

CSE/DSC 234

Data-Centric AI and AI Engineering

Arun Kumar

Topic 4: Model Customization

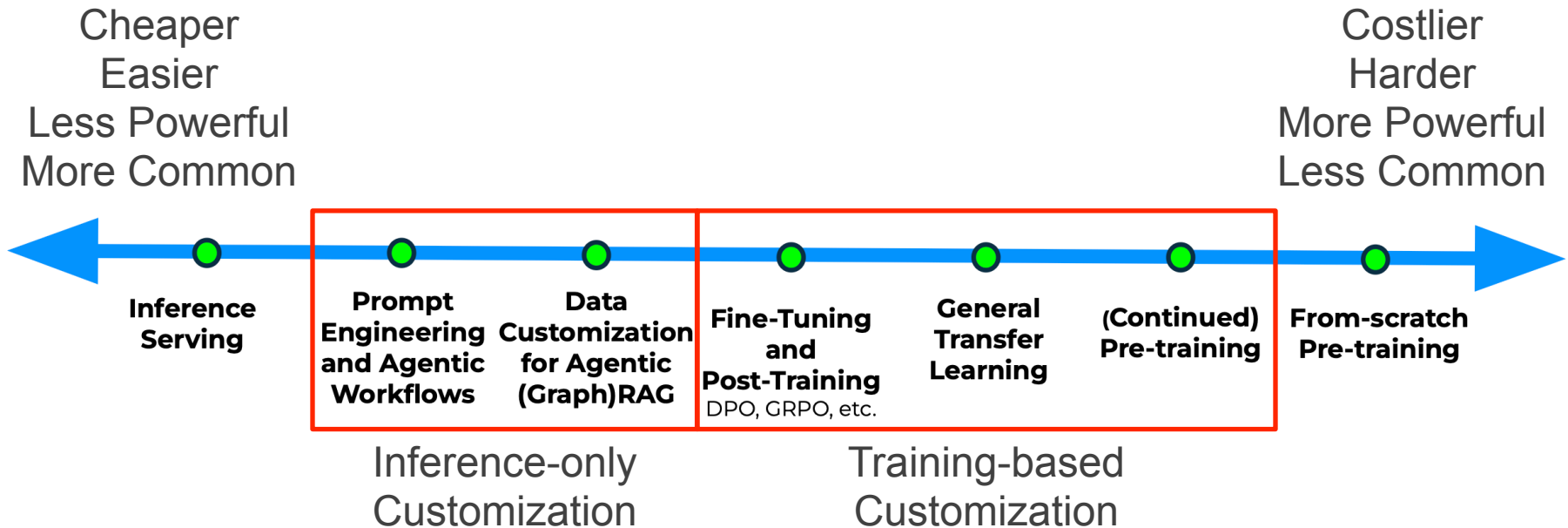
Outline

- ➔ ❖ LLM Customization Spectrum
 - ❖ Continued Pre-Training (CPT)
 - ❖ Supervised Fine-Tuning (SFT)
 - ❖ LoRA
 - ❖ Post-Training/Reinforcement Fine-Tuning (RFT)

Why Customize?

