Fine-Tuning in 2025

Who am I?

- Decade of experience with training frameworks
- Technical lead for Hugging Face Accelerate
- Tinkerer

Why do we even need to?

In many cases, APIs and prompt engineering can be enough

- Careful crafting == easy wins
- RAG/etc help fill some of those gaps too

So… is it needed?

Situtations where you need to

- **Knowledge distillation**
- Strong models != strong on your programming language

Knowledge distillation

Knowledge distillation

1. Create a task you to do, that big models are good at (bash completion)
2. Take a good **big** model off the shelf (OAI, Kimi K2, etc.) that does well
3. Create a dataset where you get the model outputs wrt the problem questions
4. Train on *both* the true outputs and the large model's outputs (a fancy loss function)

For me: 41,000 samples

Knowledge distllation

```
class DistillationLoss(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.ce_loss_fn = CrossEntropyLoss(ignore_index=-100)
        if accelerator.is_main_process:
            logger.info("Initialized DistillationLoss")

    def forward(self, student_logits, labels, topk_ids, topk_probs, ce_weight=1.0, kl_weight=
        student_logits = student_logits.float()
        topk_probs = topk_probs.float()
        B, S, V = student_logits.shape
        ce_loss = self.ce_loss_fn(student_logits.view(-1, V), labels.view(-1))

        kl_loss = 0.0
        cosine_loss = 0.0
        valid_count = 0

        for b in range(B):
            for t in range(S):
                if labels[b, t] == -100:
                    continue
                teacher_conf = topk_probs[b, t].max()
                # Skip low-confidence steps
                if teacher_conf < 0.1:
                    continue
                valid_count += 1

                # KL Loss
                student_log_probs = F.log_softmax(student_logits[b, t], dim=-1)
                teacher_dist = torch.zeros_like(student_log_probs)
                teacher_dist[topk_ids[b, t]] = topk_probs[b, t]
                kl_loss += F.kl_div(student_log_probs, teacher_dist, reduction="batchmean")


                # Cosine similarity RLKD
                student_probs = F.softmax(student_logits[b, t] / 1.5, dim=-1)
                s_vals = student_probs[topk_ids[b, t]]
                t_vals = topk_probs[b, t]
                cos_sim = F.cosine_similarity(s_vals, t_vals, dim=0)
                cosine_loss += 1 - cos_sim
```

```
        if valid_count > 0:
            kl_loss /= valid_count
            cosine_loss /= valid_count

        total_loss = ce_weight * ce_loss + kl_weight * kl_loss + rlkd_weight * cosine_loss
        return total_loss, {"ce_loss": ce_loss.item(), "kl_loss": kl_loss.item(), "rlkd_loss"
```

Low-Resource language

What is a low-resource language?

- General definition
- Programming definition
- Writing definition

From a programming perspective

- Base/Instruct
- `Elixir`
- ~5-10,000 samples
- Train

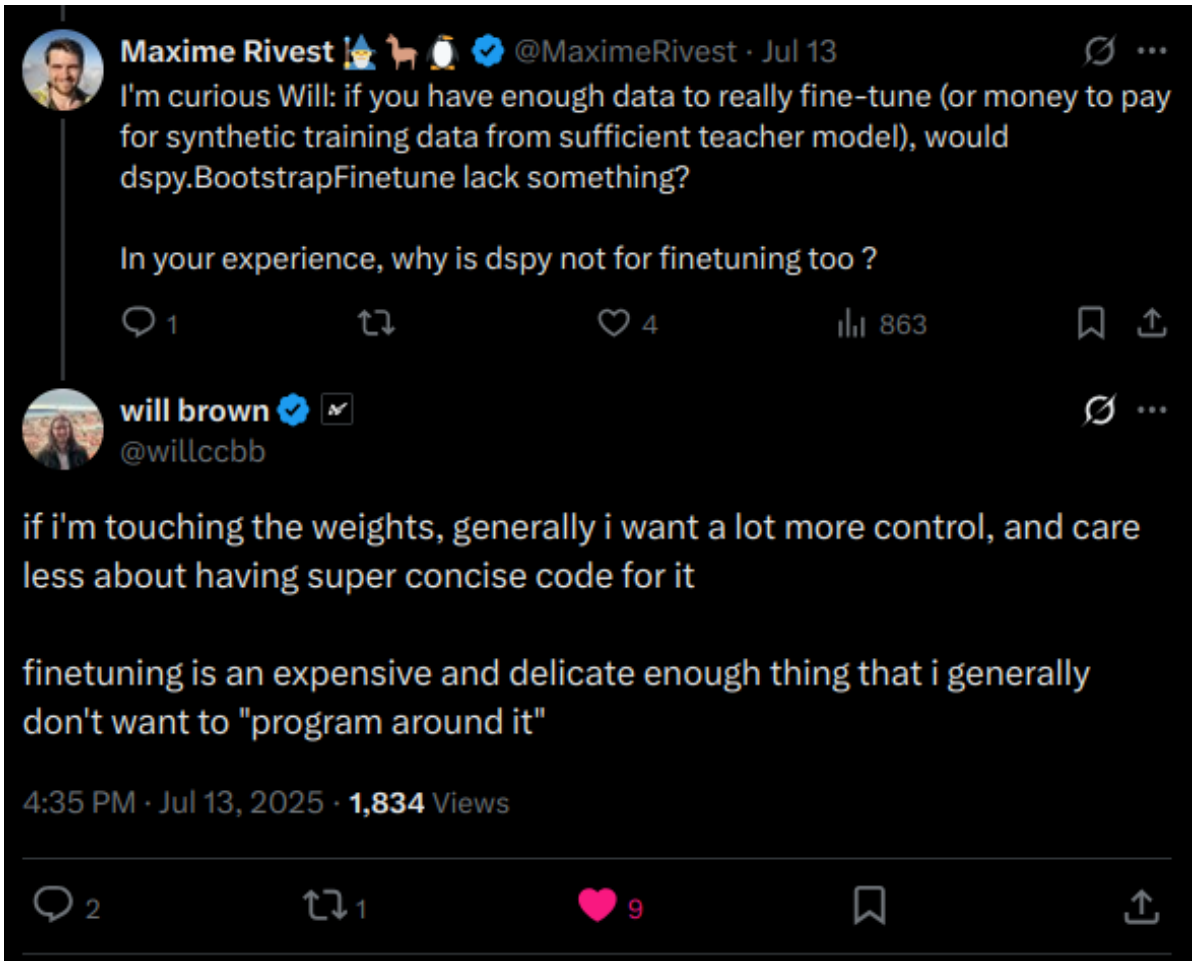
TRL, AutoTrain, Trainer, my head hurts

Many abstractions, what's the best

- If you have no code experience, use no code solutions
- If you have some code experience, use the highest abstraction you can
- If you know what you're doing, use that experience

Find Compute

- `hf-jobs`
- Spinning up your own instance (generally cheaper than FT'ing on closed-api compute)



Deploy

Always be thinking of how this will be pushed

- A model that's never deployed is just an experiment
- End the loop *somewhere* no matter what
- SimonW's `llm`

Is Fine-Tuning Dead?

Is Fine-Tuning Dead?

- No, but for most possibly
- With a few specific cases, its still needed
- It's still worth learning, especially with the rise of local models