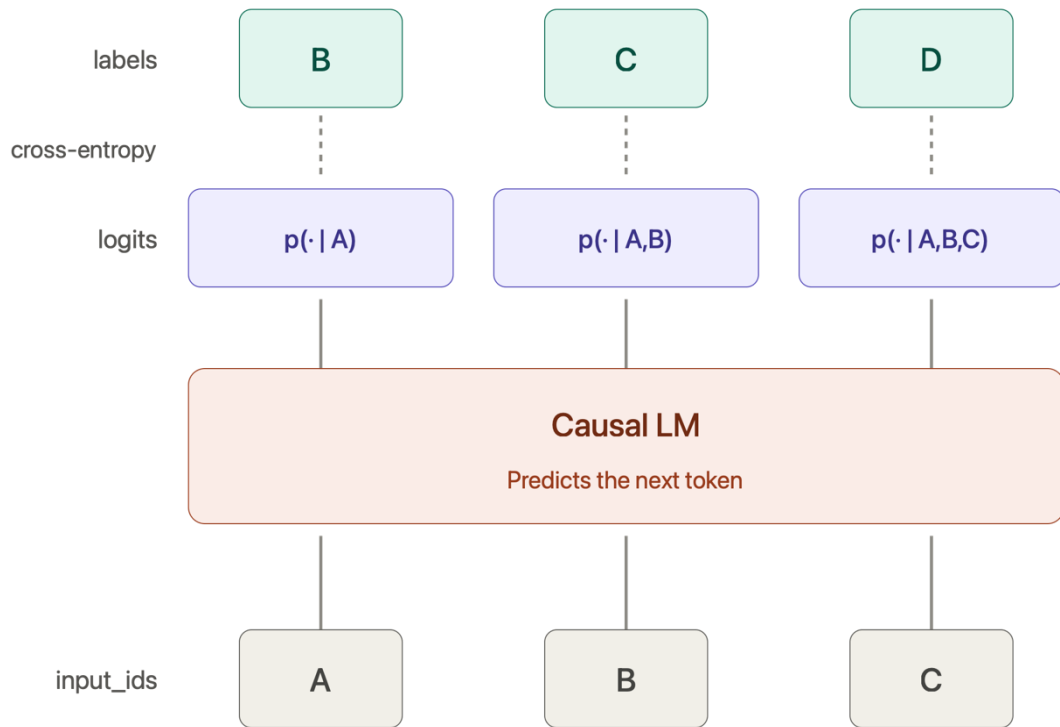
be...

Errors become part of the next context.

training $\neq$ free-running generation

Low teacher forcing is evidence of good generation

Causal shift inside the loss



```

input_ids:  [ A, B, C, D ]
logits_used: [ A, B, C   ]
labels:     [ B, C, D   ]
  
```

Causal shift inside the loss

```
input_ids:  [ A, B, C, D ]
logits_used: [ A, B, C   ]
labels:     [ B, C, D   ]
```

- Transformers usually shifts logits and labels inside the causal LM loss
- A custom loop can shift twice. Verify the model API

```
inputs = tokenizer("A B C D")
labels = inputs["input_ids"][:, 1:]
input_ids= inputs["input_ids"][:, :-1]
outputs = model(input_ids=input_ids, labels=labels)
```



```
inputs = tokenizer("A B C D")
labels = inputs["input_ids"]
input_ids= inputs["input_ids"]
outputs = model(input_ids=input_ids, labels=labels)
```



Q2. What token sequence does the model see?

CHAT TEMPLATES

Models see structured chat

Python / dataset view

```
[  
  {"role": "system", "content":  
"Be concise."},  
  {"role": "user", "content":  
"Explain GD."},  
  {"role": "assistant", "content":  
"Gradient descent..."}  
]
```



What the causal LM sees

```
<|system|> Be concise. <|end|>  
<|user|> Explain GD. <|end|>  
<|assistant|> Gradient descent...  
<|end|>
```

then → token IDs → labels → loss

A chat template is a serialization rule

messages $\longrightarrow$ template $\longrightarrow$ rendered sequence

ChatML (Chat Markup Language)

```
<|im_start|>system  
You are helpful.<|im_end|>  
<|im_start|>user  
Explain gradient descent.<|im_end|>  
<|im_start|>assistant  
Gradient descent is ...<|im_end|>
```

- Role boundaries are tokens
- End-of-message / EOS behavior is encoded
- Assistant-start header can cue generation

Why can the template change behavior?