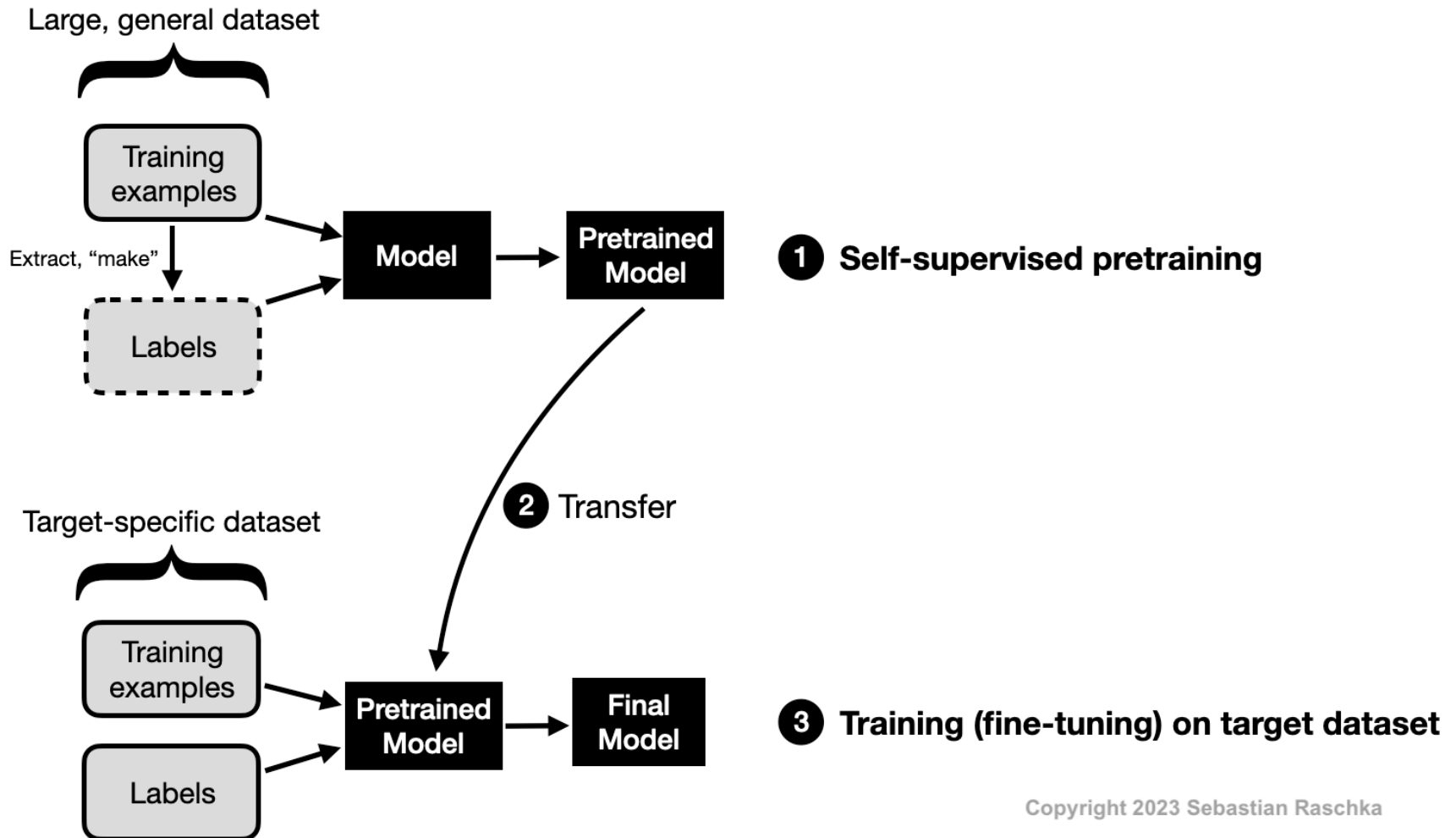
- ❖ Pure prompt/context/retrieval engineering can plateau on out-of-distribution data/jargon or domain-specific reasoning:
 - ❖ Highly domain-specific concepts, figures, mechanisms
 - ❖ Custom data ontologies and schemas
 - ❖ Even if base model has capability, consistent behavior hard
- ❖ Cost and latency concerns:
 - ❖ Runaway token spend and rate limiting headaches
 - ❖ Bad UX of slow chatbots
- ❖ Edge / no-Internet deployment
- ❖ Data+AI sovereignty

LLM Customization Spectrum



Model Object	Frozen	Modified
Prevalence in Practice	Dominant among enterprises	SFT also popular; others mostly at tech firms
Applicability	Specific to promptable LMs	Applicable to any DL model
Tools	Complex landscape	Mostly just PyTorch + HF

