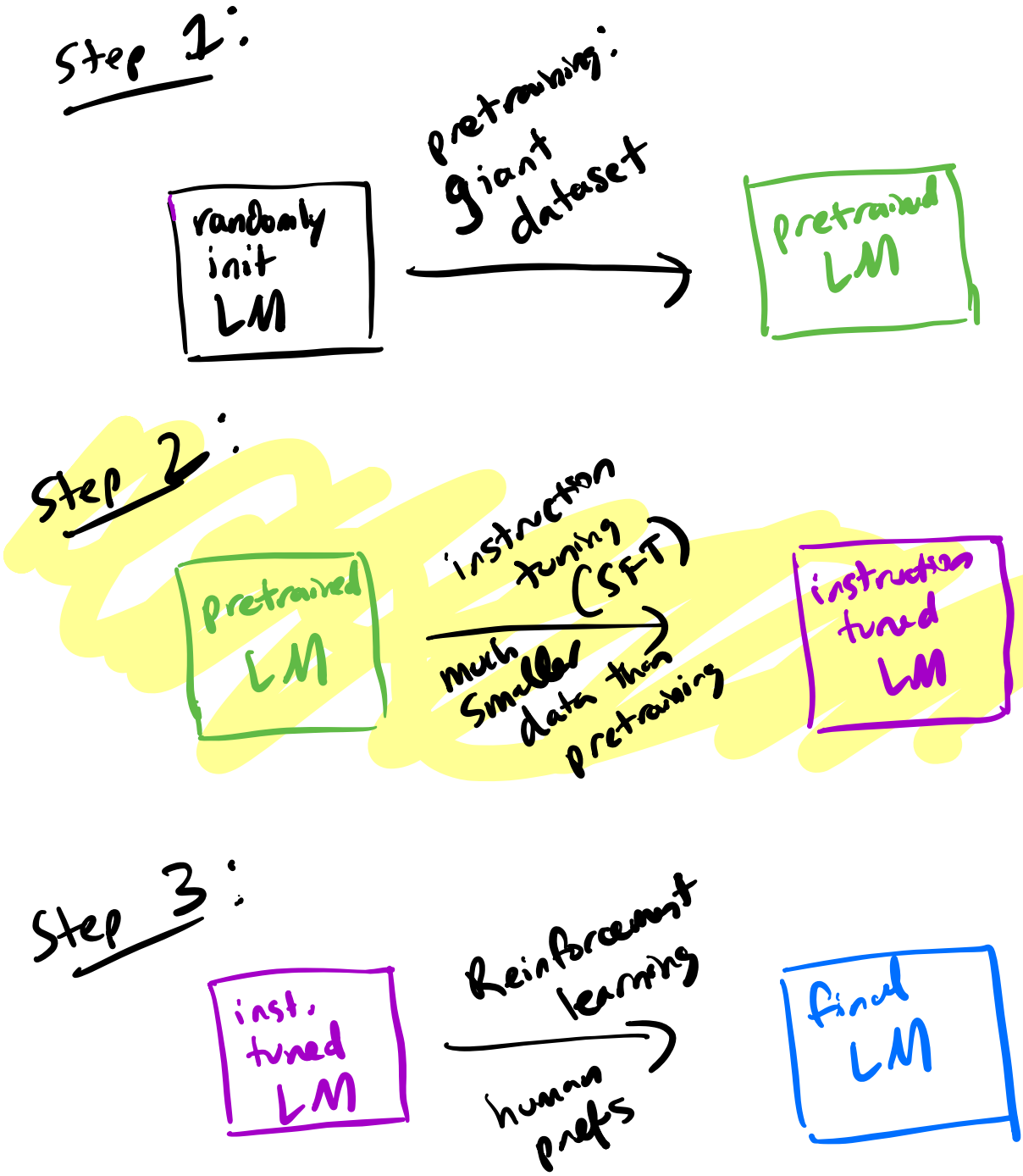
Instruction tuning / PEFT:

Pre-exam: pretraining

- ↳ huge-scale data
- ↳ giant models
- ↳ good next word predictor
- ↳ not so good: following instructions

Focus: post-training

- ↳ goal: make pretrained model follow user instructions better
- ↳ subgoal: make model "aligned" w/ human values
  - ↳ helpfulness vs. harmlessness


Step 1:
randomly init LM
pretraining: giant dataset
pretrained LM
Step 2:
pretrained LM
instruction tuning (SFT)
much smaller data than pretraining
instruction tuned LM
Step 3:
inst. tuned LM
Reinforcement learning
human prefs
final LM

Supervised fine-tuning (SFT):

↳ collect a dataset of input/output pairs for a specific task

15 + 128
143
1272 + 27
1299
⋮

↳ train model on dataset to minimize NLL on output tokens

↳ prefix LM mask

↳ basically same as pretraining

↳ when doing SFT, you risk "catastrophic forgetting"

↳ instruction tuning

↳ same as SFT, but data contains huge diversity of tasks

<instruction> → "add these two numbers"
<input context> → "127 + 13"
<output> → 140

ex. 1

"Solve my HW2"
"HW2.pdf"
Python code for HW2

↳ Tulu 3

problem: SFT on huge pretrained models is expensive

↳ adjusting billions of params thru backprop

parameter-efficient fine-tuning (PEFT)