Matched interface

```
<|user|> ... <|end|>  
<|assistant|>
```

- role boundary is familiar
- assistant-start cue is in distribution
- stopping tokens match training

Mismatched interface

```
### Human: ...  
### AI:
```

- unfamiliar separators
- possible role leakage / continuation
- different stopping behavior

Chat formatting is part of the learned interface
Train-inference consistency matters

Base checkpoint vs. instruct checkpoint

Starting from a BASE model

- No strong assistant protocol may exist yet
- SFT can teach role markers + interaction pattern
- You can choose your own coherent serialization scheme

Starting from an INSTRUCT model

- An interaction protocol is already learned
- Usually preserve its expected template
- Changing it adds avoidable distribution shift

“Use the correct template” = respect the interface your checkpoint has learned

Q3. Which tokens contribute to the loss?

TRAINING

Which tokens get loss

SYSTEM

You are concise.

USER

Explain GD.

ASSISTANT

Gradient descent...

<eos>

Option A predict everything

Option B predict assistant only

Both options use the same model
However, the supervision budget is allocated differently

From tokens to labels

input_ids

<sys>	Be	concise	<user>	Explain	GD	<asst>	Gradient	descent	<eos>
101	322	891	204	553	88	411	927	663	2

labels

-100	-100	-100	-100	-100	-100	-100	927	663	2
------	------	------	------	------	------	------	-----	-----	---

supervised positions

In common HF/PyTorch setups,
 label = -100 means “ignore this token in cross-entropy”

Full-sequence vs. assistant-only loss

Full-sequence loss

$$\mathcal{L} = \mathcal{L}_{sys} + \mathcal{L}_{user} + \mathcal{L}_{assistant}$$

- Learn to model context tokens too
- Simple standard LM objective
- Can mix adaptation + response learning

Assistant-only loss

$$\mathcal{L} = \mathcal{L}_{assistant|sys,user}$$

- Context is conditioning, not target
- Focuses supervision on generated role
- Useful when context $\gg$ answer

Masking is an objective-design choice and not a universal rule

Why mask context? Think in token budgets

Document / context

Question

Answer

If every token gets equal loss weight...

≈ 95% of supervised positions are not the answer.

Assistant-only objective

Spend the token-level supervision on what the model should generate.

But if domain modeling of the context is itself useful,
full-sequence loss may be intentional and helpful

Q4. What engineering choices control the run?

HYPER-PARAMETERS

What you configure in an SFT run

Checkpoint base / instruct

Serialization template + special tokens

Objective labels / masking

Context max length + truncation

Optimization LR + scheduler

Batching microbatch + accumulation

Duration epochs / steps

Efficiency precision + checkpointing

Evaluation split + metrics

Tracking logs + checkpoints

Learning rate (the most important number)