Customization is Transfer Learning



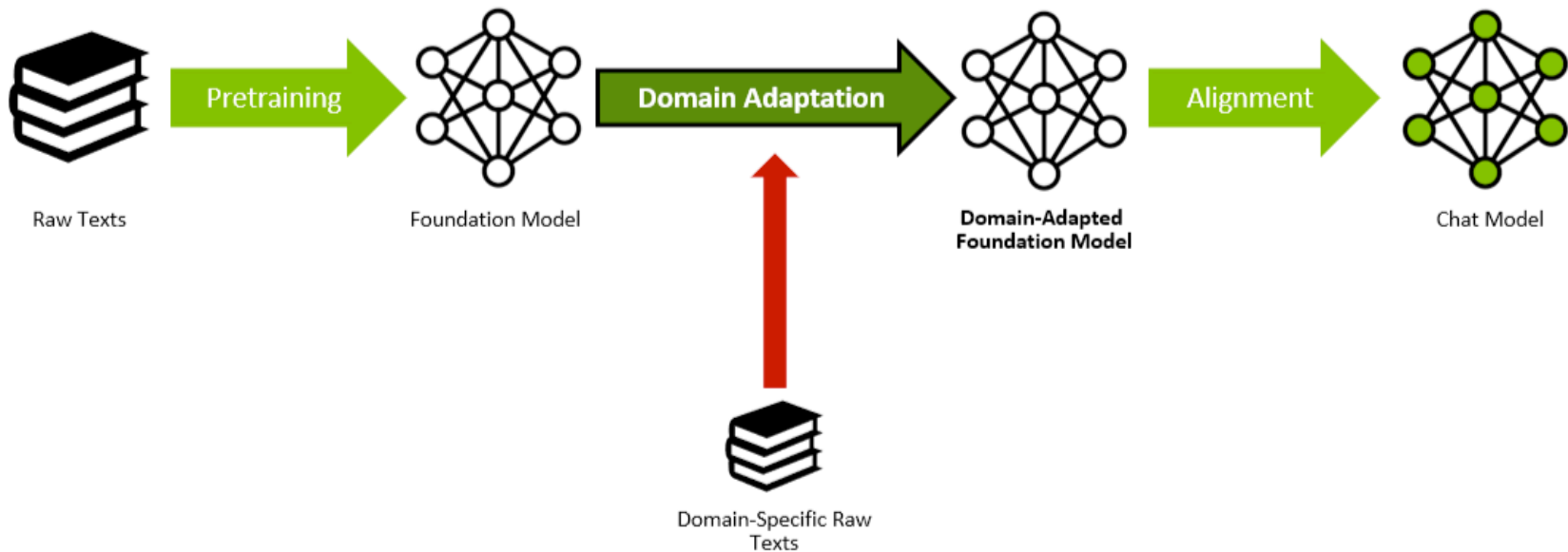
Copyright 2023 Sebastian Raschka

Outline

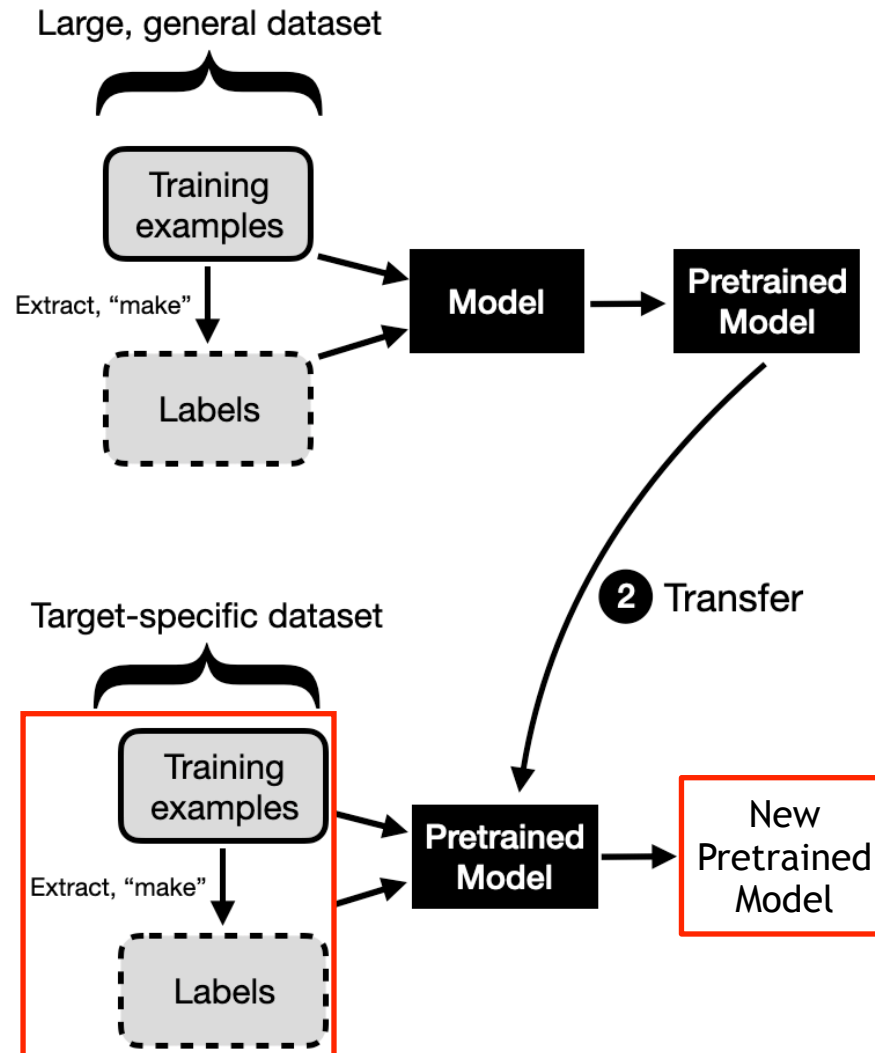
- ❖ LLM Customization Spectrum
- ➔ ❖ Continued Pre-Training (CPT)
- ❖ Supervised Fine-Tuning (SFT)
 - ❖ LoRA
- ❖ Post-Training/Reinforcement Fine-Tuning (RFT)