↳ reduce # of params adjusted during FT

↳ prompting: no params are adjusted

↳ control behavior by modifying prompt

↳ demonstrations

↳ instruction
input
output

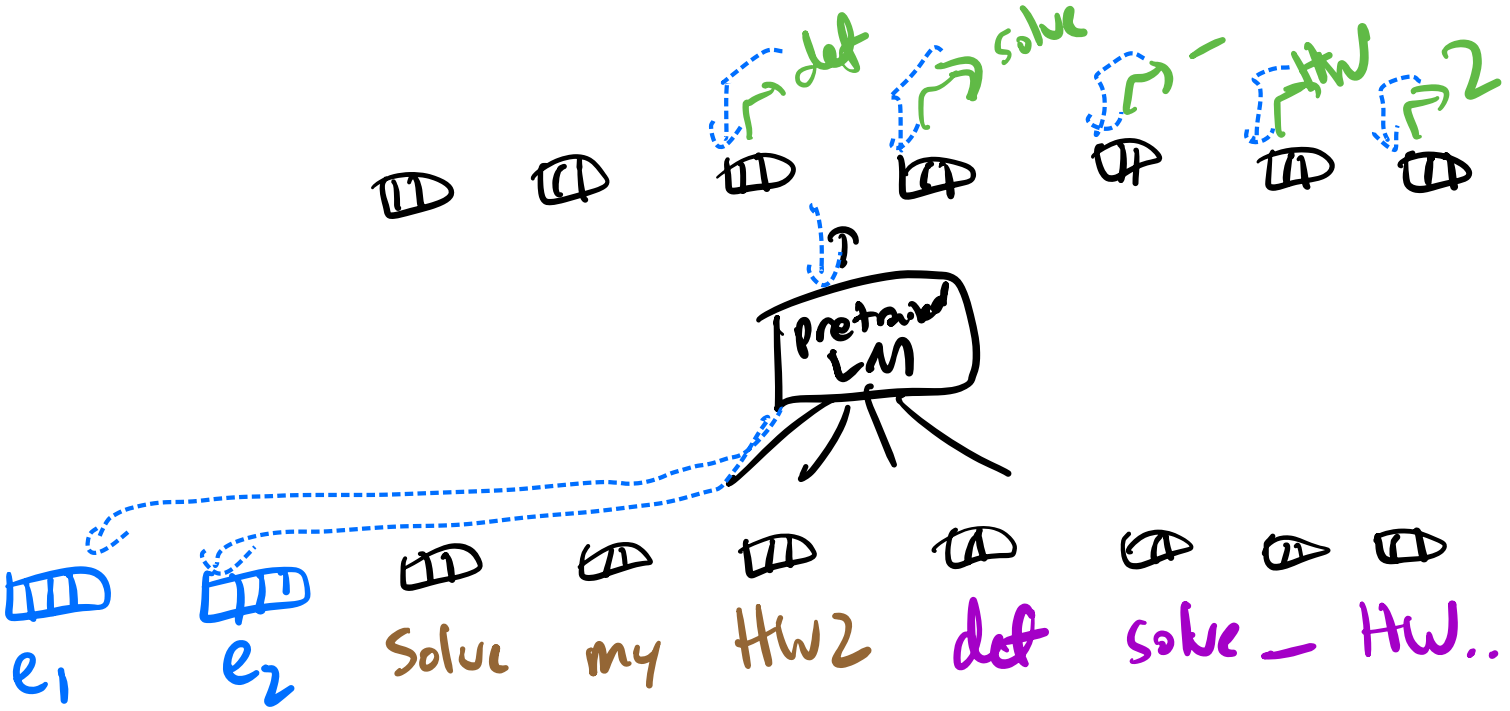
↳ fails for complex tasks

↳ requires extremely large pretrained models

↳ prompt tuning:

↳ freeze entire model

↳ add small # of new params that are tld.



only compute $\frac{dL}{de_1}, \frac{dL}{de_2}$

$$e_{1_{new}} = e_{1_{old}} - \eta \frac{dL}{de_1} \quad \dots$$

e_1, e_2 define all the new task(s)

↳ LoRA: low-rank adaptation

$h = f(Wx) \Rightarrow$ linear layer

W: $m \times n$ $\quad \frac{dL}{dW}$ is also $m \times n$

$$W_{new} = \underbrace{W_{old}}_{m \times n} - \eta \underbrace{\frac{dL}{dW}}_{m \times n}$$

idea: what if we factorize W into two low-rank matrices $\underbrace{A}_{m \times r}$ and $\underbrace{B}_{n \times r}$

$r \Rightarrow$ rank

$r \ll\ll\ll\ll m, n$

matrix product: AB^T is also $m \times n$

in LoRA:

$$h = f(Wx) \quad \text{becomes} \quad h = f\left(\left(W_{\text{pretrained}} + AB^T\right)x\right)$$

learn

$W_{\text{pretrained}}$: frozen

AB^T: learned additive offset

compute $\frac{dL}{dA}$, $\frac{dL}{dB}$

$\frac{dL}{dA}$: $m \times r$; $\frac{dL}{dB}$: $n \times r$

these are much smaller than $\frac{dL}{dW}$

↳ Separate A, B for each weight matrix in our model

↳ typically only W_q, W_k, W_v

test-time:

$$W_{FT} = W_{Pretrained} + AB^T$$

$$h = f(W_{FT} x)$$

↳ still have to fit $W_{Pretrained}$ in memory

quantization:

↳ intuition: rounding floats to ints and scaling

$$x = [-1.0, 0.5, 2.0]$$

4-bit = 16 diff. values

[0-15] integers

1. $x_{min} = -1.0$

   $x_{max} = 2.0$

   Scale $s = \frac{x_{max} - x_{min}}{15} = \frac{3}{15} = 0.2$

Zero-point $z = \text{round}\left(0 - \frac{x_{min}}{s}\right)$

$= 0 - \frac{-1.0}{0.2} = 5$

$q = \text{round}\left(\frac{x}{s} + z\right)$

$\hat{x} = s \cdot (q - z)$

$x = [-1.0, 0.5, 2.0]$

$q = [0, 8, 15]$

$\hat{x} = [-1.0, 0.6, 2.0]$