too low

useful range

too high

Too low

- slow/minimal behavioral change
- wasted compute

Reasonable

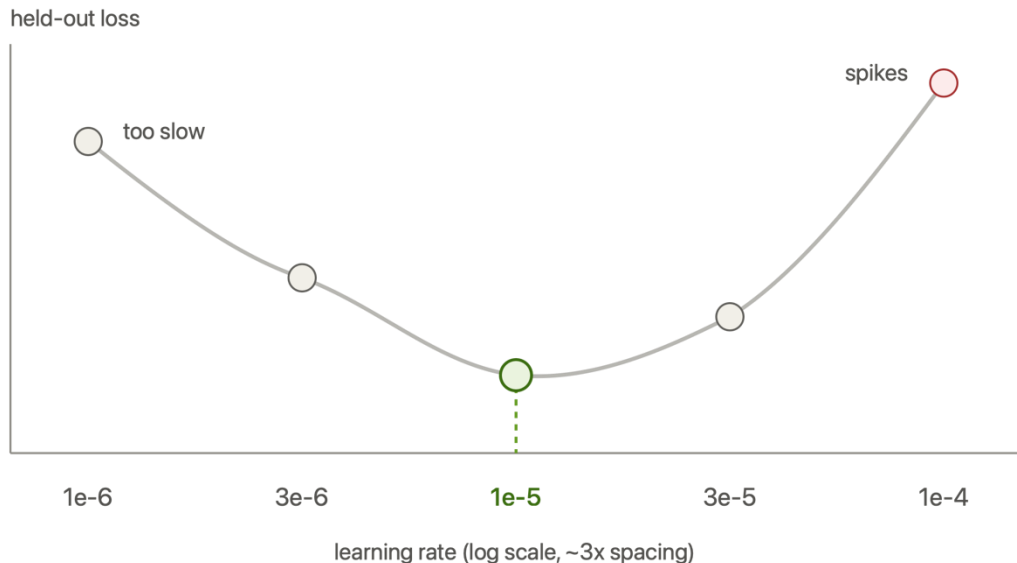
- steady loss decrease
- held-out metrics improve

Too high

- instability / spikes
- forgetting or overshooting

No universal “SFT learning rate”
model size, optimizer, batch, dataset, and full-FT vs. PEFT all matter.

Then how do I set it?