Continued Pretraining

- ❖ *Continue* training of pre-trained LLM on new dataset
 - ❖ No labels needed; same *self-supervision* objective function
- ❖ Adapts model's latent representation to new domain distribution



Schematic of CPT



CPT: Design Decisions and Tradeoffs

- ❖ **When it matters:** Base LLM is weak on domain's jargon, conventions, structure, e.g., legal, biomed, internal code, niche SQL dialects; teaches knowledge, not behavior
- ❖ **Key Design Decisions and Tradeoffs:**
 - ❖ **Data Mix:** Fraction of in-domain vs. general; too pure -> catastrophic forgetting; too dilute -> no meaningful shift
 - ❖ **Corpus Size:** Main cost knob; unlabeled data is massive
 - ❖ **Stopping Condition:** Evaluate on both general and domain-specific capabilities

Outline

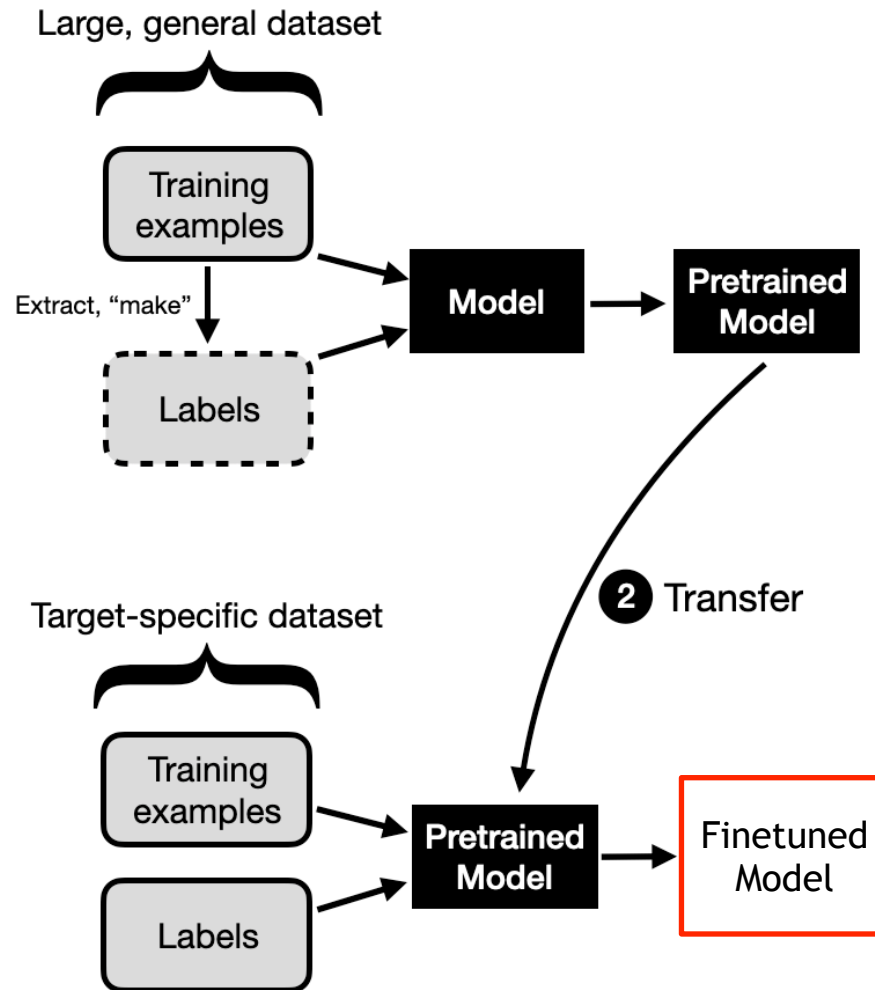
- ❖ LLM Customization Spectrum
- ❖ Continued Pre-Training (CPT)
-  ❖ Supervised Fine-Tuning (SFT)
 - ❖ LoRA
- ❖ Post-Training/Reinforcement Fine-Tuning (RFT)