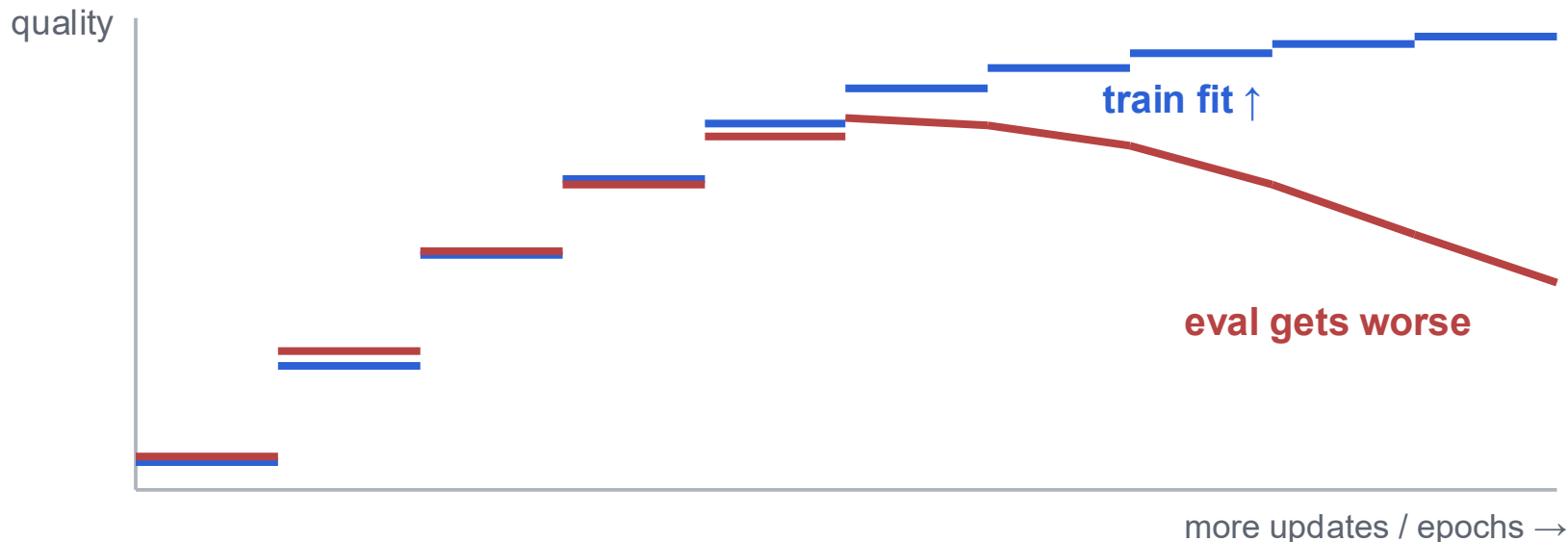


- **Start from a similar recipe.** Look for a published SFT run with similar settings (model family, size, training method, dataset size)
- **Typical first guesses**
 - Full FT, ~7B: 1e-5 to 2e-5 (Alpaca SFT used 2e-5)
 - Full FT, larger models: go lower, roughly 1e-6 to 1e-5
 - LoRA / QLoRA: 1e-4 to 2e-4, about 10x higher than full FT
 - Warmup ~3% of steps, then decay
- **Run a short sweep.** 3 to 5 values on a log scale around your starting point short runs on a slice of training

Epochs/Steps

SFT often uses only a few passes (typically 1-3 epochs)

Alpaca: 3 epochs (52k), llama 2: 2 epochs (27.5k), Tulu 3: 2 epochs (940k)



Strong initialization means “more training” is not automatically better

Effective batch size

- what one optimizer step sees

$$B_{\text{eff}} = B_{\text{device}} \times N_{\text{devices}} \times N_{\text{accum}}$$

per batch
4 sequences

GPUs
2

grad accumulation
8 steps

effective batch
64 sequences

- LR and step count depend on B_{eff} , not per-GPU batch
- Match B_{eff} when copying a recipe and re-check it when you change GPUs.

Sequence length

- Always inspect length distributions and rendered examples before training
- Three failure modes

Memory / compute

Attention and activations grow with sequence length.

Truncation

A run can silently cut off the answer or later turns.

“Training ran” ≠ “target survived preprocessing”

Padding waste

Short examples can waste most tokens without packing.

Real-world SFT examples

Qwen2.5

> 1 million SFT samples

then multistage reinforcement learning

- open-weight base + instruct families
- post-training improves instruction following and structured generation
- objective is familiar; pipeline is industrial-scale

Tülu 3



- complete open post-training recipe
- evaluation and decontamination emphasized
- SFT remains the first adaptation stage

Q5. How do we know the fine-tuning worked?

EVALUATION

What changes after SFT?