Supervised Fine-Tuning (SFT)

- ❖ Workhorse of LLM customization
- ❖ Adapts model's latent representation to new domain distribution
 - ❖ But *supervised*, i.e., needs task-specific (input, output) pairs
 - ❖ Brings down generality of AGI model back to *task-specificity*
- ❖ Closer to canonical “training” from classical ML/DL
 - ❖ Same bias-variance-noise tradeoffs on training data



Schematic of SFT (Full Parameter)



SFT Data Sources

- ❖ Common practice is to get signal from appl. production logs
 - ❖ Prompting/context eng. allows you ship an MVP
- ❖ Synthesis with more powerful LLM
- ❖ SME hand-crafted and/or rule-based heuristics
- ❖ Same tradeoffs as creating evals data (Topic 1)

Outline

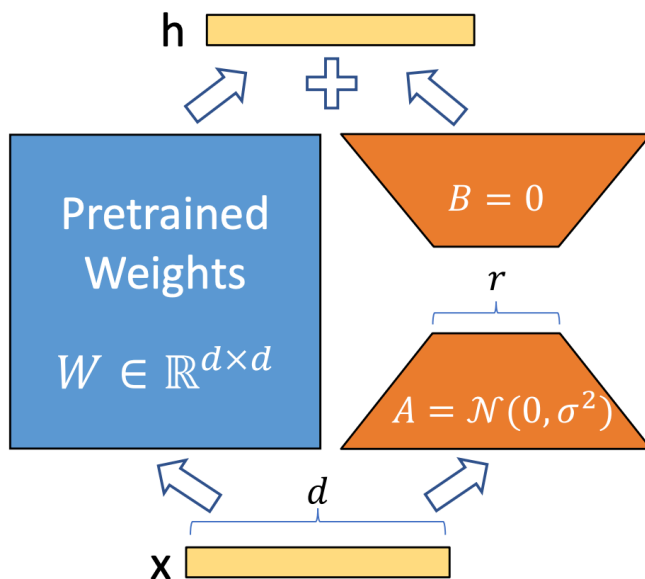
- ❖ LLM Customization Spectrum
- ❖ Continued Pre-Training (CPT)
- ❖ Supervised Fine-Tuning (SFT)
- ➔ ❖ LoRA
- ❖ Post-Training/Reinforcement Fine-Tuning (RFT)

Issues with Plain SFT

- ❖ Plain SFT (“full parameter FT”) often has pitfalls:
 - ❖ Overfitting
 - ❖ Catastrophic forgetting
 - ❖ Higher cost of training vs. inference-only customization
 - ❖ Model size, GPU cost, hosting/DevOps complexity
- ❖ One SFT run per application/task amplifies above issues

Low-Rank Adapters (LoRA)

- ❖ **Main Idea:** Freeze base LLM weights and instead train smaller set of additive parameters per layer, aka *adapters*
 - ❖ Adapters have roughly ~0.1-1% of base weights
 - ❖ Results often not that far from full FT but fraction of cost



$$\bar{B} \in \mathbb{R}^{d \times r}, \bar{A} \in \mathbb{R}^{r \times k}$$

$$h = W_0 x + \Delta W x = W_0 x + B A x$$

LoRA FT Accuracy vs. Parameters

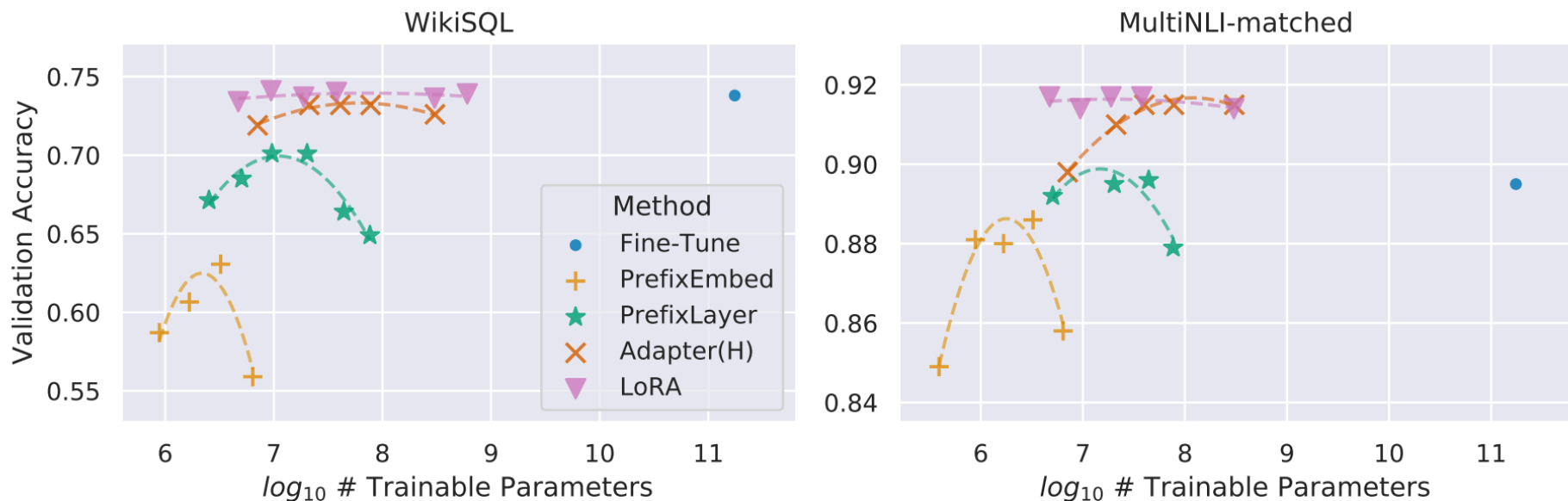
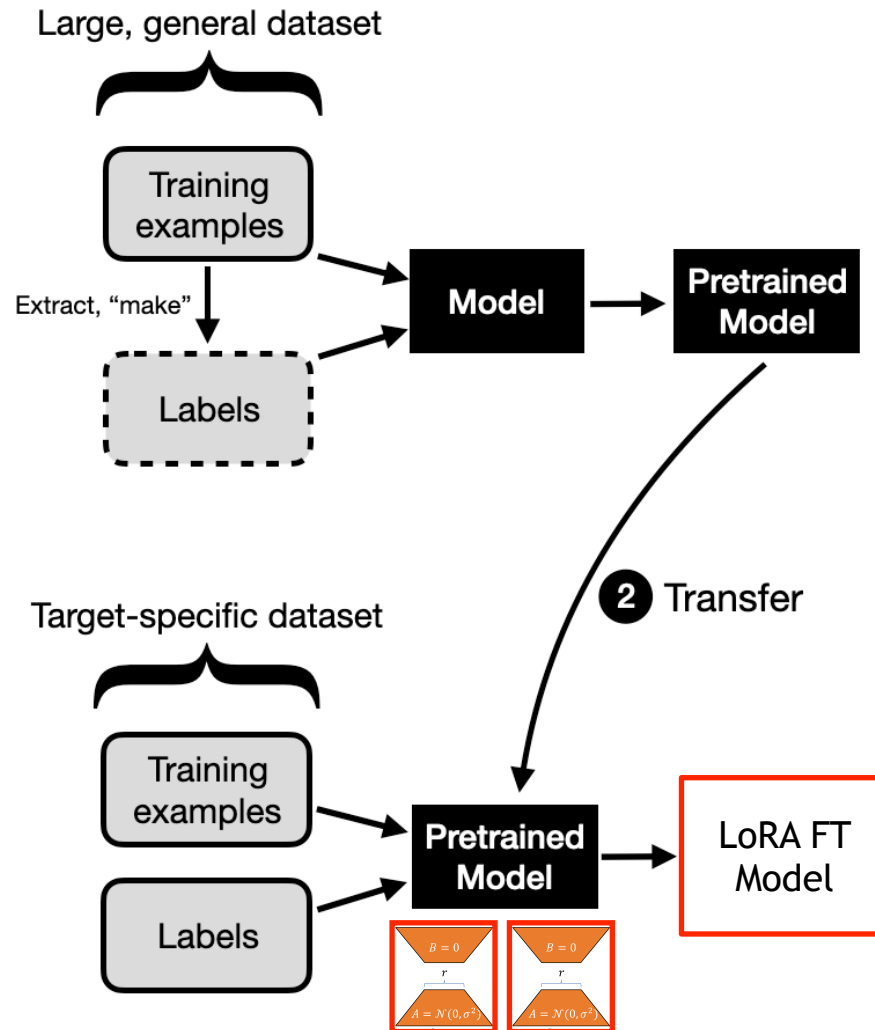