Behavior / interface

Interface answers instead of generic continuation

Instruction following obeys format / task request

Style verbosity, tone, structure

Protocol roles, tool-call syntax, refusal pattern

Capability

Task skill specific mapping can improve

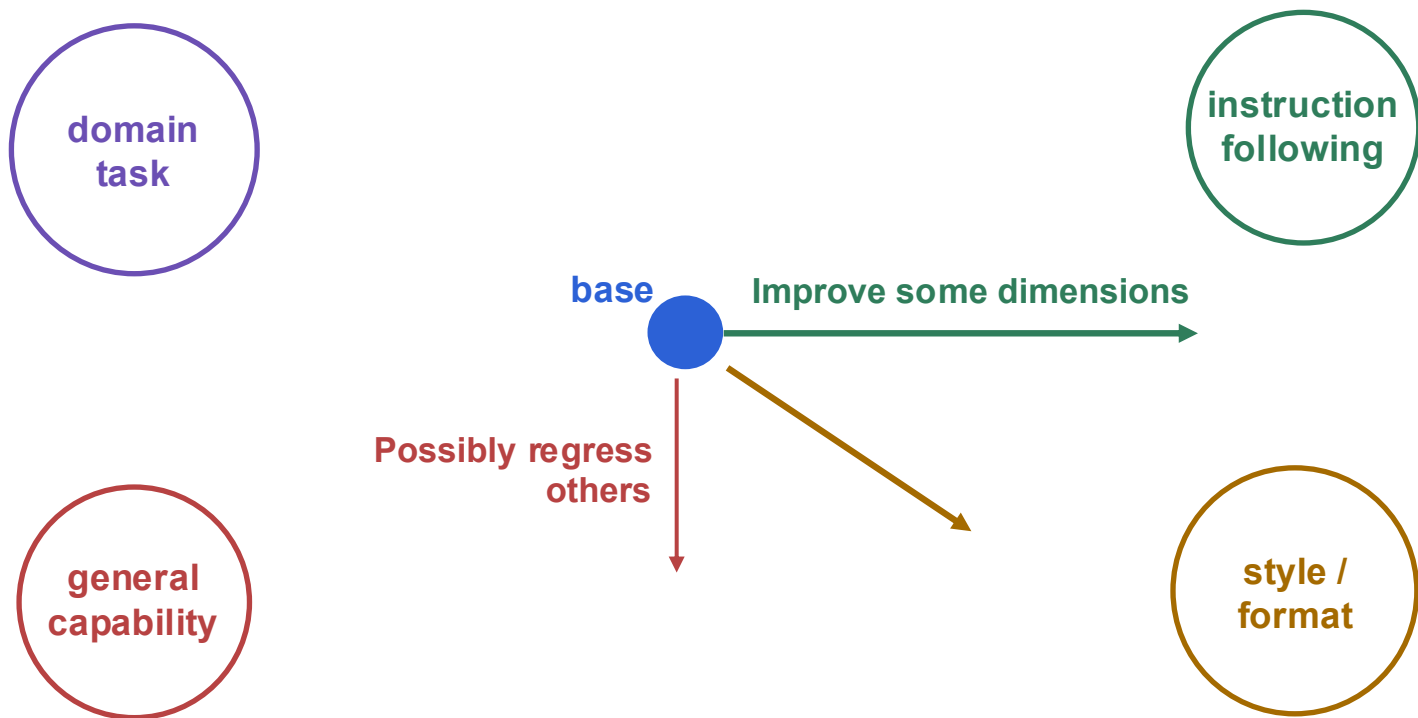
Elicitation latent capability becomes easier to invoke

Knowledge some facts/patterns can be learned

Regressions unrelated behaviors can get worse

It reshapes conditional behavior over the training distribution

Fine-tuning is not monotonic improvement



Evidence of training success

1

Optimization

Did the intended objective decrease?

2

Held-out task

Did behavior improve on unseen examples?

3

Generation

Does free-running output look right?

You need to check all three

What should you monitor during training?

- Training loss: Is optimization happening?
- Eval loss: Does held-out likelihood improve?
- Learning rate: Where are we on the schedule?
- Grad norm: Any instability / spikes?
- Tokens/step: How much supervision processed?

Experiment tracking

- Treat every run as an experiment
- Log enough to reconstruct what happened
- What should be attached to a run?
 - model checkpoint
 - hyperparameters
 - data version / split
 - metrics + curves
 - checkpoint / step
 - notes / code revision

Good loss $\neq$ good behavior

Model A

eval CE loss **1.12**

Output

```
Sure! Here is the answer:  
{ intent: correct_item }  
Hope that helps!
```

INVALID FORMAT

Model B

eval CE loss **1.18**

Output

```
INTENT: correct_item  
ITEM: backpack  
REQUEST: exchange
```

VALID FORMAT + CORRECT

**Evaluate the behavior you actually care about
(pass@k, label accuracy, LLM/human eval)**

Before you fine-tune your own model

- Consider the following questions

- 1 What behavior do I want to change?
- 2 What checkpoint and interaction protocol am I starting from?
- 3 What exact sequence will the model see?
- 4 Which tokens receive supervision?
- 5 Could important targets be truncated?
- 6 What optimization budget will I use?
- 7 What will I monitor while training?
- 8 What held-out metric and generations define success?
- 9 What baseline / checkpoint will I compare against?

LAB 5

Lab 5: Inspect and run full SFT

- Fine-tune a 0.5B instruct model to compress a short passage into exactly 3 bullet points
- The provided corpus contains short community announcements
- 5 Tasks
 - Task 1: inspect the data and understand the task
 - Task 2: inspect the rendered inputs and truncation
 - Task 3: inspect inputs, labels, logits, and loss
 - Task 4: run full SFT
 - Task 5: judge the result three ways (training, validation loss, and free generations before/after training). Decide whether they agree.