Figure 2: GPT-3 175B validation accuracy vs. number of trainable parameters of several adaptation methods on WikiSQL and MNLI-matched. LoRA exhibits better scalability and task performance. See Section F.2 for more details on the plotted data points.

Schematic of SFT (LoRA)



Advantages of LoRA

- ❖ LoRA FT has become the default method of “SFT” in practice
 - ❖ **Quantized LoRA (QLoRA):** Base model weights quantized in addition; complementary trick
- ❖ Multiple practical advantages of LoRA/QLoRA:
 - ❖ **Accuracy Benefits:** Lower risk of overfitting or forgetting
 - ❖ **Training Runtime:** Much faster than full FT (even 10x)
 - ❖ **Memory Footprint:** Smaller GPUs suffice for training
 - ❖ **Storage:** Adapters are MBs, not GBs; host base model once and numerous adapters per task or group
 - ❖ **“Multi-LoRA” Deployment:** Merge different adapters into one set of base weights at inference time on the fly

SFT: Design Decisions and Tradeoffs

- ❖ **When it matters:** Need even higher accuracy on specific tasks; lower cost and/or latency
- ❖ **Key Design Decisions and Tradeoffs:**
 - ❖ Sourcing and mixing labeled data (quality > quantity)
 - ❖ Full FT vs. LoRA
 - ❖ Traditional training tradeoffs (e.g., hyperparameter tuning)

SFT in Project 2

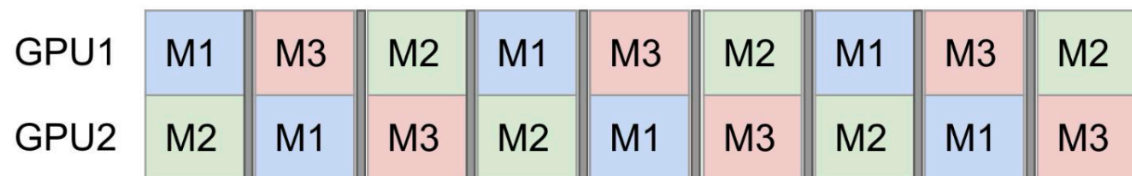
- ❖ Project 2 involves domain-specific jargon and constraints
 - ❖ Bespoke database schema identifiers, SQL queries
 - ❖ Query latency matters — human in the loop
- ❖ Key knobs to focus on:
 - ❖ **LoRA Rank:** Capacity of adapter; 8 to 64 typical
 - ❖ **Target Modules:** Which layers get adapters; usually attention projections; often defaults suffice but worth trying
 - ❖ **Learning Rate** and **LR Scheduler:** Typical SGD knobs
- ❖ Tip: Do not replicate base model over and over!

SFT with RapidFire AI

Task Parallel
(e.g., W&B, Ray Tune)



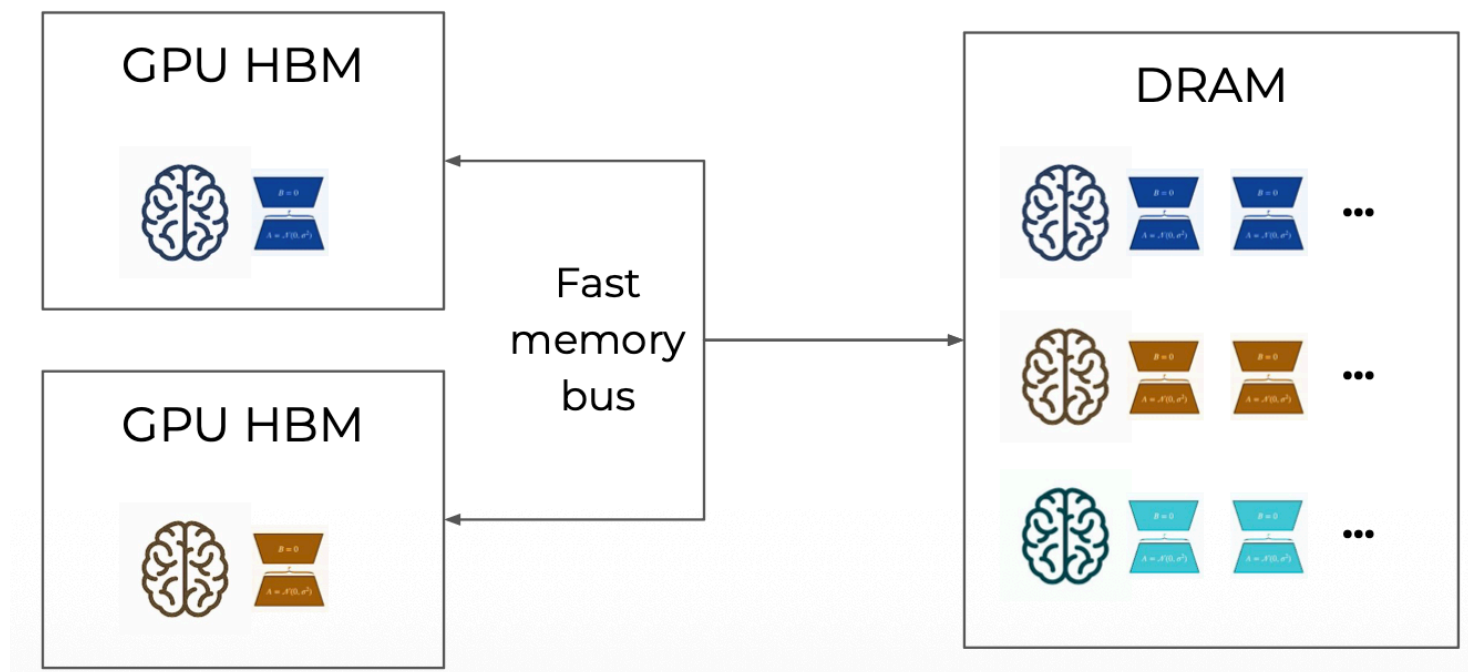
RapidFire AI
(same configs)



Key Question: How to minimize overheads of alternating across sub-tasks?

SFT with RapidFire AI

Basic Idea: Shared memory dictionary in DRAM to spill/load per-config adapters and unique base models



SFT with RapidFire AI

SFT Method	GPU	Llama-3.2-1B	Gemma-2B	Llama-3.1-8B	Llama-2-13B
Adapter Only Disk File -> Shared Mem					
LoRA with r=32	A100	1.1s -> 0.1s	1.3s -> 0.2s	3.5s -> 0.3s	5.2s -> 0.6s
	H100	1.1s -> 0.1s	1.6s -> 0.1s	3.6s -> 0.3s	5.5s -> 1.1s
Full Model Disk File -> Shared Mem					
Full Fine-tuning	A100	4.1s -> 1.7s	8.9s -> 4.1s	30.4s -> 12.7s	OOM
	H100	5.1s -> 1.9s	9.8s -> 3.4s	26.2 -> 11.4s	OOM

Result: Overheads amount to < **5%** of total runtime even for 8 chunks

Outline

- ❖ LLM Customization Spectrum
- ❖ Continued Pre-Training (CPT)
- ❖ Supervised Fine-Tuning (SFT)
 - ❖ LoRA
- ➔ ❖ Post-Training/Reinforcement Fine-Tuning (RFT)